Insertion temps réel d'un animateur dans un monde virtuel

Jean-Marc Hasenfratz

MÉMOIRE D'HABILITATION

présenté par :

Jean-Marc Hasenfratz

pour obtenir le diplôme

d'Habilitation à Diriger des Recherches
de

l'UNIVERSITÉ JOSEPH FOURIER
(GRENOBLE I)

Spécialité : Informatique

Insertion en temps réel d'un animateur dans un monde virtuel

Date de la soutenance : 10 octobre 2005

Composition du jury :

Président :	Claude Puech, Pr. Université Joseph Fourier, Grenoble
Rapporteurs :	Alan Chalmers, Pr. University of Bristol, United Kingdom
	Bernard Péroche, Pr. Université Claude Bernard, Lyon
	Xavier Pueyo, Pr. Universitat de Girona, España
Examinateurs :	Pascal Guitton, Pr. Université de Bordeaux
	François Sillion, DR INRIA, Artis, Grenoble

Habilitation préparée au sein du projet Artis
Projet commun entre le CNRS, l'INRIA, l'INPG et l'UJF

À ma mère

Remerciements

"Il était une fois un petit alsacien en DEA à Strasbourg auquel on proposa de faire un thèse à Limoges – il était nul en géo et ne savait même pas où était cette ville ! Son futur Directeur fut franc : «je ne peux rien te promettre, il n'y a pas d'école doctorale en Informatique à Limoges, tu n'auras donc pas de bourse du Ministère, je ferai mon possible pour que tu puisses faire des vacations. La seule chose que je peux te promettre, c'est une machine pour toi seul, une Indy». Le petit étudiant fit son stage de DEA à Limoges, le sujet de recherche lui convenait : "*Le lancer de faisceaux en synthèse d'images*". Il fit un dossier de demande de bourse qu'il présenta à la Région Limousin, il fut boursier. C'était parti pour trois années de travail sous la direction de Djamchid Ghazanfarpour auquel il est toujours reconnaissant pour son enseignement et grâce auquel, il peut aujourd'hui écrire ces pages. Le 5 janvier 1998, le petit doctorant soutient sa thèse entouré de toute sa famille et d'amis de toujours. Un jour, par hasard, il répondit à une annonce parue sur la liste AFIG : «Cherchons Post-Doc pour travailler sur projet européen dans le domaine de la radiosité au sein de l'équipe *i*MAGIS». «Oh là là, jamais ils ne vont vouloir de moi, ils sont bien trop forts, mais bon, je vais tenter ma chance quand même, sait-on jamais». «Bonjour, moi c'est Claude Puech, moi c'est François Sillion, moi c'est Georges Drettakis [...] on attend votre réponse». Le petit bonhomme n'en croyait pas ses oreilles, il allait travailler à Grenoble, dans l'équipe *i*magis dans une superbe ambiance. De temps à autre, il y avait la "salade du chef" ou bien un "vénérable" pour faire des jeux de mots à la cafète – bon, je saute plusieurs années – Septembre 2000, le petit bonhomme devient Maître de Conférences à l'IUP Commerce et Vente. Janvier 2003, le projet *Artis* naît. Je prends de plus en plus de café avec Claude et François, ils m'apprennent le "métier", je les harcèle de questions, ils me guident. Printemps 2005, je rédige ce mémoire et continue à grandir..."

C'est ici l'occasion de remercier les personnes qui m'ont permis de présenter ces travaux.

Merci à Claude Puech pour m'avoir accueilli dans son équipe *i*MAGIS, pour m'avoir suggéré de présenter cette habilitation, pour avoir accepté de présider le jury, pour ses nombreux conseils avisés... Je voudrais plus particulièrement te remercier, Claude, pour ta disponibilité et ta simplicité. Il me semble que tu es un "exemple" pour nombre d'entre nous.

Merci à François Sillion pour m'avoir permis de travailler avec lui dans différents projets européens, pour m'avoir permis d'effectuer mes recherches sur un thème nouveau dans l'équipe, pour m'avoir soutenu dans les moments de doute et pour avoir su canaliser les caprices d'un "Monsieur plus". Merci "Vénérable" !

Merci à Messieurs Alan Chalmers, Bernard Péroche et Xavier Pueyo, rapporteurs et Pascal Guitton, membres du jury, de m'avoir fait l'honneur et le plaisir d'avoir évalué mon travail.

Merci aux co-auteurs des différents papiers sans qui aucune publication n'aurait été possible. Je regrette que l'ordre des auteurs ait une telle importance dans ces publications puisque toujours, chacun d'entre eux s'y est investi pleinement.

Merci à Marc Lapierre pour avoir si efficacement concrétisé pendant près de 3 ans toutes les idées qui me traversaient l'esprit et sans qui les projets CYBER et CYBER-II n'auraient pu aboutir

Merci à toute l'équipe de l'IUP Commerce et Vente pour m'avoir permis d'effectuer ma recherche dans de bonnes conditions et pour la bonne ambiance que j'y retrouve semaines après semaines.

Merci à mes correctrices, Angéline, Marie-Paule et Patricia, sans lesquelles ce mémoire compterait un nombre de fautes d'orthographe qui nécessiterait certainement un *long int* pour le stocker !

Merci à Patricia pour sa disponibilité, son professionnalisme, l'aide au quotidien dans un système administratif parfois «étrange» !, mais aussi pour son amitié et nos discussions non professionnelles qui me rappellent de temps à autre qu'il n'y a pas que la Recherche dans la vie !

Merci aussi à tous ceux qui ne sont pas nommément cités mais qui ont participé, de près ou de loin, à mon travail : les collègues de travail d'Artis, ceux d'Evasion, ceux faisant partie des ACI..., les stagiaires, les doctorants...

Enfin, c'est à mon Ange que j'adresse mes remerciements les plus tendres. Elle, dont le soutien est inconditionnel, me permet de surmonter les batailles que je livre à de vieux démons. Merci à toi, ma femme.

Avant propos

Ce document englobe mes travaux de recherche depuis 2001 même si exceptionnellement, je ferai référence à mes travaux et publications effectués lors de mon Post-Doc. Les projets auxquels j'ai participé et qui ont constitué le cadre de mes recherches sont les suivants :

- **ARCADE** : "*Making Radiosity Usable*", projet européen ESPRIT Reactive Long Term Research Programme, project #24944, octobre 1997-septembre 2000
  http://artis.imag.fr/Projects/Arcade/
- **SIMULGEN** : "*Realistic Simulation of Light for General Environments*", projet européen ESPRIT Open Long Term Research Programme, Project #25772, octobre 1997 - septembre 1998
  http://iiia.udg.es/Simulgen/1stphase.htm
- **DEREVE** et **DEREVE-II** : "*Rendu réaliste et non-photoréaliste, temps réel, de larges scènes animées*", projet Région Rhône-Alpes, DEREVE 1999-2002, DEREVE-II 2003-2006.
- **GrImage** : "*Grid and Image, PC Cluster for Vision, Visualization and Virtual Reality*", 2003 - 2007
  http://www.inrialpes.fr/sed/grimage/
- **Cyber** : "*Insertion temps réel d'un animateur dans un monde virtuel*", Action Concertée Incitative (ACI) «Jeunes Chercheurs» du Ministère de la Recherche, février 2002 - janvier 2005
  http://artis.imag.fr/Projects/Cyber/
- **Cyber-II** : "*Insertion temps réel d'un animateur dans un monde virtuel*", Action Concertée Incitative (ACI) «Masse de données» du Ministère de la Recherche, septembre 2003 - août 2006
  http://artis.imag.fr/Projects/Cyber-II/

Le fait d'avoir été le responsable de l'ACI «Jeunes Chercheurs» CYBER m'a permis de développer mon propre projet de recherche. J'ai pu construire, développer et orienter ma recherche dans le sens qui me semblait le plus intéressant. L'ACI «Masse de données» CYBER-II est la continuité de la première ACI, mais avec des ambitions très nettement supérieures.

Ces deux ACI sont au coeur de ma recherche, c'est à travers elles que j'ai encadré, doctorants, ingénieurs et stagiaires, et publié un certain nombre d'articles. C'est donc tout naturellement que ce mémoire d'Habilitation à Diriger des Recherches porte sur le sujet des deux ACI : *"Insertion en temps réel d'un animateur dans un monde virtuel"*.

Sommaire

1 Introduction

Depuis des années, nous imaginons qu'un jour nous allons pouvoir nous déplacer dans des mondes "virtuels", que tout en restant à Grenoble, nous allons pouvoir visiter la pyramide de Chéops ou la maison de Salvador Dalí à Figueres... Nous voudrions déambuler dans les couloirs du Louvre, nous arrêter devant Mona Lisa, décrocher ce tableau et l'admirer sous tous les angles... Nous voudrions faire le tour du futur tramway de Bordeaux, entrer à l'intérieur et imaginer une ballade en centre ville sous la lumière chaleureuse de l'automne ou un soir au coucher du soleil... Nous voudrions...

Tous ces rêves ont un point commun, il faut que l'on puisse se voir évoluer dans un monde réel ou imaginaire, y trouver sa place et interagir avec lui. Pour cela, une puissance de calcul très importante et du matériel spécifique (caméras hautes résolutions, réseaux hauts débits, très grands écrans, etc.) sont nécessaires et c'est seulement depuis peu que nous en disposons. Nous pouvons donc enfin nous atteler à la tâche. Sous ce côté rêveur, je vais présenter mes travaux de recherche effectués ces quatre dernières années. Ils ont été réalisés au travers des projets CYBER.

Figure 1 : Deux animateurs et un ballon sont à insérer dans le monde virtuel.

1 Description du projet CYBER

Le projet CYBER se place dans le cadre de la réalité virtuelle/augmentée. Il s'agit "d'insérer" une personne dans un monde virtuel. Cette personne doit **être totalement libre** de ses mouvements, elle ne doit donc pas porter d'équipement particulier comme des capteurs. La personne doit pouvoir **être accompagnée d'autres personnes et/ou d'objets quelconques** (voir *Figure 1*), nous ne nous limitons pas à l'insertion "d'éléments" connus *a priori* (par la suite, nous parlerons d'un "animateur" pour désigner une ou plusieurs personnes avec ou sans objets sup-

plémentaires). L'animateur doit pouvoir **interagir** avec le monde virtuel, il faut donc un traitement **temps réel** (nous considérons que 25 images par seconde est la limite inférieure de vitesse d'affichage). L'animateur inséré dans le monde virtuel doit s'intégrer parfaitement, il faut donc que le rendu soit le plus **réaliste** possible en particulier, il faudra traiter les ombres portées et les conditions d'éclairage.

En pratique, un ensemble de caméras filme l'animateur sous différents angles. À partir de ces flux vidéo, nous pouvons reconstruire la géométrie 3D de l'animateur. Ces différents flux sont aussi utilisés au moment de l'insertion de l'animateur dans le monde virtuel afin que l'on puisse tourner autour de lui. Des ombres portées de l'animateur dans le monde virtuel sont calculées et affichées. L'ensemble des processus est présenté ci-contre (voir *Figure 2*).

2 Organisation du mémoire

Nous allons prendre le projet CYBER comme prétexte pour aborder mes thèmes de recherche de ces dernières années. Nous commencerons par une étude relativement technique mais importante sur l'élaboration de la **plate-forme** matérielle avant d'aborder les problèmes d'**acquisition des silhouettes** de l'animateur et de sa **modélisation**. Nous traiterons ensuite du **rendu réaliste** de l'animateur et des **interactions** possibles avec le monde virtuel.

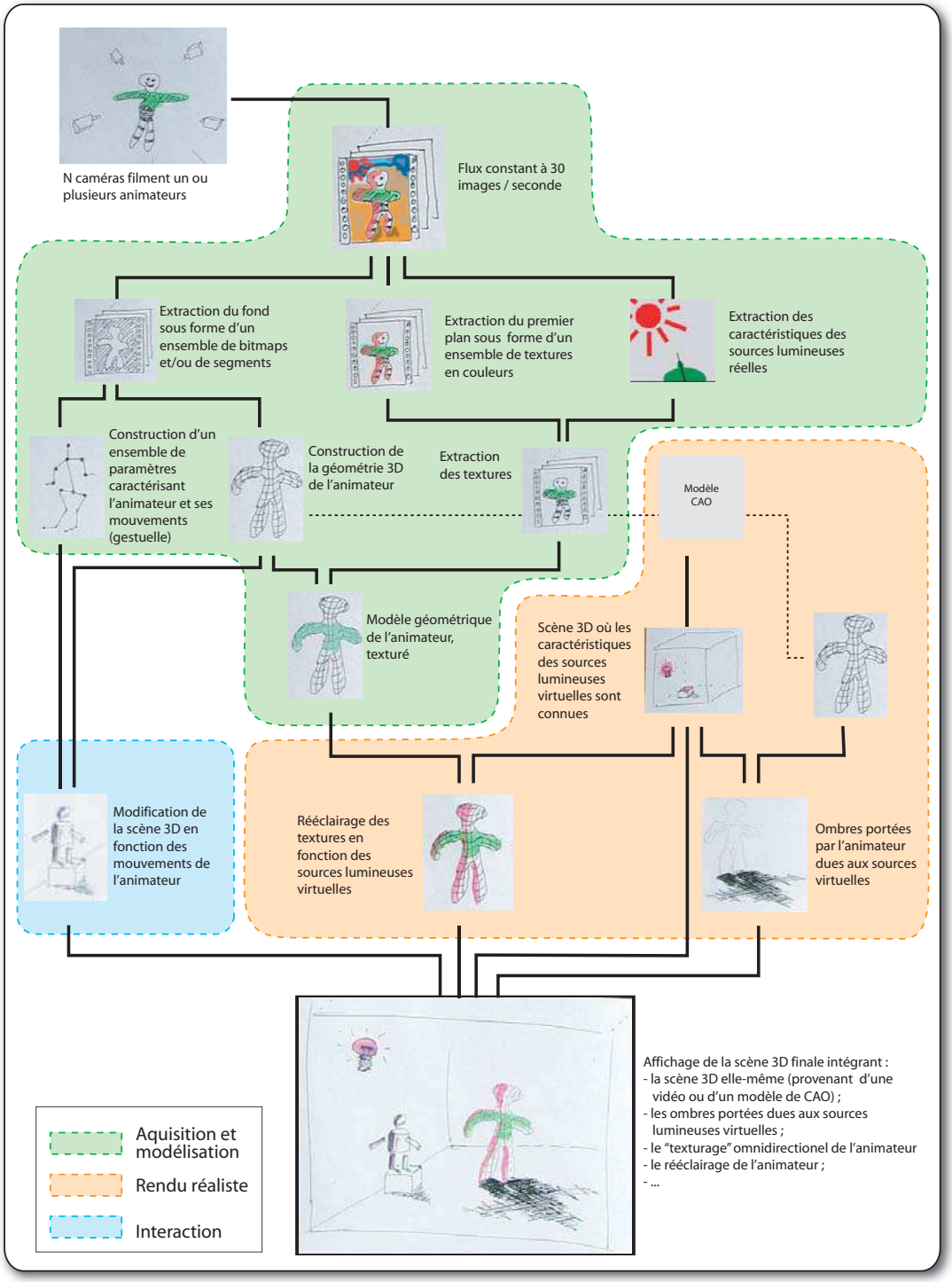


Figure 2 : Schéma général du projet CYBER

2 Plate-forme CYBER

1 Introduction

Le côté technique est généralement passé sous silence lors des recherches, le plus souvent parce que son importance est faible et que l'on considère cette partie comme un "simple problème d'ingénieur". Dans notre cas, les choix matériels font partie intégrante de notre recherche. En effet, au commencement du projet, début 2002, il a fallu définir les besoins matériels en termes d'occupation du bâtiment, de type d'éclairage, de caméras, etc. Les choix effectués allaient contraindre nos recherches, les solutions choisies devaient donc être pérennes. Cette période d'analyse des besoins a pris du temps mais nous a permis par la suite d'avoir à disposition une plate-forme évolutive et répondant à tous nos besoins. Marc Lapierre, ingénieur sur les ACI CYBER nous a énormément aidé dans cette tâche.

Notons que la première ACI a démarré en février 2002 et que la deuxième se terminera en août 2006, soit une durée totale du projet de 56 mois. La présentation correspond donc à la situation actuelle[1] sachant que le projet évolue régulièrement au cours des mois.

Ce chapitre peut être lu comme un manuel pour bien construire son studio virtuel. Certes, ce n'est pas de la recherche fondamentale mais c'est une bonne manière de s'immerger dans le sujet. Nous allons étudier les principaux éléments de la plate-forme : la salle, les éclairages, les caméras et les ordinateurs. Nous allons consider, pour chacun d'eux, les contraintes à respecter, nos choix et leurs motivations et résumer l'expertise acquise.

1. La rédaction de ce mémoire a été terminée le 12 juillet 2005.

Figure 1 : Salle de réunion avant réaménagement.

Figure 2 : Animateur et écran de projection.

2 La salle

La salle a été le premier souci car il n'existait dans le bâtiment aucun espace prévu à cet effet. Nous avons donc transformé une salle de réunion (voir *Figure 1*) en "salle de capture pour la réalité virtuelle".

Les contraintes à respecter étaient les suivantes :

- La salle devait être suffisamment grande pour avoir du recul pour placer les caméras de sorte à filmer la totalité du volume où évoluerait l'animateur (8 m^3).
- Le plafond devait être suffisamment haut pour placer des éclairages et des caméras au-dessus de l'animateur.
- Les objets spéculaires étaient à supprimer pour éviter des problèmes de reflets des éclairages et de saturation des images.
- Il fallait placer un écran et un vidéo projecteur pour que l'animateur puisse se voir évoluer (voir *Figure 2*).
- Il fallait définir une zone pour un public (chercheurs travaillant sur des expérimentations ou visiteurs) qui ne soit pas dans le champ des caméras.
- Il fallait pouvoir supprimer la lumière du jour pour ne pas dépendre d'un éclairage naturel changeant, difficile à gérer lors de l'extraction de fond.

À partir de ces contraintes, nous avons dessiné un plan de cette nouvelle salle (voir *Figure 3*).

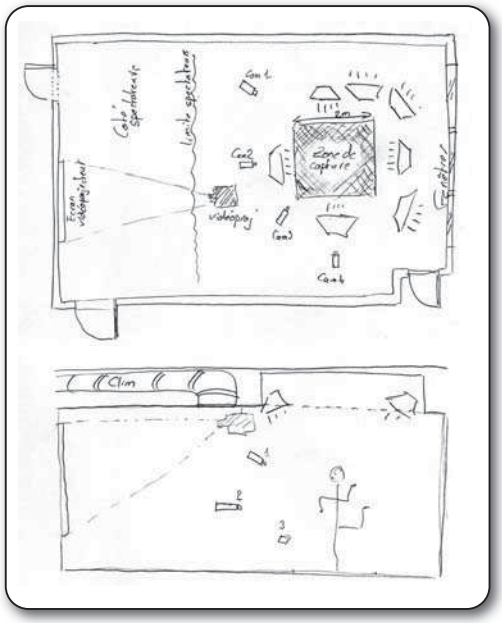


Figure 3 : Plan de la salle.

Les choix effectués ont été les suivants :

- Nous avons rehaussé au maximum le plafond (voir *Figure 4*).
- Des rampes ont été installées au plafond pour y fixer les éclairages et pouvoir les déplacer facilement (voir *Figure 5*).
- Un vidéo projecteur a été accroché au plafond (voir *Figure 5*).
- Des stores ont été installés pour supprimer l'influence de l'éclairage extérieur.
- Les cadres en aluminium des fenêtres ont été repeints avec une peinture mate pour éviter les reflets.

Figure 4 : Rehaussement d'une partie du plafond.

Cette salle était parmi les premières du genre dans un lobaratoire de recherche, nous n'avions donc que très peu d'exemple sur lesquels s'inspirer. Avec le recul, certains choix seraient à corriger.

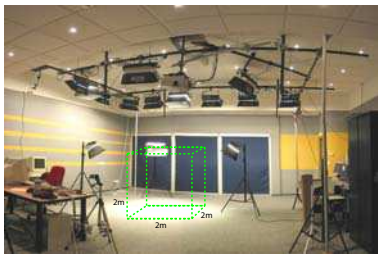


Figure 5 : Salle réaménagée permettant une zone de capture de 8m^3.

Expertise :

- Le sol est une moquette très spéculaire ce qui ne facilite pas l'extraction de fond, il faudrait une surface mate ;
- Les stores que nous avons posés sont bleu foncé ce qui prête à confusion lorsque l'on dit que nous faisons de l'extraction sur fond quelconque. De plus, l'animateur porte souvent des habits sombres qui se détachent mal du bleu foncé. Une couleur beaucoup moins présente dans les vêtements aurait été plus judicieux. Dans notre cas, nous aurions pu utiliser le même jaune que les bandes sur le mur.

3 Les caméras

Le choix des caméras a probablement été la décision la plus difficile à prendre. En effet, il existait à époque que très peu de laboratoire de recherche disposant d'une plate-forme similaire de laquelle s'inspirer et les informations techniques n'étaient pas disponibles.

Les contraintes à respecter étaient les suivantes :

- Le débit des caméras devait être d'un minimum de 25 images par seconde pour espérer faire du temps réel.
- La focale devait être compatible avec la taille de la salle et du volume de capture.
- Le coût unitaire devait être modéré puisqu'il nous fallait un minimum de quatre caméras.
- La partie logicielle ("pilote") devait exister et être fiable.
- Les images ne devaient pas être entrelacées pour améliorer la qualité de l'extraction de fond.

- La connectique devait pouvoir atteindre au minimum 5 mètres.

Début 2002, peu de constructeurs proposaient des caméras numériques en couleurs. Nous avions donc le choix entre utiliser des caméras analogiques avec des cartes d'acquisition ou nous lancer dans l'achat de caméras numériques.

Le choix a été le suivant :

Figure 6 : Caméra numérique Sony DWF-VL500.

- Nous avons choisi des caméras numériques pour éviter les problèmes d'entrelacement des images.
- Nous avons opté pour les caméras au standard IEEE 1394 (ou Firewire) utilisées sous Windows pour l'existence et la fiabilité des pilotes.
- Nous avons donc acheté quatre caméras Sony DFW-VL500, 640x480 YUV(4:2:2)[1], avec un zoom 12x capable de produire des image à 30fps (voir *Figure 6*).

Expertise :

Il existe actuellement un choix plus grand de caméras numériques aussi bien chez Sony que Marlin ou d'autres (voir *Tableau 2.1*). De plus, la nouvelle interface *camera link* permet des débits plus importants. Enfin, les pilotes sous Linux sont disponibles et fiables.

TABLEAU 2.1. Principaux modèles de caméras numériques disponibles en avril 2005

Modèle	Interface	Résolution	Fps	Ko/img	Débit[1]
SONY DFW VL500	IEEE 1394	640x480	30	600	140 Mb/s
SONY DFW X700	IEEE 1394	1024x768	15	1 536	180 Mb/s
SONY DFW sX900	IEEE 1394	1280x960	7.5	2 400	140 Mb/s
AVT MARLIN 046C	IEEE 1394	780x580	37	884	255 Mb/s
AVT MARLIN 145B2	IEEE 1394	1392x1040	10	2 828	220 Mb/s
JAI CV-M7+CL	caméra link	1380x1030	24	1 735	325 Mb/s

1. Encodage YUV(4:2:2) : Y représente la luminance, U et V respectivement les composantes rouge et bleue (chrominances). En mode 4:2:2 nous utilisons 4 valeurs Y suivi de 2 valeurs U et 2 valeurs V pour coder la couleur. Il existe différentes manières de compresser ces données soit sur 12 bits (MPEG) soit sur 16 bits (Cinepak). Pour plus d'informations, voir le site http://alcazar.xbecom.com/videogarage/spec/YUV.html

1. Pour les interfaces ieee1394, le débit est calculé sur la base d'images au format YUV4:2:2 avec une moyenne de 16 bits par pixel. Le format *caméra link* quant à lui utilise ici 10 bits par pixel.

4 L'éclairage

Pour éclairer la salle, nous avons pris contact avec une société construisant des plateaux de télévision. Nous avons aussi étudié attentivement la documentation "*Catalogue Concepteur de Lumière*" proposée sur le site *www.balcar.com*. Nous avons pu ainsi comprendre comment utiliser les éclairages pour limiter les ombres portées. L'idée est de saturer de lumière les murs et le sol pour "brûler" les ombres portées de l'animateur.

Les contraintes à respecter étaient les suivantes :

- L'éclairage devait être homogène et sans variation dans le temps pour que l'extraction de fond fonctionne correctement.
- Le nombre de sources lumineuses devait être suffisamment important pour éclairer toute la salle et réduire au maximum les ombres portées.
- Vu le nombre de sources lumineuses, celles-ci devaient dégager un minimum de chaleur.

Nous nous sommes très rapidement dirigés vers le matériel utilisé par les professionnels sur les plateaux de télévision.

Figure 7 : Balcar accrochés à la rampe et Balcar sur trépieds..

Les choix effectués ont été les suivants :

- Un ensemble de 7 Balcar Duolite DMX, 125 W, 3800 cd accrochés au plafond pour éclairer les murs (voir *Figure 7*).
- Un ensemble de 4 Balcar Quadlite DMX 250 W, 8300 cd (voir *Figure 8*) accrochés au plafond pour éclairer l'animateur et le sol.
- Une autre série de 3 Balcar Quadlite DMX 250 W, 8300 cd fixés sur des pieds et braqués sur le sol au niveau de la zone de capture pour diminuer au maximum les ombres portées.

Figure 8 : Balcar Quadlite, 125 W, 3800 cd.

Expertise :

Une très grande quantité de lumière est effectivement indispensable pour diminuer les ombres portées. Notons qu'il est quasiment impossible de supprimer la totalité des ombres portées, particulièrement au niveau des pieds. Le choix de la température des lampes utilisées est important. En effet lors de différents tournages de films, nous avons eu des problèmes pour obtenir une bonne balance des blancs. Ceci parce que nos éclairages sont de type "lumière du jour" alors que celui du vidéo projecteur

utilisé pour projeter le monde virtuel est plutôt de type tungstène. Le choix des tubes doit donc être en accord avec les autres sources de lumières potentielles (principalement les vidéos projecteurs).

5 Les calculateurs

Le choix des calculateurs a été relativement simple puisque nous avions décidé du type de caméras à utiliser (numériques avec des interfaces IEEE 1394). De plus, nous disposions d'un supercalculateur SGI Onyx.

Les contraintes étaient les suivantes :

- Chaque machine, associée à une caméra, devait être capable de traiter en temps réel le flux vidéo ainsi que l'extraction de fond ;
- Une machine devait être capable de recevoir via un réseau à 1Gb/s l'ensemble des silhouettes vues par la caméra, d'effectuer la reconstruction, de traiter les interactions et d'effectuer le rendu de l'ensemble de la scène.

Le choix a été le suivant :

- Un ensemble de 4 PC standards associés aux 4 caméras.
- Le supercalculateur SGI Onyx 3400 (voir *Figure 9*) pour l'intégration 3D — les PC de l'époque n'étaient pas assez puissants pour remplacer une telle machine.

Figure 9 : Supercalculateur SGI Onyx 3400 (photo © INRIA/R. Lamoureux)

L'ensemble des machines était connecté à un réseau 1 Gbits/s (voir *Figure 10*).

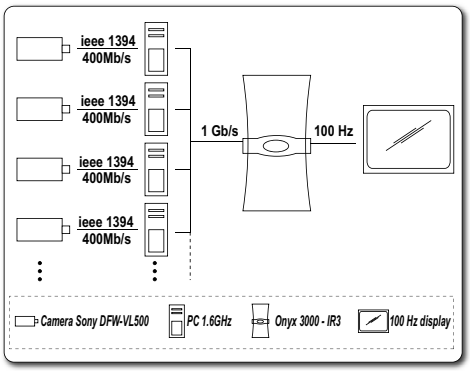


Figure 10 : Architecture centrée sur d'un seul gros calculateur.

Expertise :

L'architecture utilisée a permis d'obtenir rapidement de bons résultats. Il est sûr qu'au vu de l'évolution des PC en puissance de calcul et en trai-

tement graphique, l'utilisation d'un supercalculateur n'est plus justifiée. Une grappe de PC semble un choix bien moins onéreux et plus facile à faire évoluer dans le temps.

6 Contributions

Dans ce chapitre, nos contributions sont principalement d'ordre technique. Nous n'avons pas eu à nous poser des questions de "recherche fondamentale". Néanmoins, nous avons testé un ensemble de configurations matérielles et proposé une solution cohérente pour construire une plate-forme d'acquisition fonctionnelle, fiable et pérenne. Il est évident que le matériel évolue très vite, en particulier les caméras et les ordinateurs, mais les contraintes sont toujours celles que nous avons énoncées. L'ossature générale reste donc valable et correspond à notre contribution.

Figure 11:12 vidéo projecteurs (photo © INRIA/Jim Wallace).

7 Et maintenant ?

Les objectifs actuels sont plus ambitieux. Nous voulons plus de précision dans la reconstruction et un affichage de meilleure qualité. Ceci nous a amené à utiliser une grappe de plus d'une dizaine de PC, un mur d'images composé de 4x3 vidéos projecteurs (voir *Figure 11*) correspondant à une résolution de l'ordre de 4800x3000 pixels à la place des 1024x780 pixels habituels (voir *Figure 12*). Cette évolution matérielle fait partie du projet GrImage (Grid and Image) [GrImage]. Une description complète de cette infrastructure est proposée dans [HM04].

Figure 12: "Mur d'images" de 2x2.7m (photo © INRIA/Jim Wallace).

3

Acquisition des silhouettes

Définition *Silhouette* : contour représentant le profil de l'animateur du point de vue d'une des caméras.

1 Introduction

Dans ce chapitre, nous allons aborder deux thèmes de recherches importants : l'**extraction d'objets sur un fond quelconque** et le **calcul de leurs silhouettes** depuis différents point de vues.

L'extraction d'objets sur un fond quelconque consiste à différencier la région de l'image contenant l'information pertinente de l'arrière-plan de l'image. Les méthodes existantes reposent sur l'hypothèse d'un arrière-plan statique qui peut être appris *a priori* (voir *Figure 1*). La possibilité d'utiliser un fond quelconque à la place des fonds bleus ou verts habituels est très importante pour diverses raisons :

- elle apporte une plus grande liberté pour les tournages en disposant d'un plateau qui n'est plus limité à la surface peinte ou recouverte de tissus bleu ou vert ;
- le coût financier est moindre ;
- de nouvelles applications deviennent possibles comme le suivi de joueurs sur un terrain de jeu, les jeux vidéos où le joueur serait filmé chez lui et directement incrusté...
- le suivi et la surveillance vidéo (dans des bâtiments, sur des autoroutes...).

Nous allons aborder dans ce chapitre l'acquisition des silhouettes dans le projet CYBER, l'extraction de fond et la détermination des silhouettes.

Figure 1 : Extraction sur fond quelconque (photomontage).

Nous présenterons nos contributions et les approches sur lesquelles nous voudrions travailler.

2 Acquisition des silhouettes dans CYBER

L'extraction sur fond quelconque et la détermination de la silhouette sont deux points très liés dans notre application. Techniquement, nous disposons, en entrée, des flux d'images d'une scène dynamique contenant un personnage et nous devons produire en temps réel un ensemble de silhouettes (voir *Figure 2*).

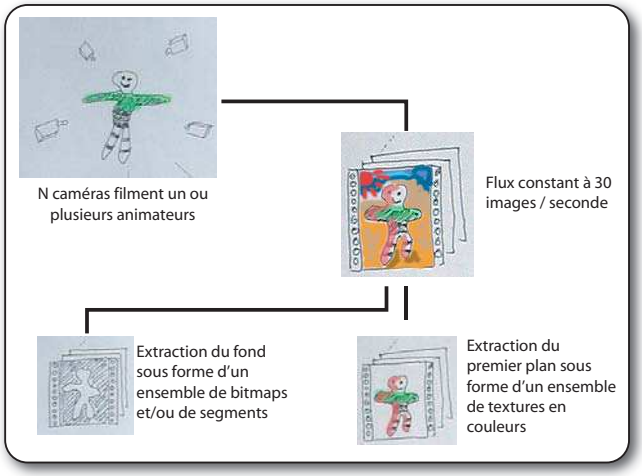


Figure 2:Module "Acquisition". À partir de n caméras, nous construisons n silhouettes ainsi que n images texturées.

Nous ne nous limiterons pas à un seul format de silhouette afin de pouvoir explorer plusieurs types de reconstructions, par contre, nous respecterons la contrainte temps réel.

Nous expliquerons par la suite pourquoi la précision de l'extraction de fond n'a pas besoin d'être très grande lors du processus de construction du modèle géométrique de l'animateur (voir *Chapitre 4–Modélisation, 4.3 Précision des silhouettes, page 28*). Cette extraction servira aussi à habiller le modèle géométrique. Nous verrons que dans ce cas, ce sera plutôt la résolution et la qualité de l'image (son piqué) qui seront importantes (voir *Chapitre 5–Rendu réaliste, 6.2 Qualité des images, page 47*).

3 Extraction de fond

Lorsque l'on se place dans le cadre de productions cinématographiques, nous trouvons soit des fonds bleus soit des fonds verts [Par02]. En

effet, ce sont le plus souvent des acteurs que l'on veut isoler du fond. Les couleurs bleues ou vertes facilitent alors la séparation de la couleur chair de la peau avec celle du fond.

3.1 Fond uniforme

Nos premiers travaux effectués dans ce domaine l'ont été au cours du stage de DEA [Fai01]. Nous avons travaillé sur l'algorithme de "chroma key" [SB96], [BL99]. Cette étude nous a permis de nous familiariser avec les techniques d'extraction de fond (voir *Figure 3*). En fait, nous avons vérifié qu'il était relativement facile de faire une extraction sur fond uniforme à condition de maîtriser l'éclairage (suffisamment puissant, bien orienté, etc.). L'avancée scientifique a été réalisée au niveau de l'optimisation de calcul pour obtenir des résultats en temps réel.

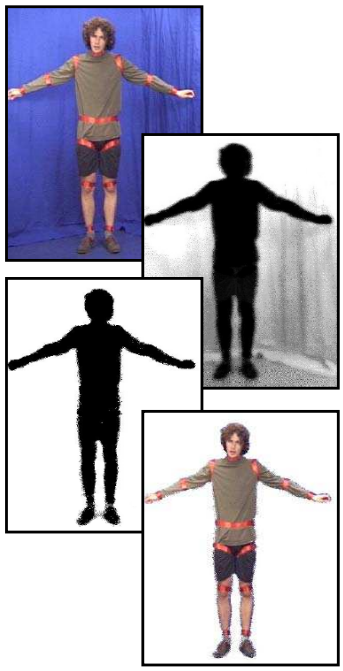


Figure 3 : Image sur fond bleu, composante alpha, alpha filtré et image résultante.

L'extraction sur fond uniforme est une technique éprouvée depuis des années, de nombreux systèmes professionnels temps réel existent sur le marché et remplissent leur fonction à merveille. Généralement, tous les traitements sont directement effectués par de l'électronique dédiée. La publication de A.R. Smith et J.F. Blinn [SB96] permet de se faire une idée très précise sur le sujet avec des références aux différents brevets sur cette technique.

3.2 Fond quelconque

L'extraction sur fond quelconque reste du domaine de la recherche. La littérature sur le sujet est importante. Nous pouvons retenir la comparaison entre différentes méthodes faite dans [TKBM99]. D'autres références intéressantes sur le sujet sont [WFSM02], [CPP03], [YPLL04].

Les hypothèses de travail sont généralement les suivantes : l'arrière plan ne bouge pas (seules des ombres peuvent s'ajouter) et les caméras sont calibrées une fois pour toute. Le principe d'une extraction sur fond quelconque se décompose en trois étapes (voir *Figure 4*) :

- Les caméras "apprennent" l'arrière plan en filmant quelques secondes l'environnement "vide". Notez qu'une seule image du fond pour chacune des caméras ne suffit pas. En effet, les capteurs électroniques oscillent dans le temps et les couleurs mesurées ne sont donc pas constantes. Il faut donc filmer quelques secondes pour moyenner ces couleurs.
- Les nouveaux éléments (objets, personnes...) sont introduits sur la zone de tournage.

- Une “différence” entre les images apprises et ce qui est filmé, est calculée. Elle permet de retrouver les nouveaux éléments. Le problème est de ne pas prendre en compte les ombres portées dues à l’ajout d’éléments dans la scène pour extraire exactement les contours des nouveaux objets ou personnes.

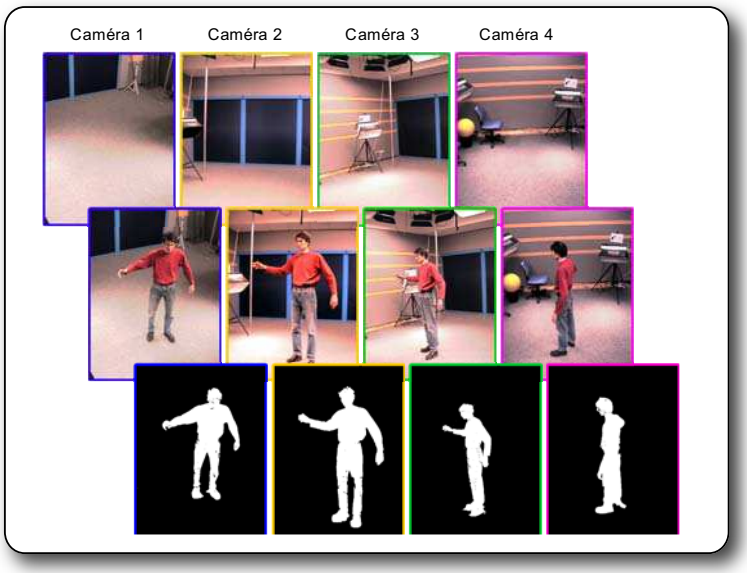


Figure 4 : Extraction sur fond quelconque à partir de 4 caméras disposées autour de l’animateur.

Dans le cas du projet CYBER, nous pouvons alléger les contraintes. En effet, il n’est pas nécessaire d’avoir une extraction très performante puisque le bruit disparaît au moment de la reconstruction volumique comme nous le verrons pas la suite (voir *4.3 ”Précision des silhouettes”, page 28*).

Les détails techniques de l’implémentation de l’extraction sur fond quelconque ont été publiés (voir *”Real-Time Capture, Reconstruction and Insertion into Virtual World of ...”*, paragraphe “*4.2 Backgroud Substraction*”, *page 95*).

4 Détermination des silhouettes

Une fois l’extraction de fond effectuée, nous disposons d’un ensemble d’images noir et blanc correspondant, pour chacune des caméras, aux masques des nouveaux éléments. Nous verrons dans le chapitre suivant (voir *Chapitre 4–Modélisation, page 23*) que, suivant l’approche utilisée, nous utiliserons des contours sous forme de segments ou des représentations sous forme de bitmaps.

4.1 Silhouettes sous forme de segments

Il y a plusieurs raisons pour vouloir travailler avec des silhouettes sous la forme de segments plutôt que directement avec les bitmaps. Ainsi la taille de la structure de données est plus petite et donc moins coûteuse à transmettre sur le réseau. Les algorithmes de reconstruction du modèle 3D peuvent travailler avec des polygones.

Il existe de nombreux algorithmes de segmentation d'images, dans le cas de CYBER, nous utilisons [DR95].

4.2 Silhouettes sous forme de bitmap

Le résultat direct de l'extraction de fond est une bitmap (avec des valeurs pouvant être comprises entre 0 et 1) à laquelle on peut associer ou non des valeurs de transparence. Comme nous le verrons par la suite (voir *Chapitre 4–Modélisation, 4.1 Approche volumique, page 26*), il peut être très intéressant de travailler directement sur ces images pour construire le modèle géométrique de l'animateur.

4.3 Transmission des silhouettes

Dans le cas du projet CYBER, nous utilisons une architecture distribuée où les informations transitent sur un réseau dédié. Même si celui-ci est à très haut débit (1 Gbit/seconde), il peut être rapidement saturé lorsque l'on augmente le nombre de caméras. Il est donc intéressant de faire une étude sur la proportion de surface occupée par l'animateur et par extrapolation du nombre de caméras que l'on peut utiliser simultanément sans saturer le réseau.

Nous nous plaçons dans l'hypothèse où il n'y a qu'un seul animateur à filmer et à insérer dans le monde virtuel. Nous allons quantifier la surface occupée par un animateur et le poids des images à transmettre. Nous distinguerons les bitmaps pouvant être utilisées lors du processus de reconstruction du modèle géométrique et les images en couleurs servant à l'habillage du modèle dans l'étape de rendu réaliste. Par extrapolation, nous évaluerons le nombre de caméras que l'on peut espérer utiliser réellement.

4.3.1 Proportion de surface occupée par un animateur

Considérons les résultats de l'extraction de fond effectuée lors d'une de nos manipulations (voir *Figure 5*). Nous pouvons constater que la surface occupée par l'animateur (pixels blancs) varie suivant la caméra entre

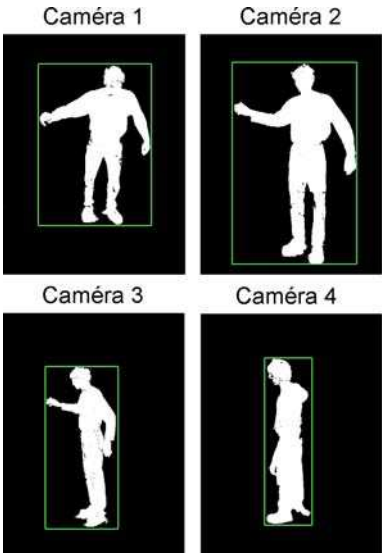


Figure 5 :Quatre images avec les boîtes englobant les silhouettes (en vert).

10% et 21% de la surface totale de l'image (voir *Tableau 3.1*, colonne "*Silhouette*").

Une première optimisation très classique est de considérer uniquement la boîte englobante de la silhouette. Seuels les positions, tailles et contenus de ces boîtes seront transmis sur le réseau. En considérant le même exemple que précédemment (voir *Figure 5*), les surfaces des boîtes englobantes occupent de 17% à 56% de l'image complète (voir *Tableau 3.1*, colonne "*Boîtes englobantes*"). Cette optimisation est triviale à mettre en place et permet de gagner dans la plupart des cas plus de 50% de bande passante (voir *Tableau 3.1*). Nous pouvons aussi constater que le poids d'une bitmap correspond à moins de 10% du poids de l'image en couleur.

TABLEAU 3.1. Proportion maximale de la surface de la boîte englobante et de sa silhouette

	Image 640x480	Boîte englobante	Silhouette
Caméra 1	100%	41%	15%
Caméra 2	100%	56%	21%
Caméra 3	100%	26%	10%
Caméra 4	100%	17%	10%
Maximum	**100%**	**56%**	**21%**
Bitmap[1]	**37.5 Ko**	**21 Ko**	**7.88 Ko**
Couleur[2]	**600 Ko**	**336 Ko**	**126 Ko**

1. Poids d'une bitmap avec un bit par pixel
2. Poids d'une image au format YUV:4:2:2 (16 bits par pixel)

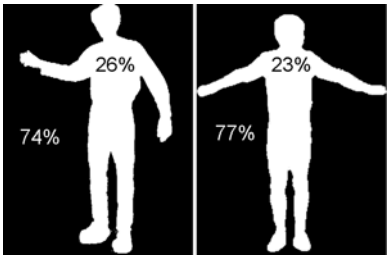


Figure 6 : À gauche : l'animateur occupe toute la hauteur de l'image, à droite : l'animateur occupe toute la largeur de l'image.

Considérons les cas maximisant la surface occupée par l'animateur (voir *Figure 6*) correspondant aux configurations suivantes :

- une caméra est face à l'animateur ;
- un animateur ayant une taille de la tête au pied correspondant à la hauteur de l'image ;
- un animateur écartant ses bras de sorte à couvrir toute la largeur de l'image.

Dans ces cas, un quart seulement (26% et 23%) de la surface de l'image est occupé par l'animateur. Ces cas restent néanmoins purement théoriques puisque le champ de vision des caméras est nettement plus large que celui utilisé dans cet exemple extrême. En effet, l'animateur doit pouvoir élever les bras puis les écarter sans que l'on ait besoin de changer la focale des caméras. Nous reprendrons donc dans la suite, l'extrait de séquence précédent (voir *Figure 5*).

4.3.2 Nombre de caméras utilisables simultanément

Etudions à présent les débits nécessaires en fonction du nombre et du type de caméras utilisées et des données transmises (images entières, boîtes englobantes ou contenu des silhouettes). Une première approche consiste à envoyer les résultats bruts, en couleur, sur le réseau, c'est-à-dire sans aucune compression. Ceci a l'avantage d'éviter de passer du temps en compression/décompression. Bien sûr, si besoin était, il faudrait étudier soigneusement le type de compression à utiliser. Considérons parmi les caméras proposées (voir *Tableau 2.1, page 8*) uniquement celles permettant un débit temps réel, c'est-à-dire supérieur à 20 images par seconde (voir *Tableau 3.2*).

TABLEAU 3.2. Modèles de caméras numériques soutenant un débit de plus de 20 images par seconde.

Modèle	Résolution	Fps	Ko/img	Débit [Mb/s]
SONY DFW VL500	640 x 480	30	600 Ko	140 Mb/s
AVT MARLIN 046C	780 x 580	37	884 Ko	255 Mb/s
JAI CV-M7+CL	1380 x 1030	24	1 735 Ko	325 Mb/s

Dans le graphe ci-dessous (voir *Figure 7*), nous avons représenté les débits réseau nécessaires en fonction du nombre de caméras. Nous avons supposé transmettre les images dans leur résolution maximale, transmettre uniquement la boîte englobante (en supposant qu'elle couvrait au maximum 57% de l'image complète) et transmettre uniquement le contenu des silhouettes (en supposant qu'il couvrait au maximum 21% de l'image complète).

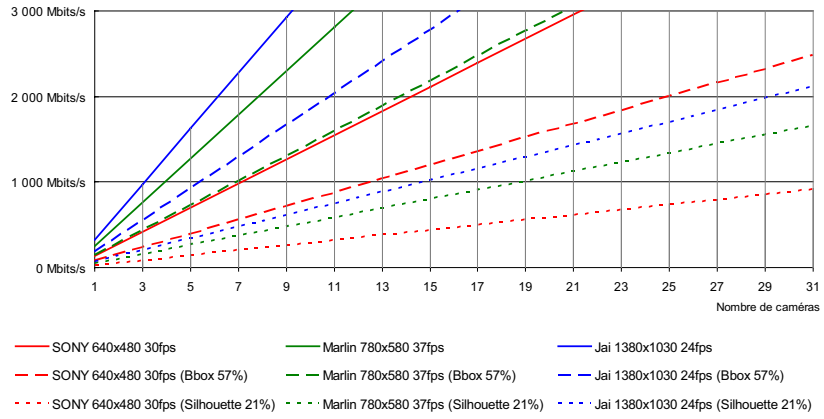


Figure 7:Débits des caméras en fonction de leur type et de leur nombre et des données transmises (images entières, boîtes englobantes ou contenu des silhouettes)

Ces résultats sont intéressants puisqu'ils montrent clairement les limites d'un réseau à 1 Gbits/s. Ainsi, transmettre les images dans leur résolution maximale n'est plus possible lorsque l'on désire travailler avec plus de sept caméras. Au contraire, le simple fait d'utiliser les boîtes englobantes permet de travailler théoriquement avec 7 caméras en haute résolution (1380x1030 pixels) et 13 caméras en 640x480 pixels à 30 Hz. Finalement, si l'on transmet uniquement le contenu des silhouettes, nous pouvons espérer utiliser quinze caméras en haute résolution et plus de trente en basse résolution (640x480 pixels).

Cette étude est bien sûr théorique, elle se base sur l'analyse des images produites à un instant d'une séquence avec un seul animateur et avec seulement quatre caméras, mais elle permet de fixer un peu mieux les idées et les limites d'un réseau à haut débit (1 GBits/s).

5 Contributions

Nous avons développé un algorithme d'extraction sur fond quelconque. Celui-ci est robuste et fonctionne en temps réel [HLGB03] (voir *"Real-Time Capture, Reconstruction and Insertion into Virtual World of ...", page 91*), [HLS04] (voir *"A Real-Time System for Full Body Interaction", page 125*). Cet algorithme permet d'obtenir les silhouettes de l'animateur sous la forme de bitmap ou de segments.

Nous avons aussi proposé une étude théorique pour quantifier l'information utile à transmettre sur le réseau afin de fixer un cadre de recherche pour les prochaines extensions de la plate-forme (voir *4.3 "Transmission des silhouettes", page 17*).

6 Et maintenant ?

Une approche sur laquelle je souhaiterais travailler est l'augmentation de la résolution des silhouettes. En effet, nous avons besoin de précision pour distinguer par exemple les doigts des mains pour obtenir une interaction fine avec le monde virtuel. De plus, nous travaillons avec des "écrans" ayant des résolutions de plus en plus grandes comme des "murs d'images" (voir *Chapitre 2–Plate-forme CYBER, 7. Et maintenant ?, page 11*) comportant près 4800x3000 pixels [GrImage]. Il devient alors difficile d'utiliser des images d'un animateur provenant de caméras ayant une résolution sensiblement plus faible (actuellement la résolution maximale serait d'environ 1300x1000 pixels – voir *Tableau 2.1, page 8*). Nous pouvons bien sûr parier sur l'augmentation de la résolution des capteurs numériques mais de la même manière, nous pouvons parier sur l'augmentation des dimensions et résolutions des "murs d'images". Il y aura donc toujours une course à la résolution comme il y a une course à la puissance.

Pour gagner en résolution, soit nous achetons systématiquement les caméras ayant la plus haute résolution et probablement les plus chères ce qui "résout" ponctuellement le problème, soit nous adoptons une attitude de chercheur pour trouver une solution pérenne. C'est bien sûr cette dernière qui m'intéresse et pour cela, je voudrais essayer de coupler plusieurs caméras ayant des résolutions convenables et des prix abordables. Ainsi un bloc de deux ou quatre caméras d'une résolution d'environ 1300x1000 permettrait d'obtenir respectivement des images d'environ 1300x2000 pixels ou 2600x2000 pixels – en pratique la résolution serait légèrement moindre pour prendre en compte le recouvrement des régions (voir *Figure 8*). Chacune de ces caméras ne filmerait qu'une partie du volume à capturer.

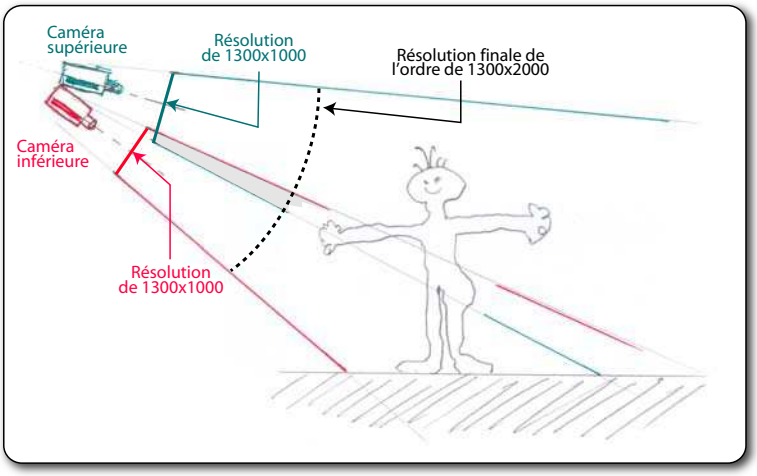


Figure 8 : Utilisation d'un bloc de deux caméras pour doubler la résolution.

Le problème scientifique serait alors de développer des algorithmes de calibrage des caméras extrêmement précis pour minimiser les problèmes au niveau du recouvrement des régions filmées. Des travaux dans ce domaine existent déjà mais dans un cadre très différent du nôtre. En effet, [WJVT+05], proposent de regrouper une centaine de caméras de faible résolution (640x480) pour obtenir des images allant jusqu'à 6900x3500 pixels. Néanmoins, l'installation d'un tel système est très complexe et d'un prix prohibitif.

Si l'on utilise les systèmes permettant d'obtenir des images de haute résolution, il n'est pas sûr que les algorithmes d'extraction de fond passent à l'échelle et restent temps réel. Enfin, comme nous l'avons étudié précédemment (voir *4.3 "Transmission des silhouettes", page 17*), nous aurons certainement des problèmes de saturation du réseau.

4 Modélisation

1 Introduction

Un *modèle* de l'animateur est nécessaire pour être incrusté dans le monde virtuel. Le modèle peut être simple : les studios virtuels utilisent par exemple un plan vertical sur lequel la texture du présentateur est plaquée. La carte graphique se charge alors d'afficher ce quadrilatère texturé et la géométrie du monde virtuel. Les problèmes simples de visibilité sont donc traités automatiquement par le processeur graphique. Un animateur pourra se placer derrière un pilier sans problème particulier, par contre, il ne pourra pas enlacer ce même pilier. En effet, le polygone sur lequel est projetée la texture de l'animateur (voir *Figure 1*) ne peut pas représenter des parties de l'animateur à la fois devant et derrière le pilier. D'autre part, des effets comme les ombres portées ou le déplacement de la caméra virtuelle autour du modèle sont interdits sous peine de voir la supercherie (voir *Figure 1*). Cette approche est donc très limitée.

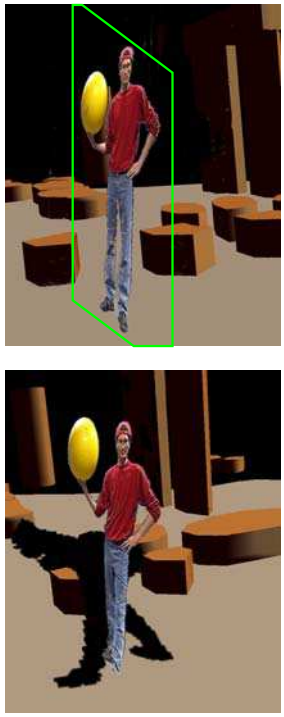


Figure 1 : Haut : la texture de l'animateur est plaquée sur un polygone. Bas : les ombres portées ne sont pas en accord avec l'image inse-

Des modèles plus évolués sont par conséquent nécessaires. Nous considérons ici deux catégories d'approches en fonction des connaissances *a priori* disponibles sur les objets à modéliser : les **modèles non-articulés** pour lesquels l'objet est supposé être constitué de plusieurs parties non-connexes de types inconnus et les **modèles articulés** pour lesquels l'objet est supposé être constitué de parties de types connus reliées par des articulations. Cette classification est bien évidemment non exhaustive mais correspond à deux thèmes de recherche que nous avons explorés.

Comme nous le verrons par la suite, un modèle non-articulé de l'animateur peut être obtenu en temps réel. Si de plus, le modèle est volumique, il permettra l'ajout d'ombres portées (voir *Chapitre 5–Rendu réaliste, 3. Ombres, page 37*), un habillage omnidirectionnel de l'animateur (voir *Chapitre 5–Rendu réaliste, 4. "Texturage" omnidirectionnel, page 41*) et des inte-

ractions avec le monde virtuel (voir *Chapitre 6–Interaction avec le monde virtuel, 4. Exemple d'interaction, page 51*).

Un modèle articulé permettra quant à lui une reconnaissance de la gestuelle, son interprétation et des interactions complexes avec le monde virtuel. Nous avons peu travaillé sur ce type de modélisation, c'est pourquoi seule une tentative de construction d'un squelette est proposée (voir *4. "Modèles non-articulés", page 26*).

Dans ce chapitre, nous allons aborder les thèmes suivants : la modélisation dans le cadre du projet CYBER, le calibrage des caméras, la construction de modèles non-articulés et la construction de modèles articulés. Nous évoquerons aussi les problèmes dus à la synchronisation ou non des caméras ainsi que les possibilités de parallélisation de la construction.

2 Modélisation dans le projet CYBER

Dans le cadre du projet CYBER, la modélisation recherchée doit être avant tout effectuée en temps réel et si possible de bonne qualité.

Le module "d'*extraction*" fournit *n* silhouettes sous forme d'images ou de segments. Il s'agit alors, à partir de ces informations reçues en entrée, de construire un module "*Modélisation*" offrant une représentation articulée et/ou géométrique de l'animateur (voir *Figure 2*).

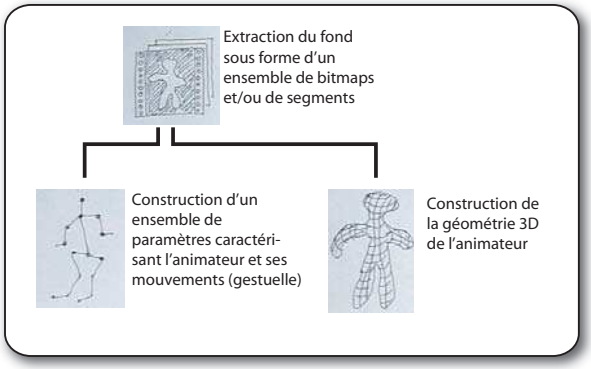


Figure 2:Module "Modélisation". À partir de silhouette, nous caculons un squelette et/ou un modèle géométrique.

Le nombre de caméras ainsi que leurs positions sont des paramètres importants dans la modélisation. Il semble évident que plus nous disposerons de caméras et plus leurs positions seront judicieuses, plus la modélisation sera précise et sans ambiguïté. Dans le cas du projet CYBER, nous avons commencé par travailler avec quatre caméras. Nous les avons positionnées de manière intuitive de sorte à découper au mieux les silhouettes

de l'animateur (voir *Figure 3*). Dans le cas où les caméras ne servent qu'à la reconstruction, il n'est pas nécessaire d'en placer face à face puisque les silhouettes seront presque identiques. En effet, l'effet de perspective est négligeable en comparaison de la précision des silhouettes calculées. Nous avons donc positionné nos caméras tous les 45° en les répartissant en hauteur. Notons que quatre caméras seulement permettent d'obtenir une modélisation très correcte.

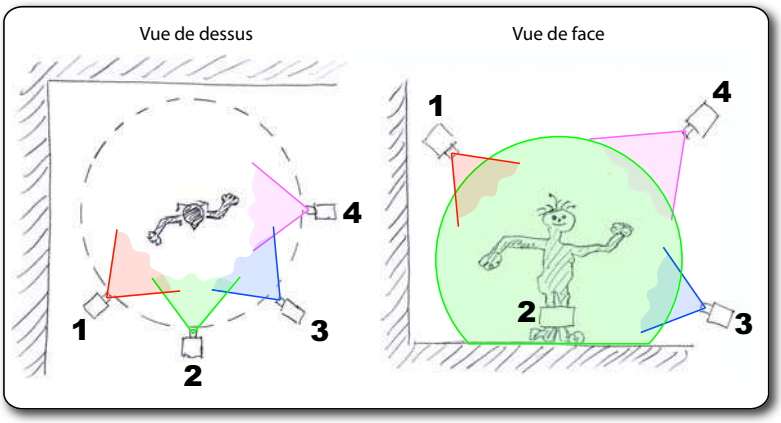


Figure 3 : Position des quatre caméras pour obtenir une reconstruction acceptable.

Cette discussion sur les caméras, leurs positions et leurs nombre sera complétée par la suite puisqu'elle doit tenir compte du fait que ces caméras peuvent être utilisées pour un rendu "*réaliste*" (voir *Chapitre 5–Rendu réaliste, 4.3 Comment positionner les caméras ?, page 43*).

3 Calibrage des caméras

Le calibrage des caméras consiste à déterminer les caractéristiques propres des caméras, les focales, les distorsions..., ainsi que leurs positions et orientations respectives. Ces informations sont indispensables pour modéliser l'élément filmé. Le calibrage de caméras est largement étudié dans le domaine de la vision par ordinateur. De nombreuses solutions existent en repérant des points particuliers sur une surface plane, en général un damier [Tsa86], [Zha00]. Dans le cas de CYBER, nous avons utilisé la bibliothèque du domaine publique [OpenCV]. Le calibrage est effectué de manière très simple en utilisant un damier d'un mètre de côté composé de carreaux de 10 cm (voir *Figure 4*).

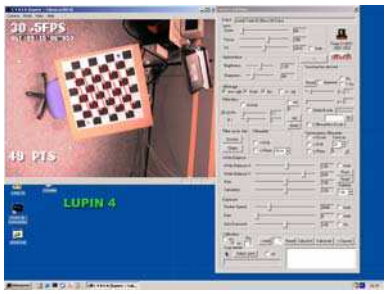


Figure 4 : Application permettant de piloter les caméras (focale, diaphragme...) et de calculer les matrices de calibrage.

4 Modèles non-articulés

Lorsque plusieurs points de vue sont disponibles, il est possible de construire l'enveloppe visuelle [Lau94] de l'objet en calculant l'intersection des faisceaux d'observation associés aux silhouettes. Cette approximation est plus ou moins précise en fonction du nombre de points de vue disponibles. Plusieurs approches existent pour déterminer cette enveloppe visuelle :

- Les approches **volumiques** [CKBH00], [Dye01], [Lok01], [SCMS01] : l'idée est de sculpter un volume dans l'espace. À partir d'un volume initial composé de cellules élémentaires, les *voxels*, nous éliminons les cellules se projetant, dans une, ou plusieurs, des différentes images considérées, à l'extérieur de la silhouette.
- Les approches **surfaciques** [CG99], [MBM01], [BF03], [FB03] : l'idée est de déterminer la surface de l'enveloppe visuelle correspondant aux différents points de vue et délimitant le volume d'intersection des faisceaux d'observations. Ces approches peuvent être fondées sur des algorithmes de calcul d'intersection de polyèdres ou de polygones.

Bien que populaires, les approches basées sur les enveloppes visuelles présentent des inconvénients. En premier lieu, elles ne produisent pas un modèle de la surface observée mais une enveloppe de cette dernière et, l'approximation que cela constitue n'est pas toujours très précise, en particulier lorsque peu de caméras sont disponibles. En deuxième lieu, les parties concaves d'un objet n'apparaissent pas dans l'enveloppe visuelle du fait qu'elles ne sont pas présentes dans les silhouettes de l'objet. De récentes approches volumiques [Kut00], [KS00] proposent de considérer l'information photométrique en réponse à ce dernier problème.

Un état de l'art des différentes approches existantes est proposé dans [HLS04] (voir *"A Real-Time System for Full Body Interaction"*, paragraphe "2. *Previous works*", *page 128*)

Nous allons à présent étudier plus en détails les deux approches, volumique et surfacique et discuter de l'importance limitée de la précision des silhouettes pour la reconstruction de nos modèles.

4.1 Approche volumique

L'approche volumique consiste à diviser régulièrement le volume dans lequel évolue l'animateur en cubes ou *voxels*. L'idée est alors de sculpter cette grille en supprimant, pour chaque caméra, les voxels n'appartenant pas à la silhouette vue par celle-ci (voir *"Real-Time Capture, Reconstruction*

and Insertion into Virtual World of ...”, paragraphe “*5. Shape Estimation*”, *page 96*). Cette technique est généralement appelée “voxel carving” (voir *Figure 5*).

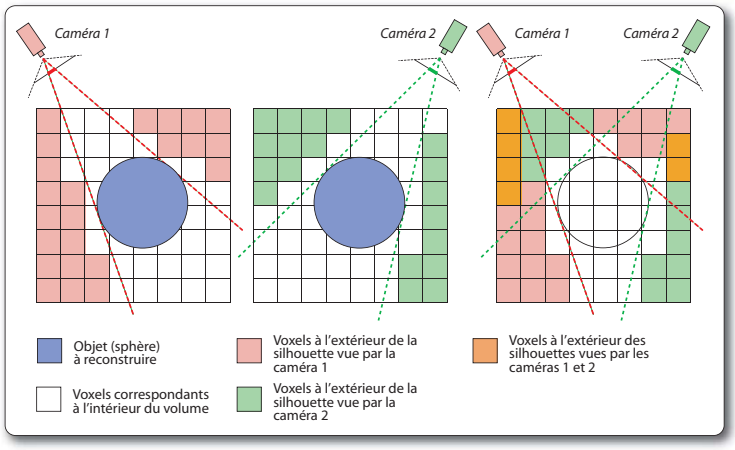


Figure 5 : Principe (en 2D) du “voxel carving” pour une sphère.

Une fois les voxels formant l'animateur trouvés (voir *Figure 7*), nous utilisons un algorithme de “marching cubes” (voir *”Real-Time Capture, Reconstruction and Insertion into Virtual World of ...”*, paragraphe “*5.3 Marching cube on binary voxels*”, *page 98*) pour lisser la surface (voir *Figure 6*).

Pour obtenir des résultats en temps réel, nous avons optimisé l'algorithme de “voxel carving” en utilisant directement les silhouettes sous forme de bitmap et en effectuant le plus d'opérations possibles sur la carte graphique. L'approche que nous avons proposée fonctionne sur tout type de carte proposant les fonctionnalités OpenGL. La description générale est proposée dans (voir *”Real-Time Capture, Reconstruction and Insertion into Virtual World of ...”*, paragraphe “*5.1 Hardware Assisted Voxelisation*”, *page 96*). Une description beaucoup plus technique comportant une discussion pour d'autres applications que celle du projet CYBER a été soumise à publication (voir *“Fast Voxel Carving Using OpenGL Facilities”, page 145*).

4.2 Approche surfacique

Dans une approche de reconstruction surfacique, chaque silhouette obtenue pour chacune des positions des caméras sert à construire un faisceau. L'intersection de l'ensemble des faisceaux est calculée et forme l'enveloppe visuelle de l'objet (voir *Figure 8*).

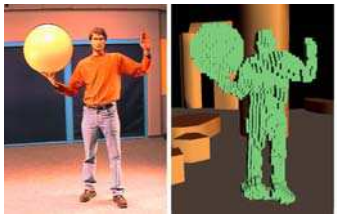


Figure 7 : À gauche : vue prise par une des quatre caméras. À droite : reconstruction voxelique.

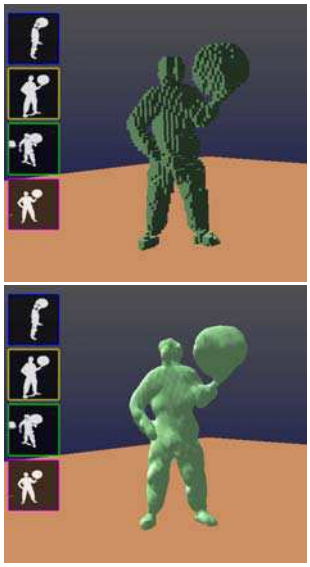


Figure 6 : En haut : résultat brut du “voxel carving”, en bas : résultat après lissage par “marching cubes”. Les quatre vignettes sur la gauche correspondent aux quatre silhouettes utilisées pour la reconstruction.

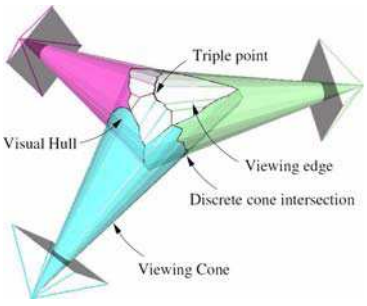


Figure 8 : Reconstruction surfacique d'une sphère à partir de trois caméras (extrait de [FMBR04]).

Figure 9 : Reconstruction surfacique d'une personne (extrait de [ABFM+04]).

Une description d'un certain nombre d'approches surfaciques est décrite dans [CG99]. De nouvelles solutions sont proposées dans [MBM01], [BF03], [FB03].

Une reconstruction surfacique temps réel d'un animateur a été développée sur une grappe de PC dans [ABFM+04]. Les résultats obtenus avec quatre caméras et le module d'extraction de fond développé dans le projet CYBER sont proposés ci-dessous (voir *Figure 9*).

4.3 Précision des silhouettes

Lors du processus d'extraction de fond, il est possible que du bruit ou un mauvais réglage des seuils de détection entraînent des erreurs. On peut ainsi obtenir des images avec des zones ne correspondant pas à la présence de l'animateur (voir *Figure 10*).

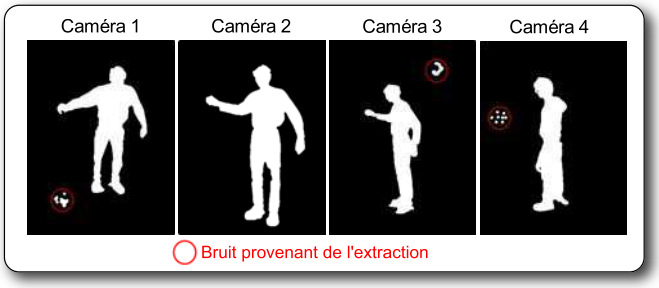


Figure 10 : Les images provenant des caméras 1, 2 et 3 sont bruitées (cf. cercle rouge). Les régions bruitées ne correspondant pas à une même position dans l'espace, ces erreurs n'auront aucune conséquence.

À partir du moment où ce bruit ne correspond pas, sur toutes les images d'extraction, à une même région dans l'espace, ce bruit ne portera pas à conséquence (voir *Figure 11*).

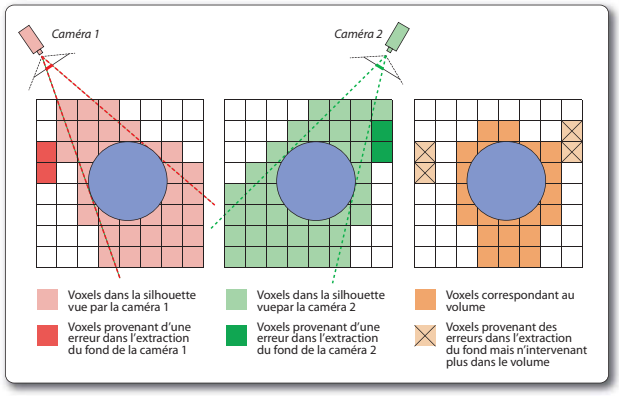
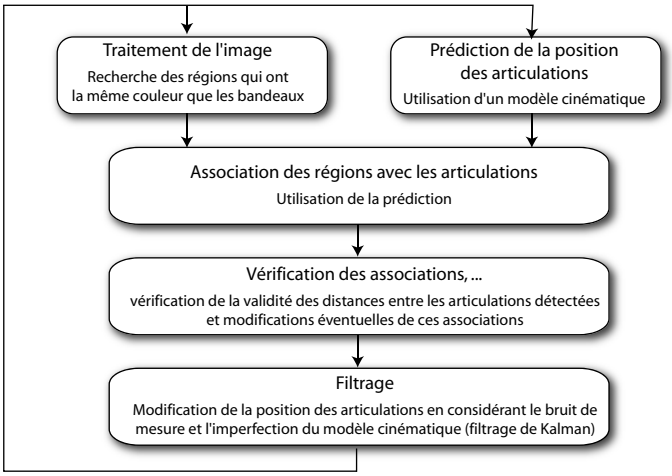


Figure 11 : Certaines erreurs lors de l'extraction disparaissent au moment de la reconstruction.

Ainsi, des images légèrement bruitées, permettent d'obtenir un modèle de bonne qualité. Ceci vaut quel que soit l'approche, surfacique ou volumique.

5 Modèles articulés

Dans cette partie, nous allons présenter les travaux effectués lors d'un stage de DEA en 2001 [Fai01]. Un des buts du stage était d'être capable de filmer, avec une seule caméra, un animateur et de reconstruire en temps réel son squelette simplifié à 14 articulations (voir *Figure 12*). Comme nous ne voulions pas placer de capteurs sur l'animateur, pour ne pas le gêner dans ses mouvements, nous avons travaillé sur le suivi des 14 articulations. Pour simplifier la reconnaissance de ces points particuliers, nous avons utilisé 14 bandeaux de différentes couleurs (voir *Figure 13*). L'algorithme de suivi des articulations était le suivant :



Les résultats étaient plutôt bons, le suivi des bandeaux couplé avec une prédiction de leurs positions permettait de construire, en temps réel, le squelette de l'animateur.

Ces premiers travaux ont été effectués avant que la salle de capture ne soit construite ; la salle utilisée était petite et l'éclairage était limité. Il arrivait que des pixels rouges apparaissent au niveau des yeux ou des lèvres pendant quelques images. Ces pixels étaient alors confondus avec un bandeau rouge. Il est néanmoins intéressant de constater que l'algorithme mis en place retrouvait le squelette exact et s'adaptait très bien.

Les limites de l'approche ont été atteintes et nous en avons retenu les points suivants que nous développerons par la suite :

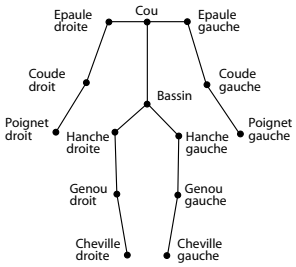


Figure 12 : Modèle humain comportant 14 articulations.

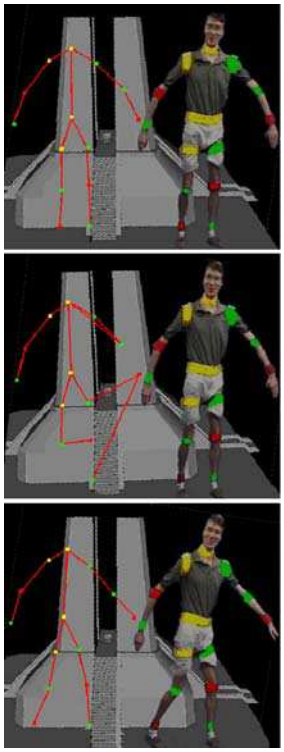


Figure 13 : Trois images extraites d'une séquence d'environ dix images. L'algorithme de reconstruction du squelette peut se tromper lorsqu'il perd un bandeau mais rattrape automatiquement son erreur.

- Un mauvais éclairage introduit des couleurs erronées comme des yeux ou des lèvres rouges. Ces "fausses couleurs" peuvent alors être confondues avec des bandeaux.
- Une seule caméra ne permet pas de suivre en permanence tous les bandeaux, même si la prédiction autorise la perte de bandeaux durant quelques images.

En effet, considérons trois images prises sur une série de dix (voir *Figure 13*). La première et la dernière images correspondent respectivement au début et à la fin de la séquence. Ces deux images étant de bonne qualité, le squelette est cohérent. Dans l'image du milieu, des pixels rouges apparaissent au niveau du visage et la cheville droite n'est pas suffisamment éclairée pour que son bandeau soit reconnu. Le système associe alors la cheville droite au genou gauche, le genou gauche au poignet gauche et le poignet gauche au pixel rouge du visage. Le squelette ne correspond plus à une position possible. Nous avons donc ajouté des contraintes morphologiques en fixant les distances entre les articulations. Malheureusement, certaines positions sont mal interprétées (voir *Figure 14*). Il est aussi possible que les bandeaux ne soient plus visibles, la reconstruction devient alors incohérente (voir *Figure 15*).

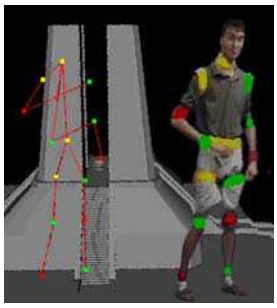


Figure 14 : Les contraintes anatomiques ne suffisent pas toujours pour assurer une bonne reconstruction du squelette.

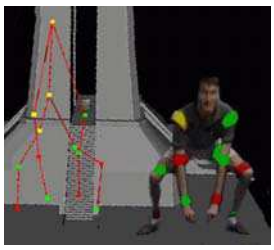


Figure 15 : La disparition d'un bandeau rend le squelette incohérent.

En conclusion, la construction d'un modèle articulé tridimensionnel est un problème complexe à résoudre, d'autant plus que nous ne disposions que d'une seule caméra. Nous avons divisé le problème de la reconstruction 3D en deux parties. Nous avons cherché à déterminer la position des articulations dans l'image. Pour cela, nous avons fixé des rubans de couleur au niveau des articulations de l'animateur. Un filtrage de Kalman a permis de suivre les articulations. Une fois que l'on connaissait la position 2D des articulations, les contraintes anatomiques nous ont permis de déterminer les positions des articulations dans l'espace. Depuis ces recherches, nous n'avons plus travaillé sur la reconnaissance du squelette. Nous avons préféré nous concentrer sur les modèles non-articulés et sur le rendu réaliste.

6 Synchronisation

En théorie, la synchronisation des caméras doit être parfaite pour obtenir une reconstruction exacte. En effet, si les images ne sont pas prises exactement au même instant, l'objet reconstruit risque d'être incomplet comme nous pouvons le constater avec l'exemple d'une balle filmée par deux caméras non synchronisées (voir *Figure 16*). Le moyen le plus sûr pour obtenir une telle synchronisation est d'utiliser un générateur d'impulsions relié aux prises de synchronisation externe des caméras. Les caméras numériques que nous utilisons ont un débit de 30 Hz non synchronisé, malheureusement ce débit tombe à 15 Hz lorsque l'on active la synchronisation matérielle. Notons qu'un très grand nombre de caméras numériques réagissent de la même manière.

Pour passer outre cette limitation, nous avons utilisé une synchronisation logicielle (voir *"A Real-Time System for Full Body Interaction"*, paragraphe "*4.2.3 Synchronisation*", *page 131*). Pour cela, nous utilisons un système de cache et de datation des images provenant des caméras (voir *"A Real-Time System for Full Body Interaction"*, paragraphe "*3.2. Software Architecture*", *page 129*).

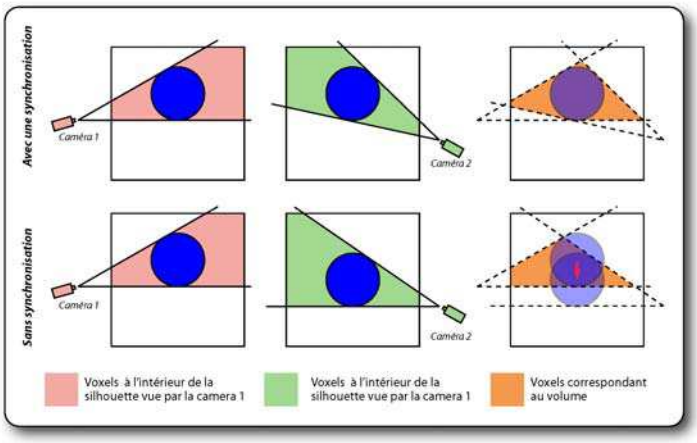


Figure 16 : Ligne du haut : les deux caméras sont synchronisées, la reconstruction est exacte. Ligne de bas : la balle bleue a le temps de se déplacer entre les images prises par les deux caméras, la reconstruction est incomplète, seule une partie de la balle est reconstruite.

Etudions à présent les conséquences d'une telle synchronisation logicielle. Lors de l'acquisition des images à partir des différentes caméras non synchronisées fonctionnant à 30 Hz, nous pouvons assurer que le décalage maximum est strictement inférieur à 33 ms (voir *Figure 17*). Si l'objet ne bouge pas, la reconstruction est bien sûr exacte. Théoriquement, les problèmes peuvent arriver lorsque l'objet bouge rapidement. Néanmoins, nous n'avons constaté aucun problème "choquant" et les résultats étaient plutôt réalistes. Ceci peut s'expliquer par le fait qu'il faut prendre en compte le temps d'exposition (indépendant de la fréquence d'acquisition) pour chaque caméra, et le fait qu'un objet en mouvement entraîne un flou de bougé. La silhouette produite est alors plus grande que la silhouette réelle de l'objet et corrige ainsi l'erreur théorique. Finalement, la synchronisation logicielle entraîne, en théorie, une reconstruction incomplète de l'objet mais en pratique, le flou de bougé, inhérent au temps d'obturation, moyenne cette erreur et la reconstruction reste de bonne qualité.

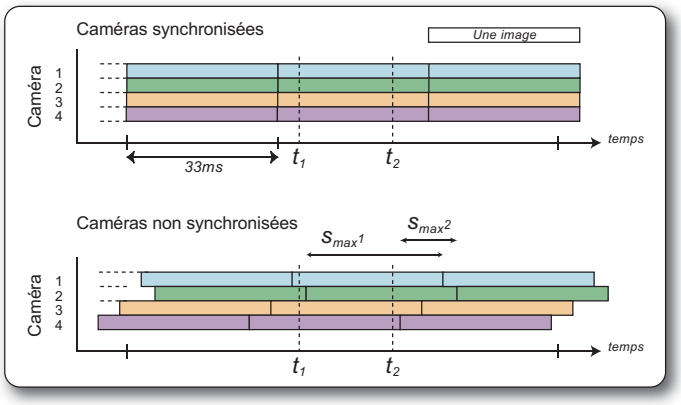


Figure 17:Meilleur et pire cas de synchronisation. Soit D_i le délai entre t_k et la dernière image reçue depuis la caméra i ($0<=D_i<33$ ms). Le décalage $S_{i,j}$ entre les caméras i et j est $||D_i-D_j||$. Dans le cas synchronisé, toutes les silhouettes sont reçues en même temps : $S_{i,j}=0$ ms quel que soit t_k. Dans le cas non-synchrone, les images sont décalées. A t_1 le décalage maximum est $S_{max1}=S_{1,2}$~30 ms, à t_2 le décalage maximum $S_{max}=S_{2,4}$ << 33 ms : S_{maxk} est donc toujours inférieur à 33 ms.

7 Parallélisation

La modélisation géométrique étant un processus coûteux en temps et l'affichage devant s'effectuer à 25 images par seconde minimum, nous avons été amené à travailler sur la parallélisation des traitements. Au début de nos travaux, nous disposions de PC peu performants et d'une SGI Onyx (voir *Chapitre 2–Plate-forme CYBER, 5. Les calculateurs, page 10*). Nous connaissions les performances de l'Onyx pour l'avoir "malmené" lors de nos travaux sur la radiosité (voir *"Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer", page 75*). Nous savions que son architecture permettait de traiter en temps réel les différents flux vidéos, d'effectuer la reconstruction du modèle et l'affichage de la scène.

La distribution des processus a été la suivante. Sur chaque PC, un processus gère l'extraction de fond et le filtrage puis transmet les silhouettes à l'Onyx via une liasons réseau haut débit (1 Gb/s). Sur l'Onyx, un processus par PC a en charge un cache avec les différentes silhouettes datées et transmet au processus de reconstruction la dernière silhouette reçue. En pratique, nous n'avons pas constaté de perte d'image due à un traitement trop lent. Enfin, le processus principal effectue la reconstruction du

modèle, modifie la scène en fonction des interactions et s'occupe du rendu final (voir *Figure 18*).

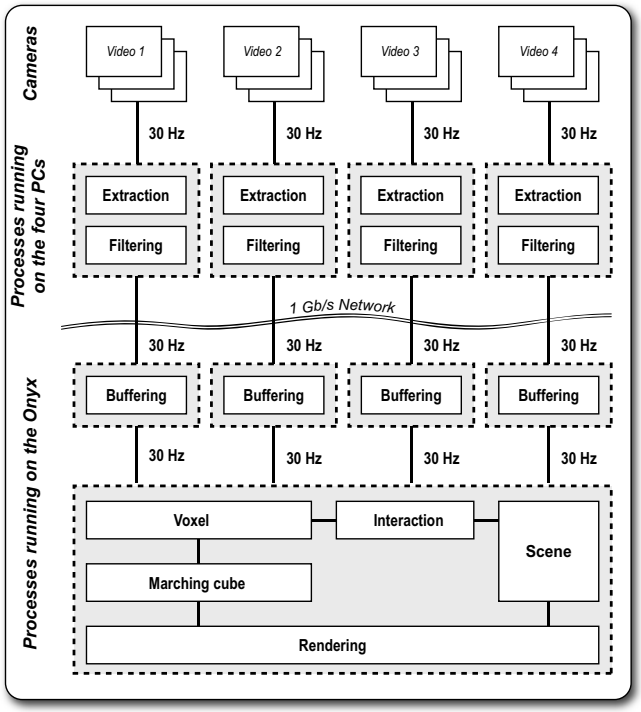


Figure 18: Répartition des différents processus.

Cette parallélisation fonctionne très bien et a permis d'obtenir tous les résultats que l'on a publiés. Cependant, la puissance de l'Onyx est limitée et son prix est prohibitif (à l'achat et à la maintenance). À l'heure actuelle, lorsque l'on a besoin de puissance de traitement, nous nous tournons plutôt vers des architectures évolutives et nettement moins onéreuses comme les grappes de PC.

8 Contributions

Nous avons commencé par travailler sur un modèle articulé à partir d'un simple flux vidéo et de marqueurs sur l'animateur (bandeaux de couleurs). Cette étude a montré les limites d'une telle approche cependant elle a permis de montrer que l'on pouvait extraire un squelette 3D à condition d'introduire des contraintes morphologiques [Fai01].

Dans [HLGB03] (voir *Chapitre 8–Articles, 3. Real-Time Capture, Reconstruction and Insertion into Virtual World of ..., page 91*), nous avons proposé une modélisation temps réel à base de voxels et de seulement quatre caméras.

Nous avons développé des algorithmes temps réel pour reconstruire un modèle non-articulé de l'animateur. En particulier, nous avons proposé une nouvelle méthode de "voxel carving" utilisant pleinement les fonctionnalités matérielles disponibles sur la plupart des cartes graphiques (voir *Chapitre 8–Articles, 7. Fast Voxel Carving Using OpenGL Facilities, page 145*).

Nous avons montré qu'une synchronisation logicielle permettait une reconstruction de qualité [HLS04] (voir *Chapitre 8–Articles, 5. A Real-Time System for Full Body Interaction, page 125*).

Enfin, notre bonne connaissance de l'architecture DSM ("*Distributed Shared Memory*"), disponible par exemple sur les SGI Onyx [SH00] (voir *Chapitre 8–Articles, 1. A Practicle Analysis of Clustering Strategies for Hierarchical Radiosity, page 59*) nous a permis de proposer une architecture logicielle efficace pour une modélisation temps réel.

9 Et maintenant ?

La précision de la reconstruction du modèle géométrique de l'animateur dépend principalement des caméras : leur résolution, leur nombre et leur position. L'utilisation de quatre caméras est un minimum ; nous arrivons à obtenir un modèle volumique correct pour un animateur seul. Par contre, si nous désirons insérer plusieurs animateurs, il sera nécessaire d'augmenter sensiblement le nombre de caméras pour diminuer les angles morts. Nous serons confrontés aux problèmes de débit réseau (voir *Chapitre 3–Acquisition des silhouettes, 4. Transmission des silhouettes, page 17*) et de positionnement des caméras (voir *Chapitre 5–Rendu réaliste, 4. Comment positionner les caméras ?, page 43*). C'est probablement ce dernier point qui sera le plus intéressant scientifiquement. Comment trouver les positions optimales (ou les moins mauvaises) des caméras ? Comment varie la précision du modèle en fonction du nombre et de la position des caméras ?

Nous avons cherché à extrapoler un squelette de l'animateur. L'approche présente un certain nombre de lacunes. Depuis, plusieurs travaux ont été publiés [TMSS02], [CTMS03] pour trouver le squelette d'un seul animateur. Mais est-il possible de reconnaitre les squelettes de plusieurs animateurs ? d'un animateur jouant avec une balle ? Cet axe de recherche reste ouvert.

5 Rendu réaliste

1 Introduction

Le “rendu réaliste” consiste, dans notre cas, à afficher sur un périphérique graphique une scène virtuelle et un ou plusieurs éléments réels sans que l’on puisse remarquer d’incohérences. Nous voulons que l’intégration du réel dans le virtuel soit la plus convaincante possible.

Dans le monde de l’audiovisuel et des émissions de télévision en particulier, nous pouvons constater que le réalisme n’est pas encore d’actualité. Prenons pour exemple l’émission Thalassa diffusée par France 3 en 2003 (en 2005 rien n’a vraiment changé). En regardant l’émission, nous remarquons immédiatement que le décor n’est pas réel, que Georges Pernoud est filmé en studio sur fond bleu et qu’il est simplement incrusté sur un décor virtuel.

Etudions de plus près deux images extraites de l’émission pour comprendre ce qui “choque” lors de la diffusion (voir *Figure 1*).

- Nous avons l’impression que l’animateur flotte sur le sol. En fait, ceci est dû au manque d’ombres portées comme nous le verrons par la suite.
- Quelle que soit l’image projetée sur l’écran en arrière plan, l’aspect de l’animateur ne change pas (cela est d’autant plus visible lors de la diffusion). En effet, une image à dominante rouge projetée sur l’écran en arrière plan devrait se traduire par un rougissement de l’animateur sur son côté droit.
- A l’inverse, le teint de Georges Pernoud est trop chaud par rapport au monde virtuel bleu et donc froid dans lequel il évolue.

Figure 1 : Extrait de l’émission Thalassa sur France 3.

- Les plans proposés par le producteur sont peu variés (flagrant lors de la diffusion). Ceci est dû au fait que pour obtenir une incrustation sans distorsion, la caméra de plateau doit être face à l'animateur.

Dans ce chapitre "Rendu réaliste", nous allons reprendre ces différents problèmes et apporter un ensemble de solutions. Dans l'ordre, nous allons traiter les points suivants :

- Rendu réaliste dans le projet CYBER ;
- Ombres portées ;
- "Texturage" omnidirectionnel de l'animateur.

2 Rendu réaliste dans le projet CYBER

Le module "*Rendu réaliste*" (voir *Figure 2*) est celui faisant appel au plus de ressources : la géométrie de l'animateur, les flux vidéo, la géométrie de la scène...

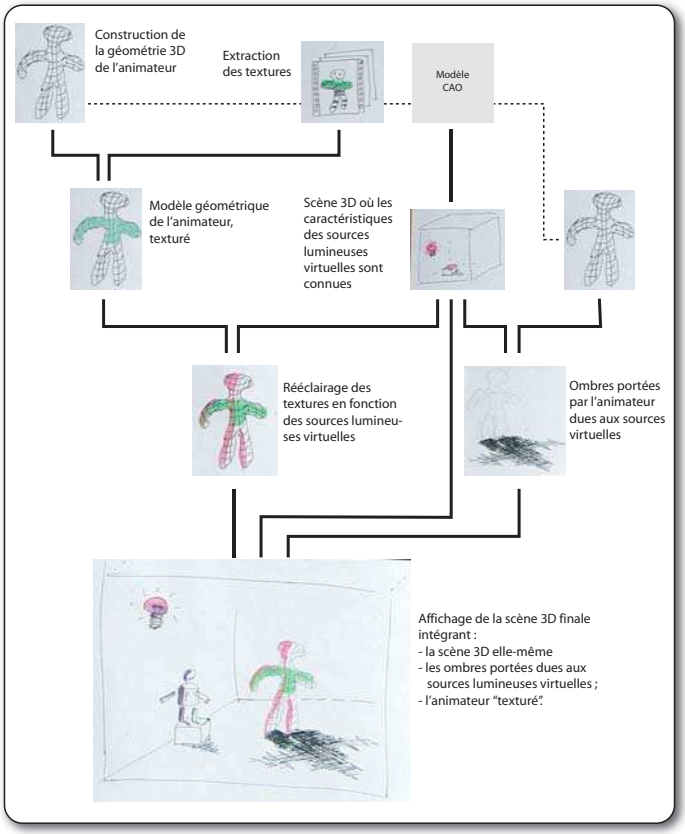


Figure 2:Module "Rendu réaliste".

Les avancées scientifiques réalisées dans le cadre du projet CYBER doivent pouvoir être présentées à des visiteurs. Nous verrons par la suite (voir *4.3 "Comment positionner les caméras ?", page 43*) que vouloir utiliser cette plate-forme à la fois pour des expérimentations scientifiques et des démonstrations impose des contraintes supplémentaires.

3 Ombres

L'animateur peut produire deux types d'ombres portées, les ombres réelles et les ombres virtuelles. Les premières sont dues aux éclairages du studio. Elles sont toujours présentes même en disposant l'éclairage de manière optimale. Elles peuvent être supprimées très facilement lorsque l'on utilise une technique de "chromakey". Si le fond n'a pas une couleur uniforme, il est nécessaire de "jouer" sur les contrastes des caméras, les positions et la puissance des éclairages, pour limiter ce type d'ombre. Le second type d'ombre provient des sources lumineuses du monde virtuel. Elles doivent être calculées et prises en compte lors du rendu final (voir *Figure 3*). Ceci est possible uniquement lorsque l'on dispose d'un modèle géométrique 3D. Dans un contexte de production télévisuelle, nous ne disposons généralement pas d'un tel modèle, les ombres portées ne peuvent donc pas provenir des sources lumineuses du monde virtuel (*cf.* l'exemple de l'émission Thalassa présenté en introduction de ce chapitre)

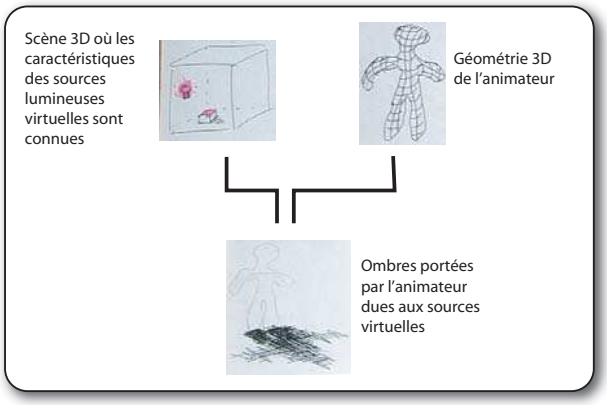


Figure 3:Les ombres de l'animateur dues aux sources appartenant au monde virtuel doivent être calculées et prises en compte dans le rendu final.

3.1 Importance des ombres

Les ombres sont étudiées depuis bien longtemps, les premières caractérisations écrites se trouvent dans le *Codex Urbinas* de Léonard de Vinci de 1490. En fait, ceci n'est pas surprenant puisque les ombres portées

sont indispensables à la compréhension d'une image. En effet, elles apportent quatre informations :

- la position (voir *Figure 4*) ;
- la forme de l'objet (voir *Figure 5*) ;
- la forme de la surface recevant l'ombre (voir *Figure 6*) ;
- l'existence d'un objet qui n'est pas visible dans l'image (voir *Figure 7*).

Figure 4 : L'ombre apporte une information visuelle sur la position du robot par rapport au sol.

Figure 5 : L'ombre apporte une information sur la géométrie cachée. De gauche à droite, nous pouvons remarquer, grâce aux ombres portées, que le robot tient un anneau, tient la "teapot" ou ne tient rien dans sa main gauche.

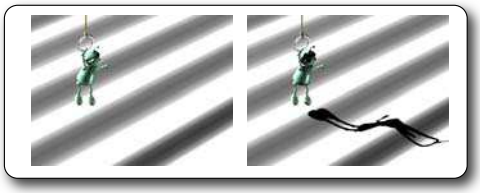


Figure 6 : L'ombre portée permet de mieux comprendre la forme du sol.

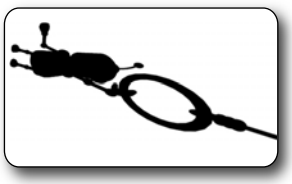


Figure 7 : L'ombre indique que le robot se trouve hors du champ visuel.

Une discussion plus complète sur l'importance des ombres portées est proposée dans "l'état de l'art" sur les ombres douces en temps réel que nous avons publié (voir *"A survey of Real-Time Soft Shadows Algorithms"*, paragraphe "*1. Introduction*", *page 104*).

3.2 Ombres douces

Produire des ombres douces (voir *Figure 8*) est un sujet traité depuis longtemps en synthèse d'image. Par contre, le fait d'ajouter la contrainte temps réel est relativement récent, pour preuve le «grand» nombre de publications sur le sujet à la conférence *Siggraph 2003*. Cette effervescence est en fait directement liée aux possibilités de programmation des cartes graphiques de dernières générations.

Figure 8 : Ombres dures et douces produites par trois sources de tailles différentes.

Nous avons présenté un "état de l'art" sur les ombres douces temps réel lors de la conférence Eurographics en 2003 [HLHS03]. Cette présentation a été complétée par les publications présentées à cette même conférence pour donner lieu à une publication exhaustive sur le sujet [HLHS03b] (voir *"A survey of Real-Time Soft Shadows Algorithms", page 101*).

Dans cette publication, nous avons entre autres expliqué comment sont produites les ombres douces, pourquoi elles sont importantes et les effets que peuvent produire plusieurs sources lumineuses de couleurs différentes. Nous avons mis en avant un certain nombre de problèmes généralement "oubliés" dans les algorithmes d'ombres douces temps réel (voir *"A survey of Real-Time Soft Shadows Algorithms"*, paragraphe "*2.4 Important issues in computing soft shadows*", *page 106*)[1]. Nous reprendrons quelques uns de ces problèmes.

Lorsqu'une source lumineuse est beaucoup plus grande que l'objet éclairé, son ombre peut être quasiment douce (voir *Figure 9*) et presque aucune zone d'ombre dure n'est produite. Cet effet est rarement obtenu avec les algorithmes temps réel parce que la plupart du temps, ils considèrent le centre de la source pour produire les régions de pénombre et/ou considèrent uniquement les polygones les plus proches de la source lumineuse.

Figure 9 : Lorsque la source lumineuse est sensiblement plus grande que l'objet qu'elle éclaire, il n'est plus possible de considérer son centre pour essayer de produire une ombre douce réaliste.

1. Un certain nombre de pages Internet on été écrites pour servir de support à des cours, elles contiennent en particulier des images et des vidéos montrant l'importance des ombres – http://artis.imag.fr/Research/RealTimeShadows/

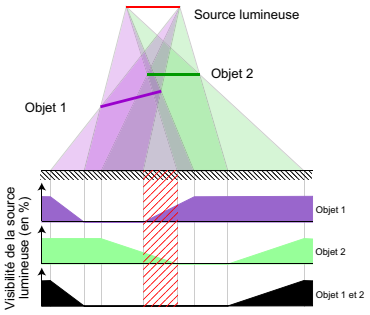


Figure 10 : L'ombre de deux objets n'est pas une simple combinaison des deux ombres prises séparément.

Le calcul de l'ombre de deux objets n'est pas une simple combinaison des deux ombres prises séparément (voir *Figure 10*). Ceci impose donc de traiter tous les objets d'une scène en même temps.

Dans cet "état de l'art", nous avons proposé une classification des différentes approches en fonction de leur rapidité, de leur qualité visuelle, du type de sources lumineuses qu'elles considèrent, du type de scènes utilisables et de la nécessité de disposer de fonctionnalités matérielles particulières [HLHS03b] (voir *"A survey of Real-Time Soft Shadows Algorithms"*, *Table 1*, *page 119*).

Nous avons travaillé sur d'autres approches pour produire des ombres douces temps réel qui prennent en compte les problèmes présentés ci-dessus. Nous avons soumis à publication une nouvelle approche pour produire ces ombres douces en utilisant massivement les possibilités de programmation des dernières générations de cartes graphiques (voir *"Soft Shadow Maps: Efficient Sampling of Light Source Visibility", page 153*). Cet algorithme discrétise les surfaces des objets portant des ombres pour calculer la proportion de lumière retenue par chaque élément de surface. En pratique, la complexité de cette approche est en O(n) où n est le nombre de polygones des objets portant des ombres douces (voir *Figure 11*).

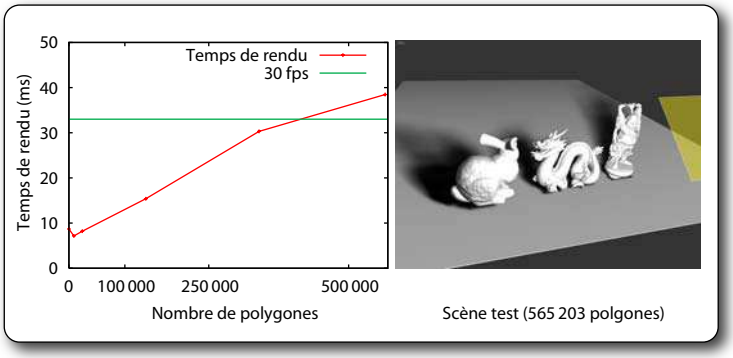


Figure 11 : Influence du nombre de polygones sur la vitesse du rendu.

De plus, cet algorithme permet d'obtenir des ombres douces réalistes même pour des sources lumineuses plus grandes que les objets éclairés (voir *Figure 12*). Ceci grâce à un traitement en deux passes : la première

traite les polygones orientés vers la source lumineuse et la deuxième traite les autres polygones portant des ombres (voir *Figure 12*).

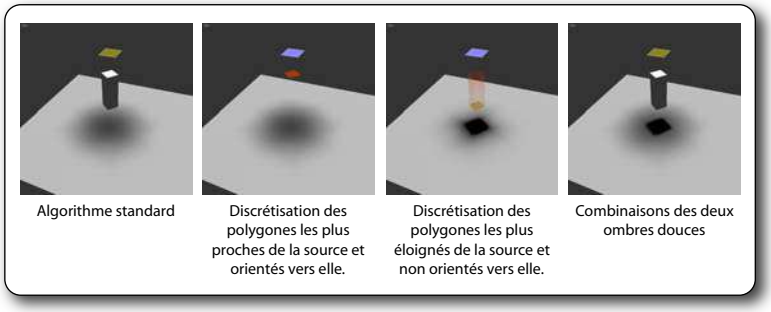


Figure 12:Les algorithmes standards ne fonctionnent pas toujours (à gauche). Notre approche en deux “passes” produit des ombres réalistes (à droite).

4 “Texturage” omnidirectionnel

Pour que le rendu soit réaliste, il faut que l’on retrouve les couleurs et les textures réelles de l’animateur dans le monde virtuel. En effet, il n’est pas envisageable d’afficher la simple reconstruction géométrique sans l’habiller. Nous allons donc utiliser les flux vidéo provenant des caméras ayant servis dans la phase de modélisation (voir *Figure 13*) pour “texturer”[1] la géométrie de l’animateur. De plus, pour que l’on puisse se déplacer tout autour du modèle et le voir sous tous les angles, l'habillage doit être complet, c'est-à-dire omnidirectionnel. Nous utiliserons dans cette partie les modèles et textures acquis dans le cadre du projet GrImage [GrImage].

Figure 13:Position de six caméras servant à la construction du modèle 3D et au “texturage”.

L’étape de “texturage omnidirectionnel” (voir *Figure 14*) étant liée aux données produites lors de l’acquisition, nous allons rappeler les hypothèses de travail :

- La géométrie de l’animateur est disponible sous la forme d’un ensemble de polygones, mais nous ne disposons pas forcément des informations d’adjacence des faces.
- Les caméras sont calibrées et nous connaissons leurs positions (à quelques centimètres près). Elles sont synchronisées (de manière matérielle ou logicielle).

1. Dans la suite du chapitre, je prendrai la liberté de ne plus utiliser “*texturage*” (ou “*texturer*” pour ne pas alourdir la lecture. Lorsqu’il n’y aura pas de confusion possible, j’utiliserai le terme *habiller.*

- Pour chaque caméra, nous disposons d'un flux d'images couleurs correspondant à l'intérieur de la silhouette.

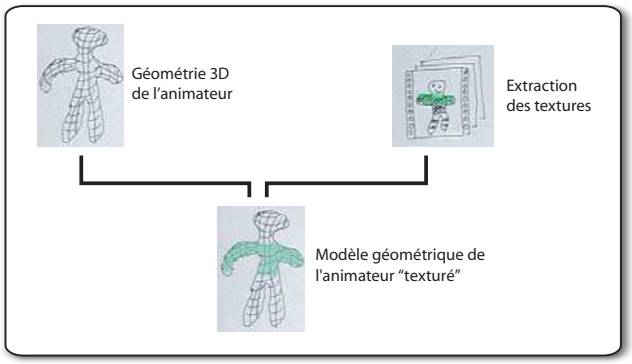


Figure 14 : Module "Texturage omnidirectionnel". À partir du modèle géométrique reconstruit et des flux vidéos nous "habillons" l'animateur.

Les travaux sur l'habillage du modèle géométrique ont débuté grâce à deux stages de DEA en 2003 et 2004. Ces travaux ont permis de faire un état des lieux des approches existantes et d'en implémenter un certain nombre. Nous avons ainsi pu mettre en évidence les problèmes à résoudre :

- Comment assurer une bonne jonction entre les textures provenant de caméras différentes ?
- Comment positionner et choisir les caméras ?
- Comment traiter les problèmes de visibilité ?
- Comment conserver une cohérence temporelle des couleurs ?
- Comment tenir compte de l'éclairage provenant du monde virtuel ?

Ces travaux de recherche étant en cours, nous n'avons pas encore le recul suffisant pour proposer la meilleure approche. Par contre, nous pouvons faire le point sur les problèmes présentés ci-dessus et proposer des solutions ou des perspectives de recherche. Nos premiers résultats seront publiés en septembre 2005 [OH05] (voir *"Omnidirectional texturing of human actors from multiple view video seq.", page 137*).

Figure 15 : La jonction entre des textures provenant de caméras différentes peut être visible. Ici, nous pouvons observer un «zébrage» noir dû à ce problème.

4.1 Jonction entre les textures

Pour habiller l'ensemble du modèle 3D, nous utilisons les différents flux vidéo à disposition. Deux polygones partageant une même arête peuvent, en théorie, être texturés avec des images provenant de caméras différentes. D'une part, la jonction entre les images utilisées peut alors être visible et nuire au réalisme (voir *Figure 15*). D'autre part, le modèle poly-

gonal peut être composé de triangles très allongés (voir *Figure 16*) ce qui augmente ces artefacts.

Plusieurs approches ont déjà été proposées dans la littérature [CTMS03], [DYB98], principalement en “mélangeant” les couleurs des polygones partageant une même arête (voir *”Omnidirectional texturing of human actors from multiple view video seq.”*, paragraphe “*2.2 Multi-view texture mapping*”, *page 147*). Le principal problème est de définir correctement le “mélange”. En effet, nous pouvons faire une simple moyenne entre les textures des différentes caméras. Cette solution ne donne pas de bons résultats en particulier lorsque l’on superpose trois textures ou plus. En effet, le mélange d’un trop grand nombre de textures introduit du flou dans l’image résultante. Il faut donc attribuer judicieusement des poids à chacune des textures.

Figure 16 : Modèle géométrique de l’animateur. Certains polygones sont très allongés.

4.2 Problèmes de visibilité

Le choix des caméras utilisées pour texturer l'avatar est relativement complexe. En effet, pour un polygone donné, nous pourrions utiliser la caméra la “plus orthogonale” au polygone. Malheureusement, cette approche simpliste ne fonctionne pas. En effet, si un autre polygone se situe entre la caméra choisie et le polygone à texturé, ce dernier ne sera pas entièrement visible depuis la caméra mais sera néanmoins texturé avec ce qu'elle voit (voir *Figure 17*). Le choix des caméras servant à habiller l'avatar doit donc obligatoirement prendre en compte les problèmes de visibilité (voir *Figure 18*). La plate-forme d'acquisition est actuellement équipée de six caméras, à terme, nous devrions en disposer d'une dizaine, voire plus. Une augmentation du nombre de caméras permet d'augmenter le nombre de point de vue et donc d'améliorer la couverture visuelle de l'animateur. On peut ainsi espérer filmer l'animateur sous “toutes ses coutures”.

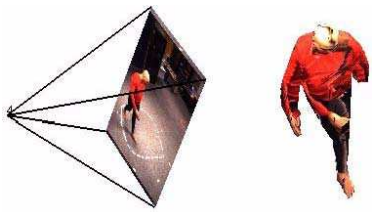


Figure 17 : Problème de visibilité, la caméra utilisée (à gauche sur la figure) pour texturer le bras droit de l'avatar est la même que celle utilisée pour texturer la jambe droite. Il en résulte une projection erronée du bras sur la cuisse de l'avatar.

Figure 18 : Autre exemple où un objet (le damier) se trouve entre la caméra et les polygones à texturer (le visage).

4.3 Comment positionner les caméras ?

Nous avons vu qu’avec peu de caméras, et à condition de les positionner judicieusement, nous pouvions obtenir une reconstruction satisfaisante de l’animateur (voir *Chapitre 4–Modélisation, 2. Modélisation dans le projet CYBER, page 24*). Par contre, ce faible nombre de caméras ne convient pas à l’habillage du modèle. En effet, certaines régions de l’animateur sont “à l’ombre” de toutes les caméras, nous ne disposons alors d’aucune information de texturage. Nous avons étudié le nombre de caméras visibles depuis la surface du modèle en travaillant avec une configuration utilisant six caméras (voir *Figure 19*). La conclusion est simple, il faut beaucoup plus de caméras pour couvrir tous les “angles morts” dus aux mouvements de l’animateur. Dans notre exemple, le simple fait de

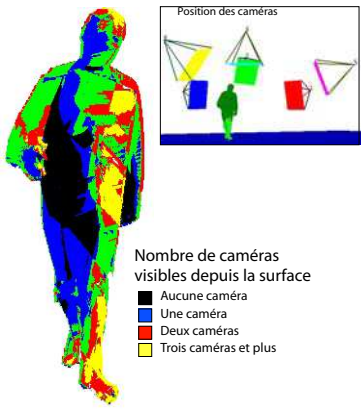


Figure 19: Nombre de caméras visibles depuis la surface du modèle.

lever le bras droit cache le flanc droit (régions noires voir *Figure 19*) de l'animateur de toutes les caméras, nous ne pouvons donc plus le texturer.

Il est donc nécessaire d'augmenter sensiblement le nombre de ces caméras. Le problème est alors de les positionner au mieux en fonction de ce que l'on veut filmer : un ou plusieurs animateurs avec ou sans objets supplémentaires (comme des balles), etc.

D'autres contraintes sont à prendre en compte : la position de l'écran sur lequel est projetée la scène et la position du public (chercheurs, visiteurs...). En effet, si l'écran de projection est dans le champ d'une des caméras, nous ne pourrons plus appliquer l'algorithme d'extraction sur fond statique quelconque. Ainsi, il est nécessaire de bien orienter la caméra 4 de notre exemple (voir *Figure 20*) pour ne pas risquer de filmer l'écran. La solution est de placer la caméra en hauteur et de la faire pointer vers le bas à la limite de l'écran, mais nous risquons d'avoir des angles morts.

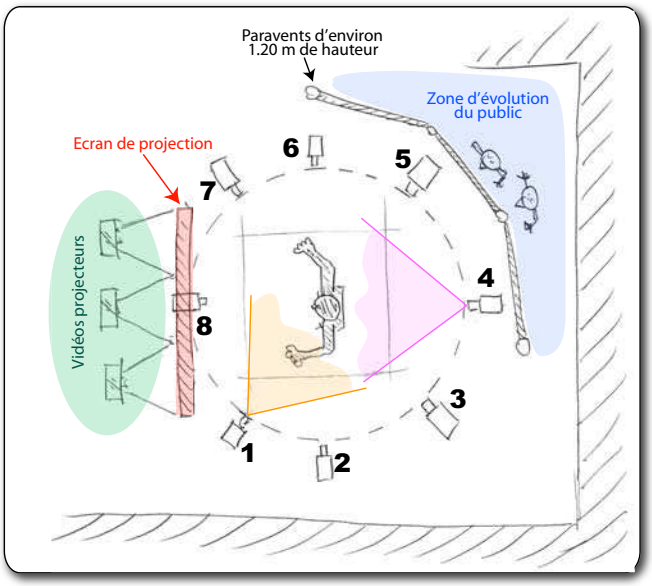


Figure 20: Positions des caméras. La caméra 1 risque de filmer le public, la caméra 4 risque de filmer l'écran de projection.

Le problème est le même pour tous les objets ou personnes en mouvement. Ainsi, du public pourrait nuire à la bonne extraction de fond. La caméra 1 de notre exemple (voir *Figure 20*) risque de filmer ce public. Une solution est alors d'utiliser des paravents pour augmenter les surfaces statiques du fond tout en autorisant des personnes à circuler à proximité de la zone de capture (voir *Figure 20*).

4.4 Cohérence des couleurs dans le temps

L'animateur étant en mouvement, les silhouettes vues depuis chaque caméra changent à chaque image, la géométrie de l'avatar n'est donc jamais la même. En particulier, le nombre et la forme des triangles qui composent l'avatar varient en permanence. Ceci introduit des discontinuités temporelles. L'effet le plus visible est une variation de couleur de certains polygones due à un changement de caméra utilisée pour leurs "texturages" (voir *Figure 21*).

Figure 21 : Deux images prises en tournant très légèrement la caméra. Nous constatons un changement net des textures.

Les polygones ayant une durée de vie d'une seule image, il n'est pas possible de les suivre dans le temps et il n'est donc pas possible de se fonder sur une numérotation des polygones. Ceci reviendrait à utiliser une approche dans "l'espace objet". La solution que nous étudions actuellement se place dans "l'espace image". Nous cherchons à suivre dans le temps la couleur des pixels affichés. L'idée est alors de pondérer la couleur des pixels à l'instant *t* en fonction des couleurs de ces mêmes pixels à l'instant *t-1*, *t-2*...*t-n*. Ainsi, nous pouvons espérer éviter les sauts de couleur. Bien sûr, la pondération sera un choix crucial pour le bon fonctionnement du processus. Notons que tous les changements brusques de couleur d'un pixel ne doivent pas être lissés, ils peuvent correspondre à un mouvement réel de l'animateur. On peut donc envisager d'utiliser des pondérations dynamiques dans le temps en fonction de critères comme des seuils, des mouvements brusques de la caméra, etc.

Une approche que nous voulons étudier, est le suivi des points dans le temps pour prédire leurs positions. Nous pourrons nous inspirer de [VBK05] même si l'approche décrite utilise un modèle voxelique et n'est pas temps réel. Une autre technique est présentée dans [TCMS03]. Cette solution aurait l'avantage de traiter les discontinuités temporelles mais aussi, dans une certaine mesure, le manque d'information dans les angles morts. En effet, par extrapolation, nous pouvons envisager de combler les vides avec les couleurs prédites.

5 Contributions

La première contribution a été de mettre en évidence ce qui "choque" dans les émissions télévisuelles utilisant des studios virtuels (voir *1. "Introduction", page 35*). À partir de ces constats, nous avons principalement travaillé sur le point le plus critique, à savoir les ombres portées. Nous avons commencé par publier un état de l'art exhaustif des algorithmes temps réel d'ombres douces [HLHS03], [HLHS03b] (voir *"A survey of Real-Time Soft Shadows Algorithms", page 101*). Nous y avons proposé une classification des algorithmes pour faciliter le choix de l'approche à utiliser en fonction des besoins du programmeur.

Forts de cette expérience et des problèmes scientifiques que nous en avons dégagés, nous avons proposé une nouvelle approche pour des ombres douces temps réel en discrétisant les objets pour évaluer la proportion de lumière retenue par chaque élément de surface (voir *"Soft Shadow Maps: Efficient Sampling of Light Source Visibility", page 153*).

Enfin, nous travaillons actuellement sur le texturage omnidirectionnel de la géométrie 3D de l'animateur. Nous avons mis en évidence un certain nombre de points sur lesquels nous concentrer. Les premiers résultats ont fait l'objet d'une publication [OH05] (voir *"Omnidirectional texturing of human actors from multiple view video seq.", page 137*).

6 Et maintenant ?

6.1 Rééclairage

Pour obtenir un rendu réaliste, les éléments insérés dans le monde virtuel (dans notre cas, un animateur) ne doivent pas ressortir, ils doivent parfaitement s'intégrer. Dans notre cas, c'est l'apparence de notre animateur qui doit être prise en compte ainsi que les modifications d'éclairage qu'il peut entraîner sur le reste de la scène. En effet, lorsqu'il passe à proximité d'une source lumineuse, il doit bien sûr projeter une ombre mais aussi changer de couleur en fonction des caractéristiques des sources lumineuses présentes. Ainsi, si une source a une dominante rouge, l'animateur doit rougir. C'est ce que l'on va appeler le "rééclairage" (voir *Figure 22*).

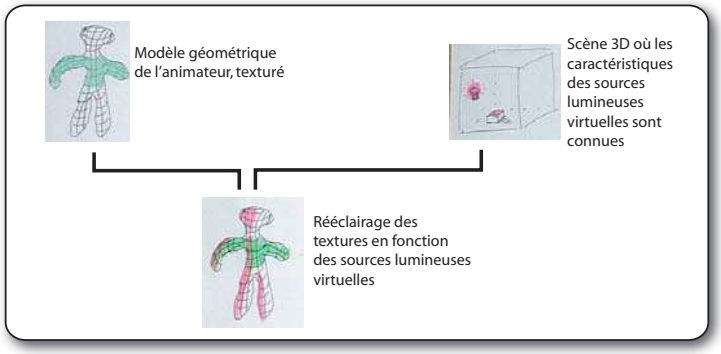


Figure 22:Module de "Rééclairage". Nous utilisons le modèle géométrique ainsi que les caractéristiques des sources lumineuses du monde virtuel pour "rééclairer" l'animateur.

Nous pouvons distinguer deux types d'échanges lumineux à simuler dans nos mondes virtuels : les *éclairages directs ou locaux* et les *éclairages indirects ou globaux*.

Éclairage local

Les *effets locaux* sont produits directement par les sources de lumière : éclairement direct, ombres portées et auto-ombrage. Nous avons principalement travaillé sur l'aspect "ombre douce" qui est le plus critique. Il reste néanmoins encore beaucoup de travail à faire. Nous continuons à explorer de nouvelles approches aussi bien dans "l'espace image" que dans "l'espace objet". Parmi les problèmes à résoudre, le plus difficile est probablement de traiter les "auto-ombrages" (ombre d'un objet portée sur lui-même) d'une manière physiquement exacte (voir *Figure 23*). En effet, nous savons produire des effets d'auto-ombrage plausibles [Assa03] mais pas réel, c'est-à-dire ne respectant que très peu les lois optiques.

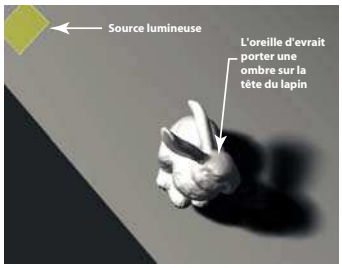


Figure 23:L'auto-ombrage n'est pas pris en compte dans cet exemple.

Les sources lumineuses généralement utilisées en synthèse d'images sont de simples quadrilatères avec une distribution énergétique homogène. Cependant, ce type de sources est une violente simplification. En effet, nous trouvons dans le monde réel des éclairages "statiques" comme des néons, des spots... mais aussi des éclairages "dynamiques" comme un feu de cheminée, un ciel nuageux… Ces sources peuvent être géométriquement complexes (voir *Figure 24*) ou ne pas être "modélisables" sous la forme d'un ensemble de polygones (comme des nuages), il faut donc trouver une modélisation adéquate. Ces modèles peuvent correspondre à une description énergétique ou à une géométrie très simple simulant le même éclairage. Pour proposer de nouveaux modèles, nous pouvons nous appuyer sur nos travaux dans le domaine de la radiosité et en particulier sur le "clustering" [HDSD99], [SH00] (voir *"A Practicle Analysis of Clustering Strategies for Hierarchical Radiosity", page 59*).

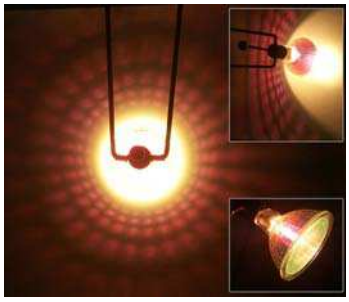


Figure 24:Exemple de lampe de bureau que l'on veut modéliser et utiliser dans le monde virtuel.

Éclairage global

Les *effets globaux* proviennent d'échanges subtils entre l'environnement et l'animateur plongé dans le monde virtuel. Par exemple, lorsqu'un objet est éclairé, il réfléchit une partie non négligeable de cette lumière, modifiée par sa matière, vers les objets qui l'entourent. L'apparence de ces objets est alors modifiée par cette lumière colorée. C'est ce que l'on appelle communément l'éclairage indirect. Il est très important pour ajouter du réalisme dans la scène [SP93], [TL04] (voir *Figure 25*). Pour tenir compte de ces échanges globaux, nous pouvons là encore nous appuyer sur nos connaissances en matière de radiosité [HDSD99], [SH00] (voir *"A Practicle Analysis of Clustering Strategies for Hierarchical Radiosity", page 59*).

Figure 25:L'ambiance de l'environnement virtuel n'est pas prise en compte lors de l'incrustation ce qui rend l'insertion choquante (photomontage)

6.2 Qualité des images

Nous utilisons actuellement des caméras de résolution modérée qui ne sont pas appropriées à un rendu sur un mur d'images par exemple. Il nous faut donc augmenter la qualité des images. Nous pouvons nous tourner vers des solutions comme celles proposées par [WJVT+05] en utilisant un ensemble de 128 caméras mais ce système est très coûteux et peu envisa-

geable dans notre contexte. Je préfère imaginer une solution à base de blocs de quelques caméras (deux ou quatre) pour augmenter la qualité des images (voir *Chapitre 3–Acquisition des silhouettes, 6. Et maintenant ?, page 20*).

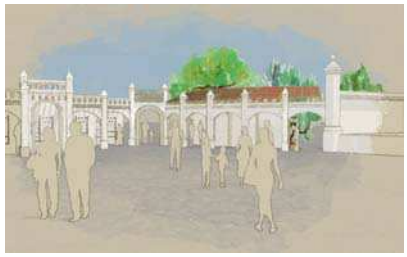
6.3 Rendu "non photoréaliste"

Nous avons présenté en introduction le "rendu réaliste" comme l'affichage d'une scène virtuelle et d'éléments réels sans que l'on puisse remarquer d'incohérences. En fait, la frontière avec le rendu "non photoréaliste" est très proche. En particulier, nous pouvons imaginer sans difficulté intégrer notre modèle 3D de l'animateur dans une scène qui aurait un rendu dans un style donné, comme de l'aquarelle, du dessin à la plume, etc. (voir *Figure 26*). Mais c'est une recherche à long terme puisqu'il faut tout d'abord résoudre le problème de navigation dans des modèles rendus de manière non photoréaliste (même si des travaux sont en cours pour aller dans ce sens [KDMF03], [DFR04]).

© wang-wang@vip.sina.com

© HLM Design International Ltd

© Carlos Marrero, Fort Myers, Florida USA



© Javier Sandoval, Construsol Digital Services, Mexico

Figure 26 : Exemple de personnages adoptant le même rendu non-photoréaliste que le monde virtuel dans lequel il évolue. (Ces images proviennent de la "Piranesi Gallery" et ont été créées à l'aide du logiciel "Piranesi" http://www.informatix.co.uk/piranesi/)

6

Interaction avec le monde virtuel

1 Introduction

Dans ce chapitre, l'**interaction avec le monde virtuel** concerne uniquement la modification de la géométrie de la scène ou du modèle reconstruit. En particulier, les interactions de type "échanges lumineux" ne sont pas discutées ici, elles font partie du chapitre précédent (voir *Chapitre 5–Rendu réaliste, 6.1 Rééclairage, page 46*).

L'interaction (géométrique) a été traitée dans nos recherches à des fins plutôt ludiques. En effet, ma spécialité étant le "rendu d'images de synthèse" plutôt que "l'interaction homme/machine", nous avons exploré très "naïvement" les possibilités de notre plate-forme. Cependant, les interactions entre l'animateur et le monde virtuel dans lequel il est plongé contribuent très largement à la sensation d'immersion. Il est donc important de développer les recherches dans ce sens.

Nous allons donc reprendre les hypothèses dans le cadre du projet CYBER, étudier les problèmes de latence et proposer quelques exemples d'interaction avec le monde virtuel que nous avons développés.

2 L'interaction dans le projet CYBER

Dans le projet CYBER, la contrainte a toujours été le temps réel afin de pouvoir interagir avec le monde virtuel. Nous avons aussi interdit tout capteur intrusif qui pourrait gêner l'animateur dans ses mouvements. À terme, nous devrions reconnaître le squelette de l'animateur et utiliser cette structure pour effectuer des interactions (voir *Figure 1*). À l'heure actuelle nous ne disposons que d'une géométrie sous forme de polygones ou de voxels sans information morphologique. Nous ne sommes donc

pas capables de distinguer un pied d'une main ! Les interactions sont donc limitées.

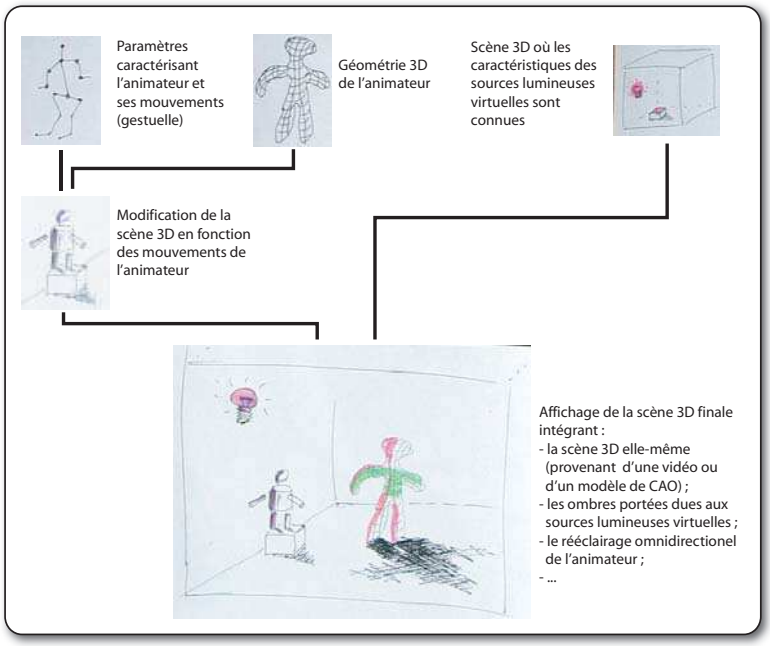


Figure 1 :Module "Interaction" dans le projet CYBER.

3 Temps réel et latence

Pour que l'animateur puisse interagir avec le monde virtuel, il est indispensable que l'ensemble des processus soit exécuté en temps réel. Dans notre implémentation [HLS04] (voir *"A Real-Time System for Full Body Interaction", page 125*), nous arrivons à traiter l'ensemble des informations : acquisition, construction du modèle, rendu et interaction en moins de 30 ms (voir *Figure 2*).

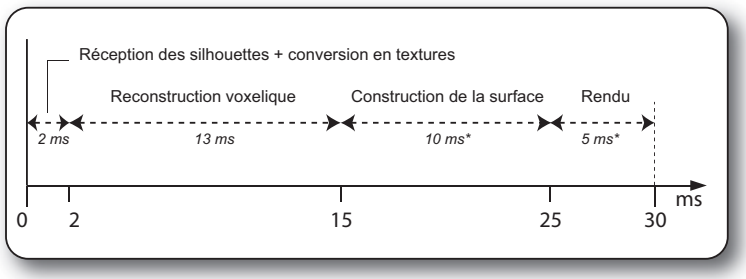


Figure 2:Temps passé dans chaque processus. (*) Le temps de construction du modèle et son rendu dépendent de la complexité de sa surface – les chiffres proposés sont des bornes supérieures.

La latence du système est un point crucial de l'interaction. En effet, si l'animateur bouge son bras et qu'il faut quelques secondes avant de voir le mouvement dans le monde virtuel, aucune interaction ne sera possible. Nous avons donc optimisé les transmissions des information pour obtenir une latence constante d'environ 4 images, c'est-à-dire 0.12 s (voir *"A Real-Time System for Full Body Interaction"*, paragraphe "*5. Interaction*", page *page 132*).

4 Exemple d'interaction

Nous nous sommes fondés sur le modèle voxelique pour effectuer nos interactions. Le principe est très simple, il suffit qu'un certain nombre de voxels de la grille soient occupés par l'animateur pour déclencher une action à cette position. Nous avons développé plusieurs interactions, cependant l'apport scientifique étant faible, nous n'en présenterons que deux [HLS04] (voir *"A Real-Time System for Full Body Interaction", page 125*).

"Slider" virtuel

L'interaction la plus évoluée consiste à simuler un "slider" virtuel. Plus le "slider" est haut et plus le nombre de balles tombant est important (voir *Figure 3*). Le jeu est alors de les faire rebondir sur une raquette (réelle) ou n'importe quelle partie du corps (voir *Figure 4*).

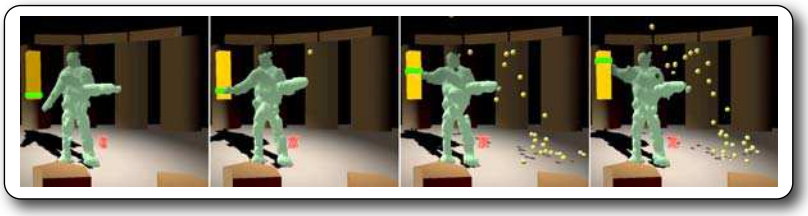


Figure 3 : Manipulation d'un "slider" virtuel pour régler le flux de balles.

Figure 4 : L'animateur fait rebondir les balles virtuelles sur sa raquette réelle.

Effets spéciaux

Le deuxième type d'interaction est basique, il s'agit de toucher un objet de la scène (ici un 'x' rouge) pour provoquer une "explosion" de l'animateur en utilisant un système de particules (voir *Figure 5*).

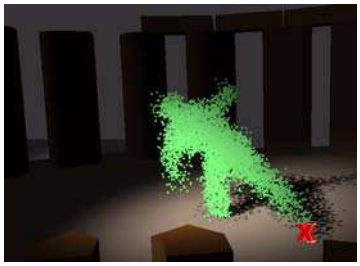


Figure 5 : Explosion de l'animateur en utilisant un système de particules.

5 Contributions

Nous avons proposé une chaîne complète, temps réel avec une latence très faible permettant de traiter l'acquisition vidéo, d'effectuer la construction du modèle, de calculer des ombres portées et d'interagir avec des éléments du monde virtuel [HLGB03], [HLS04] (voir *Chapitre 8–Articles,*

3. Real-Time Capture, Reconstruction and Insertion into Virtual World of ..., page 91).

Nous avons présenté un certain nombre d'interactions pour lesquelles il n'est pas nécessaire de connaître la morphologie de l'animateur [HLS04] (voir *Chapitre 8–Articles, 5. A Real-Time System for Full Body Interaction, page 125*)

6 Et maintenant ?

Pour développer l'interaction avec le monde virtuel, il sera nécessaire de connaître la morphologie de l'animateur, probablement en retrouvant son squelette [TMSS02], [CTMS03]. Nous pourrons ainsi connaître la position des mains et envisager des interactions plus fines. Cependant, ceci risque d'être difficile si nous voulons conserver la possibilité de travailler sans connaissance *a priori* des objets et/ou animateurs que l'on désire capturer et insérer dans le monde virtuel.

7 Conclusion

Durant ces dernières années et au travers des deux ACI CYBER et CYBER-II, j'ai travaillé sur plusieurs domaines. En premier lieu, les besoins matériels de notre plate-forme d'acquisition ont dûs être établis de sorte qu'elle réponde à nos contraintes et qu'elle soit pérenne. Cette élaboration, puis la construction de cette salle dédiée, m'ont permis de me familiariser avec les éclairages de studio et les caméras numériques. Les autres domaines sur lesquels j'ai orienté ma recherche correspondent davantage à des problèmes scientifiques. Il s'agit, en effet, d'acquisition, de modélisation, de rendu réaliste et d'interaction. Je vais reprendre à présent mes contributions puis présenter mes axes de recherche à venir.

1 Contributions

L'**acquisition de silhouettes** a été le premier sujet de recherche. Nous avons pris le parti de ne pas travailler avec des fonds de couleur uniformes mais plutôt avec des fonds quelconques. Ces recherches ont rempli nos attentes puisque nous sommes capables d'extraire les silhouettes de l'animateur en temps réel. Il y a bien sûr quelques limitations pratiques. Il ne faut pas que l'animateur porte des habits dont la couleur se confond avec celle du fond de la salle. De plus, l'éclairage doit limiter au maximum les ombres portées (surtout au niveau des pieds) pour ne pas entraîner des erreurs de reconstruction. Nous avons discuté de la précision nécessaire lors de l'acquisition des silhouettes pour obtenir un modèle satisfaisant de l'animateur.

Nous avons abordé la **modélisation** sous deux angles : la création d'un modèle articulé et celle d'un modèle non-articulé. Le premier a permis de montrer clairement les limites lorsque l'on ne dispose que d'une seule caméra. En effet, des résultats temps réel peuvent être obtenus mais

notre approche utilisant des bandeaux de couleurs ne permet pas d'obtenir une reconnaissance fiable à tout moment. Concernant la modélisation non-articulée, nous avons proposé un certain nombre d'algorithmes temps réel permettant de produire un modèle volumique sous forme de voxels ou un modèle surfacique sous forme d'un ensemble de polygones. Les deux approches ont chacune leurs avantages et inconvénients. Dans les deux cas, le faible nombre de caméras utilisées s'avère être une limitation à la précision de la reconstruction. C'est pourquoi, nous proposons d'augmenter à la fois le nombre de caméras ainsi que leur résolution.

Le travail sur le **rendu réaliste** temps réel, nous a permis de publier un état de l'art des algorithmes produisant des ombres douces en temps réel. À partir de là, nous avons proposé une nouvelle approche temps réel, "basée image" et physiquement plus proche de la réalité. Nous travaillons sur l'habillage de la géométrie de l'animateur pour augmenter le réalisme. C'est actuellement notre principal sujet de recherche.

Nous avons élaboré une chaîne complète de traitements depuis l'acquisition jusqu'au rendu réaliste en **temps réel**. Il ne restait plus qu'à travailler sur l'**interaction** de l'animateur avec le monde virtuel. C'est ce que nous avons fait au travers de démonstrations très simples d'un "slider" virtuel, de jeux de balles ou d'effets spéciaux. Ces interactions utilisant un modèle voxelique, ont montré la faisabilité d'interagir avec le monde virtuel. Cependant, il faut bien reconnaître que tant que nous ne serons pas capables de reconnaître la morphologie de l'animateur, les interactions resteront limitées.

2 Et maintenant ?

Ces travaux ont apporté un certain nombre de réponses mais ont suscité des interogations ouvrant la porte à de nouveaux axes de recherches. Ainsi, parmi les points sur lesquels je souhaite poursuivre mes recherches, nous trouvons le texturage omnidirectionnel de l'animateur. Ce sujet a été largement développé dans le chapitre sur le "*Rendu réaliste*". J'ajouterai simplement qu'il n'est peut-être pas nécessaire de passer par un modèle géométrique explicite pour effectuer ce rendu réaliste. Une approche utilisant les particularités des nouvelles cartes graphiques semble envisageable.

Toujours dans ce chapitre de "*Rendu réaliste*", nous avons présenté les effets de l'éclairage global et les interactions lumineuses animateur/monde virtuel. Le problème reste que ce type d'interactions lumineuses est très complexe et très difficile à réaliser en temps réel. Néanmoins, nous pouvons espérer qu'avec une modélisation adéquate des sources lumineuses, nous pourrons obtenir de bons résultats.

Enfin, si nous voulons interagir “physiquement” avec le monde virtuel, il nous faudra obligatoirement reconnaître les éléments reconstruits, que ce soit un ou plusieurs animateurs avec un ou plusieurs objets. Ce champ de recherche est certainement le plus vaste.

8

Articles

Liste des publications depuis 1999 et ayant un rapport avec le sujet traité dans ce mémoire.

[HDSD99] Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, George Drettakis, "*A Practical Analysis of Clustering Strategies for Hierarchical Radiosity*", Computer Graphics Forum (Proc. of Eurographics '99), 18(3), pp. 221-232, Sep 1999

Voir page 59

[SH00] François Sillion, Jean-Marc Hasenfratz, "*Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer*", Third Eurographics Workshop on Parallel Graphics and Visualisation, Girona - Sep 2000, pp 61-74

Voir page 75

[HLGB03] Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, Edmond Boyer, "*Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors*", Vision, Video and Graphics, pp. 49-56, 2003

Voir page 91

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion, "*A survey of Real-Time Soft Shadows Algorithms*", Computer Graphics Forum, 22(4), pp 753-774, Dec. 2003

Voir page 101

[HLS04] Jean-Marc Hasenfratz, Marc Lapierre, François Sillion, "*A Real-Time System for Full Body Interaction with Virtual Worlds*", Eurographics Symposium on Virtual Environments, pp. 147-156, 2004

Voir page 125

[OH05] Alexandrina Orzan, Jean-Marc Hasenfratz, "*Omnidirectional texturing of human actors from multiple view video sequences*", Proceedings of the Conference on Computer-Human Interaction, Sept. 2005

Voir page 137

[HLS05] Jean-Marc Hasenfratz, Marc Lapierre, François Sillion, "*Fast Voxel Carving Using OpenGL Facilities*", **Rejected** to Journal of Graphics Tools, ?(?), 2005

Voir page 145

[ALHH+05] Lionel Atty, Marc Lapierre, Nicolas Holzschuch, Chuck Hansen, J-Marc Hasenfratz, Francois Sillion , "*Soft Shadow Maps: Efficient Sampling of Light Source Visibility*", **Rejected** to Eurographics Symposium on Rendering (2005)

Voir page 153

1 A Practicle Analysis of Clustering Strategies for Hierarchical Radiosity

[HDSD99] Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, George Drettakis,
"A Practicle Analysis of Clustering Strategies for Hierarchical Radiosity",
Computer Graphics Forum (Proc. of Eurographics '99), 18(3), pp. 221-232, Sep 1999

EUROGRAPHICS '99 / P. Brunet and R. Scopigno
(Guest Editors)

Volume 18 (1999), Number 3

A Practical Analysis of Clustering Strategies for Hierarchical Radiosity

Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, George Drettakis

iMAGIS – GRAVIR/IMAG-INRIA, Grenoble, France

Abstract

The calculation of radiant energy balance in complex scenes has been made possible by hierarchical radiosity methods based on clustering *mechanisms. Although clustering offers an elegant theoretical solution by reducing the asymptotic complexity of the algorithm, its practical use raises many difficulties, and may result in image artifacts or unexpected behavior. This paper proposes a detailed analysis of the expectations placed on clustering and compares the relative merits of existing, as well as newly introduced, clustering algorithms. This comparison starts from the precise definition of various clustering strategies based on a taxonomy of data structures and construction algorithms, and proceeds to an experimental study of the clustering behavior for real-world scenes. Interestingly, we observe that for some scenes light is difficult to simulate even with clustering. Our results lead to a series of observations characterizing the adequacy of clustering methods for meeting such diverse goals as progressive solution improvement, efficient ray casting acceleration, and faithful representation of object density for approximate visibility calculations.*

1. Introduction and Motivation

In scenes with great geometric complexity containing hundreds of thousands or even millions of polygons global illumination algorithms require the grouping, or clustering, of the individual primitives. In this way light exchanges can be treated at the level of the clusters and thus the computational complexity of the radiosity solution becomes manageable[14, 12]. Unfortunately, approximations made by hierarchical radiosity algorithms using clustering are very sensitive to the quality of the cluster hierarchy. Due to the complexity of the algorithms and data structures (by definition we are working with very complex models), no experimental analysis of the behavior of clustering algorithms has been undertaken today. Willmott and Heckbert [15] provided an inspiring study for progressive refinement radiosity and hierarchical radiosity without clustering, but evidently this study was restricted in the type of scene considered.

In this practice-and-experience paper, we investigate clustering algorithms for hierarchical radiosity in a practical context. In particular, we have chosen an experimental approach, by comparing the performance of different clustering algorithms. We have concentrated our attention on models provided by real-world applications, in an attempt to uncover problems which real users of radiosity will encounter.

We propose a taxonomy of clustering algorithms, based both on the type of data structure used, and the type of construction algorithm. We then proceed to define requirements for a clustering algorithm. In particular, a clustering algorithm should provide the user with an intuitive time-quality tradeoff: the more time is spent on a solution, the better the quality of the solution. Several other desirable properties are also identified, such as limiting the overlap of clusters, optimizing the "tightness" of the fit of clusters around objects and appropriate size of the clusters with respect to the contained objects.

Once the requirements have been defined, we proceed with a series of experiments run on models used mainly in real-world applications. Various parameters are measured, including the image quality for varying simulation parameters, the cluster construction time, the quality of the hierarchy using different criteria and the speed of ray-tracing for each cluster hierarchy.

The results of these experiments have allowed us to observe a number of interesting properties of clustering, which are discussed in detail. This in-depth study of clustering leads to the understanding that there exists no universal, ideal clustering method, while explaining the relative merits of various approaches.

2. A Taxonomy of Clustering Algorithms

Clustering for hierarchical radiosity was introduced by Smits *et al.* [14] and Sillion [11, 12]. Clustering algorithms can be classified by considering two aspects important to their usage:

1. The choice of data structure used. Broadly speaking, two categories have been presented: regular, typically axis-aligned subdivisions of space and hierarchies of bounding volumes (HBV), which are more closely adapted to the object geometry. Examples of such structures include *k*-d trees and Octrees. Hybrids have also been proposed but have not been used to date in clustering for radiosity.
2. The type of construction algorithm. Again, two basic categories have been used: top-down and bottom-up construction.

2.1. Data Structure Choices

The data structures used for clustering have been mostly inherited from traditional spatial subdivision structures used in graphics.

2.1.1. Octrees and *k*-d Trees

Regular structures such as octrees or *k*-d trees have several advantages:

- They are fast to build (see Section 5.2), and easy to construct since the form of the clusters is (nearly always) predefined.
- They can provide fast ray-tracing since they use traditional ray-traversal mechanisms (see Section 5.2).

Their disadvantages are not specific to clustering for illumination, but due to the rather inflexible nature of their construction:

- Objects which intersect cell boundaries do not have a trivial placement in the tree. As a consequence a heuristic needs to be determined to place the object at an appropriate level.
- The tree can be very deep if the scene contains objects with large differences in scale.
- Since the shape of sub-clusters is prescribed by the subdivision mechanism, many empty clusters can be created. These clusters consume memory and resources since they are considered in all radiosity operations.
- Cluster boundaries do not tightly fit the set of contained objects, resulting in poor-quality estimates of the optical density, for the volumetric estimation of visibility [12].

The first clustering algorithm to use *k*-d trees for illumination was presented by Sillion [11]. This approach uses a traditional, axis-aligned *k*-d tree into which objects are inserted. The strategy chosen for placement of objects intersecting cell boundaries is to put objects at the lowest level entirely containing them. For many models, this can have very negative consequences since a large number of objects can end up at very high levels in the hierarchy, with adverse effects on computation speed. Since the refinement algorithm must, when refining a link to a cluster, create new links for each of its children, a high branching factor may result in long computational times.

2.1.2. Hierarchy of Bounding Volumes

Algorithms based on bounding volumes hierarchies have been used by several researchers [14, 13, 7]. These algorithms use rectangular axis-aligned bounding boxes, and share the following qualities:

Advantages:

- If built correctly, the cluster hierarchy adapts well to the organization of the scene into individual objects (possibly each having its own sub-cluster hierarchy).
- An "intuitive" hierarchy can be produced for a scene with very different object sizes, without creating empty clusters.
- Clusters can be made to tightly fit their contents.

Disadvantages:

- Overlapping clusters are generally unavoidable.
- Bad clusters often result when the scene is considered as a set of individual polygons, without taking advantage of the object structure (*e.g.*, clusters mixing parts of different nearby objects).

Hybrid data structures have also been developed for clustering objects in different domains (notably for ray-tracing acceleration). Cazals *et al.* [2, 3] construct hierarchies of uniform grids, and Klimazewski *et al.* [10] present a similar approach. These methods are however designed to optimize ray-tracing by constructing regular grid structures, and are thus unsuitable "as is" for clustering. Some ideas however, in particular those concerning grouping of objects, by Cazals *et al.* [2, 3], could be applied in part to future clustering algorithms.

2.2. Construction Algorithms

In this section we re-visit the algorithms described above by construction algorithm type. The basic approaches are top-down and bottom-up construction. We will briefly discuss some issues of manual clustering, which is often used in industry and even in research.

2.2.1. Top-down Clustering

Typical top-down construction algorithms include the *k*-d tree (KDT) construction used by Sillion [12]. An initial cell is created, and objects are subsequently added into the cell by appropriately subdividing the cell so that the object "fits" in a sub-cell. As mentioned above, objects crossing cell boundaries are placed high up in the hierarchy.

Christensen *et al.* [4] build a hierarchy of bounding volumes starting with the bounding box of the entire scene. The bounding box is split into eight octants. For each surface contained in this bounding box, if the size of the object is smaller than that of an octant, the object is inserted into the octant containing its centroid. Otherwise, the object is attached as a direct child of the cluster at this level. A new bounding box of each octant is computed, and the algorithm continues recursively in the same manner. In this paper we refer to this algorithm and data structure as TF-OCT ("tight-fitting octree").

2.2.2. Bottom-up Clustering

Bottom-up construction of clusters is inherently more complex, since it requires the examination of the existing objects and their mutual spatial relationships. Since it is in a certain sense an optimization process, the complexities of algorithms suited to such constructions rapidly become quadratic or higher in the number of objects to be processed.

Smits *et al.* [14] mention they use a "modified Goldsmith-Salmon algorithm" without describing the specifics [14]. Sillion and Drettakis employ a hierarchy of regular grids to filter the objects and clusters in order of increasing size, and produce candidate clusters based on spatial proximity [13]. The same algorithm is used by Gibson and Hubbold [7]. In the rest of this paper we refer to this method as PROXI, for "Proximity clustering".

Note that as we choose to group objects based on a minimization function (*e.g.*, minimize the volume of the clusters created with respect to the objects being inserted) [13, 7], an optimal solution requires an exhaustive test of all the combinations of groupings of the objects being considered. Clearly, this expense is extremely costly. A more appropriate grouping approach could be that of Cazals *et al*[3].

2.2.3. Manual Construction

The difficulty of clustering is such that automatic methods are not able to treat all scenes effectively. As a consequence user intervention is inevitable at some stage in the process. Examples of such intervention are the definition of "natural clusters" such as those defined by the group of polygons belonging to a single "object" (a chair for example) or a logical group such as the set of all objects on top of a desk. The special case of touching objects is also important, since it is a way of defining object hierarchies.

Some of these interventions can be handled at input (often the set of objects defining a chair object is defined as a group by the modeling program and then instanced). Such information should be incorporated by the clustering system and used to its advantage when available. Other cases are much harder (*e.g.*, the "objects on the desk" case, or touching objects).

2.3. Improved Approaches

An extended version of KDT, which we call OBT for "Overlapping Binary Tree", was derived as an attempt to merge some of the benefits of Christensen's octree construction and the binary trees obtained with KDT. The algorithm is very similar to the KDT construction, but here, objects crossing the cell boundaries are pushed down in the hierarchy only if the ratio of their size and the considered cell size, is higher than an overlap parameter which can be set by the user (when set to 0, a KDT hierarchy is obtained). The clusters produced by this algorithm can be larger than those of KDT, and therefore may overlap, but the parameter provided gives us control on the maximum potential overlap between adjacent clusters. Specifically, for a given value of the overlap parameter λ, the following relations are true:

- maximal size of the KDT cell to which an object of size S is assigned:

$$x = \frac{2S}{\lambda}$$

- maximal overlap between two adjacent OBT clusters of original size d:

$$o = \frac{8}{3}\lambda d$$

These relations ensure that large clusters will not be overwhelmed by large (and spatially sparse) collections of small objects.

As a consequence, the OBT algorithm produces a deeper hierarchy than KDT, and thus even more empty clusters. In order to get rid of empty clusters, we developed another algorithm that uses an auxiliary OBT hierarchy to build HBV clusters in a second pass. In this second step, we keep only the bounding box of the contents (objects and child clusters) of each non-empty OBT cluster. Therefore, each cluster in the obtained HBV hierarchy will have at most two child clusters. We call the resulting new structure and algorithm OKDT (Overlapping KDT).

Unfortunately, OKDT proved to be unable to separate objects formed of thin, long polygons, such as cylinders, because they were unlikely to be inserted deep in the hierarchy. As a consequence, several clusters contain too many surfaces. This had adverse consequences on ray-casting and refinement time (average branching factor being central for hierarchical algorithms). Therefore, in order to improve our algorithm we also tried applying a PROXI clusterisation process on each cluster containing more than a dozen polygons. We call this algorithm OKDT-P (Overlapping KDT with Proximity second pass).

3. Requirements for a Clustering Algorithm

The most important goal for a good clustering algorithm is to group the objects in a way which represents light transfer to or from the group in the most faithful manner. At the heart

of the hierarchical radiosity algorithm is the assumption that it is possible to replace a "complete" calculation by a simpler one, performed using simplified representations such as clusters. This idea can only be exploited if the resulting simplification only introduces modest perturbations in the calculation.

Unfortunately, a precise definition, or even a set of quantitative yardsticks allowing us to evaluate the quality of such a representation do not currently exist. Instead, a set of heuristics have been developed by various researchers in this domain (*e.g.* [4, 13, 7]). Considering the existing body of work, we can identify two sets of desirable properties for a clustering technique: those affecting the quality of the simulation, and those affecting the overall efficiency of the applications.

3.1. Requirements Regarding the Quality of the Results

We outline below some of the required properties in order to arrive at a satisfactory simulation of light transfer:

1. Monotonicity with respect to light transfer precision. If a light transfer previously represented at a certain level becomes represented at a finer level of the cluster hierarchy, the precision of the light transfer should be increased. If we consider the Time-error graph obtained by plotting the computation time as a function of the solution error, for different tolerance thresholds, we would like to obtain a *smooth and monotonic* function.
2. Overlapping clusters should be avoided as much as possible. Overlapping is problematic mainly because it implies the treatment of the transfer of light of a volume to itself, which is difficult to represent and to express in terms of the hierarchical radiosity formalism. This is especially true in the case where error bounds are estimated to drive the hierarchical refinement.
3. The nature of object group shapes should be preserved as much as possible. Clusters which contain large regions of empty space and scattered small objects should be avoided. Although this may seem evident, many automatic clustering algorithms have trouble respecting this requirement.
4. Objects should always belong to a cluster of "appropriate" size, with respect to their own dimensions. This requirement is difficult to quantify, but is especially important in the context of approximate visibility calculation, where the attenuation of light passing through clusters is estimated based on a volumetric analogy. This analogy relies on the calculation of an optical density for each cluster, which is only meaningful for clusters with well-distributed (i.e. "random") collections of similarly-sized objects [11].

3.2. Efficiency Considerations

Considering the major impact of cost considerations on the usability of clustering radiosity systems, particular attention must be paid to the two following aspects:

3.2.1. Building the Hierarchy

First, we are of course concerned about the efficiency of hierarchy construction. Even though the actual clustering phase is generally a preprocess, and can sometimes be stored with the model, it is still preferable to have efficient construction algorithms:

- Fast cluster construction is important in the modelling/design stage where the model changes significantly and thus long clustering times hinder lighting experimentation.
- For extremely large models it may be impractical to store an additional high-overhead data structure describing the cluster hierarchy.
- Finally, the application of hierarchical radiosity with clustering to dynamically changing environments [5] may require at least a partial rebuild of the cluster hierarchy at interactive rates.

3.2.2. Acceleration of Ray Casting

Another important problem is the potential use of the cluster hierarchy as a supporting data structure to accelerate ray casting queries. Such queries are used for image generation or more often, in the case of radiosity, for visibility calculations when computing form-factors. Indeed, ray casting is the method of choice for visibility estimation in hierarchical radiosity, because the hierarchical algorithm fragments the calculation into a large number of individual queries, each relative to a different emitter/receiver pair [9]. Global approaches such as hemi-cube calculations are therefore inappropriate for hierarchical radiosity.

The requirements of ray-tracing acceleration are often contradictory with those of clustering for illumination. For example, structures which minimise the number of intersections on average in a statistical sense, such as the algorithm of Goldsmith and Salmon [8], result in clusters which contain elongated bounding boxes containing large empty spaces. These clusters are inappropriate for light transfer estimation or optical density estimation, as outlined above.

A possible solution is the creation of two separate structures, one for clustering and one for ray-tracing acceleration. This however would be undoubtedly far too expensive in memory for very large models. In practice, we have observed that the performance cost implied by the use of the clustering structure for ray acceleration is most often acceptable. Some comparative results concerning the performance of various cluster hierarchies as ray tracing accelerators are presented in Section 5.

4. Experimental methodology for evaluation clustering algorithms

4.1. Methodology

As stated earlier, clustering is necessary to make the illumination computation of industrial scenes tractable. Its most

important drawback is that it makes error control very difficult, where it would be necessary to allow fast solutions to be calculated with an acceptable precision. Given the previously described clustering algorithms, we need to determine the critical parameters for their behavior.

Until now, most of the scene models used in the literature were designed by researchers for research purpose. We feel that performing our study on such scenes would have in some way "hidden" most of the problems involved by clustering. In raw “industrial” scenes, poor quality initial meshing, dense distribution of objects or randomly ordered polygons can make clustering algorithms barely usable.

Thus, our approach has been to perform a sufficient number of experiments on a set of complex, "real life" scenes in order to understand and compare each algorithms behavior in terms of quality-versus-time tuning. We limited the range of our experiments to fairly coarse and fast calculations where the impact of clustering is significant.

4.2. Clustering Strategies Studied

The following table summarizes the set of clustering strategies considered in this paper.

PROXI: Proximity cluster. Bottom-up construction after size filtering.

OKDT: Overlapping *k*-d tree. Top-down allowing partial overlap.

OKDT-P: Overlapping *k*-d tree with limited branching. OKDT modified to re-cluster cells with many children, using PROXI locally.

TF-OCT: Tight-fitting octree. Top-down, layer-by-layer octree construction, with re-fitting of octree cell before subdivision.

We did not submit the KDT algorithm to our tests because our experience with it proved that its average branching factor was far too high for it to be usable with scenes as large as the one we used.

4.3. Test Scenes Chosen

We performed our tests on four different scenes, shown in Figure 1. Three of them are rather large industrial-type models while the fourth one is provided as a comparison to show that clustering usually behaves very well when applied to small scenes designed for research purposes. We have taken these scenes as representative scenes for the clustering problem, allowing us to identify the different problems of the algorithms which we will test.

AIRCRAFT (184,456 polygons)
Model of an aircraft cabin (courtesy of LightWork Design Ltd). All objects have been tessellated into (rather small) triangles to account for the rounded shapes.

VRLAB (30,449 polygons)
A virtual reality lab with two floors and mostly overhead lighting (courtesy of Fraunhofer Institut für Graphische Datenverarbeitung). This scene has a mixture of large polygons, likely to be subdivided, and very small patches (on chairs and desktop computers).

CAR (216,157 polygons)
A model of a car interior with very small details, lit by a single overhead console fixture (courtesy of BMW).

OFFICE (5,260 polygons)
A model of a simple office scene. This model is much smaller than the other three and is provided to show that clustering performs well on this kind of small scene usually found in the literature.

4.4. Tests Performed

For each of these scenes, and for each clusterizer we decided to run the following experiments, designed to measure key aspects of clustering:

- **Data structure quality**: We listed the number of clusters in the hierarchy, as well as the average number of surfaces and clusters in each cluster node. These figures are needed to estimate both the memory cost of the hierarchy (compared to the cost of the actual geometry of the scene) and its efficiency for hierarchical radiosity. Recall that the computation speed of cluster based hierarchical radiosity depends on the average branching factor because the refinement algorithm must, when refining a link to a cluster, create new links for each child of this cluster.
- **Measure of time needed to build the cluster hierarchy.**
- **Solution quality**: To evaluate image quality we chose to use a "visual quality" error evaluation on images obtained for a given set of viewpoints rather than a view-independant error metric since we want to study the visual quality rather than the accuracy of the energy transfer quantities. Images were compared to reference images (resulting from a maximum precision calculation) using the following metric: we transform all pixels from RGB space into chromaticity space XYZ. The global image error is then the L_2 norm of the pixel-by-pixel difference image between the current and the reference image, evaluated in CIELUV space [6].
- **Algorithm usability**: To study the "quality versus time" behavior of each algorithm, we rendered each scene using each clustering algorithm, changing only the parameters controlling our refinement process (we used a BF-like refinement algorithm [9], and an error-bound based refinement, both of them showed similar behavior on clusters), measuring the time needed and evaluating the resulting image quality using the method previously described. Since we are interested in the effects of the cluster hierarchy, we do not perform tests on very precise solutions, which involve only surface-to-surface energy exchanges. Instead, we chose our refinement parameters in such a way that the proportion of energy gathered through links

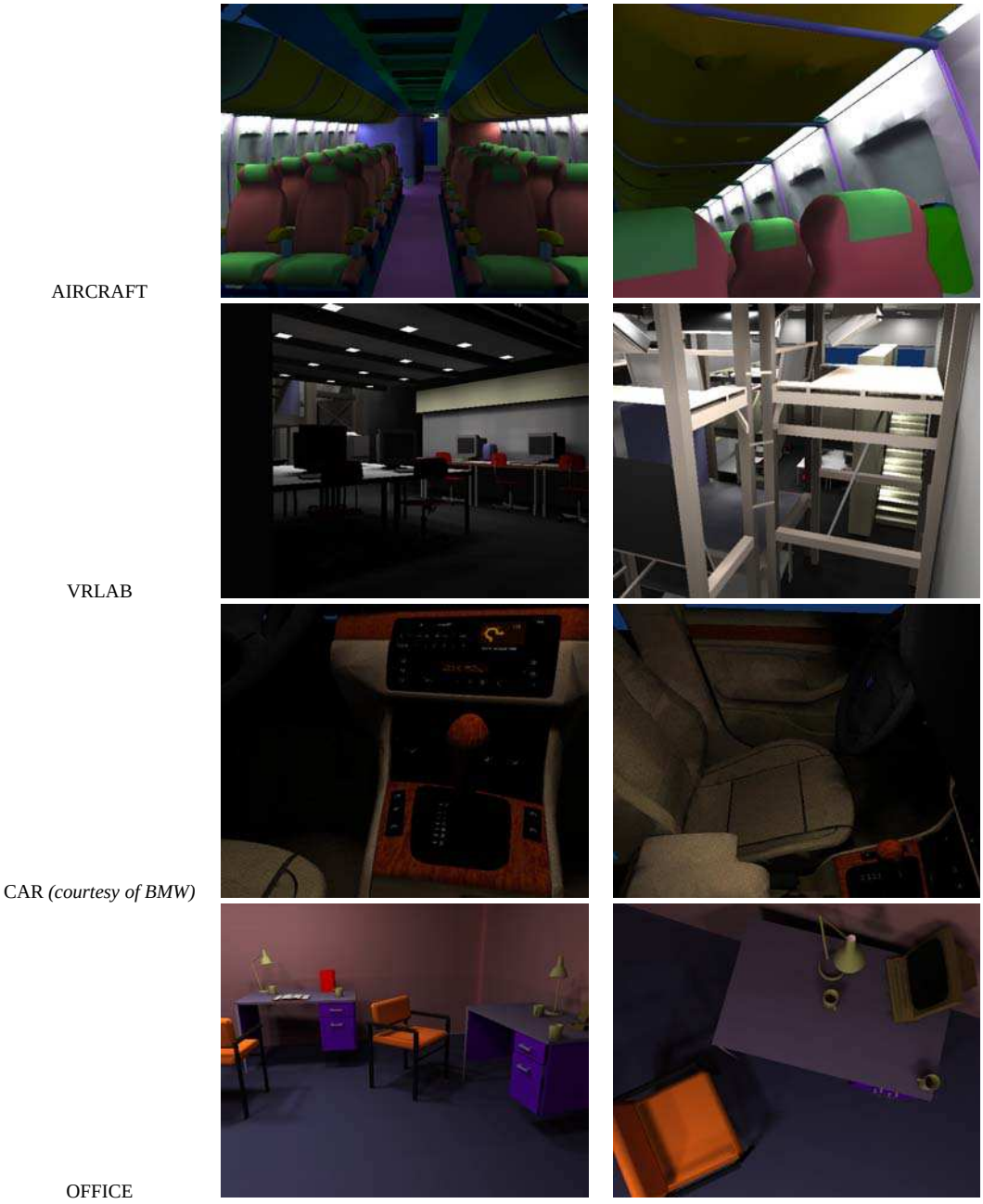


Figure 1: *The four test scenes.*

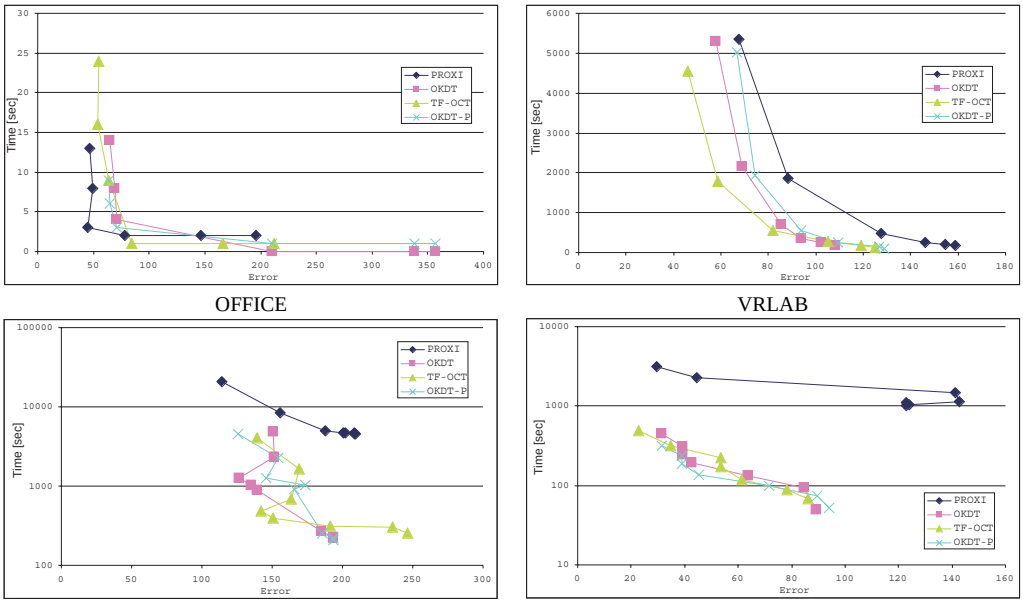


Figure 2: *Time-error curves*

involving clusters is significant (most of the time varying between 30% and 100%).

- **Ray casting acceleration**: As we said in Section 3.2.2, we have seen that it is desirable to use the cluster hierarchy as an acceleration structure to answer visibility requests (see Section 3.2). We decided to test the efficiency of each structure for ray-casting. Therefore, we chose 100,000 random pairs of surfaces in each scene, and measured the total time needed to search for a possible occlusion between each pair.

We also ran a separate set of experiments to evaluate the efficiency of the cluster hierarchy when using approximated volumetric visibility [12]. This alternative to classical exact visibility computation accelerate rendering times when precise shadow area determination is not needed. Instead of the previously defined scenes, we used a model of several trees (Figure 5) representative of some applications where an average representation of light transfers may be sufficient. It is also a good test case for the volumetric visibility algorithm: the set of leaves being a good approximation of a turbid media. We calculated the images using each clustering algorithm and then compared the difference with a reference image obtained using exact surface visibility.

5. Results

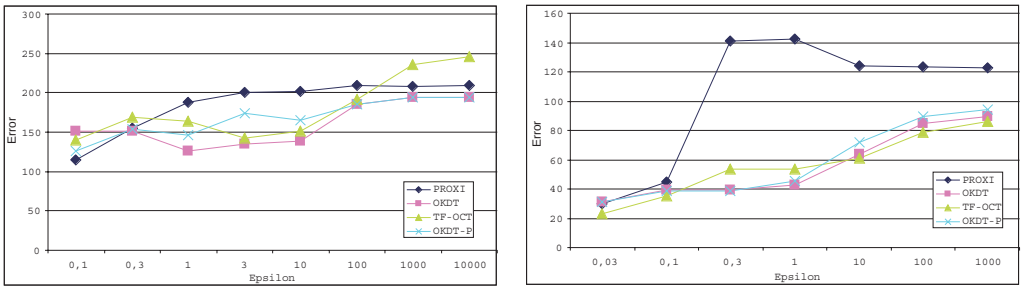
In this section, we present a number of observations drawn from the tests explained previously. We begin with an analysis of the quality aspects of the simulations, looking at the evolution of our error measure with the user-defined error tolerance. We then consider in more detail the capacity of different clustering techniques to assist visibility calculations, with a particular emphasis on computational efficiency.

We ran all algorithms on a Silicon Graphics computer (MIPS R10000 at 250 MHZ) with 4GB of memory.

5.1. Evolution of Solution Error

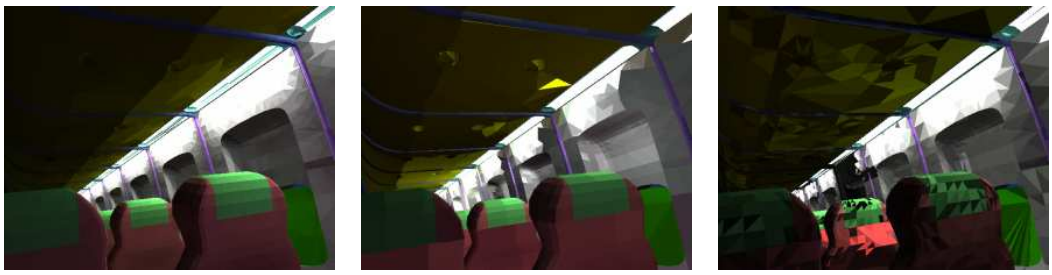
Recall from Section 3 that a major demand on the clustering mechanism (together with the chosen refinement strategy) is that the evolution of computation time and solution quality should be regular and monotonic as the user changes the error tolerance (Figure 3). Ideally, this would result in a very regular and monotonic time/error curve. Such curves are presented in Figure 2 for the four test scenes.

Looking at the four Time-error curves, we see two very different types of behaviors: for the OFFICE and VRLAB scenes, all curves are regular and fairly monotonic, whereas for AIRCRAFT and CAR the variation of error is more erratic. Indeed, looking at a plot of error as a function of the



AIRCRAFT Error CAR Error

Figure 3: *Error curves as function of user-supplied tolerance*



Reference TF-OCT PROXI

Figure 4: *Example solutions exhibiting different (visual) forms of error (see also color plate).*

user-supplied error tolerance, we find that the error in the solution does not always decrease as we reduce the tolerance. Because of limited space, we present only the error plots for AIRCRAFT and CAR (Figure 3); the corresponding curves for OFFICE and VRLAB are monotonically decreasing.

Why can the error increase when we decrease our tolerance? this unfriendly behavior occurs when links refined as a result of the error tolerance change produce a less accurate representation of radiosity exchanges. This is largely a question of refinement criteria, but is also influenced by the clustering strategy, as well as the distribution of objects in the scene. We observe that our scenes can be classified into two types: AIRCRAFT and CAR consist of many small polygons, because of a previous tessellation of the objects. VRLAB and OFFICE, on the other hand, contain objects of varying size, from large walls to small furniture components. Based on our experience and the results of the above experiments, we observe that clustering algorithms have more difficulty with the first type of scene (“polygon soup”). This is especially true of AIRCRAFT because 3D space is very densely populated, resulting in many interactions between clusters which are not separated by a significant distance. These interactions also present a particular challenge to the refinement criterion.

As an illustration of the typical errors created in an approximate solution for such scenes, consider the images in Figure 4. The two approximate solutions have a similar error under our measure, yet they appear quite different visually. The solution using TF-OCT clusters exhibits marked radiosity variations along axis-aligned boundaries, corresponding to the octree cells; on the other hand, the solution using PROXI shows a high variance of radiosity and a speckle pattern, due to the fact that nearby small objects can belong to many different and overlapping clusters, with markedly different radiosities.

5.2. Performance and Visibility Calculation with Clusters

We now consider the performance behavior of the clustering strategies in terms of construction time and as auxiliary structures for visibility calculations.

Construction Time

As explained in Section 2.2, bottom-up construction is a very expensive process since it amounts to an optimization procedure.

Observed computation times for the construction of the

cluster hierarchies support this prediction: the PROXI clustering strategy construction is always much slower than OKDT and TF-OCT, which operate top-down on simple recursive subdivision schemes. These two techniques always take less than 1% of the PROXI time. Interestingly, OKDT-P takes between 20% and 80% of the PROXI time, depending on the distribution of objects in the scene.

Approximate Visibility Calculations

The acceleration of visibility calculations using *equivalent extinction properties* of the clusters has been proposed by Sillion[12]. In this approach, the transmittance factor between two points in the scene is evaluated by considering the entire segment between the points, and its intersections with all clusters, then combining the corresponding attenuation values, in an analogy with partially absorbing volumes. This method, also adopted by Christensen *et al.* [4], is often faster than true ray casting using the surfaces, because of the smaller number of clusters and the ease of computation of ray-cluster intersections.

We computed approximate visibility using clusters in a scene dominated by direct lighting (from the sun), as shown in Figure 5. In this case, the shadow pattern on the floor is essentially an "X-ray image" of the cluster hierarchy, which greatly helps in the comprehension of the cluster distribution.

We observe a clear hierarchy in terms of shadow quality, in the order PROXI (best, notice high quality of trunk shadows), OKDT-P, OKDT, TF-OCT (poorest). Please see images in color section. This is consistent with the intuitive notion that PROXI starts from the objects and build clusters bottom-up, thereby building clusters that are very tight around the objects.

OKDT (and TF-OCT even more so) exhibits some incorrect shadows of large, blocky clusters, due to the constraints in the spatial subdivision. In this respect, OKDT-P effectively improves on OKDT, with a better fit around the objects and more precise shadows.

Ray Casting Acceleration

Interestingly, the computation times shown in Figure 5 increase with the quality of the shadows. This is consistent with the general observation that hierarchical structures with lower branching factors have more hierarchical levels and perform better for ray tracing acceleration.

This reasoning is supported by the analysis of cluster statistics on our test scenes. Figure 6 shows the variation of the following three quantities with the clustering technique, for each test scene:

- total number of clusters
- average number of child elements per cluster
- performance of ray tracing acceleration. This is measured by shooting a large number (100,000) of random rays through the scene and computing ray-surface intersections.

We first observe an obvious inverse correlation between the total number of clusters and the average number of children. In addition, TF-OCT has the largest branching factor for the cluster hierarchy because of its octal subdivision scheme. OKDT also has a fairly high number of children on average, because its construction mechanism offers no way to control this branching factor. Conversely, PROXI has a built-in mechanism limiting the number of children of any given cluster (this operates by grouping objects into overlapping sub-clusters). Therefore it exhibits the lowest branching factor. OKDT-P is intermediate, as expected, because by construction it avoids clusters with many children, handing them to the PROXI clusterizer. Still it avoids the overall large number of clusters of PROXI. A consistent best performer in terms of acceleration is therefore OKDT-P.

Finally, we note the conflicting nature of the two desires for (a) efficient ray tracing acceleration and (b) suitability for radiosity calculations (compare Figure 6 and Figure 2).

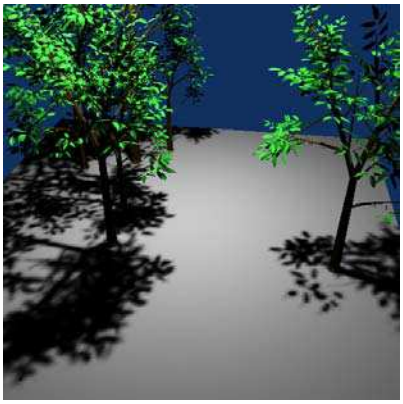
6. Conclusions and Future Work

We have presented an experimental analysis of clustering algorithms for hierarchical radiosity. A taxonomy of clustering algorithms was proposed, followed by a set of requirements for a good clustering algorithm. Guided by these requirements, we developed an experimental methodology based on an image-space quality measure. Extensive tests were run on scenes for the most part developed in real-world application contexts.

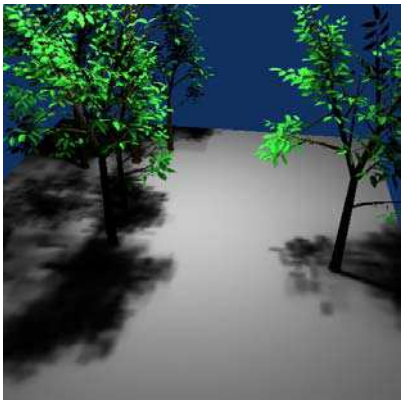
Drawing concrete conclusions from experimental tests such as those performed here is always a delicate task. Nonetheless, there are certain elements which we believe are clear enough to be singled out:

Clustering works well in many cases: in particular, for scenes containing objects of different sizes and a sufficient number of large initial surfaces (walls, floors etc.), all the clustering algorithms tested appear to perform well. The Time-error graphs are smooth and monotonic for these cases, presenting the user with an intuitive time-quality tradeoff.

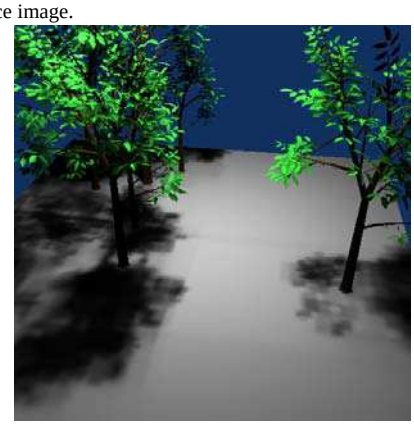
For scenes containing many small objects ("polygon soup"), existing clustering algorithms are less well-behaved. In particular, more time spent computing a solution does not always result in higher quality (see Section 5.1). This is even more troublesome since the scenes in question are typical of industrial "real-world" models, which are often the result of a fine tessellation of some unspecified and unrecoverable modeling format. It is clear that a new approach is required to treat such models, in order to build a hierarchy that follows the definition of objects. Reconstruction of individual



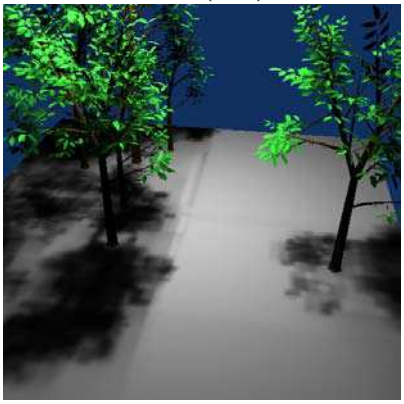
Reference image.



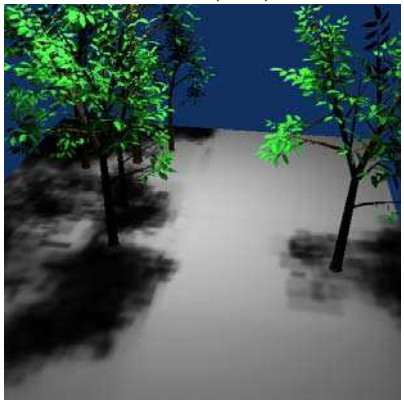
PROXI (629 s)



OKDT-P (402 s)



OKDT (281 s)



TF-OCT (166 s)

Figure 5: *Influence of the clustering method on approximate visibility calculations (see also color plate).*

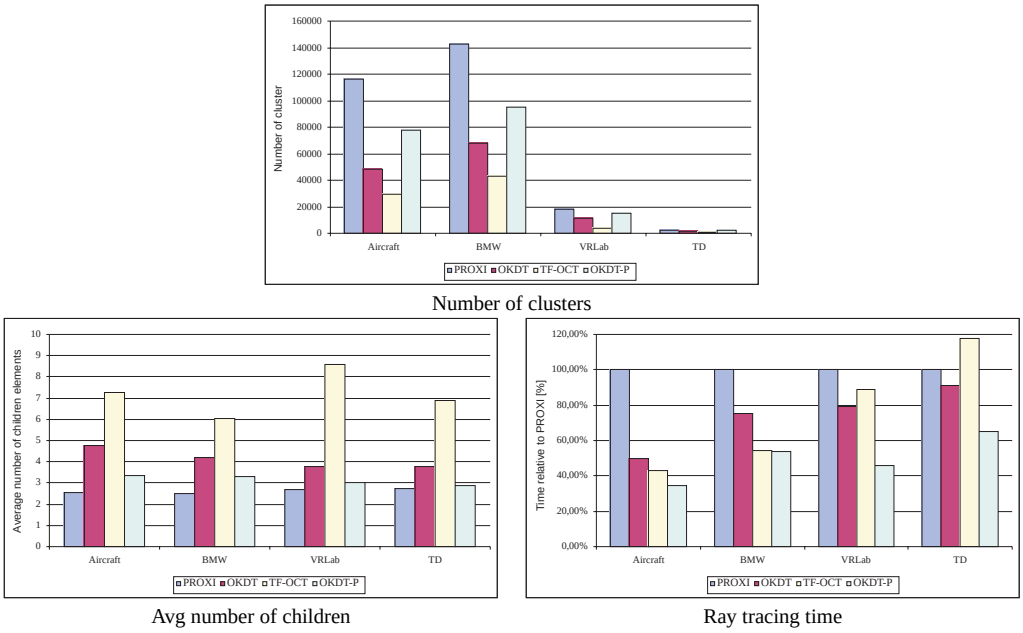


Figure 6: *Statistics on the cluster hierarchies*

objects is possible based on connectivity and surface properties, and multi-resolution object models could be developed to provide a hierarchy of representations.

Of the clustering algorithms tested, the hierarchical bounding volumes (PROXI) approach seems to have the most predictable behavior in almost all cases. In particular, the Time-error curve is almost always monotonic and smooth. In addition, due to the nature of construction, it fits objects more tightly, which is a desirable property for clustering. However the overhead for PROXI is significant if not prohibitive in most cases: a much longer construction time (compared to all others tested), longer solution times, and in some cases a higher absolute error for very approximate simulations.

In terms of ray-casting cost, it appears that OKDT-P is the most rapid structure. Thus, if ray-casting cost is an issue (for example in interactive updates where efficiency is paramount), this may be the clustering algorithm of choice.

Finally, in terms of the quality of approximate visibility, a clear hierarchy was found with the following order PROXI (best), OKDT-P, OKDT, TF-OCT (poorest).

We hope these first conclusions will be useful to researchers and developers who wish to use clustering for hierarchical radiosity. Clearly, much remains to be done in this domain.

The error metric adopted for our tests is one of many possibilities. It is evident that different applications have different notions of error (for example in lighting design where an exact measure of energy prevails over image quality), and these different requirements will lead to different choices for clustering. These issues must be further investigated.

The initial, first-order, classification of scene “type” with respect to their behavior in the context of a clustering algorithm is an interesting avenue of research. Ideally, extensive experimentation would allow us to determine which algorithm is suitable for a given scene. This is however a very ambitious task, so even initial results would be worthy of further research.

The development of a novel clustering approach treating scenes containing many small unrelated polygons is also an interesting challenge.

To conclude, we believe that our analysis has shown the utility of clustering for many cases, identified some weaknesses of current algorithms and identified certain important properties of each algorithm with respect to their suitability for different tasks.

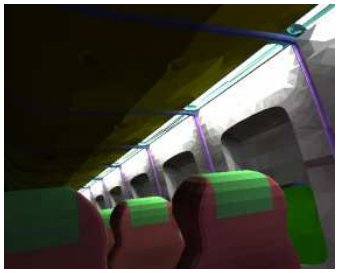
7. Acknowledgments

This work was supported by the European Union under the Esprit Long Term Research contract # 24944 “*ARCADE:*

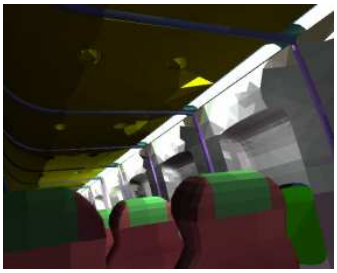
making radiosity usable" [1]. Scenes were kindly provided by Lightwork Design Ltd, Fraunhofer Institute for Computer Graphics and BMW. iMAGIS is a joint research project of CNRS/INRIA/UJF/INPG.

References

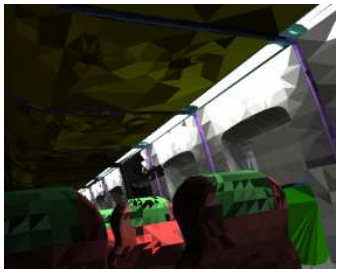
1. ARCADE: making radiosity usable. European Union. Esprit Long Term Research project #24944. `http://www-imagis.imag.fr/ARCADE`.
2. Frédéric Cazals, George Drettakis, and Claude Puech. Filtering, clustering and hierarchy construction: a new solution for ray tracing very complex environments. In F. Post and M. Göbel, editors, *Computer Graphics Forum (Proc. of Eurographics '95)*, volume 15, Maastricht, the Netherlands, September 1995.
3. Frédéric Cazals and Claude Puech. Bucket-like space partitioning data structures with applications to ray-tracing. In *Proceedings of the 13th International Annual Symposium on Computational Geometry (SCG-97)*, pages 11–20, New York, June4–6 1997. ACM Press.
4. Per Henrik Christensen, Dani Lischinski, Eric J. Stollnitz, and David H. Salesin. Clustering for Glossy Global Illumination. *ACM Transactions on Graphics*, 16(1):3–33, January 1997.
5. George Drettakis and François Sillion. Interactive update of global illumination using a line-space hierarchy. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '97 (Los Angeles, CA), pages 57–64. ACM SIGGRAPH, New York, August 1997.
6. Mark D. Fairchild. *Color Appearance Models*, chapter 3, pages 90–93. Addison Wesley, 1998.
7. Simon Gibson and Roger J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Computer Graphics Forum*, 15(5):297–310, December 1996.
8. Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, May 1987.
9. Pat Hanrahan, David Saltzman, and Larry Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, August 1991. Proceedings SIGGRAPH '91 in Las Vegas (USA).
10. Krzysztof S. Klimaszewski and Thomas W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, January 1997.
11. François Sillion. Clustering and volume scattering for hierarchical radiosity calculations. In G. Sakas, P. Shirley, and S. Müller, editors, *Photorealistic Rendering Techniques*. Springer Verlag, 1995. Proceedings of Fifth Eurographics Workshop on Rendering (Darmstadt, Germany, June 1994).
12. François Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3), September 1995. (a preliminary version appeared in the fifth Eurographics workshop on rendering, Darmstadt, Germany, June 1994).
13. François Sillion and George Drettakis. Feature-based control of visibility error: A multiresolution clustering algorithm for global illumination. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '95 (Los Angeles, CA), pages 145–152. ACM SIGGRAPH, New York, August 1995.
14. Brian Smits, James Arvo, and Donald P. Greenberg. A clustering algorithm for radiosity in complex environments. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '94 (Orlando, FL), pages 435–442. ACM SIGGRAPH, New York, July 1994.
15. Andrew J. Willmott and Paul S. Heckbert. An empirical comparison of progressive and wavelet radiosity. In Julie Dorsey and Philipp Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 175–186, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.



Reference

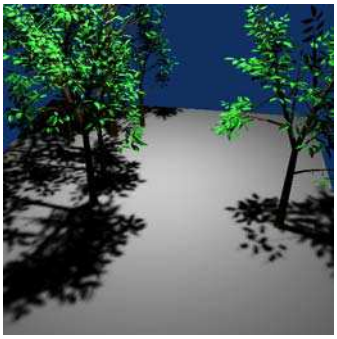


TF-OCT

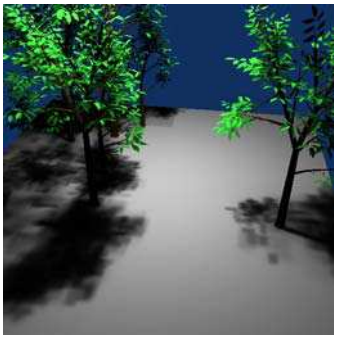


PROXI

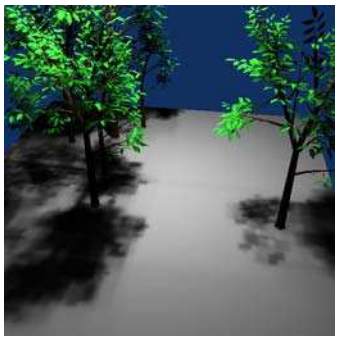
Figure 4: Example solutions exhibiting different (visual) forms of error.



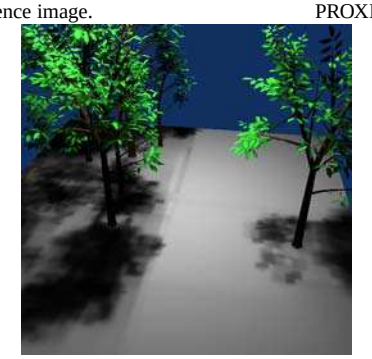
Reference image.



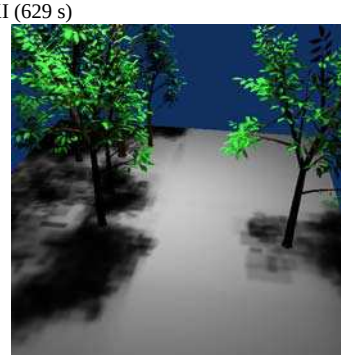
PROXI (629 s)



OKDT-P (402 s)



OKDT (281 s)



TF-OCT (166 s)

Figure 5: Influence of the clustering method on approximate visibility calculations.

2 Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer

[SH00] François Sillion, Jean-Marc Hasenfratz, "Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer", Third Eurographics Workshop on Parallel Graphics and Visualisation, Girona - Sep 2000, pp 61-74

Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer

François X. Sillion, Jean-Marc Hasenfratz

iMAGIS *- GRAVIR/IMAG
INRIA Rhône-Alpes
ZIRST, 655, avenue de l'Europe
38330 Montbonnot Saint Martin
France

Abstract

We introduce a simple, yet efficient extension to the hierarchical radiosity algorithm for the simulation of global illumination, taking advantage of a distributed shared memory (DSM) parallel architecture. Our task definition is based on a very fine grain decomposition of the refinement process at the level of individual pairs of hierarchical elements, therefore allowing a very simple implementation from an existing code with minimal modifications. We describe a generic refinement scheme based on a *scheduler*, allowing both easy parallelization and reordering of refinement tasks, which is useful for interactive and user-driven applications. We show that a very simple task grouping mechanism suffices to avoid excessive time waste in synchronization. Results obtained on an SGI Origin computer with 64 processors validate the approach, with excellent speedups using the full capacity of the machine.

1 Introduction

The radiosity method is a numerical simulation technique capable of determining the distribution of global illumination in a three-dimensional scene composed of diffuse reflectors [SP94]. Recent advances, in the form of hierarchical (wavelet) formulations and the introduction of clustering techniques, have made it possible to compute radiosity solutions in large scenes. Still these hierarchical algorithms are relatively slow on complex industrial scenes: usable solutions can be computed in tens of minutes, but very high-quality solutions typically require hours of calculation [HDSD99].

It is therefore quite natural to use parallel computers to reduce computation times – a state of the art is proposed by [RCJ98]. Two major approaches have been studied in previous work, corresponding to different parallel architectures and to different granularities: clusters of independent processors, or massively parallel machines with distributed shared memory. We review below the most significant approaches, and note that the high price of large parallel computers can only be accepted if very good speed-ups are obtained.

*iMAGIS is a joint research project of CNRS, INRIA, INPG and UJF.

Different authors describe parallel implementations of hierarchical radiosity on clusters of workstations. Funkhouser describes an algorithm where multiple hierarchical radiosity solvers work in parallel [Fun96]. A master process distributes sets of polygons over the set of workstations to determine an approximate and partial radiosity solution. The master then collects and merges solutions. This mechanism is iterated until convergence. This approach is well adapted to very complex scenes which could not be duplicated on all processors, provided visibility heavily restricts the potential interactions between subsets. This is particularly true of architectural scenes, with appropriate visibility preprocessing. The validation is done by tests over eight SGI workstations on the Soda Hall model with a speed-up of 5.5. Feng *et al.* also exploited the spatial coherence of the scene with 3D cells visible sets [FY97]. Tests on 8 DEC 3000 workstations with the PVM (Parallel Virtual Machine) programming environment show that the speed-up degrades with the number of processors due mostly to the Ethernet connection.

There is not as much work on hierarchical radiosity based on distributed shared memory. Bohn *et al.* proposed a parallel hierarchical radiosity approach on a *Connection Machine 5*, with a relatively modest speedup of 8.4 on 64 processors [BG95].

Renambot *et al.* proposed a parallel hierarchical radiosity based on a geometrical splitting of the scene. The radiosity is computed within each sub-scene. Exchange of energy between sub-scenes is performed by means of virtual interfaces and visibility masks. The size of sub-scenes can be adapted in order to fit into cache or local memory [RAPP97]. This algorithm was tested on a SGI Origin2000 with 32 processors. An accurate and interesting analysis of the hardware counters of the R10000 processor is presented. This seems to be a good approach, but requires a good knowledge of the different parameters to set and a very specific implementation. Cavin *et al.* proposed a parallel shooting wavelet radiosity algorithm on a large number of processors [CAP98]. A precise study of load balancing is proposed and a well adapted version for the Origin 2000 is presented in [Cav99]. Results show a good behavior for up to 32 processors (speedup of 24), with an unexplained but severe degradation afterwards. Singh *et al.* [SGL94] place pairs of patches which could interact into queues. Every processor has its own queue. When the treatement of a pair produces sub-patches, there are enqueued on the same processor. When a queue is empty, the associated processor steals tasks from other processors. Speed-up seems to be very good but tests are made only on one scene with a small number of polygons (174). A precise memory cache study is proposed, but the used of a (very) small scene makes conclusions difficult to generalize.

With the development of off-site computation facilities and companies, more and more opportunities exist to perform heavy simulations at specialized sites. Therefore the issue of acquisition cost for large parallel computers such as the SGI Origin is largely solved, and the development of an efficient parallel algorithm for hierarchical radiosity becomes of interest to a large user community. We placed our work in the context of such a machine, with tests on a 64-processor Origin2000, and aim to provide a simple and efficient algorithm, allowing the easy adaptation of existing codes.

The paper is organized as follows. The next section briefly recalls important mechanisms of hierarchical radiosity. Section 3 describes the extension of hierarchical radiosity into a parallel refinement algorithm. Results and discussion are in section 4 and finally, section 5 concludes and presents future work.

2 Hierarchical Radiosity

To understand our parallel implementation of hierarchical radiosity, we briefly describe the sequential version of the algorithm. The goal of the method is to compute a suitable approximation of the lighting distribution in the scene, by projecting it onto a set of basis functions. Basis functions, and the surface elements that form their support, are arranged in a hierarchy.

The core of the algorithm consists of refining the set of interactions between surface elements (although formally the interactions are between basis functions, we will refer to surface elements –equivalent to constant basis functions–for a more intuitive discussion). Refinement proceeds by pushing interactions down the hierarchy, until the radiosity exchange is "sufficiently well" approximated. The quality of an implementation largely rests on the choice of refinement criteria, but this issue is orthogonal to the parallelization method described here.

Interactions can be represented explicitly using "links", where a link embodies the impact of a surface element on another. The set of links attached to an element therefore represents the various sources of illumination for this element. The refinement process operates on a given set of links (initially and when we use clustering, a single link represents all the energy transfered in the scene [Sil95]), evaluates their quality using estimates of visibility, geometrical and energetic considerations, and decides for each link to either keep it or replace it with links at lower hierarchical levels.

Light energy can be transfered across the resulting set of links, iteratively until a global solution is obtained. Note however that since contributions are "gathered" at all levels of the hierarchy, a consolidation pass ("push-pull") can be needed at each iteration to ensure each element has a consistent view of all energy received higher and lower in the hierarchy. A typical execution of the program consists of several refinement iterations, each of which executing a refinement stage, followed by an energy propagation stage. Such iterations are useful because the refinement criterion uses the current estimate of radiosity in its decisions, and this estimate improves with each iteration. Note however that very few refinement iterations usually suffice to reach a high-quality solution (our results were computed for 4 iterations): they should not be confused with shooting iterations (e.g. in [Cav99]) which only compute the contribution of a single object, therefore requiring thousands of iterations for convergence.

Links therefore collectively represent the interactions in the scene, and are typically very numerous, placing heavy load on memory resources. As an alternative, shooting methods do not store links but instead propagate energy immediately upon calculation of transfer coefficients (form factors) [SSSS98]. The disadvantage is that links must then be recomputed in subsequent iterations, therefore it can be beneficial to keep some of them [GD99], using heuristic criteria.

Since the energy gathering and pushpull stages are easily parallelized (they basically consist of a simple array traversal), we focus in this paper on the hierarchical refinement stage.

3 Extension of Hierarchical Radiosity into a Parallel Refinement Algorithm

3.1 General considerations

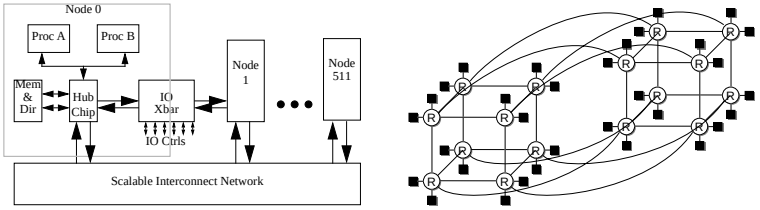
From the survey of the various approaches to the parallelization of radiosity algorithms, we observe the classical distinction between algorithms that explicitly manage data exchange (message-passing algorithms), and shared memory algorithms operating on a single data space. In practice, the latter class actually resorts to the operating system to implement virtual, or distributed, shared memory across the processors.

Algorithms assuming shared memory are obviously easier to implement, since data access is only an issue in terms of concurrency, therefore only requiring some synchronization or locking mechanisms. This is in contrast to distributed algorithms where low-level data management must be integrated in the algorithm. In the context of radiosity calculations, the global nature of light propagation (where each object can potentially illuminate many other objects in different areas of the scene) makes it very difficult to organize and monitor data locality, unless a very strong spatial structure is present as in some architectural scenes [Fun96]. Distributed Shared Memory (DSM) systems therefore appear particularly suited to radiosity calculations.

Even for the simpler shared memory case, however, a principal difficulty remains the segmentation of the work into tasks that can be efficiently distributed to the processors. A major factor influencing the overall efficiency is the granularity of the chosen tasks. Ideally, we want to divide the work into tasks of uniform complexity, to avoid situations where one processor is still working on its task while all others have finished. In the case of hierarchical radiosity calculations, the complexity of the calculations is not known in advance, since we precisely aim to adjust the effort spent on each radiant interaction, keeping it to the minimum needed to achieve the desired accuracy. The computation is therefore very dynamic, generating more calculations in areas of high importance.

Cavin *et al.* described a successful parallel implementation of wavelet radiosity on a DSM computer [CAP98, Cav99], and actually report that some of their "iterations" take hours, while others take only a few seconds. Furthermore, because of the large granularity of their algorithm they had to resort to complex operations on their data structures in the form of temporary copies and "lures", to avoid wasting time in synchronization locks. Therefore, using a fine granularity with very lightweight tasks seems a more promising response to the load balancing issue, assuming tasks can be quickly assigned to idle processors. A finer task segmentation would be difficult in a shooting wavelet radiosity approach where input surfaces are used as top-level shooting objects. By contrast, our use of a *complete* hierarchy in the scene, made possible by the clustering algorithm, allows us to consider the entire radiosity solution phase as a sequence of atomic refinement operations of very small, and nearly uniform, complexity. In essence, energy transfers (modeled either as link refinements or shooting operations) can be performed at any level of the hierarchy.

In summary, we chose a very fine granularity at the level of individual interactions involving an emitter-receiver pair: a task consists of treating such an interaction, which means either establishing a link (transferring energy, in a shooting approach) or deciding to reconsider the interaction at a finer level. Note however that in the latter case, several new tasks are created,



a. Block diagram. b. Topology of the DSM Origin 2000.

Figure 1: Scalable DSM Origin 2000 (*figures extracted from [LL97]*)

and the corresponding effort is therefore not part of the original task.

3.2 The ccNUMA architecture

To validate our approach, we conducted tests on an SGI Origin 2000 computer. The Origin is a scalable multiprocessor computer (up to 1024 processors) with Distributed Shared Memory (DSM), based on the SN0 (Scalable Node 0) architecture [LL97]. Figure 1 a. describes this architecture. The basic building block is the node board, composed of two MIPS R10000 processors, each with separate 32 KB first level (L1) instruction and data caches on the chip with 32-byte cache line, and a unified (instruction and data), commonly 4 MB, two-way set associative second level (L2) off-chip cache with 128-byte cache line. Each node contains 64MB to 4Go of main memory, accessible through a custom circuit called the hub. Large SN0 systems are built by connecting the nodes together via a scalable interconnection network. Connecting two nodes is done by connecting their hub chips through a router, which can be itself connected up to six hubs or other routers. Figure 1 b. shows the topology of the 64 processor Origin 2000 used for this paper.

People with extensive experience in using this ccNUMA architecture report that it has a very good properties of data locality [CAP98, Cav99] and is well adapted for a hierarchical radiosity algorithm.

3.3 Work flow of the refinement stage

The hierarchical radiosity algorithm lends itself well to a recursive implementation, due to the hierarchical structure of mesh elements and wavelet basis functions. However as discussed above such recursive implementations are prone to severe imbalance, as some interactions will require much more effort than others.

We introduce a generic mechanism for radiosity calculations, which we call a *scheduler*. The scheduler is basically a repository of potential links, or transfer interactions, with appropriate methods to request one or several of these links, and to submit new links. The set of links in the scheduler at any time therefore represent the set of pending tasks. A minimal modification of a hierarchical radiosity code is sufficient to use the scheduler: we basically reorganize the

recursive traversal of the set of interactions, with the recursive creation of lower-level interactions, into a loop that continually requests a task from the scheduler, makes a decision about its subdivision and potentially submits resulting new tasks to the scheduler.

It is easily seen that a scheduler based on a stack data structure simply mimics the recursive behavior of the original algorithm. However, and even for purely sequential implementations, there are several advantages to using the scheduler. First, the scheduler offers a global view of all pending tasks at any given time. This allows more global decisions to be made during the course of the solution. Second, task extraction can be made according to various selection criteria, with maximal flexibility. For instance, the scheduler can implement a priority list, with criteria such as error estimates ("refine links with most error first"), importance or user-defined priorities. This is especially useful for interactive design sessions where the user might shift focus from one area of the scene to another, or modify the scene [DS97]. In such cases the ordering of tasks can be dynamically modified.

In the DSM model, a unique scheduler exists in memory, and we shall see later how to synchronize accesses to its data from the multiple processes.

Architecture of the radiosity program

The architecture of our radiosity program is depicted in Figure 2. We employ the simple *sproc* system call to create threads sharing their entire address space. The main process deals with the graphical user interface (for interactive sessions) and high-level control of the computation (running multiple iterations, monitoring convergence). Each iteration is handled by the solver process, which itself controls a number of identical refinement processes (refiners). The solver process is responsible for the initial loading of the scheduler, with all existing links at the start of the iteration. All refiners continuously obtain tasks from the scheduler, perform the corresponding refinement decisions, and add new tasks to the scheduler as needed. Note that there is no scheduling process, but rather a data structure whose access is controlled by a synchronization mechanism as we will see below.

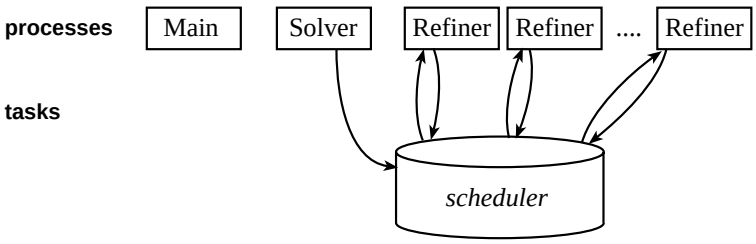


Figure 2: *Organization of our radiosity code. We use $n+2$ threads for n simultaneous refiners. The solver feeds the scheduler structure with an initial set of tasks (links), and each refiner requests tasks from the scheduler and returns new tasks as appropriate.*

3.4 Synchronization

A mentioned above, our architecture accommodates a number of refinement processes operating on a shared address space. Appropriate synchronization and locking mechanisms are needed to ensure complete data consistency. We found it sufficient to implement the following three locking mechanisms, isolating the three corresponding code fragments.

1. scheduler: since all refiners access the same pool of tasks through the scheduler interface, both the extraction and the incorporation of tasks should be protected against concurrent access. A semaphore with a single access right is used to control the operation of the scheduler.

2. hierarchical data structure. The hierarchy of elements representing the objects in the scene must be guarded against concurrent modifications of any given element. However, simultaneous modifications of different elements are allowed. Therefore we can use a pool of semaphores indexed by a hashing function to avoid congestion. The critical code section is the subdivision of a surface element.

3. Interactions. When using links to represent transfer interactions, each element possesses a list of links, which must be guarded against concurrent modification (links can either be added as a result of refinement, or deleted when they are selected for refinement and "pushed down" in the hierarchy). For the shooting version of the algorithm, which does not use links, the equivalent operation is the update of radiosity values. Similar to the previous case, we can use a pool of semaphores to avoid congestion.

3.5 Performance

Our scheduling mechanism, based on the acquisition of tasks from the scheduler, leaves only a small number of sources for performance degradation. These are mainly (a) the time spent waiting for semaphore acquisition in the various locks described above, and (b) performance issues at the system level, in particular concerning memory access across the different caches and nodes of the Origin computer. We do not consider the second problem at this time, and focus on synchronization performance.

Clearly the code implementing task extraction and incorporation in the scheduler is critical, as it can be performed for a single refiner at once. Therefore it is important to minimize the corresponding effort. Since the order in which links are refined is not critical in general, we optimized our code to operate on blocks of tasks, therefore in essence grouping a set of links to be refined. It should be noted that this conserves the desired property that all tasks be of similar complexity, and that the grouping is arbitrary and does not necessarily correspond to high-order hierarchy elements. As expected, the overall performance is improved by using blocks of links, since the time needed to operate the scheduler is not negligible with respect to the time for refining a single link. Quite naturally, the size of the block is not important as long as it provides "enough work" to overshadow the scheduler operation: Figure 7 shows that blocks of 10 links already provide a significant improvement, whereas increasing to 100 links make no difference.

Using blocks of links takes care of the acquisition of tasks by each refiner: for the submission of new tasks (as a result of refinement), we use a buffering mechanism. New tasks are accumulated in a table that is unique to each process, and only after all links in the block have been refined is this table integrated in the scheduler (with appropriate synchronization).

For the other critical code portions, namely the modification of the hierarchy through surface element splits, and the representation of energy exchanges (links or radiosity updates), we found that using a pool of semaphores indexed by a hashing function works very well and basically removes all contention. We used a table of about twice the number of refiners, and an indexing function based on the memory address of the concerned elements.

4 Results and Discussion

4.1 Test Scenes Chosen

We performed our tests on three different scenes, shown in Figure 3. Two of them are rather large industrial-type models while the third one is more artificial.

- **AIRCRAFT** (184,456 original polygons)
  Model of an aircraft cabin (courtesy of LightWork Design Ltd). All objects have been tessellated into (rather small) triangles to account for the rounded shapes.

- **VRLAB** (30,449 original polygons)
  A virtual reality lab with two floors and mostly overhead lighting (courtesy of Fraunhofer Institut für Graphische Datenverarbeitung). This scene has a mixture of large polygons, likely to be subdivided, and very small patches (on chairs and desktop computers).

- **OFFICE** (5,260 original polygons)
  A model of a simple office scene. This model is much smaller than the other three and is provided to show that clustering performs well on this kind of small scene usually found in the literature.

4.2 Measurements

We tested our algorithm on a ccNUMA Silicon Graphics Origin 2000 computer [LL97] with 64 processors. In fact, we limit our tests to 40 processors to avoid locking the Origin for our own usage! Each processor is a 195 MHz R10000 with L1 data cache of 32 Kbytes, 4 Mbytes of secondary unified data cache L2 and 384 MBytes of local memory.

Refinement time was measured with the *times* system call, which returns clock ticks. Memory access time, cache usage etc. were measured through the 31 hardware counters of the R10000 using the *perfex* software tool.

Note that we have replaced the standard SGI memory allocation functions by the corresponding functions in the *GNU library* (*GNU C Library, Wolfram Gloger and Doug Lea*) to avoid thread-safe memory manipulation locks.

VRLab.

Office.

Aircraft.

Figure 3: The three test scenes used.

For all scenes we chose execution parameters to require about 5 hours of total CPU time, which corresponded to different levels of refinement for different scenes (with for instance very many links in the Office scene which is comparatively much simpler).

4.3 CPU time and speed-up

Figure 4 shows the total time (summed over all refiners) used for each refinement iteration, for different numbers of processors[1]. We clearly see that the time is constant for the VRLab and quite constant for the Office. The Aircraft has a "strange" behavior: time increases with the number of processors, at different rates for different iterations (and with a perfect constant time for the first iteration).

In Figure 5a. we show the speed-up for each scene. Logically, VRLab has a quite perfect speed-up of 39.4 on 40 processors (in fact, we have obtained a speed-up of 49.2 on 50 processors) and the Office has a speed-up of 25.7 on 30 processors. The aircraft speedup is not as good, reaching 24 for 40 processors. We can see in Figure 5b. that the speed-up of Aircraft decreases with the number of processors for the last three iterations: respectively, we obtain speed-ups of 35.1, 23.4 and 17.3 on 40 processors for iterations 2, 3 and 4.

To explain this behavior, we have investigated memory usage, thinking the problem may be due to cache performance or data locality issues. However as shown in Figure 6 this is not a

[1] A technical problem prevented us to obtain results for the Office scene with 40 processors. However it is clearly not related to our algorithm.

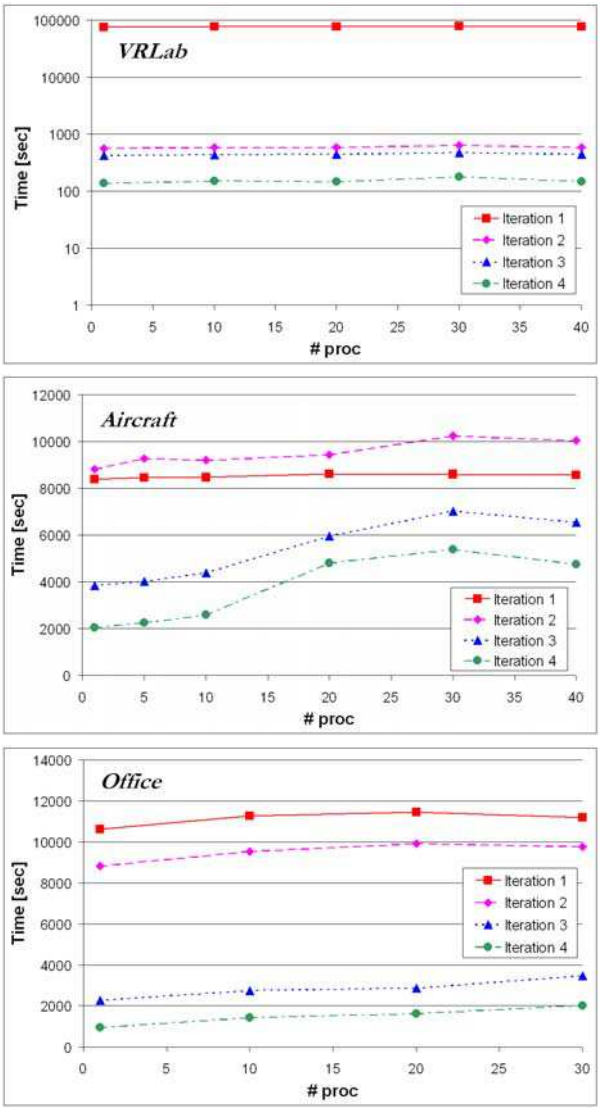
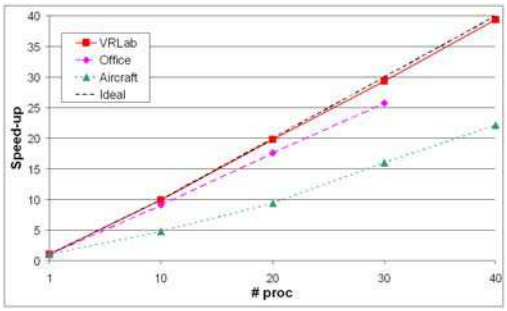
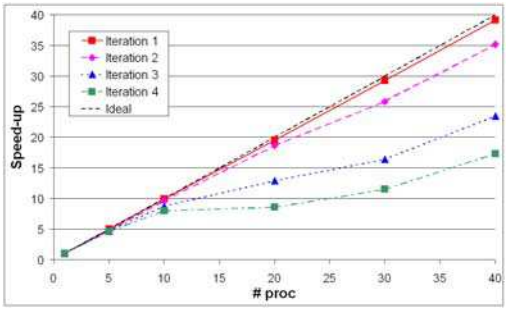


Figure 4: CPU Refinement time for the three test scenes.



a. Speed-up for all test scenes (four iterations).



b. Aircraft speed-up for each iterations.

Figure 5: Speed-up.

good explanation: we have plotted both the number of links created (and stored, in our implementation), as well as the total memory usage, for each scene and at the end of each iteration (cumulative). We clearly see that memory consumption is directly correlated to the number of links (not surprisingly, following analyses of [WH97, SSSS98]). However the aircraft scene has the smallest number of links and memory size, and fits easily in the local memory of a single processor board. Conversely, Office uses the most memory but exhibits a perfect speedup even for subsequent iterations. At this point we are looking for an explanation to this behavior. We also looked at the percentage of total time spent accessing memory, as provided by *perfex*, but for the aircraft scene this percentage *decreases* (from about 65% to about 45 %) as we increase the number of processors... In fact, we have analyzed all 31 hardware counters without discovering a satisfactory explaination of this behaviour.

We note however that if a memory locality issue can be identified, the *scheduler* could easily incorporate other criteria to better balance the data over the processors by ordering links differently. This does not appear to be necessary given the good results already obtained, unless maximal performance is absolutely necessary.

Finally we present in Figure 7b results for different block sizes in the tasks distributed by

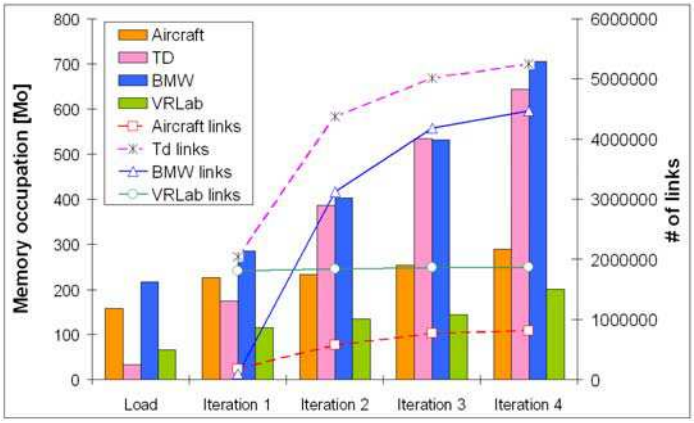


Figure 6: *Memory used before and during the iterations.*

the scheduler. As we increase this size from 1 to 10, we observe a natural improvement, but increasing it to 100 offers no more improvement.

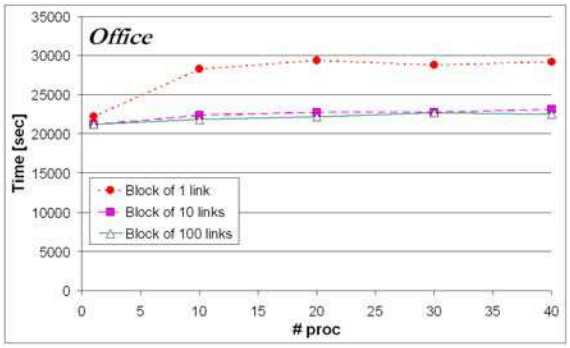


Figure 7: *Influence of the size of link blocks on overall CPU time.*

5 Conclusions and future work

We have presented a parallel algorithm for hierarchical radiosity calculations on a computer with distributed shared memory. The work is divided into very simple atomic tasks, consisting of the refinement decision on a single interaction (or a small group of such interactions). The distribution is easily managed with a single *scheduler* structure, with appropriate synchronization. Incidentally the use of a scheduler is beneficial in many radiosity applications including interactive steering by the user. The resulting organization encapsulates the parallel behavior

of the algorithm into the scheduler and thread management portions of the code, and allows full flexibility in the rest of the simulation, including choice of wavelet bases, shooting or link-based algorithms, visibility determination techniques and refinement oracles. Therefore our proposed algorithm can be very easily implemented on top of an existing radiosity simulation code.

Our results indicate good to excellent speedups depending on the scenes, for up to 40 processors. This is particularly encouraging considering that no effort was made to localize tasks and thereby improve memory access time. Future work includes the understanding of the peculiar behavior observed for the aircraft scene, in which speedups are consistently decreasing for successive iterations of the refinement algorithm. While our analyses so far have not yet identified a data locality problem, we note that should such an issue be identified, the scheduler mechanism can incorporate any ordering based for instance on data locality. Therefore if we can predict a grouping of link refinement tasks that improves performance it can readily be implemented in the scheduler.

6 Acknowledgments

Peter Kipfer contributed to the design and early implementation of this work. We would like to thank the Centre Charles Hermite for providing us access to its computational resources, and Laurent Alonso for his advice on performance questions. This work was supported in part by the European Union's ESPRIT project #24944, ARCADE ("Making Radiosity Usable").

References

[BG95] Christian-A. Bohn and Robert Garmann. A Parallel Approach to Hierarchical Radiosity. In V. Skala, editor, *Proceedings of the Winter School of Computer Graphics and CAD Systems '95*, pages 26–35, Plzen, Czech Republic, February 1995. University of West Bohemia.

[CAP98] Xavier Cavin, Laurent Alonso, and Jean-Claude Paul. Parallel wavelet radiosity. In *Proceedings of the Second Eurographics Workshop on Parallel Graphics and Visualisation*, pages 61–75, Rennes, France, September 1998. Eurographics.

[Cav99] Xavier Cavin. Load balancing analysis of a parallel hierarchical algorithm on the origin2000. In *Fifth European SGI/Cray MPP Workshop*, Bologna, Italy, September 1999.

[DS97] George Drettakis and François Sillion. Interactive update of global illumination using a line-space hierarchy. In *Computer Graphics Proceedings, Annual Conference Series:* SIGGRAPH '97 (Los Angeles, CA), pages 57–64. ACM SIGGRAPH, New York, August 1997.

[Fun96] Thomas A. Funkhouser. Coarse-Grained Parallelism for Hierarchical Radiosity Using Group Iterative Methods. In *Computer Graphics Proceedings, Annual Conference Series, 1996 (ACM SIGGRAPH '96 Proceedings)*, pages 343–352, 1996.

[FY97] Chen-Chin Feng and Shi-Nine Yang. A parallel hierarchical radiosity algorithm for complex scenes. In *Proceedings of the Third Parallel Rendering Symposium (PRS '97)*, pages 71–78, Phoenix, AZ, October 1997. IEEE Computer Society.

[GD99] Xavier Granier and George Drettakis. Controlling memory consumption of hierarchical radiosity with clustering. In *Graphics Interface*, pages 58–65, June 1999.

[HDSD99] Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, and George Drettakis. A practical analysis of clustering strategies for hierarchical radiosity. In *Computer Graphics Forum (Proc. Eurographics '99)*, volume 18(3), September 1999.

[LL97] James Laudon and Daniel Lenoski. The sgi origin: a ccnuma highly scalable server. In *Proceedings of the 24th international symposium on Computer architecture*, pages 241–251, Denver, CO USA, June 1997. ACM.

[RAPP97] Luc Renambot, Bruno Arnaldi, Thierry Priol, and Xavier Pueyo. Towards efficient parallel radiosity for dsm-based parallel computers using virtual interfaces. In *Proceedings of the Third Parallel Rendering Symposium (PRS '97)*, pages 79–86, Phoenix, AZ, October 1997. IEEE Computer Society.

[RCJ98] E. Reinhard, A. Chalmers, and F. Jansen. Overview of parallel photo-realistic graphics. In *Computer Graphics Forum (Proc. Eurographics '98)*, pages 1–25, 1998.

[SGL94] Jaswinder P. Singh, Annop Gupta, and Marc Levoy. Parallel Visualization Algorithms: Performance and Architectural Implications. *IEEE Computer*, 27(7):45–55, July 1994.

[Sil95] François X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Computer Graphics*, 1(3), September 1995. (a preliminary version appeared in the fifth Eurographics workshop on rendering, Darmstadt, Germany, June 1994).

[SP94] François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann publishers, San Francisco, 1994.

[SSSS98] M. Stamminger, H. Schirmacher, P. Slusallek, and H.-P. Seidel. Getting rid of links in hierarchical radiosity. *Computer Graphics Forum (Proc. Eurographics '98)*, 17(3):C165–C174, September 1998.

[WH97] Andrew Willmott and Paul Heckbert. An empirical comparison of progressive and wavelet radiosity. In Julie Dorsey and Phillip Slusallek, editors, *Rendering Techniques '97 (Proceedings of the Eighth Eurographics Workshop on Rendering)*, pages 175–186. Springer Wien, 1997. ISBN 3-211-83001-4.

3 Real-Time Capture, Reconstruction and Insertion into Virtual World of ...

[HLGB03] Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, Edmond Boyer, "Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors", Vision, Video and Graphics, pp. 49-56, 2003

Vision, Video, and Graphics (2003)
P. Hall, P. Willis (Editors)

Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors

JM. Hasenfratz[1] and M. Lapierre[1] and J.-D. Gascuel[1] and E. Boyer[2]

[1] Artis GRAVIR/IMAG, a joint project of CNRS, INPG, INRIA, UJF.
[2] MOVI GRAVIR/IMAG, a joint project of CNRS, INPG, INRIA, UJF.
http://www-artis.imag.fr/CYBER

Abstract

In this paper, we show how to capture an actor with no intrusive trackers and without any special environment like blue set, how to estimate its 3D-geometry and how to insert this geometry into a virtual world in ***real-time****. We use several cameras in conjunction with background subtraction to produce silhouettes of the actor as observed from the different camera viewpoints. These silhouettes allow the 3D-geometry of the actor to be estimated by a voxel based method. This geometry is rendered with a marching cube algorithm and inserted into a virtual world. Shadows of the actor corresponding to virtual lights are then added and interactions with objects of the virtual world are proposed. The main originality of this paper is to propose a complete pipeline that can computes up to 30 frames per second.*
Since the rapidity of the process depends mainly on its slowest step, we present here all these steps. For each of them, we present and discuss the solution that is used. Some of them are new solutions, as the 3D shape estimation which is achieved using graphics hardware. Results are presented and discussed.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation – Bitmap and framebuffer operations I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual reality I.4.1 [Image Processing and Computer Vision]: Scene Analysis – Tracking

1. Introduction

The CYBER project† inserts a live actor into a virtual word in real-time. To achieve this operation with realism, the animator must be integrated perfectly in its virtual environment. This perfect insertion requires more than a simple composite of the video flow over the rendered 3D-scene. First, we must add shadows of the actor as would be casted by the lights in the virtual scene. This prevents the impression that the actor flies above the floor. We must also relight the video flow to take account of the virtual environment lighting. Without that, combined pictures are not convincing and a virtual red light near the actor does not impact its visual aspect. Finally, just combining 3D environment and video data does not allow any interaction with the virtual scene.

This topic is very important for television industry using virtual sets and on-line presentations, for games where the player is rendered into the game, for education where the student visits a virtual place.

In this paper, our objective is to demonstrate that it is feasible to capture an actor with no intrusive trackers and without any special environment or material, estimate shape, and combine it with a virtual world up to 30 times per second.

The paper is organized as follows: previous work are reviewed in section 2. Section 3 describes the hardware and software architecture which are used. Section 4 is devoted to the capture stage; in particular camera calibration and background subtraction are discussed. Section 5 presents our original approach to real-time shape estimation which is based on graphics hardware. Section 6 proposes two examples of actor integration into a virtual world: high quality real-time shadows of the actor and real-time interaction with

† http://www-artis.imag.fr/CYBER

virtual objects in the scene. Finally we discuss the results before concluding.

2. Previous Work

Several approaches have been proposed to insert a live actor into a virtual scene at interactive rates. 3D information can be retrieved using stereo cameras which generate range images[4, 7]. This technique is able to reconstruct concave regions and the Virtualized Reality system[24] shows that it can work on large dynamic objects. Matusik[14] computes a visual hull from multiple camera views, using epipolar geometry, and generates a 3D textured model (each camera retrieving a colored silhouette).

Geometry information can also be retrieved by *Voxel Coloring*[17]: it consists in testing color consistency of voxels from the surface of the geometry among all the points of view.

Another approach is to use voxel carving: a volume is discretized into voxels, which can be used as a step to fit a skeleton[20], or to analyze human movements by fitting ellipsoids[1].

An interesting application by Lok[11] is to combine advanced interactions within the virtual world.

These methods have some drawbacks: concave regions which are difficult to capture or the number of views which are needed, but combining them as in Li[10] can reduce these problems and can improve the speed as well as the quality of the estimated geometry.

All these methods are working at interactive rates, but not at 30Hz on full VGA resolution images and with standard cameras. The solution that we propose does not have such limitation, and we will describe how to maintain this frame rate all along the process.

3. System architecture

3.1. Hardware architecture

Our hardware architecture is described in Figure 1. It is composed of four standard IEEE 1394 cameras (Sony DFW–VL500) running at a resolution of 640x480 pixels. The color mode which is used is YUV4:2:2. Each camera is connected to a Pentium4 PC running at 1.6 GHz. We use a 1Gbit/sec Ethernet network between the PCs and a SGI Onyx 3000–IR3. The Onyx is configured to use 8 R12000 processors. The output of the Onyx is a 100Hz display screen.

The actor can move inside a 2m large square. We do not use blue background (see Figure 2) but we use a controlled lighting environment with a grid of fluorescent lights. The cameras are not externally synchronized.

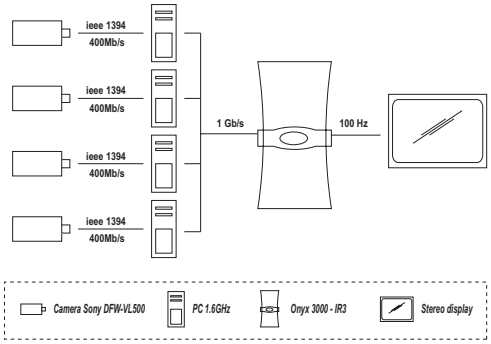


Figure 1: *Hardware configuration of the system.*

Figure 2: *The experimental setup. Cameras are surrounded by red circles in the image.*

3.2. Software architecture

PCs are running Windows 2000 with an IEEE 1394 library developed at Carnegie Mellon University[23] to drive the cameras. The OpenCV[15] library is used for camera calibration.

The different modules are shown in Figure 3. Cameras are set in continuous mode and produce 30 frames per second. On each PC, an infinite loop process subtracts the background (see Section 4.2) and filters the images. Results are sent to the Onyx via the network. Background subtraction and filtering takes less than 33ms per frame, and thus a data flow of 30Hz can be achieved. This rate is perfectly constant as the whole process if waiting for the camera capture, and this guarantees coherency between cameras (gap between two cameras is less than 1/30 second). Working with an external trigger would have lowered the frame rate to 15Hz.

Five processes run in parallel on the Onyx. Four of them are just buffering images coming from the *subtraction/filtering* modules (via the network). The fifth process is an infinite loop taking alternatively data from each buffer and therefore synchronizing the flows coming from the cameras. This last process estimates the 3D geometry of the actor

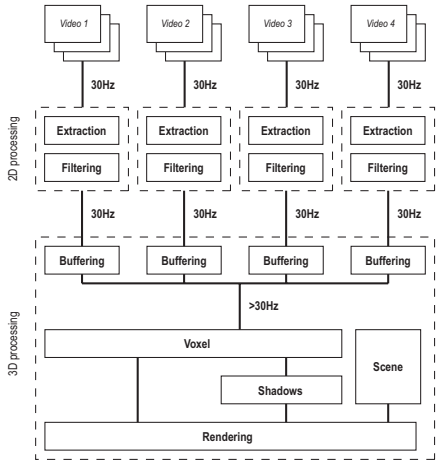


Figure 3: *The different modules involved (dashed rectangles correspond to processes.)*

(see Section 5) and performs the integration with the virtual world (see Section 6).

4. Capture

4.1. Camera calibration

Calibration is required to recover positions, orientations and internal parameters of the cameras involved in the reconstruction process. These information are needed to project voxels into image planes and verify which of them belong to the silhouette cones and therefore to the estimated shape. The calibration information are encoded in projection matrices, one per image, which are estimated in a preliminary step. Note that we therefore assume that all the camera parameters remain constant during the real time 3D environment and video data combining step. In order to estimate the projection matrices, we use a calibration pattern with known Euclidean characteristics: a chess-board. Since exact Euclidean characteristics of the scene are not needed for combining virtual and real images but only its geometry up to a scale, we do not use the exact Euclidean dimensions of the chess-board's squares but just the fact that they are of equal dimensions. Estimating projection matrices from coplanar points with known positions is then a well known problem, see [22, 26] for instance. We use the *Open Computer Vision Library*[15]. Such library offers solutions for both corner extraction and projection matrix estimation. Note that camera distortion could also be taken into account using the OpenCV library. However, such distortion does not appear to be significant in our case and we neglect it. Finally, we mention that we are currently implementing a new calibration system based on a more flexible pattern than the chess-board which is difficult to position efficiently in the scene.

4.2. Background subtraction

In our application, relevant and irrelevant information in the images correspond to foreground and background information respectively, where the background is assumed to be static and can therefore be learned *a priori*. Thus, extracting relevant information -the silhouettes- consists in detecting foreground, or equivalently background, pixels in the images. The solution generally adopted for that purpose is to verify pixels' values and detect changes. Indeed a background pixel value should be constant over the time while foreground pixel value should vary at some time. Following this principle, several approaches exist that check the intensity functions at each pixel. Robust ones use temporal filters to detect changes as well as spatial filters to cluster pixels and eliminate false detection (see [21] for a comparative study). However, in our context the critical issue is not robustness but how fast the operation is. Indeed, even if artifacts exist, they are usually not consistent over the image set and thus removed during the reconstruction step. Consequently, we use a simple but fast method. Background pixels are assumed to follow Gaussian distributions, possibly correlated, in the YUV space. Such distributions are learned *a priori* from a set of background images. The fact that a pixel belongs to the silhouette or the background is then checked by thresholding its Mahalanobis distance to the background mean position in the YUV space. Note that in order to take into account shadows during the subtraction, constraints on the intensity (the Y parameter) values could easily be relaxed. As a result, we obtain a flow of black and white pictures representing the silhouettes as viewed by the different cameras (see Figure 4). This operation takes in average 22ms per image.

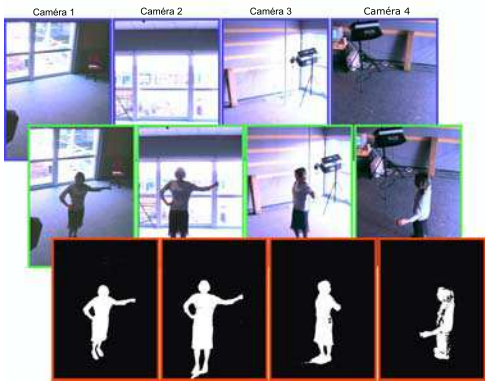


Figure 4: *Background subtraction: the first row shows background images, the second row some images taken during runtime, and the third row displays the corresponding extracted silhouette bitmaps.*

We have also implemented and tested binary morpholog-

ical filters to remove artifacts such as isolated pixels. However, the benefit is not really important while time consuming is significant. Finally, the silhouette extraction takes less than 23 ms per frame. The resulting bitmaps are sent to the buffering modules via the local network. The flow is cadenced by internal cameras clocks which deliver images at 30Hz. The output of the buffering modules is then used for shape estimation as explained in the next section.

5. Shape estimation

The next step is to estimate the scene geometry in order to compute illumination interactions. We are given, at each instant, several silhouettes corresponding to different camera viewpoints. The shape that can be estimated from these silhouettes is the visual hull [8] of the objects under consideration. The visual hull is in fact the maximal solid shape consistent with the object silhouettes. Such an approximation of the scene captures all the geometric information available from the silhouettes. Several approaches have been proposed to compute visual hulls. They can be roughly separated into two categories: volume based approaches and surface based approaches.

The first category includes methods that approximate visual hulls by collections of elementary cells called voxels and carve them according the silhouette information (See [18] and [5] for reviews). Approaches in this category can handle objects with complex topologies. Note that due to the space discretizations, they lead to approximations only of the visual hull, and that such approximations can be computationally expensive when precision is required.

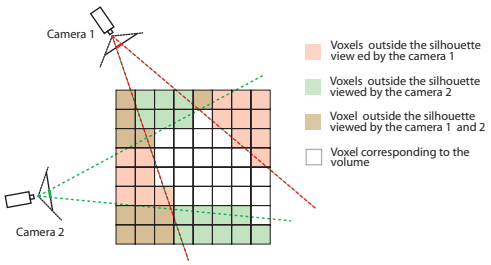


Figure 5: *Principle of voxel carving.*

The second category of approaches estimates elements of the visual hull's surface by intersecting the viewing cones associated the silhouettes [2, 13]. These category of approaches suffer from numerical instabilities [9]. Consequently, they often lead to surface models which are incomplete or corrupted, in particular when considering objects with complex topologies. Also, the visual hull itself is an approximation of the scene geometry and an exact description of an approximation of the scene is not necessarily required to compute illumination interaction. Thus, we have chosen for our application a voxel-carving approach (see Figure 5).

The accuracy of reconstructed geometry is depending on the way cameras are situated in the scene: in our case we wanted to be able to reconstruct a whole body, so the reconstructed space is $2m^3$ large. At a 64^3 resolution each voxel in this case is 3cm large. We could work on smaller objects with a better precision by focusing cameras on a reduced space, for example to reconstruct only the hands of the actor with a 0.5cm accuracy.

The following part describes how graphics hardware can be efficiently used for a voxel-carving implementation.

5.1. Hardware assisted voxelisation

Classically, a cube of N^3 voxels is considered as the discrete space to be carved. In this paper, we also use N^3 cells but instead of voxels, we consider N images of N^2 pixels. These images are located in the middle of each voxel slices. With this approach, we can benefit from graphic hardware at different stages of the process as shall be seen. In this section, we consider N=64 pictures of N by N pixels. A discussion of the parameter N is proposed in Section 7.

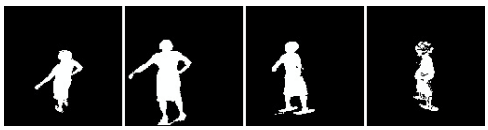


Figure 6: *Four textures corresponding to the silhouettes coming from the different buffer modules.*

We read one silhouette in each buffer and transform it into an OpenGL texture of 128x128 pixels as in Figure 6.

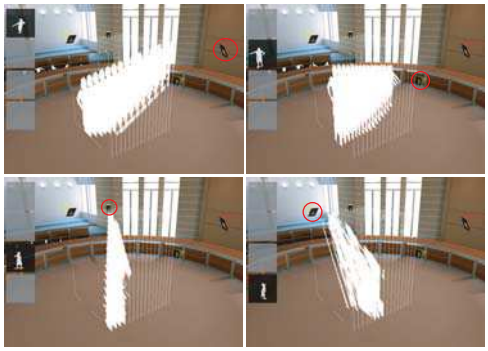


Figure 7: *Projections of the silhouettes onto the different slices for each camera. Red circles identify the involved cameras in each image.*

For each camera, we project the silhouette textures in voxel-space using graphics hardware facilities. This approach is similar to the one used by Lok[11] but with fewer

constraints. Figure 7 shows the depth projections on 64 planes corresponding to the grid of 64^3 voxels. Note that to ensure correct texture projections, the glHint parameter in OpenGL must be set to the highest level.

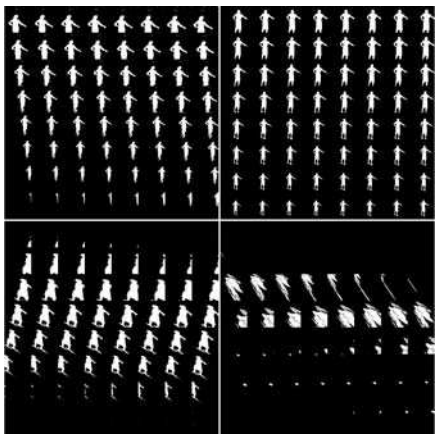


Figure 8: *Texture projection tiled in a single 512x512 texture map (8x8 textures of 64x64 pixels).*

All the projected textures corresponding to the same camera viewpoint are tiled in a single texture map of 512x512 pixels. This size corresponds to 64 projections of the silhouette textures composed by 64x64 pixels (see Figure 8).

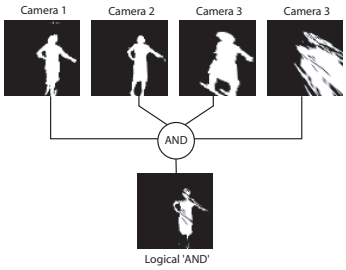


Figure 9: *Logical* AND *operation on four textures projected to the same slice.*

To determine the volume corresponding to the scene visual hull we must find the intersection between the projection of all bitmap images. Again, we take here advantage of the hardware: blending facilities of OpenGL can simulate the AND operation that is used for the intersection: by calling `glBlendFunc(GL_DST_COLOR, GL_ZERO)`, successive renderings of projected silhouettes will be combined with previous ones using the AND operator.

Figure 9 shows the result of the *AND* logical operation with the different projected texture on the same plane. Note that we always work with the same planes representing the slices of the original voxel grid.

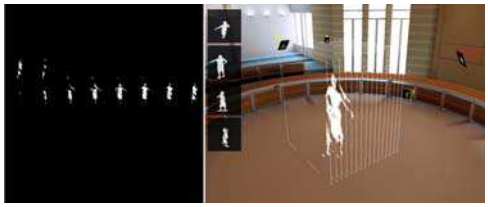


Figure 10: *Logical* AND *operation on four tiled textures. On the left, the tiled texture is shown, on the right all the textured quads located in the virtual world are displayed.*

Figure 10 presents the result of the *AND* logical operation on the entire texture and its equivalent operation in the voxel-space.

Figure 11: *Voxel representation of the* AND *operation on four tiled textures.*

Finally, to obtain the voxels representing the actor we just consider each white pixel in the final tiled texture map as a voxel (see Figure 11).

5.2. Pseudocode for the reconstruction

The different steps of the shape estimation algorithm are as follow:

```
Disable depth test
Disable blend
For each camera
  Transform silhouette to texture
  Set texture matrix to camera parameters
  For each slice
    Set working area to a tile
    Render a quad with texture projected
  Enable blend
Read whole frame buffer (glReadPixels)
Convert white pixels to voxels
```

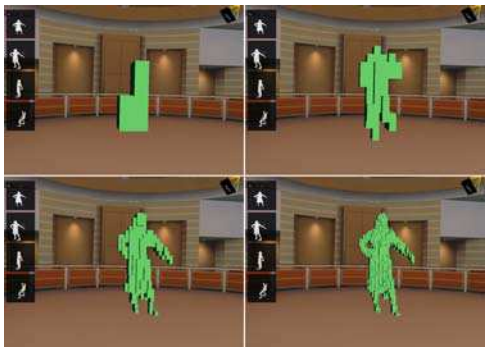


Figure 12: *Different resolutions of the voxel grid (8^3, 16^3, 32^3, 64^3).*

5.3. Marching cube on binary voxels

To produce a triangulated surface from the voxel data, we use the classical marching cube[12] algorithm. From an implicit function defined over space (the voxels reconstructed so far), the goal is to derive a smoother surface by reconstructing triangle strips (see figure 13).

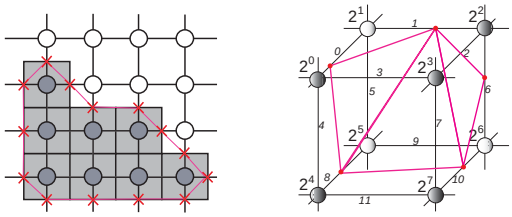


Figure 13: *Left: the relation between the input voxels (shaded squares), the in or out values on each grid corner, and the reconstructed contour. Right: in 3D, a table is used to list triangle strips. The code of the cell, computed from the corner values, is used to index a fixed table of edge lists. In the case shown (`10011101` = 157), we have `strips[157] = {6, 10, 1, 8, 0, END}`.*

The implementation used provides a slight improvement over the original algorithm for *OpenGL* rendering, because we pre-compute stripping into the *strips* table. A simple combinator pass over the table combines triangles sharing one edge into strips, and reduces the total number of vertices by 29% (and the data send by only 21%, because of the `STOP` code(s) inserted when more than one strip is needed for a cell). But in practice, a speedup of approximatively 2 is observed, because the triangle do share edges, but also because all the cells don't have the same compression ratio, nor the same probability.

The final result is shown in figure 14. The roughness observed (and particularly the normal quantization) is due to the binary nature of the input data. A smoothing should be used at that level to improve the surface regularity.

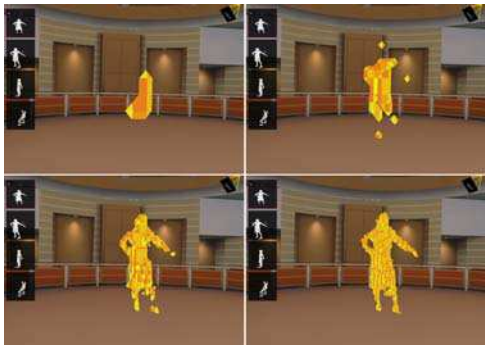


Figure 14: *Different resolutions of the marching cube (8^3, 16^3, 32^3, 64^3).*

6. 3D Graphics integration

The 3D model can be used for visual purposes: it can be rendered "as is" in a virtual world with points or cubes (see Figures 12 and 14), but it also permits volumetric effects and interaction.

Figure 15: *Marching cube representation: Shadow due to a virtual light. Left: the actor seems to be "flying" above the floor. Right: shadows remove this impression*

To integrate the avatar in a realistic way in a virtual scene we must add cast shadows. These shadows help understanding relative object position in the virtual scene. In particular, it prevents the actor from "flying above" the floor (see Figures 15). Many real-time techniques, based on geometry[3, 6] or on a bitmap representation[25, 16] can be used. Since the geometry is changing at each frame, we chose a simple projective shadow technique which is fast, does not depend on geometry complexity and cast shadows on any geometry. Shadows in the scene are precalculated and we only need to calculate at each frame the shadows cast by the avatar. This is done by rendering a black and white texture representing the avatar as seen from the light source and by projecting this image on the scene geometry following the light direction.

Another advantage to dispose of 3D information is that

Figure 16: *Example of 3D interaction. The actor could interact with the five virtual cubes.*

we can calculate collision between the avatar and its virtual environment. We implemented a virtual synthesizer, symbolized by five semi-transparent cubes that play different sounds when the avatar collides them (see Figure 16). Another example implemented was interaction with a virtual light, with a 3D interface (same cubes) allowing a light source to move around the avatar. Note that we don't match a skeleton to the avatar (like Theobalt[20]), so we can't detect which part of the body is colliding. We only used the percentage of filled voxel in different particular regions of our grid of voxel.

7. Results

As seen in section 4.2, it takes less than 25ms to capture and extract the actor, that permits taking advantage of the maximum performance from cameras. On the server part, we show in detail how we reconstruct also at 30fps.

The following table shows precisely how much time is spent in each processing step. The reconstruction time is the most important. It is distributed between the number of renders done (N per camera, where N is the resolution of voxel space) and the transfer from frame buffer (N^3 pixels) to our voxel data structure in main memory. As each of this call is time consuming, we do it only once by previously rendering the whole N^3 pixels.

step	*time*	*total*
scene rendering (34.000 triangles)	6ms	6ms
+ receiving 4 textures	2ms	8ms
+ reconstruction 64^3	15ms	23ms
+ rendering as cubes	3ms	26ms

Time for each step, with 4 cameras and 64^3 *voxels*

We integrate our reconstruction into several virtual scenes with different resolutions to see the limits of our method. The following table show results obtained in a 34.000 polygons scene (see Figure 12). We can see that the 30fps constraint is respected even at the highest voxel resolution.

voxel-space resolution	*time(fps)*	*with shadow*
64^3	26ms(33Hz)	41ms(20Hz)
32^3	13ms(50Hz)	18ms(50Hz)
16^3	10ms(50Hz)	14ms(50Hz)

Reconstruction and rendering performance

8. Conclusion

As demonstrated in section 7, we have been able to achieve the real-time constraint all over the pipelined processing. We should emphasis that it was a tough work to spot out the bottlenecks, and to solve them. In particular, we are running at the maximum camera's rate and definition, and at the maximum of the texture size allowed on the *Onyx's IR3*. So this is the best we could possibly obtain from that setup.

To go even further, we plan to upgrade the setup in the following ways:

- By connecting a video camera to the Onyx server, we will render a color image mapped onto the geometry. As 3D geometry of the avatar is known, we can relight each pixel of this image according to the virtual lighting.
- Add more contour cameras (e.g. 2 or 4 by PCs) to have tighter volume carving, and less ambiguities. The next setup targeted will start with 20 cameras.
- Use higher definition contour cameras (e.g. *Sony VL900*) running at 2048*x*2048 @ 10Hz. And use time multiplexing to keep a good frame rate.
- Use several graphic cards to compute the voxel carving, and improve geometry extraction by running the marching cube on a continuous *inside-ness* function.

Furthermore, video integration into the virtual scene could be improved by using real-time soft smooth shadows, for instance inspired from [19].

We're also working on interacting with the virtual worlds: because we have a real-time reconstruction of the actor's shape, we are able to compute contacts between the real actor and the virtual set. This open us the exciting world of live virtual experiences.

Acknowledgments

Thanks to Isabelle, who gracefully accepted to dance in front of our cameras. This work was supported by the "ACI Jeunes Chercheurs" of the Department of the Research.

References

1. Kong Man Cheung, Takeo Kanade, J.-Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. In *Proceedings of the*

2000 IEEE Conference on Computer Vision and Pattern Recognition (CVPR '00), volume 2, pages 714 – 720, June 2000.

2. R. Cipolla and P.J. Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 1999.
3. Franklin C. Crow. Shadow algorithms for computer graphics. In *Proceedings of the 4th annual conference on Computer graphics and interactive techniques*, pages 242–248. ACM Press, 1977.
4. Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM Press, 1996.
5. C.R. Dyer. Volumetric Scene Reconstruction from Multiple Views. In L.S. Davis, editor, *Foundations of Image Understanding*, pages 469–489. Kluwer, Boston, 2001.
6. Heidmann. Real shadows, real time. In *Iris Universe*, pages 23–31. Silicon Graphics Inc., 1991.
7. Sing Bing Kang and Richard Szeliski. 3-d scene data recovery using omnidirectional multibaseline stereo. Technical Report CRL 85/6, Cambridge Research Lab, 1996.
8. A. Laurentini. The Visual Hull Concept for Silhouette-Based Image Understanding. *pami*, 16(2):150–162, February 1994.
9. S. Lazebnik, E. Boyer, and J. Ponce. On How to Compute Exact Visual Hulls of Object Bounded by Smooth Surfaces. In *cvpr01*, volume I, pages 156–161, December 2001.
10. Ming Li, Hartmut Schirmacher, Marcus Magnor, and Hans-Peter Seidel. Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *Proc. 5th Conf. on Multimedia Signal Processing*, 2002.
11. Benjamin Lok. Online model reconstruction for interactive virtual environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 69–72. ACM Press, 2001.
12. William E. Lorensen and Harvey E. Cline. Marchnig cubes: A high resolution 3d surface construction algorithm. *Computer Graphics*, 21(4):163–169, July 1987.
13. W. Matusik, C. Buehler, and L. McMillan. Polyhedral Visual Hulls for Real-Time Rendering. In *Eurographics Workshop on Rendering*, pages 115–126, 2001.
14. Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.
15. Open computer vision library. http://sourceforge.net/projects/opencvlibrary/.
16. Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252. ACM Press, 1992.
17. Steven Seitz and Charles Dyer. Photorealistic Scene Reconstruction by Voxel Coloring. *International Journal of Computer Vision*, 25(3), November 1999.
18. G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafe. A survey o methods for volumetric scene reconstruction from photographs. In *International Workshop on Volume Graphics*, 2001.
19. Cyril Soler and François Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics Proceedings*, pages 321–332, Jul 1998. Annual Conference Series, SIGGRAPH'98.
20. C. Theobalt, M. Magnor, P. Schüler, and H.-P. Seidel. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. *Proc. IEEE Pacific Graphics 2002), Beijing, China*, pages 96–103, oct 2002.
21. K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and Practice of Background Maintenance. In *iccv99*, pages 255–261, 1999.
22. Roger Y. Tsai. An Efficient and Accurate Camera Calibration Technique for 3d Machine Vision. *Proceedings of IEEE Coonference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.
23. Iwan Ulrich, Christopher Baker, Bart Nabbe, and Illah Nourbakhsh. Ieee-1394 digital camera windows driver. http://www-2.cs.cmu.edu/ iwan/1394/.
24. Sundar Vedula, Peter Rander, Hideo Saito, and Takeo Kanade. Modeling, combining, and rendering dynamic real-world events from image sequences. In *Proc. 4th Conference on Virtual Systems and Multimedia (VSMM98)*, pages 326 – 332, November 1998.
25. Lance Williams. Casting curved shadows on curved surfaces. In *Proceedings of the 5th annual conference on Computer graphics and interactive techniques*, pages 270–274. ACM Press, 1978.
26. Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 22(11):1330–1334, 2000.

4 A survey of Real-Time Soft Shadows Algorithms

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion, "A survey of Real-Time Soft Shadows Algorithms", Computer Graphics Forum, 22(4), pp 753-774, Dec. 2003

Volume 22 (2003), Number 4 pp. 753–774

A Survey of Real-time Soft Shadows Algorithms

J.-M. Hasenfratz†, M. Lapierre‡, N. Holzschuch§ and F. Sillion§

Artis GRAVIR/IMAG-INRIA**

Abstract

Recent advances in GPU technology have produced a shift in focus for real-time rendering applications, whereby improvements in image quality are sought in addition to raw polygon display performance. Rendering effects such as antialiasing, motion blur and shadow casting are becoming commonplace and will likely be considered indispensable in the near future. The last complete and famous survey on shadow algorithms — by Woo et al.[52] *in 1990 — has to be updated in particular in view of recent improvements in graphics hardware, which make new algorithms possible. This paper covers all current methods for real-time shadow rendering, without venturing into slower, high quality techniques based on ray casting or radiosity. Shadows are useful for a variety of reasons: first, they help understand relative object placement in a 3D scene by providing visual cues. Second, they dramatically improve image realism and allow the creation of complex lighting ambiances. Depending on the application, the emphasis is placed on a guaranteed framerate, or on the visual quality of the shadows including penumbra effects or "soft shadows". Obviously no single method can render physically correct soft shadows in real time for any dynamic scene! However our survey aims at providing an exhaustive study allowing a programmer to choose the best compromise for his/her needs. In particular we discuss the advantages, limitations, rendering quality and cost of each algorithm. Recommendations are included based on simple characteristics of the application such as static/moving lights, single or multiple light sources, static/dynamic geometry, geometric complexity, directed or omnidirectional lights, etc. Finally we indicate which methods can efficiently exploit the most recent graphics hardware facilities.*

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Color, shading, shadowing, and texture, I.3.1 [Computer Graphics]: Hardware Architecture – Graphics processors, I.3.3 [Computer Graphics]: Picture/Image Generation – Bitmap and framebuffer operations

Keywords: *shadow algorithms, soft shadows, real-time, shadow mapping, shadow volume algorithm.*

1. Introduction

Cast shadows are crucial for the human perception of the 3D world. Probably the first thorough analysis of shadows was Leonardo Da Vinci's[48] (see Figure 1), focusing on paintings and static images. Also of note is the work of Lambert[35] who described the geometry underlying cast shadows (see Figure 1), and more recently the paper from Knill *et al.*[34].

With the emergence of computer graphics technology, researchers have developed experiments to understand the impact of shadows on our perception of a scene. Through different psychophysical experiments they established the important role of shadows in understanding:

- the position and size of the occluder[49, 38, 27, 30, 31];
- the geometry of the occluder[38];
- the geometry of the receiver[38].

Wanger[49] studied the effect of shadow quality on the perception of object relationships, basing his experiments on

† University Pierre Mendès France – Grenoble II
‡ University Joseph Fourier – Grenoble I
§ INRIA
** Artis is a team of the GRAVIR/IMAG laboratory, a joint research unit of CNRS, INPG, INRIA, UJF.

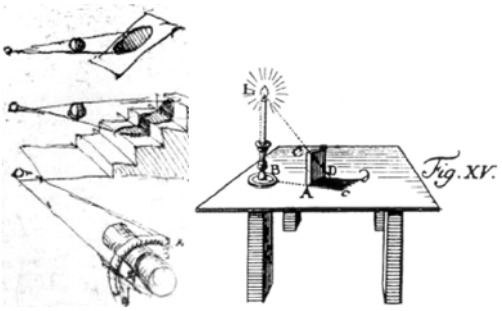


Figure 1: *Left: Study of shadows by Leonardo da Vinci*[48] *— Right: Shadow construction by Lambert*[35].

shadow sharpness. Hubona *et al.*[27] discuss the general role and effectiveness of object shadows in 3D visualization. In their experiments, they put in competition shadows, viewing mode (mono/stereo), number of lights (one/two), and background type (flat plane, "stair-step" plane, room) to measure the impact of shadows.

Kersten *et al.*[30, 31] and Mamassian *et al.*[38] study the relationship between object motion and the perception of relative depth. In fact, they demonstrate that simply adjusting the motion of a shadow is sufficient to induce dramatically different apparent trajectories of the shadow-casting object.

These psychophysical experiments convincingly establish that it is important to take shadows into account to produce images in computer graphics applications. Cast shadows help in our understanding of 3D environments and soft shadows take part in realism of the images.

Since the comprehensive survey of Woo *et al.*[52], progress in computer graphics technology and the development of consumer-grade graphics accelerators have made real-time 3D graphics a reality[3]. However incorporating shadows, and especially realistic soft shadows, in a real-time application, has remained a difficult task (and has generated a great research effort). This paper presents a survey of shadow generation techniques that can create soft shadows in real time. Naturally the very notion of "real-time performance" is difficult to define, suffice it to say that we are concerned with the display of 3D scenes of significant complexity (several tens of thousands of polygons) on consumer-level hardware *ca.* 2003. The paper is organized as follows:

We first review in Section 2 basic notions about shadows: hard and soft shadows, the importance of shadow effects showing problems encountered when working with soft shadows and classical techniques for producing hard shadows in real time. Section 3 then presents existing algorithms for producing soft shadows in real time. Section 4 offers a discussion and classifies these algorithms based on their different abilities and limitations, allowing easier algorithm selection depending on the application's constraints.

2. Basic concepts of hard and soft shadows

2.1. What is a shadow?

Consider a light source L illuminating a scene: *receivers* are objects of the scene that are potentially illuminated by L. A point P of the scene is considered to be in the *umbra* if it can not see any part of L, *i.e.* it does not receive any light directly from the light source.

If P can see a part of the light source, it is in the *penumbra*. The union of the umbra and the penumbra is the shadow, the region of space for which at least one point of the light source is occluded. Objects that hide a point from the light source are called *occluders*.

We distinguish between two types of shadows:

attached shadows, occuring when the normal of the receiver is facing away from the light source;

cast shadows, occuring when a shadow falls on an object whose normal is facing toward the light source.

Self-shadows are a specific case of cast shadows that occur when the shadow of an object is projected onto itself, *i.e.* the occluder and the receiver are the same.

Attached shadows are easy to handle. We shall see later, in Section 4, that some algorithms cannot handle self-shadows.

2.2. Importance of shadow effects

As discussed in the introduction, shadows play an important role in our understanding of 3D geometry:

- Shadows help to **understand relative object position and size** in a scene[49, 38, 27, 30, 31]. For example, without a cast shadow, we are not able to determine the position of an object in space (see Figure 2(a)).
- Shadows can also help us **understanding the geometry of a complex receiver**[38] (see Figure 2(b)).
- Finally, shadows provide useful visual cues that help in **understanding the geometry of a complex occluder**[38] (see Figure 3).

2.3. Hard shadows *vs.* soft shadows

The common-sense notion of shadow is a binary status, *i.e.* a point is either "in shadow" or not. This corresponds to *hard shadows*, as produced by point light sources: indeed, a point light source is either visible or occluded from any receiving point. However, point light sources do not exist in practice and hard shadows give a rather unrealistic feeling to images (see Figure 4(c)). Note that even the sun, probably the most common shadow-creating light source in our daily life, has a significant angular extent and does not create hard shadows. Still, point light sources are easy to model in computer

(a) Shadows provide information about the relative positions of objects. On the left-hand image, we cannot determine the position of the robot, whereas on the other three images we understand that it is more and more distant from the ground.

(b) Shadows provide information about the geometry of the receiver. Left: not enough cues about the ground. Right: shadow reveals ground geometry.

Figure 2: *Shadows play an important role in our understanding of 3D geometry.*

(a) (b) (c)

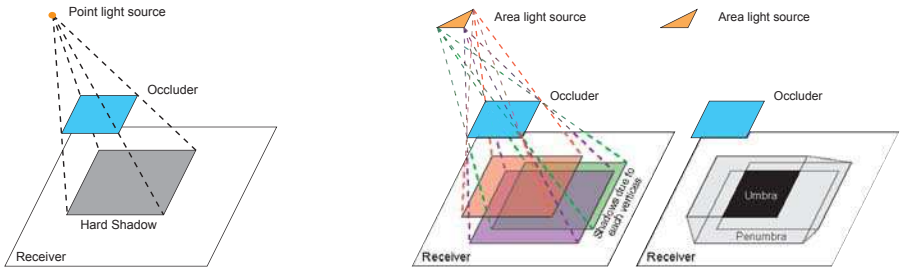
Figure 3: *Shadows provide information about the geometry of the occluder. Here we see that the robot holds nothing in his left hand on Figure 3(a), a ring on Figure 3(b) and a teapot on Figure 3(c).*

graphics and we shall see that several algorithms let us compute hard shadows in real time.

In the more realistic case of a light source with finite extent, a point on the receiver can have a partial view of the light, *i.e.* only a fraction of the light source is visible from that point. We distinguish the *umbra* region (if it exists) in which the light source is totally blocked from the receiver, and the *penumbra* region in which the light source is partially visible. The determination of the umbra and penumbra is a difficult task in general, as it amounts to solving visibility relationships in 3D, a notoriously hard problem. In the case of polygonal objects, the shape of the umbra and penumbra regions is embedded in a discontinuity mesh[13] which can be constructed from the edges and vertices of the light source and the occluders (see Figure 4(b)).

Soft shadows are obviously much more realistic than hard shadows (see Figures 4(c) and 4(d)); in particular the degree of softness (blur) in the shadow varies dramatically with the distances involved between the source, occluder, and receiver. Note also that a hard shadow, with its crisp boundary, could be mistakenly perceived as an object in the scene, while this would hardly happen with a soft shadow.

In computer graphics we can approximate small or distant light source as point sources only when the distance from the light to the occluder is much larger than the distance from the occluder to the receiver, and the resolution of the final image does not allow proper rendering of the penumbra. In all other cases great benefits can be expected from properly representing soft shadows.



(a) Geometry of hard shadows

(b) Geometry of soft shadows

(c) Illustration of hard shadows

(d) Illustration of soft shadows

Figure 4: *Hard vs. soft shadows.*

2.4. Important issues in computing soft shadows

2.4.1. Composition of multiple shadows

While the creation of a shadow is easily described for a (light source, occluder, receiver) triple, care must be taken to allow for more complex situations.

Shadows from several light sources Shadows produced by multiple light sources are relatively easy to obtain if we know how to deal with a single source (see Figure 5). Due to the linear nature of light transfer we simply sum the contribution of each light (for each wavelength or color band).

Shadows from several objects For point light sources, shadows due to different occluders can be easily combined since the shadow area (where the light source is invisible) is the union of all individual shadows.

With an area light source, combining the shadows of several occluders is more complicated. Recall that the lighting contribution of the light source on the receiver involves a partial visibility function: a major issue is that no simple combination of the partial visibility functions of distinct occluders can yield the partial visibility function of the set of occluders considered together. For instance there may be points in the scene where the light source is not occluded by any object taken separately, but is totally occluded by the set of objects taken together. The correlation between the partial visibility functions of different occluders cannot be predicted easily, but can sometimes be approximated or bounded[45, 5].

As a consequence, the shadow of the union of the objects can be larger than the union of the shadows of the objects (see Figure 6). This effect is quite real, but is not very visible on typical scenes, especially if the objects casting shadows are animated.

2.4.2. Physically exact or fake shadows

Shadows from an extended light source Soft shadows come from spatially extended light sources. To model properly the shadow cast by such light sources, we must take into account all the parts of the occluder that block light coming from the light source. This requires identifying all parts of the object casting shadow that are visible from at least one point of the extended light source, which is algorithmically much more complicated than identifying parts of the occluder that are visible from a single point.

Because this visibility information is much more difficult

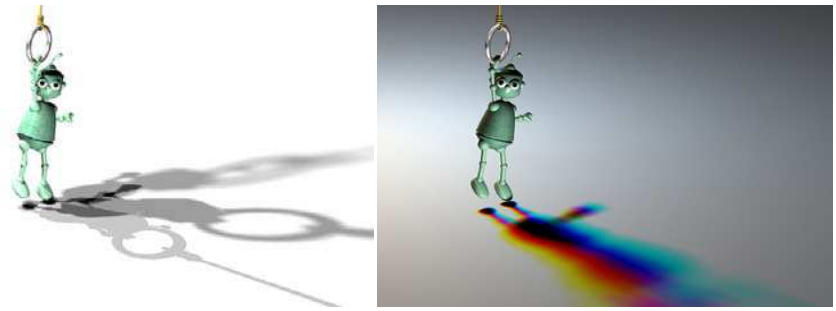


Figure 5: *Complex shadow due to multiple light sources. Note the complex interplay of colored lights and shadows in the complementary colors.*

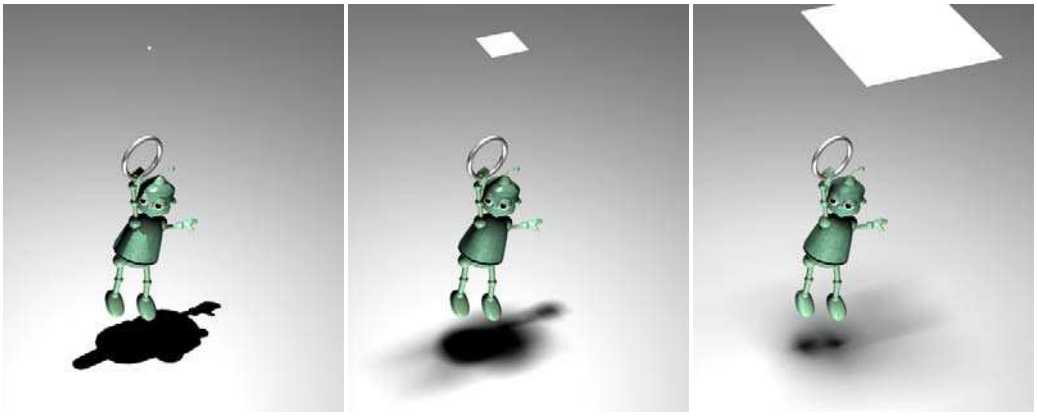


Figure 7: *When the light source is significantly larger than the occluder, the shape of the shadow is very different from the shape computed using a single sample; the sides of the object are playing a part in the shadowing.*

to compute with extended light sources than with point light sources, most real-time soft shadow algorithms compute visibility information from just one point (usually the center of the light source) and then simulate the behavior of the extended light source using this visibility information (computed for a point).

This method produces shadows that are not physically exact, of course, but can be close enough to real shadows for most practical applications. The difference between the approximation and the real shadow is harder to notice if the objects and their shadow are animated — a common occurrence in real-time algorithms.

The difference becomes more noticeable if the difference between the actual extended light source and the point used for the approximation is large, as seen from the object casting shadow. A common example is for a large light source, close enough from the object casting shadow that points of the light source are actually seeing different sides of the object (see Figure 7). In that case, the physically exact shadow is very different from the approximated version.

While large light sources are not frequent in real-time algorithms, the same problem also occurs if the object casting shadow is extended along the axis of the light source, *e.g.* a character with elongated arms whose right arm is pointing toward light source, and whose left arm is close to the receiver.

In such a configuration, if we want to compute a better looking shadow, we can either:

- Use the complete extension of the light source for visibility computations. This is algorithmically too complicated to be used in real-time algorithms.
- Separate the light source into smaller light sources[24, 5]. This removes some of the artefacts, since each light source is treated separately, and is geometrically closer to the

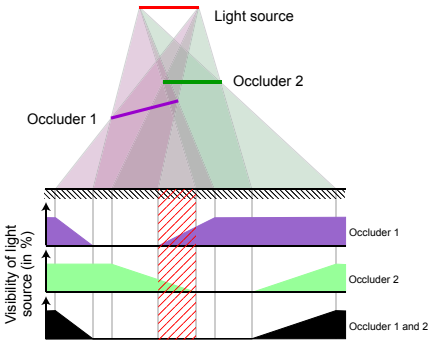


Figure 6: *The shadow of two occluders is not a simple combination of the two individual shadows. Note in particular the highlighted central region which lies in complete shadow (umbra) although the light source is never blocked by a single occluder.*

point sample used to compute the silhouette. The speed of the algorithm is usually divided by the number of light sources.

- Cut the object into slices[45]. We then compute soft shadows separately for each slice, and combine these shadows. By slicing the object, we are removing some of the visibility problems, and we allow lower parts of the object — usually hidden by upper parts — to cast shadow. The speed of the algorithm is divided by the number of slices, and combining the shadows cast by different slices remains a difficult problem.

Approximating the penumbra region When real-time soft shadow algorithms approximate extended light sources using points, they are in fact computing a hard shadow, and extending it to compute a soft shadow.

There are several possible algorithms:

- extend the umbra region outwards, by computing an *outer penumbra* region,
- shrink the umbra region, and complete it with an *inner penumbra* region,
- compute both inner penumbra and outer penumbra.

The first method (outer penumbra only) will always create shadows made of an umbra and a penumbra. Objects will have an umbra, even if the light source is very large with respect to the occluders. This effect is quite noticeable, as it makes the scene appear much darker than anticipated, except for very small light sources.

On the other hand, computing the inner penumbra region can result in light leaks between neighboring objects whose shadows overlap.

Illumination in the umbra region An important question is the illumination in regions that are in the umbra — completely hidden from the light source. There is no light reaching theses regions, so they should appear entirely black, in theory.

However, in practice, some form of ambient lighting is used to avoid completely dark regions and to simulate the fact that light eventually reaches these regions after several reflections.

Real-time shadow methods are usually combined with illumination computations, for instance using the simple OpenGL lighting model. Depending on whether the shadow method operates before or after the illumination phase, ambient lighting will be present or absent. In the latter case the shadow region appears completely dark, an effect that can be noticeable. A solution is to add the ambient shading as a subsequent pass; this extra pass slows down the algorithm, but clever re-use of the Z-buffer on recent graphics hardware make the added cost manageable[40].

Shadows from different objects As shown in Section 2.4.1, in presence of extended light sources, the shadow of the union of several objects is larger than the union of the individual shadows. Furthermore, the boundary of the shadow caused by the combination of several polygonal objects can be a curved line[13].

Since these effects are linked with the fact that the light source is extended, they can not appear in algorithms that use a single point to compute surfaces visible from the light source. All real-time soft shadow algorithms therefore suffer from this approximation.

However, while these effects are both clearly identifiable on still images, they are not as visible in animated scenes. There is currently no way to model these effects with real-time soft shadow algorithms.

2.4.3. Real-time

Our focus in this paper is on real-time applications, therefore we have chosen to ignore all techniques that are based on an expensive pre-process even when they allow later modifications at interactive rates[37]. Given the fast evolution of graphics hardware, it is difficult to draw a hard distinction between real-time and interactive methods, and we consider here that frame rates in excess of 10 fps, for a significant number of polygons, are an absolute requirement for “real-time” applications. Note that stereo viewing usually require double this performance.

For real-time applications, the display refresh rate is often the crucial limiting factor, and must be kept high enough (if not constant) through time. An important feature to be considered in shadowing algorithms is therefore their ability to guarantee a sustained level of performance. This is of course

impossible to do for arbitrary scenes, and a more important property for these algorithms is the ability to parametrically vary the level of performance (typically at the price of greater approximation), which allows an adaptation to the scene's complexity.

2.4.4. Shadows of special objects

Most shadowing algorithms make use of an explicit representation of the object's shapes, either to compute silhouettes of occluders, or to create images and shadow maps. Very complex and volumetric objects such as clouds, hair, grass etc. typically require special treatment.

2.4.5. Constraints on the scene

Shadowing algorithms may place particular constraints on the scene. Examples include the type of object model (techniques that compute a shadow as a texture map typically require a parametric object, if not a polygon), or the necessity/possibility to identify a subset of the scene as occluders or shadow receivers. This latter property is important in adapting the performance of the algorithm to sustain real-time.

2.5. Basic techniques for real-time shadows

In this State of the Art Review, we focus solely on real-time soft shadows algorithms. As a consequence, we will not describe other methods for producing soft shadows, such as radiosity, ray-tracing, Monte-Carlo ray-tracing or photon mapping.

We now describe the two basic techniques for computing shadows from *point light sources*, namely *shadow mapping* and the *shadow volume algorithm*.

2.5.1. Shadow mapping

Method The basic operation for computing shadows is identifying the parts of the scene that are hidden from the light source. Intrisically, it is equivalent to visible surface determination, from the point-of-view of the light source.

The first method to compute shadows[17, 44, 50] starts by computing a view of the scene, from the point-of-view of the light source. We store the z values of this image. This Z-buffer is the *shadow map* (see Figure 8).

The shadow map is then used to render the scene (from the normal point-of-view) in a two pass rendering process:

- a standard Z-buffer technique, for hidden-surface removal.
- for each pixel of the scene, we now have the geometrical position of the object seen in this pixel. If the distance between this object and the light is greater than the distance stored in the shadow map, the object is in shadow. Otherwise, it is illuminated.

Figure 8: *Shadow map for a point light source. Left: view from the camera. Right: depth buffer computed from the light source.*

- The color of the objects is modulated depending on whether they are in shadow or not.

Shadow mapping is implemented in current graphics hardware. It uses an OpenGL extension for the comparison between Z values, `GL_ARB_SHADOW`†.

Improvements The depth buffer is sampled at a limited precision. If surfaces are too close from each other, sampling problems can occur, with surfaces shadowing themselves. A possible solution[42] is to offset the Z values in the shadow map by a small bias[51].

If the light source has a cut-off angle that is too large, it is not possible to project the scene in a single shadow map without excessive distortion. In that case, we have to replace the light source by a combination of light sources, and use several depth maps, thus slowing down the algorithm.

Shadow mapping can result in large aliasing problems if the light source is far away from the viewer. In that case, individual pixels from the shadow map are visible, resulting in a staircase effect along the shadow boundary. Several methods have been implemented to solve this problem:

- Storing the ID of objects in the shadow map along with their depth[26].
- Using deep shadow maps, storing coverage information for all depths for each pixel[36].
- Using multi-resolution, adaptative shadow maps[18], computing more details in regions with shadow boundaries that are close to the eye.
- Computing the shadow map in perspective space[46], effectively storing more details in parts of the shadow map that are closer to the eye.

The last two methods are directly compatible with existing OpenGL extensions, and therefore require only a small amount of coding to work with modern graphics hardware.

An interesting alternative version of this algorithm is to

† This extension (or the earlier version, `GL_SGIX_SHADOW`, is available on Silicon Graphics Hardware above Infinite Reality 2, on NVidia graphics cards after GeForce3 and on ATI graphics cards after Radeon9500.

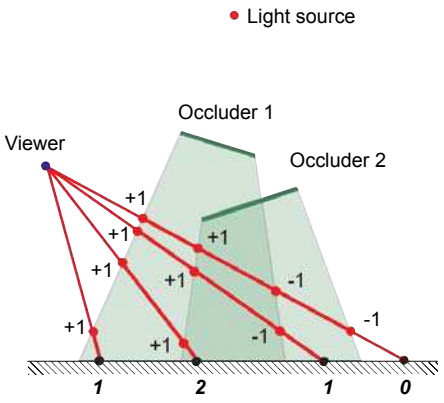


Figure 9: *Shadow volume.*

warp the shadow map into camera space[55] rather than the usual opposite: it has the advantage that we obtain a modulation image that can be mixed with a texture, or blurred to produce antialiased shadows.

Discussion Shadow mapping has many advantages:

- it can be implemented entirely using graphics hardware;
- creating the shadow map is relatively fast, although it still depends on the number and complexity of the occluders;
- it handles self-shadowing.

It also has several drawbacks:

- it is subject to many sampling and aliasing problems;
- it cannot handle omni-directional light sources;
- at least two rendering passes are required (one from the light source and one from the viewpoint);

2.5.2. The Shadow Volume Algorithm

Another way to think about shadow generation is purely geometrical. This method was first described by Crow[12], and first implemented using graphics hardware by Heidmann[23].

Method The algorithm consists in finding the silhouette of occluders along the light direction, then extruding this silhouette along the light direction, thus forming a *shadow volume*. Objects that are inside the shadow volume are in shadow, and objects that are outside are illuminated.

The shadow volume is calculated in two steps:

- the first step consists in finding the silhouette of the occluder as viewed from the light source. The simplest method is to keep edges that are shared by a triangle facing the light and another in the opposite direction. This actually gives a superset of the true silhouette, but it is sufficient for the algorithm.
- then we construct the shadow volume by extruding these edges along the direction of the point light source. For each edge of the silhouette, we build the half-plane subtended by the plane defined by the edge and the light source. All these half-planes define the shadow volume, and knowing if a point is in shadow is then a matter of knowing if it is inside or outside the volume.
- for each pixel in the image rendered, we count the number of faces of the shadow volume that we are crossing between the view point and the object rendered. Front-facing faces of the shadow volume (with respect to the view point) increment the count, back-facing faces decrement the count (see Figure 9). If the total number of faces is positive, then we are inside the shadow volume, and the pixel is rendered using only ambient lighting.

The rendering pass is easily done in hardware using a stencil buffer[23, 32, 15]; faces of the shadow volume are rendered in the stencil buffer with depth test enabled this way: in a first pass, front faces of the shadow volumes are rendered incrementing the stencil buffer; in a second pass, back faces are rendered, decrementing it. Pixels that are in shadow are "captured" between front and back faces of the shadow volume, and have a positive value in the stencil buffer. This way to render volumes is called *zpass*.

Therefore the complete algorithm to obtain a picture using the Shadow Volume method is:

- render the scene with only ambient/emissive lighting;
- calculate and render shadow volumes in the stencil buffer;
- render the scene illuminated with stencil test enabled: only pixels which stencil value is 0 are rendered, others are not updated, keeping their ambient color.

Improvements The cost of the algorithm is directly linked to the number of edges in the shadow volume. Batagelo and Júnior[7] minimize the number of volumes rendered by precalculating in software a modified BSP tree. McCool[39] extracts the silhouette by first computing a shadow map, then extracting the discontinuities of the shadow map, but this method requires reading back the depth buffer from the graphics board to the CPU, which is costly. Brabec and Seidel[10] reports a method to compute the silhouette of the occluders using programmable graphics hardware[14], thus obtaining an almost completely hardware-based implementation of the shadow volume algorithm (he still has to read back a buffer into the CPU for parameter transfer).

Roettger *et al.*[43] suggests an implementation that doesn't require the stencil buffer; he draws the shadow volume in the alpha buffer, replacing increment/decrement with a multiply/divide by 2 operation.

Everitt and Kilgard[15] have described a robust implementation of the shadow volume algorithm. Their method includes capping the shadow volume, setting $w = 0$ for extruded vertices (effectively making infinitely long quads) and setting the far plane at an infinite distance (they prove that this step

only decreases Z-buffer precision by a few percents). Finally, they render the shadow volume using the *zfail* technique; it works by rendering the shadow volume *backwards*:

- we render the scene, storing the Z-buffer;
- in the first pass, we increment the stencil buffer for all back-facing faces, but only if the face is behind an existing object of the scene;
- in the second pass, we decrement the stencil buffer for all front-facing faces, but only if the face is behind an existing object;
- The stencil buffer contains the intersection of the shadow volume and the objects of the scene.

The *zfail* technique was discovered independently by Bilodeau and Songy and by Carmack.

Recent extensions to OpenGL[15, 16, 21] allow the use of shadow volumes using stencil buffer in a single pass, instead of the two passes required so far. They also[15] provide *depth-clamping*, a method in which polygon are not clipped at the near and far distance, but their vertices are projected onto the near and far plane. This provides in effect an infinite view pyramid, making the shadow volume algorithm more robust.

The main problem with the shadow volume algorithm is that it requires drawing large polygons, the faces of the shadow volume. The fillrate of the graphics card is often the bottleneck. Everitt and Kilgard[15, 16] list different solutions to reduce the fillrate, either using software methods or using the graphics hardware, such as scissoring, constraining the shadow volume to a particular fragment.

Discussion The shadow volume algorithm has many advantages:

- it works for omnidirectional light sources;
- it renders eye-view pixel precision shadows;
- it handles self-shadowing.

It also has several drawbacks:

- the computation time depends on the complexity of the occluders;
- it requires the computation of the silhouette of the occluders as a preliminary step;
- at least two rendering passes are required;
- rendering the shadow volume consumes fillrate of the graphics card.

3. Soft shadow algorithms

In this section, we review algorithms that produce soft shadows, either interactively or in real time. As in the previous section, we distinguish two types of algorithms:

- Algorithms that are based on an image-based approach, and build upon the shadow map method described in Section 2.5.1. These algorithms are described in Section 3.1.
- Algorithms that are based on an object-based approach, and build upon the shadow volume method described in Section 2.5.2. These algorithms are described in Section 3.2.

3.1. Image-Based Approaches

In this section, we present soft shadow algorithms based on shadow maps (see Section 2.5.1). There are several methods to compute soft shadows using image-based techniques:

1. Combining several shadow textures taken from point samples on the extended light source[25, 22].
2. Using layered attenuation maps[1], replacing the shadow map with a Layered Depth Image, storing depth information about all objects visible from at least one point of the light source.
3. Using several shadow maps[24, 54], taken from point samples on the light source, and an algorithm to compute the percentage of the light source that is visible.
4. Using a standard shadow map, combined with image analysis techniques to compute soft shadows[9].
5. Convolving a standard shadow map with an image of the light source[45].

The first two methods approximate the light source as a combination of several point samples. As a consequence, the time for computing the shadow textures is multiplied by the number of samples, resulting in significantly slower rendering. On the other hand, these methods actually compute more information than other soft shadow methods, and thus compute more physically accurate shadows. Most of the artefacts listed in Section 2.4.2 will not appear with these two methods.

3.1.1. Combination of several point-based shadow images[25, 22]

The simplest method[22, 25] to compute soft shadows using image based methods is to place sample points regularly on the extended light source. These sample points are used to compute binary occlusion maps, which are combined into an attenuation map, used to modulate the illumination (calculated separately).

Method Herf[25] makes the following assumptions on the geometry of the scene:

- a light source of uniform color,
- subtending a small solid angle with respect to the receiver,
- and with distance from the receiver having small variance.

With these three assumptions, contributions from all sample points placed on the light source will be roughly equal.

The user identifies in advance the object casting shadows, and the objects onto which we are casting shadow. For each object receiving shadow, we are going to compute a texture containing the soft shadow.

Figure 10: *Combining several occlusion maps to compute soft shadows. Left: the occlusion map computed for a single sample. Center: the attenuation map computed using 4 samples. Right: the attenuation map computed using 64 samples.*

Figure 11: *With only a small number of samples on the light source, artefacts are visible. Left: soft shadow computed using 4 samples. Right: soft shadow computed using 1024 samples.*

We start by computing a binary occlusion map for each sample point on the light source. For each sample point on the light source, we render the scene into an auxiliary buffer, using 0 for the receiver, and 1 for any other polygon. These binary occlusion maps are then combined into an attenuation map, where each pixel stores the number of sample points on the light source that are occluded. This attenuation map contains a precise representation of the soft shadow (see Figures 10 and 11).

In the rendering pass, this soft shadow texture is combined with standard textures and illumination, in a standard graphics pipeline.

Discussion The biggest problem for Herf[25] method is rendering the attenuation maps. This requires $N_p N_s$ rendering passes, where N_p is the number of objects receiving shadows, and N_s is the number of samples on the light source. Each pass takes a time proportionnal to the number of polygons in the objects casting shadows. In practice, to make this method run in real time, we have to limit the number of receivers to a single planar receiver.

To speed-up computation of the attenuation map, we can lower the number of polygons in the occluders. We can also lower the number of samples (n) to increase the framerate, but this is done at the expense of image quality, as the attenuation map contains only $n-1$ gray levels. With fewer than 9 samples (3×3), the user sees several hard shadows, instead of a single soft shadow (see Figure 11).

Herf's method is easy to parallelize, since all occlusion maps can be computed separately, and only one computer is needed to combine them. Isard *et al.*[28] reports that a parallel implementation of this algorithm on a 9-node Sepia-2a parallel calculator with high-end graphics cards runs at more than 100 fps for moderately complex scenes.

3.1.2. Layered Attenuation Maps[1]

The Layered Attenuation Maps[1] method is based on a modified layered depth image[29]. It is an extension of the previous method, where we compute a layered attenuation map for the entire scene, instead of a specific shadow map for each object receiving shadow.

Method It starts like the previous method: we place sample points on the area light source, and we use these sample points to compute a modified attenuation map:

- For each sample point, we compute a view of the scene, along the direction of the normal to the light source.
- Theses images are all warped to a central reference, the center of the light source.
- For each pixel of these images:
  - In each view of the scene, we have computed the distance to the light source in the Z-buffer.
  - We can therefore identify the object that is closest to the light source.
  - This object makes the first layer of the layered attenuation map.
  - We count the number of samples seeing this object, which gives us the percentage of occlusion for this object.
  - If other objects are visible for this pixel but further away from the light they make the subsequent layers.
  - For each layer, we store the distance to the light source and the percentage of occlusion.

The computed Layered Attenuation Map contains, for all the objects that are visible from at least one sample point, the distance to the light source and the percentage of sample points seeing this object.

At rendering time, the Layered Attenuation Map is used like a standard attenuation map, with the difference that all the objects visible from the light source are stored in the map:

- First we render the scene, using standard illumination and textures. This first pass eliminates all objects invisible from the viewer.
- Then, for each pixel of the image, we find whether the corresponding point in the scene is in the Layered Attenuation Map or not. If it is, then we modulate the lighting

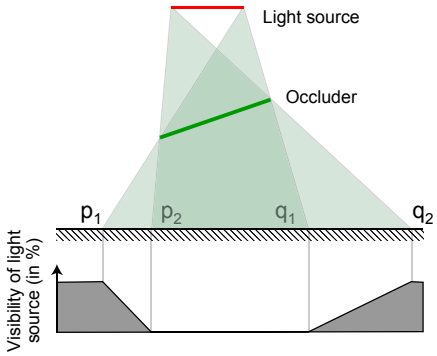


Figure 12: *Percentage of a linear light source that is visible.*

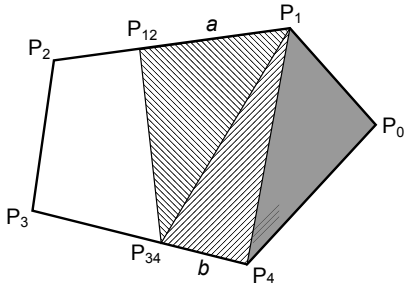


Figure 13: *Using the visibility channel to compute visibility from a polygonal light source. The shadow maps tell us that vertices P_0, P_1 and P_4 are occluded and that vertices P_2 and P_3 are visible. The visibility channel for edge $[P_1P_2]$ tells us that this edge is occluded for a fraction a; similarly, the visibility channel for edge $[P_3P_4]$ tells us that this edge is occluded for a fraction b. The portion of the light that is occluded is the hatched region, whose area can be computed geometrically using a and b.*

value found by the percentage of occlusion stored in the map. If it isn't, then the point is completely hidden from the light source.

Discussion The main advantage of this method, compared to the previous method, is that a single image is used to store the shadowing information for the entire scene, compared to one shadow texture for each shadowed object. Also, we do not have to identify beforehand the objects casting shadows.

The extended memory cost of the Layered Attenuation Map is reasonable: experiments by the authors show that on average, about 4 layers are used in moderately complex scenes.

As with the previous method, the speed and realism are related to the number of samples used on the light source. We are rendering the entire scene N_s times, which precludes real-time rendering for complex scenes.

3.1.3. Quantitative Information in the Shadow Map[24]

Heidrich *et al.*[24] introduced another extension of the shadow map method, where we compute not only a shadow map, but also a visibility channel(see Figure 12), which encodes the percentage of the light source that is visible. Heidrich *et al.*[24]'s method only works for linear light sources, but it was later extended to polygonal area light sources by Ying *et al.*[54].

Method We start by rendering a standard shadow map for each sample point on the linear light source. The number of sample points is very low, usually they are equal to the two end vertices of the linear light source.

In each shadow map, we detect discontinuities using image analysis techniques. Discontinuities in the shadow map happen at shadow boundaries. They are separating an object casting shadow from the object receiving shadow. For each discontinuity, we form a polygon linking the frontmost object (casting shadow) to the back object (receiving shadow). These polygons are then rendered in the point of view of the other sample, using Gouraud shading, with value 0 on the closer points, and 1 on the farthest points.

This gives us a visibility channel, which actually encodes the percentage of the edge linking the two samples that is visible.

The visibility channel is then used in a shadow mapping algorithm. For each pixel in the rendered image, we first check its position in the shadow map for each sample.

- if it is in shadow for all sample points, we assume that it is in shadow, and therefore it is rendered black.
- if it is visible from all sample points, we assume that it is visible, and therefore rendered using standard OpenGL illumination model.
- if it is hidden for some sample point, and visible from another point, we use the visibility channel to modulate the light received by the pixel.

Ying *et al.*[54] extended this algorithm to polygonal area light sources: we generate a shadow map for each vertex of the polygonal light source, and a visibility channel for each edge. We then use this information to compute the percentage of the polygonal light source that is visible from the current pixel.

For each vertex of the light source, we query the shadow map of this vertex. This gives us a boolean information, whether this vertex is occluded or not from the point of view of the object corresponding to the current pixel. If an edge

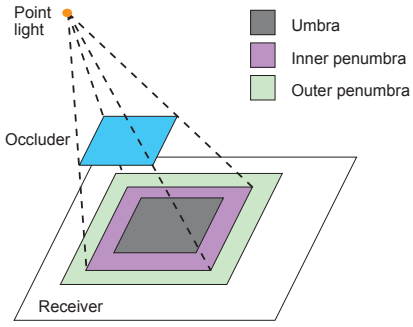


Figure 14: *Extending the shadow of a point light source: for each occluder identified in the shadow map, we compute a penumbra, based on the distance between this occluder and the receiver.*

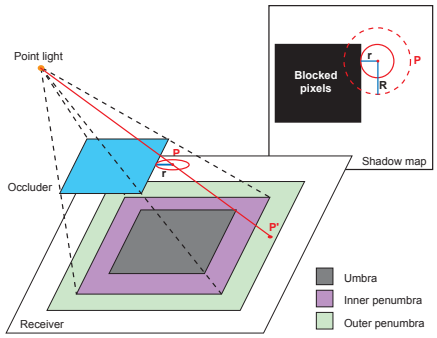


Figure 15: *Extending the shadow of a single sample: For each pixel in the image, we find the corresponding pixel P in the shadow map. Then we find the nearest blocked pixel. P is assumed to be in the penumbra of this blocker, and we compute an attenuation coefficient based on the relative distances betwen light source, occluder and P.*

links an occluded vertex to an non-occluded one, the visibility channel for this edge gives us the percentage of the edge that is occluded (see Figure 13). Computing the visible area of the light source is then a simple 2D problem. This area can be expressed as a linear combination of the area of triangles on the light source. By precomputing the area of these triangles, we are left with a few multiplications and additions to perform at each pixel.

Discussion The strongest point of this algorithm is that it requires a small number of sampling points. Although it can work with just the vertices of the light source used as sampling points, a low number of samples can result in artefacts in moderately complex scenes. These artefacts are avoided by adding a few more samples on the light source.

This method creates fake shadows, but nicely approximated. The shadows are exact when only one edge of the occluder is intersecting the light source, and approximate if there is more than one edge, for example at the intersection of the shadows of two different occluders, or when an occluder blocks part of the light source without blocking any vertex.

The interactivity of the algorithm depends on the time it takes to generate the visibility channels, which itself depends on the complexity of the shadow. On simples scenes (a few occluders) the authors report computation times of 2 to 3 frames per second.

The algorithm requires having a polygonal light source, and organising the samples, so that samples are linked by edges, and for each edge, we know the sample points it links.

3.1.4. Single Sample Soft Shadows[9, 33]

A different image-based method to generate soft shadows was introduced by Parker *et al.*[41] for parallel ray-tracing and later modified to use graphics hardware by Brabec and Seidel[9].

This method is very similar to standard shadow mapping. It starts by computing a standard shadow map, then uses the depth information available in the depth map to extend the shadow region and create a penumbra. In this method, we distinguish between the inner penumbra (the part of the penumbra that is inside the shadow of the point sample) and the outer penumbra (the part of the umbra that is outside the shadow of the point sample, see Figure 14). Parker *et al.*[41] compute only the outer penumbra; Brabec and Seidel[9] compute both the inner and the outer penumbra; Kirsch and Doellner[33] compute only the inner penumbra. In all cases, the penumbra computed goes from 0 to 1, to ensure continuity with areas in shadow and areas that are fully illuminated.

Method In a first pass, we create a single standard shadow map, for a single sample — usually at the center of the light source.

During rendering, as with standard shadow mapping, we identify the position of the current pixel in the shadow map. Then:

- if the current pixel is in shadow, we identify the nearest pixel in the shadow map that is illuminated.
- if the pixel is lit, we identify the nearest pixel in the shadow map that corresponds to an object that is closer to the light source than the current pixel (see Figure 15).

In both cases, we assume that the object found is casting a shadow on the receiver, and that the point we have found is in the penumbra. We then compute an attenuation coefficient based on the relative positions of the receiver, the occluder

and the light source:

$$f = \frac{\text{dist}(\text{Pixel}_{\text{Occluder}}, \text{Pixel}_{\text{Receiver}})}{RSz_{\text{Receiver}}|z_{\text{Receiver}} - z_{\text{Occluder}}|}$$

where R and S are user-defineable parameters. The intensity of the pixel is modulated using[8]:

- $0.5 * (1 + f)$, clamped to $[0.5, 1]$ if the pixel is outside the shadow,
- $0.5 * (1 - f)$, clamped to $[0, 0.5]$ if the pixel is inside the shadow.

For pixels that are far away from the boundary of the shadow, either deep inside the shadow or deep inside the fully lit area, f gets greater than 1, resulting in a modulation coefficient of respectively 0 or 1. On the original shadow boundary, $f = 0$, the two curves meet each other continuously with a modulation coefficient of 0.5. The actual width of the penumbra region depends on the ratio of the distances to the light source of the occluder and the receiver, which is perceptually correct.

The slowest phase of this algorithm is the search of neighbouring pixels in the shadow map, to find the potential occluder. In theory, an object can cast a penumbra than spans the entire scene, if it is close enough to the light source. In practice, we limit the search to a maximal distance to the current pixel of $R_{max} = Rz_{\text{Receiver}}$.

To ensure that an object is correctly identified as being in shadow or illuminated, the information from the depth map is combined with an item buffer, following Hourcade and Nicolas[26].

Discussion The aim of this algorithm is to produce perceptually pleasing, rather than physically exact, soft shadows. The width of the penumbra region depends on the ratio of the respective distances to the light source of the occluder and the receiver. The penumbra region is larger if the occluder is far from the receiver, and smaller if the occluder is close to the receiver.

Of course, the algorithm suffers from several shortcomings. Since the shadow is only determined by a single sample shadow map, it can fail to identify the proper shadowing edge. It works better if the light source is far away from the occluder. The middle of the penumbra region is placed on the boundary of the shadow from the single sample, which is not physically correct.

The strongest point of this algorithm is its speed. Since it only needs to compute a single shadow map, it can achieve framerates of 5 to 20 frames per second, compared with 2 to 3 frames per second for multi-samples image-based methods. The key parameter in this algorithm is R, the search radius. For smaller search values of R, the algorithms works faster, but can miss large penumbras. For larger values of R, the algorithm can identify larger penumbras, but takes longer for each rendering.

A faster version of this algorithm, by Kirsch and Doellner[33], computes both the shadow map and a shadow-width map: for each point in shadow, we precompute the distance to the nearest point that is illuminated. For each pixel, we do a look-up in the shadow map and the shadow-width map. If the point is occluded, we have the depth of the current point (z), the depth of the occluder (z_{occluder}) and the shadow width (w). A 2D function gives us the modulation coefficient:

$$I(z, w) = \begin{cases} 1 & \text{if} \quad z = z_{\text{occluder}} \\ 1 + c_{bias} - c_{scale}\frac{w}{z_{\text{occluder}} - z} & \text{otherwise} \end{cases}$$

The shadow-width map is generated from a binary occlusion map, transformed into the width map by repeated applications of a smoothing filter. This repeated filtering is done using graphics hardware, during rendering. Performances depend mostly on the size of the occlusion map and on the size of the filter; for a shadow map resolution of 512×512 pixels, and a large filter, they attain 20 frames per second. Performance depends linearly on the number of pixels in the occlusion map, thus doubling the size of the occlusion map divides the rendering speed by 4.

3.1.5. Convolution technique[45]

As noted earlier, soft shadows are a consequence of partial visibility of an extended light source. Therefore the calculation and soft shadows is closely related to the calculation of the visible portion of the light source.

Soler and Sillion[45] observe that the percentage of the source area visible from a receiving point can be expressed as a simple convolution for a particular configuration. When the light source, occluder, and receiver all lie in parallel planes, the soft shadow image on the receiver is obtained by convolving an image of the receiver and an image of the light source. While this observation is only mathematically valid in this very restrictive configuration, the authors describe how the same principle can be applied to more general configurations:

First, appropriate imaging geometries are found, even when the objects are non-planar and/or not parallel. More importantly, the authors also describe an error-driven algorithm in which the set of occluders is recursively subdivided according to an appropriate error estimate, and the shadows created by the subsets of occluders are combined to yield the final soft shadow image.

Discussion The convolution technique's main advantages are the visual quality of the soft shadows (not their physical fidelity), and the fact that it operates from images of the source and occluders, therefore once the images are obtained the complexity of the operations is entirely under control. Sampling is implicitly performed when creating a light source image, and the combination of samples is handled

by the convolution operation, allowing very complex light source shapes.

The main limitation of the technique is that the soft shadow is only correct in a restricted configuration, and the proposed subdivision mechanism can only improve the quality when the occluder can be broken down into smaller parts. Therefore the case of elongated polygons in th direction of the light source remains problematic. Furthermore, the subdivision mechanism, when it is effective in terms of quality, involves a significant performance drop.

3.2. Object-Based Approaches

Several methods can be used to compute soft shadows in animated scenes using object-based methods:

1. Combining together several shadow volumes taken from point samples on the light source, in a manner similar to the method described for shadow maps in Section 3.1.1.
2. extending the shadow volume[19, 53, 11] using a specific heuristic (Plateaus[19], Penumbra Maps[53], Smoothies[11]).
3. computing a penumbra volume for each edge of the shadow silhouette[2, 4, 5].

3.2.1. Combining several hard shadows

Method The simplest way to produce soft shadows with the shadow volume algorithm is to take several samples on the light source, compute a hard shadow for each sample and average the pictures produced. It simulates an area light source, and gives us the soft shadow effect.

However, the main problem with this method, as with the equivalent method for shadow maps, is the number of samples it requires to produce a good-looking soft shadow, which precludes any real-time application. Also, it requires the use of an accumulation buffer, which is currently not supported on standard graphics hardware.

An interesting variation has been proposed by Vignaud[47], in which shadow volumes from a light source whose position changes with time are added in the alpha buffer, mixed with older shadow volumes, producing a soft shadow after a few frames where the viewer position does not change.

3.2.2. Soft Planar Shadows Using Plateaus

The first geometric approach to generate soft shadows has been implemented by Haines[19]. It assumes a planar receiver, and generates an attenuation map that represents the soft shadow. The attenuation map is created by converting the edges of the occluders into volumes, and is then applied to the receiver as a modulating texture.

Method The principle of the plateaus method[19] is to generate an attenuation map, representing the soft shadow. The attenuation map is first created using the shadow volume method, thus filling in black the parts of the map that are occluded.

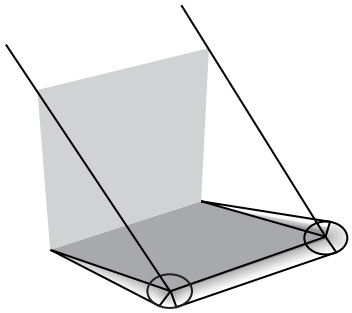


Figure 16: *Extending the shadow volume of an occluder with cones and planes.*

Then, the edges of the silhouette of the objects are transformed into volumes (see Figure 16):

- All the vertices of the silhouette are first turned into cones, with the radius of the cone depending on the distance between the occluder vertex and the ground, thus simulating a spherical light source.
- then edges joining adjacent vertices are turned into surfaces. For continuity, the surface joining two cones is an hyperboloid, unless the two cones have the same radius (that is, if the two original vertices are at the same distance of the ground), in which case the hyperboloid degenerates to a plane.

These shadow volumes are then projected on the receiver and colored using textures: the axis of the cone is black, and the contour is white. This texture is superimposed with the shadow volume texture: Haines' algorithm only computes the outer penumbra.

One important parameter in the algorithm is the way we color the penumbra volume; it can be done using Gouraud shading, values from the Z-buffer or using a 1D texture. The latter gives more control over the algorithm, and allows penumbra to decrease using any function, including sinusoid.

Discussion The first limitation of this method is that it is limited to shadows on planar surfaces. It also assumes a spherical light source. The size of the penumbra only depends on the distance from the receiver to the occluders, not from the distance between the light source and the occluders. Finally, it suffers from the same fillrate bottleneck as the original shadow volume algorithm.

A significant improvement is Wyman and Hansen[53]'s Penumbra Map method: the interpolation step is done using programmable graphics hardware[6, 20, 14], generating a

penumbra map that is applied on the model, along with a shadow map. Using a shadow map to generate the umbra region removes the fill-rate bottleneck and makes the method very robust. Wyman and Hansen report framerate of 10 to 15 frames per second on scenes with more than 10,000 shadow-casting polygons.

The main limitation in both methods[19, 53] is that they only compute the outer penumbra. As a consequence, objects will always have an umbra, even if the light source is very large with respect to the occluders. This effect is clearly noticeable, as it makes the scene appear much darker than anticipated, except for very small light sources.

3.2.3. Smoothies[11]

Chan and Durand[11] present a variation of the shadow volume method that uses only graphics hardware for shadow generation.

Method We start by computing the silhouette of the object. This silhouette is then extended using "smoothies", that are planar surfaces connected to the edges of the occluder and perpendicular to the surface of the occluder.

We also compute a shadow map, which will be used for depth queries. The smoothies are then textured taking into account the distance of each silhouette vertex to the light source, and the distance between the light source and the receiver.

In the rendering step, first we compute the hard shadow using the shadow map, then the texture from the smoothies is projected onto the objects of the scene to create the penumbra.

Discussion As with Haines[19], Wyman and Hansen[53] and Parker[41], this algorithm only computes the outer penumbra. As a consequence, occluders will always project an umbra, even if the light source is very large with respect to the occluders. As mentionned earlier, this makes the scene appear much darker than anticipated, an effect that is clearly noticeable except for very small light sources.

The size of the penumbra depends on the ratio of the distances between the occluder and the light source, and between receiver and light source, which is perceptually correct.

Connection between adjacent edges is still a problem with this algorithm, and artefacts appear clearly except for small light sources.

The shadow region is produced using the shadow map method, which removes the problem with the fill rate bottleneck experienced with all other methods based on the shadow volume algorithm. As with the previous method[53], the strong point of this algorithm is its robustness: the authors have achieved 20 frames per second on scenes with more than 50,000 polygons.

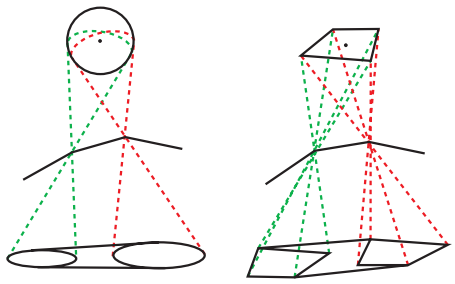


Figure 17: *Computing the penumbra wedge of a silhouette edge: the wedge is a volume based on the silhouette edge and encloses the light source.*

3.2.4. Soft Shadow Volumes[2, 4, 5]

Akenine-Möller and Assarsson[2], Assarsson and Akenine-Möller[4] and Assarsson *et al.*[5] have developed an algorithm to compute soft shadows that builds on the shadow volume method and uses the programmable capability of modern graphics hardware[6, 20, 14] to produce real-time soft shadows.

Method The algorithm starts by computing the silhouette of the object, as seen from a single sample on the light source. For each silhouette edge, we build a *silhouette wedge*, that encloses the penumbra caused by this edge (see Figure 17). The wedge can be larger than the penumbra, that is we err on the safe side.

Then, we render the shadow volume, using the standard method (described in Section 2.5.2) in a visibility buffer. After this first pass, the visibility buffer contains the hard shadow.

In a subsequent pass, this visibility buffer is updated so that it contains the soft shadow values. This is done by rendering the front-facing triangles of each wedge. For each pixel covered by these triangles, we compute the percentage of the light source that is occluded, using fragment programs[20]. For pixels that are covered by the wedge but in the hard shadow (as computed by the previous pass), we compute the percentage of the light source that is visible, and add this value to the visibility buffer. For pixels covered by the wedge but in the illuminated part of the scene, we compute the percentage of the light source that is occluded and substract this value from the visibility buffer (see Figures 18 and 19).

After this second pass, the visibility buffer contains the percentage of visibility for all pixels in the picture. In a third pass, the visibility buffer is combined with the illumination computed using the standard OpenGL lighting model, giving the soft shadowed picture of the scene.

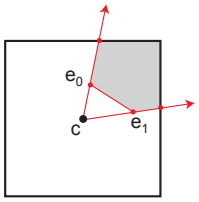


Figure 18: *Computing the area of the light source that is covered by a given edge. The fragment program computes the hatched area for each pixel inside the corresponding wedge.*

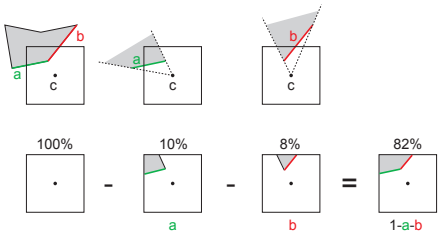


Figure 19: *Combining several connected edges. The portion of the light source that is occluded is equal to the sum of the portions of the light source occluded by the different edges.*

Discussion The complexity of the algorithm depends on the number of edges in the silhouette of the object, and on the number of pixels covered by each penumbra wedge. As a consequence, the easiest optimisation of the algorithm is to compute tighter penumbra wedges[5].

The main advantage of this algorithm is its speed. Using programmable graphics hardware for all complex computations, and tabulating complex functions into pre-computed textures, framerates of 150 frames per second are obtained on simple scenes, 50 frames per second on moderately complex scenes (1,000 shadow-casting polygons, with a large light source), with very convincing shadows. Performance depends mostly on the number of pixels covered by the penumbra wedges, so smaller light sources will result in faster rendering.

It should be noted that although a single sample is used to compute the silhouette of the object, the soft shadow computed by this algorithm is physically exact in simple cases, since visibility is computed on the entire light source. More precisely this happens when the silhouette of the occluder remains the same for all points on the light source, *e.g.* for a convex object that is distant enough from the light source.

The weak point of the algorithm is that it computes the silhouette of the object using only a single sample. It would fail on scenes where the actual silhouette of the object, as seen from the area light source, is very different from the silhouette computed using the single sample. Such scenes include scenes where a large area light source is close to the object (see Figure 7), and scenes where the shadows of several objects are combined together (as in Figure 6). In those circumstances, it is possible to compute a more accurate shadow by splitting the light source into smaller light sources. The authors report that splitting large light sources into 2×2 or 3×3 smaller light sources is usually enough to remove visible artefacts. It should be noted that splitting the light source into n light sources does not cut the speed of the algorithm by n, since the rendering time depends on the number of pixels covered by the penumbra wedges, and smaller light sources have smaller penumbra wedges.

One key to the efficiency of the algorithm is its use of fragment programs[20]. The fragment programs take as input the projections of the extremities of the edge onto the plane of the light source, and give as output the percentage of the light source that is occluded by the edge (see Figure 18). If several edges are projecting onto the light source, their contributions are simply added (see Figure 19) — this addition is done in the framebuffer. The authors have implemented several fragment programs, for spherical light sources, for textured rectangular light sources and for non-textured rectangular light sources.

4. Classification

4.1. Controlling the time

Algorithms used in real time or interactive applications must be able to run at a tuneable framerate, in order to spend less time for rendering at places where there is a lot of computation taking place, and more time when the processor is available.

Ideally, soft shadow methods used in real-time applications should take as input the amount of time available for rendering, and return a soft shadow computed to the best of the algorithm within the prescribed time limit. Since this review focuses on hot research algorithms, this feature has not been implemented in any of the algorithms reviewed here. However, all of these algorithms are tunable in the sense that there is some sort of parameter that the user can tweak, going from soft shadows that are computed very fast, but are possibly wrong, to soft shadows that can take more time to compute but are either more visually pleasing or more physically accurate.

Several of these parameters are available to a various degree in the methods reviewed:

- The easiest form of user control is the use of a different level-of-detail for the geometry of the occluders. Simpler geometry will result in faster rendering, either with image-based methods or with object-based methods. It can be expected that the difference in the shadow will not be noticeable with animated soft shadows.

Method	Time	Quality	Tunable	Light	Scene	Required Hardware
Image-based						
Multi-samples[22, 25]	I	*	Y	Polygon	1 planar receiver	
Distributed Multi-samples[28]	RT	**	Y	Planar		ShadowMap
Single sample[9, 33]	RT	*	Y	Sphere		ShadowMap
Convolution[45]	I	**	Y	Polygon		2D Convol.
Visibility Channel[24, 54]	I	**	Y	Linear, Polygon		2D Convol.
Geometry-based						
Plateaus[19]	I	**	Y	Sphere	1 planar receiver	
Penumbra Map[53]	RT	**	Y	Sphere		Vertex & Frag. Programs
Smoothie[11]	RT	**	Y	Sphere		Vertex & Frag. Programs
Soft Shadow Volumes[2, 4, 5]	RT	***	Y	Sphere, Rect.		Fragment Programs

Table 1: *Comparison of soft shadows algorithms (see Section 4 for details)*

- Another form of user control is to add more samples on the light source[22, 25, 1], or to subdivide large light sources into a set of smaller ones[2, 4, 5, 24, 54]. It should be noted that the order of magnitude for this parameter is variable: 256 to 1024 samples are required for point-based methods[22, 25, 1] to produce shadows without artefacts, while area-based methods[2, 4, 5, 24, 54] just need to cut the light source into 2×2 or 3×3 smaller sources. Either way, the rendering time is usually multiplied by the number of samples or sources.
- All image-based methods are also tuneable by changing the resolution of the buffer.
- Other parameters are method-specific:
  - the single sample soft shadows[9] method is tuneable by changing the search radius;
  - Convolution[45] is tuneable by subdividing the occluders into several layers;
  - Plateaus[19] are tuneable by changing the number of vertices used to discretize the cones and patches;
  - Smoothies[11] are tuneable by changing the maximum width of the smoothies;

4.2. Controlling the aspect

Another important information in chosing a real-time soft shadow algorithm is the aspect of the shadow it produces. Some of the algorithms described in this review can produce a physically exact solution if we allow them a sufficient rendering time. Other methods produce a physically exact solution in simple cases, but are approximate in more complex scenes, and finally a third class of methods produce shadows that are always approximate, but are usually faster to compute.

Physically exact (time permitting): Methods based on point samples on the light source[22, 25, 1] will produce physically exact shadows if the number of samples is sufficient. However, with current hardware, the number of samples compatible with interactive applications gives shadows that are not visually excellent (hence the poor mark these methods receive in table 1).

Physically exact on simple scenes: Methods that compute the percentage of the light source that is visible from the current pixel will give physically exact shadows in places where the assumptions they make on the respective geometry of the light source and the occluders are verified. For example, soft shadow volumes[4, 5] give physically exact shadows for isolated convex objects, provided that the silhouette computed is correct (that the occluder is far away from the light source). Visibility channel[24, 54] gives physically exact shadows for convex occluders and linear light sources[24], and for isolated edges and polygonal light sources[54] . Convolution[45] is physically exact for planar and parallel light source, receiver and occluder.

Always approximate: All methods that restrict themselves to computing only the inner- or the outer-penumbra are intrisically always approximate. They include single-sample soft shadows using shadow-width map[33], plateaus[19] and smoothies[11]. The original implementation of single sample soft shadows[9] computes both the inner- and the outer-penumbra, but gives them always the same width, which is not physically exact.

The second class of methods is probably the more interesting for producing nice looking pictures. While the conditions imposed seem excessively hard, it must be pointed out that they are conditions for which it is *guaranteed* that the shadow is exact in *all* the points of the scene. In most places of a standard scene, these methods will also produce physically exact shadows.

4.3. Number and shape of the light sources

The first cause for the soft shadow is the light source. Each real-time soft shadow method makes an assumption on the light sources, their shapes, their angles of emission and more importantly their number.

Field of emission: All the methods that are based on an image of the scene computed from the light source are restricted with respect to the field of emission of the light source, as a field of emission that is too large will result in distortions in the image. This restriction applies to all image-based algorithms, plus smoothies[11] and volume-based algorithms if the silhouette is computed using discontinuities in the shadow map[39].

On the contrary, volume-based methods can handle omni-directional illumination.

Shape: For extended light sources, the influence of the shape of the light source on a soft shadow is not directly perceptible. Most real-time soft shadow methods use this property by restricting themselves to simple light source shapes, such as spheres or rectangles:

- Single-sample soft shadows[9, 33], plateaus[19] and smoothies[11] assume a spherical light source. Soft shadow volumes[5] also work with a spherical light source.
- Visibility channel[24] was originally restricted to linear light sources.
- Subsequent implementation of the visibility channel works with polygonal light sources[54].
- Other methods place less restriction on the light source. Multi-sample methods[25, 1] can work with any kind of light source. Convolution[45] are also not restricted. However, in both cases, the error in the algorithm is smaller for planar light sources.
- Convolution[45] and soft shadow volumes[4, 5] work with textured rectangles, thus allowing any kind of planar light source. The texture can even be animated[4, 5].

Number: All real-time soft shadow algorithms are assuming a single light source. Usually, computing the shadow from several light sources results in multiplying the rendering time by the number of light sources. However, for all the methods that work for any kind of planar light source[25, 1, 45, 4, 5], it is possible to simulate several co-planar light sources by placing the appropriate texture on a plane. This gives us several soft shadows in a single application of the algorithm. However, it has a cost: since the textured light source is larger, the algorithms will run more slowly.

4.4. Constraints on the scene

The other elements causing shadows are the occluders and the receivers. Most real-time soft shadows methods make some assumptions on the scene, either explicit or implicit.

Receiver: The strongest restriction is when the object receiving shadows is a plane, as with the plateaus method[19]. Multi-sample soft shadow[25, 22] is also restricted to a small number of receivers for interactive rendering. In that case, self-shadowing is not applicable.

Self-shadowing: The convolution[45] method requires that the scene is cut into clusters, within which no self-shadows are computed.

Silhouette: For all the methods that require a silhouette extraction — such as object-based methods — it is implicitly assumed that we can compute a silhouette for all the objects in the scene. In practice, this usually means that the scene is made of closed triangle meshes.

4.5. New generation of GPUs

Most real-time soft shadow methods use the features of the graphics hardware that were available to the authors at the time of writing:

Shadow-map: all image-based methods use the `GL_ARB_SHADOW` extension for shadow maps. This extension (or an earlier version) is available, for example, on Silicon Graphics hardware above the Infinite Reality 2, on NVIDIA graphics cards above the GeForce 3 and on ATI graphics above the Radeon9500.

Imaging subset: along with this extension, some methods also compute convolutions on the shadow map. These convolutions can be computed in hardware if the *Imaging Subset* of the OpenGL specification is present. This is the case on all Silicon Graphics machines and NVIDIA cards.

Programmable GPU: finally, the most recent real-time soft shadow methods use the programming capability introduced in recent graphics hardware. Vertex programs[14] and fragment programs[21] are used for single-sample soft shadows[33], penumbra maps[53], smoothies[11] and soft shadow volumes[4, 5]. In practice, this restricts these algorithms to only the latest generation of graphics hardware, such as the NVIDIA GeForce FX or the ATI Radeon 9500 and above.

Many object-based algorithms suffer from the fact that they need to compute the silhouette of the occluders, a costly step that can only be done on the CPU. Wyman and Hansen[53] report that computing the silhouette of a moderately complex occluder (5000 polygons) uses 10 ms in their implementation. If the next generation of graphics hardware would include the possibility to compute this silhouette entirely on the graphics card[10], object-based algorithms[53, 11, 2, 4, 5] would greatly benefit from the speed-up.

5. Conclusions

In this State of the Art Review, we have described the issues encountered when working with soft shadows. We have presented existing algorithms that produce soft shadows in real time. Two main categories of approaches have been reviewed, based on shadow maps and shadow volumes. Each one has advantages and drawbacks, and none of them can simultaneously solve all the problems we have mentioned. This motivated a discussion and classification of these methods, hopefully allowing easier algorithm selection based on a particular application's constraints.

We have seen that the latest algorithms benefit from the programmability of recent graphics hardware. Two main directions appear attractive to render high-quality soft shadows in real time: by programming graphics hardware, and by taking advantage simultaneously of both image-based and object-based techniques. Distributed rendering, using for instance PC clusters, is another promising avenue although little has been achieved so far. Interactive display speeds can be obtained today even on rather complex scenes. Continuing improvements of graphics technology — in performance and programmability — lets us expect that soft shadows will soon become a common standard in real-time rendering.

Acknowledgments

The "Hugo" robot used in the pictures of this paper was created by Laurence Boissieux.

This work was supported in part by the "ACI Jeunes Chercheurs" *CYBER* of the French Ministry of Research, and by the "Région Rhône-Alpes" through the DEREVE research consortium.

We wish to express our gratitude to the authors of the algorithms described in this review, who have provided us with useful detailed information about their work, and to the anonymous reviewers whose comments and suggestions have significantly improved the paper.

Remark: All the smooth shadows pictures in this paper were computed with distributed ray-tracing, using 1024 samples on the area light sources.

References

1. Maneesh Agrawala, Ravi Ramamoorthi, Alan Heirich, and Laurent Moll. Efficient image-based methods for rendering soft shadows. In *Computer Graphics (SIGGRAPH 2000)*, Annual Conference Series, pages 375–384. ACM SIGGRAPH, 2000. 9, 10, 17, 18

2. Tomas Akenine-Möller and Ulf Assarsson. Approximate soft shadows on arbitrary surfaces using penumbra wedges. In *Rendering Techniques 2002 (13th Eurographics Workshop on Rendering)*, pages 297–306. ACM Press, 2002. 14, 15, 17, 18

3. Tomas Akenine-Möller and Eric Haines. *Real-Time Rendering*. A K Peters Ltd, 2nd edition, 2002. 2

4. Ulf Assarsson and Tomas Akenine-Möller. A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3), 2003. 14, 15, 17, 18

5. Ulf Assarsson, Michael Dougherty, Michael Mounier, and Tomas Akenine-Möller. An optimized soft shadow volume algorithm with real-time performance. In *Graphics Hardware*, 2003. 4, 5, 14, 15, 16, 17, 18

6. ATI. Smartshader^{TM} technology white paper. http://www.ati.com/products/pdf/smartshader.pdf, 2001. 14, 15

7. Harlen Costa Batagelo and Ilaim Costa Júnior. Real-time shadow generation using BSP trees and stencil buffers. In *SIBGRAPI*, volume 12, pages 93–102, October 1999. 8

8. Stefan Brabec. Personnal communication, May 2003. 13

9. Stefan Brabec and Hans-Peter Seidel. Single sample soft shadows using depth maps. In *Graphics Interface*, 2002. 9, 12, 17, 18

10. Stefan Brabec and Hans-Peter Seidel. Shadow volumes on programmable graphics hardware. *Computer Graphics Forum (Eurographics 2003)*, 25(3), September 2003. 8, 18

11. Eric Chan and Fredo Durand. Rendering fake soft shadows with smoothies. In *Rendering Techniques 2003 (14th Eurographics Symposium on Rendering)*. ACM Press, 2003. 14, 15, 17, 18

12. Franklin C. Crow. Shadow algorithms for computer graphics. *Computer Graphics (SIGGRAPH 1977)*, 11(3):242–248, 1977. 8

13. George Drettakis and Eugene Fiume. A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics (SIGGRAPH 1994)*, Annual Conference Series, pages 223–230. ACM SIGGRAPH, 1994. 3, 6

14. Cass Everitt. OpenGL ARB vertex program. http://developer.nvidia.com/docs/IO/8230/GDC2003_OGL_ARBVertexProgram.pdf, 2003. 8, 14, 15, 18

15. Cass Everitt and Mark J. Kilgard. Practical and robust stenciled shadow volumes for hardware-accelerated rendering. http://developer.nvidia.com/object/robust_shadow_volumes.html, 2002. 8, 9

16. Cass Everitt and Mark J. Kilgard. Optimized stencil shadow volumes. http://developer.nvidia.com/docs/IO/8230/GDC2003_ShadowVolumes.pdf, 2003. 9

17. Cass Everitt, Ashu Rege, and Cem Cebenoyan. Hardware shadow mapping. http://developer.nvidia.com/object/hwshadowmap_paper.html. 7

18. Randima Fernando, Sebastian Fernandez, Kavita Bala, and Donald P. Greenberg. Adaptive shadow maps. In *Computer Graphics (SIGGRAPH 2001)*, Annual Conference Series, pages 387–390. ACM SIGGRAPH, 2001. 7

19. Eric Haines. Soft planar shadows using plateaus. *Journal of Graphics Tools*, 6(1):19–27, 2001. 14, 15, 17, 18

20. Evan Hart. ARB Fragment Program: Fragment level programmability in OpenGL. http://www.ati.com/developer/gdc/GDC2003_OGL_ARBFragmentProgram.pdf, 2003. 14, 15, 16

21. Evan Hart. Other New OpenGL Stuff: Important stuff that doesn't fit elsewhere. http://www.ati.com/developer/gdc/GDC2003_OGL_MiscExtensions.pdf, 2003. 9, 18

22. Paul S. Heckbert and Michael Herf. Simulating soft shadows with graphics hardware. Technical Report CMU-CS-97-104, Carnegie Mellon University, January 1997. 9, 17, 18

23. Tim Heidmann. Real shadows, real time. In *Iris Universe*, volume 18, pages 23–31. Silicon Graphics Inc., 1991. 8

24. Wolfgang Heidrich, Stefan Brabec, and Hans-Peter Seidel. Soft shadow maps for linear lights high-quality. In *Rendering Techniques 2000 (11th Eurographics Workshop on Rendering)*, pages 269–280. Springer-Verlag, 2000. 5, 9, 11, 17, 18

25. Michael Herf. Efficient generation of soft shadow textures. Technical Report CMU-CS-97-138, Carnegie Mellon University, 1997. 9, 10, 17, 18

26. J.-C. Hourcade and A. Nicolas. Algorithms for antialiased cast shadows. *Computers & Graphics*, 9(3):259–265, 1985. 7, 13

27. Geoffre S. Hubona, Philip N. Wheeler, Gregory W. Shirah, and Matthew Brandt. The role of object shadows in promoting 3D visualization. *ACM Transactions on Computer-Human Interaction*, 6(3):214–242, 1999. 1, 2

28. M. Isard, M. Shand, and A. Heirich. Distributed rendering of interactive soft shadows. In *4th Eurographics Workshop on Parallel Graphics and Visualization*, pages 71–76. Eurographics Association, 2002. 10, 17

29. Brett Keating and Nelson Max. Shadow penumbras for complex objects by depth-dependent filtering of multi-layer depth images. In *Rendering Techniques 1999 (10th Eurographics Workshop on Rendering)*, pages 205–220. Springer-Verlag, 1999. 10

30. Daniel Kersten, Pascal Mamassian, and David C. Knill. Moving cast shadows and the perception of relative depth. Technical Report n^o 6, Max-Planck-Institut fuer biologische Kybernetik, 1994. 1, 2

31. Daniel Kersten, Pascal Mamassian, and David C. Knill. Moving cast shadows and the perception of relative depth. *Perception*, 26(2):171–192, 1997. 1, 2

32. Mark J. Kilgard. Improving shadows and reflections via the stencil buffer. http://developer.nvidia.com/docs/IO/1348/ATT/stencil.pdf, 1999. 8

33. Florian Kirsch and Juergen Doellner. Real-time soft shadows using a single light sample. *Journal of WSCG (Winter School on Computer Graphics 2003)*, 11(1), 2003. 12, 13, 17, 18

34. David C. Knill, Pascal Mamassian, and Daniel Kersten. Geometry of shadows. *Journal of the Optical Society of America*, 14(12):3216–3232, 1997. 1

35. Johann Heinrich Lambert. *Die freye Perspektive*. 1759. 1, 2

36. Tom Lokovic and Eric Veach. Deep shadow maps. In *Computer Graphics (SIGGRAPH 2000)*, Annual Conference Series, pages 385–392. ACM SIGGRAPH, 2000. 7

37. Céline Loscos and George Drettakis. Interactive high-quality soft shadows in scenes with moving objects. *Computer Graphics Forum (Eurographics 1997)*, 16(3), September 1997. 6

38. Pascal Mamassian, David C. Knill, and Daniel Kersten. The perception of cast shadows. *Trends in Cognitive Sciences*, 2(8):288–295, 1998. 1, 2

39. Michael D. McCool. Shadow volume recontsruction from depth maps. *ACM Transactions on Graphics*, 19(1):1–26, 2000. 8, 18

40. Steve Morein. ATI radeon hyperz technology. In *Graphics Hardware Workshop*, 2000. 6

41. Steven Parker, Peter Shirley, and Brian Smits. Single sample soft shadows. Technical Report UUCS-98-019, Computer Science Department, University of Utah, October 1998. 12, 15

42. William T. Reeves, David H. Salesin, and Robert L. Cook. Rendering antialiased shadows with depth maps. *Computer Graphics (SIGGRAPH 1987)*, 21(4):283–291, 1987. 7

43. Stefan Roettger, Alexander Irion, and Thomas Ertl. Shadow volumes revisited. In *Winter School on Computer Graphics*, 2002. 8

44. Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. *Computer Graphics (SIGGRAPH 1992)*, 26(2):249–252, July 1992. 7

45. Cyril Soler and François X. Sillion. Fast calculation of soft shadow textures using convolution. In *Computer Graphics (SIGGRAPH 1998)*, Annual Conference Series, pages 321–332. ACM SIGGRAPH, 1998. 4, 6, 9, 13, 17, 18

46. Marc Stamminger and George Drettakis. Perspective shadow maps. *ACM Transactions on Graphics (SIGGRAPH 2002)*, 21(3):557–562, 2002. 7

47. Sylvain Vignaud. Real-time soft shadows on geforce class hardware. http://tfpsly.planet-d.net/english/3d/SoftShadows.html, 2003. 14

48. Leonardo Da Vinci. *Codex Urbinas*. 1490. 1, 2

49. Leonard Wanger. The effect of shadow quality on the perception of spatial relationships in computer generated imagery. *Computer Graphics (Interactive 3D Graphics 1992)*, 25(2):39–42, 1992. 1, 2

50. Lance Williams. Casting curved shadows on curved surfaces. *Computer Graphics (SIGGRAPH 1978)*, 12(3):270–274, 1978. 7

51. Andrew Woo. The shadow depth map revisited. In *Graphics Gems III*, pages 338–342. Academic Press, 1992. 7

52. Andrew Woo, Pierre Poulin, and Alain Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, November 1990. 1, 2

53. Chris Wyman and Charles Hansen. Penumbra maps: Approximate soft shadows in real-time. In *Rendering Techniques 2003 (14th Eurographics Symposium on Rendering)*. ACM Press, 2003. 14, 15, 17, 18

54. Zhengming Ying, Min Tang, and Jinxiang Dong. Soft shadow maps for area light by area approximation. In *10th Pacific Conference on Computer Graphics and Applications*, pages 442–443. IEEE, 2002. 9, 11, 17, 18

55. Hansong Zhang. Forward shadow mapping. In *Rendering Techniques 1998 (9th Eurographics Workshop on Rendering)*, pages 131–138. Springer-Verlag, 1998. 8

5 A Real-Time System for Full Body Interaction

[HLS04] Jean-Marc Hasenfratz, Marc Lapierre, François Sillion, "A Real-Time System for Full Body Interaction", Eurographics Virtual Environments, pp. 147-156, 2004

Eurographics Symposium on Virtual Environments (2004)
S. Coquillart, M. Göbel (Editors)

A Real-Time System for Full Body Interaction with Virtual Worlds

J-M. Hasenfratz and M. Lapierre and F. Sillion

ARTIS†, GRAVIR/IMAG - INRIA

Abstract

Real-time video acquisition is becoming a reality with the most recent camera technology. Three-dimensional models can be reconstructed from multiple views using visual hull carving techniques. However the combination of these approaches to obtain a moving 3D model from simultaneous video captures remains a technological challenge. In this paper we demonstrate a complete system architecture allowing the real-time ($\geq$ 30 fps) acquisition and full-body reconstruction of one or several actors, which can then be integrated in a virtual environment. A volume of approximately $2m^3$ is observed with (at least) four video cameras and the video fluxes are processed to obtain a volumetric model of the moving actors. The reconstruction process uses a mixture of pipelined and parallel processing, using N individual PCs for N cameras and a central computer for integration, reconstruction and display. A surface description is obtained using a marching cubes algorithm. We discuss the overall architecture choices, with particular emphasis on the real-time constraint and latency issues, and demonstrate that a software synchronization of the video fluxes is both sufficient and efficient. The ability to reconstruct a full-body model of the actors and any additional props or elements opens the way for very natural interaction techniques using the entire body and real elements manipulated by the user, whose avatar is immersed in a virtual world.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation – Bitmap and framebuffer operations I.3.6 [Computer Graphics]: Methodology and Techniques – Interaction techniques I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism – Virtual reality I.4.8 [Image Processing and Computer Vision]: Scene Analysis – Tracking

1. Introduction

Inserting live-action movement in virtual worlds is a requirement for many applications. In the context of the CYBER project† we focus on the capture of live movements in real time, for the incrustation of the acquired actors and objects in a virtual world. This operation is important for instance in the television industry, for virtual sets and online presentation by a real person; for games, in order to insert a real person in the game world; for education and entertainment, to allow visits and presentation of remote places.

In this paper, we present a real-time system allowing full body interaction with the virtual world. The novelty is a fully real-time system (min. 25 frames per second) providing a 3D model of the body of an actor (and additional objects) which can be used for interaction and other calculations. Such a 3D model is useful to allow fully 3D interaction using body parts, natural gestures or additional props and objects manipulated by the user.

The paper is organized as follows: Section 2 presents the previous work in trying to capture and reconstruct an actor in real-time. In Section 3, we present the hardware and software architectures used in our system. Section 4 is devoted to the real-time reconstruction of the actor. Section 5 proposes particular applications to demonstrate possible interaction with the virtual environment. Finally, results are discussed in Section 6 before concluding and discussing possible future work.

† ARTIS is a research project in the GRAVIR/IMAG laboratory, a joint unit of CNRS, INPG, INRIA and UJF.
† http://www-artis.imag.fr/CYBER

2. Previous Work

In the last few years, several real-time systems have been presented to capture the 3D shape of a dynamic object, typically a person. We discuss these approaches by grouping them according to their main goal: reconstructing a 3D shape (as fast as possible), rendering a 3D shape, or replaying a 3D sequence. We first observe that performance can be characterized by the acquisition rate and the image display rate, possibly two very different values.

Borovikov and Davis[BD00] use a system with 16 groups of 4 cameras (3 black and white and one color) to represent a moving actor by an octree of voxels. With a cluster of 16 PCs they obtain a maximal processing rate of 10Hz. Wu and Matsuyama[WM03] propose a parallel plane-based volume intersection method to parallelize the reconstruction. With a Myrinet network, 9 PCs and 9 cameras they achieve a frame rate of 15Hz.

Different projects aim at a realistic rendering of the 3D reconstructed model. Matusik *et al.* [MBR+00] describe an image-based approach to compute and shade visual hulls from silhouette image data. 640x480 images are produced at a frequency of 8 frames per second by using 4 client computers and a quad-processor PC as server. In this approach, no explicit 3D shape is necessary because all is done in the image space. In [MBM01], Matusik *et al.* present an algorithm creating and rendering an exact polyhedral visual hull. The system runs at 15 fps with 4 cameras and 5 PCs. Li *et al.*[LMS03b, LMS03a] propose an improved hardware acceleration to render visual hulls. No explicit volume is produced, all computation and rendering is done by the GPU. This ingenious approach produces textured images at 84 fps. In [LMS03c], Li *et al.* remove artifacts on the dynamic object by using projective texture mapping. In [LSMS02], the authors use stereo to compensate for some of the inherent drawbacks of the visual hull method, such as inability to reconstruct surface details and concave regions. Matsuyama and Takai[MT02] use 9 cameras and 9 PCs on a Myrinet network to produce a "3D video" permitting a free view point visualization. Goldlücke and Magnor[GM03a, GM03b] use a voxel structure and render the actor by placing suitable textured billboards at the center of each voxel. They obtain a frame rate of 15Hz with 4 cameras, 2 client PCs and one server PC. Another interesting approach, 3D video fragments[WLG04], generates free viewpoint video by using splatting, and takes into account time and spatial coherence between frames instead of regenerating the whole scene. Real-time interaction is demonstrated by Prince[PCF+02] in a more complex setup where the reconstructed body can be seen and manipulated by another user in an Augmented Reality context.

Another step after obtaining the 3D shape of a moving actor is to reconstruct a skeleton of his body and then to fit it to a virtual body model. In [CMSS03, TMSS02], the authors fit the volumetric reconstruction to a humanoid skeleton. Cheung *et al.*[CKBH00] propose to fit ellipsoids on a volumetric reconstruction. Starck and Hilton[SH03] use a stereo approach to reconstruct the voxel shape of an actor, find its skeleton and fit his body to a virtual model. In [TSS02, TLMS03, TCM+03, TCMS03, CTMS03], the authors propose a full system to produce free view-point video of human actors. They depict the 3D shape reconstruction, the body fitting approach, and the texturing of the virtual body.

Because reconstruction of the 3D shape of the dynamic object is time critical, we have compared the different approaches in Table 1. The proposed classification is done by reconstruction algorithm: volumetric (using voxels) or polyhedral. Solutions such as [MBR+00, LMS03b, LMS03a] are not represented in this table because of their implicit reconstruction on the graphic hardware which limits the interaction with the virtual world (see discussion in Section 5).

Fitting the silhouettes to a predefined body model is not a viable approach for our application, first because it is typically restricted to a simple model and does not allow complex clothing movements; second because it places a severe limitation on what can be captured (i.e. a single human body). Instead we want to be able to capture multiple bodies or additional objects.

3. System architecture

3.1. Hardware architecture

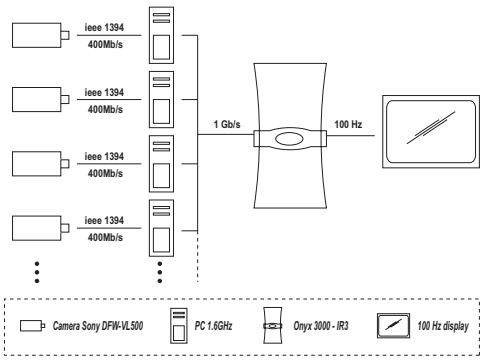


Figure 1: *Hardware configuration of the system.*

Our system is built using a number of independent components and a combination of pipeline and parallel organizations. Our current implementation uses four video cameras but this number could be scaled up as discussed later. Each video camera is connected to a PC, linked to a supercomputer for further processing. The image of the virtual world with the embedded actor is then projected on a screen (Figure 1).

The cameras are standard IEEE 1394 cameras (Sony

Ref	Goal	# PC[1]	Network	# cameras	Time[2] ms
Volumetric					
[BD00]	Reconstruction	16	Ethernet 100	14	100
[CKBH00]	Ellipsoid fitting	5+1	”high speed hub”	5	40
[MT02]	Editing	9	Myrinet 1.28GBits/s	9	64
[WM03]	Reconstruction	6 to 10	Myrinet	6	40 to 23
[TLMS03]	Skeleton + Playback	3+1	Ethernet 100	6	100 to 125
[GM03a, GM03b]	Rendering	2+1	Ethernet 100	4	66
Polyhedral					
[MBM01]	Rendering	4+1	100MBit/s	4	66
[TLMS03]	Skeleton + Playback	3+1	Ethernet 100	6	40

Table 1: *Comparison of recent systems using real-time 3D object reconstruction for different goals. (1) Notation C+S corresponds to C PC clients and S PC server. (2) ”Time” corresponds only to the reconstruction time.*

DFW–VL500) running in the YUV4:2:2 mode at a resolution of 640x480 pixels. The PCs are Pentium4-class running at 1.6 GHz. The link between the PCs and the SGI Onyx 3000–IR3 supercomputer is a 1Gbit/sec Ethernet network. The Onyx is configured to use 8 R12000 processors running at 400Mhz. The output of the Onyx is a standard video projector or a 100Hz display screen.

Figure 2: *Experimental setup. Cameras and actor capture region are surrounded by red circles.*

The actor can move freely inside a volume of approximately 2m cubed. We decided not to use a blue background to retain maximal freedom in the setup, and allow for a transportable system, but rather we use a controlled lighting environment with a grid of fluorescent lights (see Figure 2). The cameras are not externally synchronized to allow capture at their maximum rate, a software mechanism is used to ensure the synchronization of captured images (see Section 4.2.3).

3.2. Software architecture

The PCs are running Windows 2000. An IEEE 1394 library developed at Carnegie Mellon University[UBNN] is used to drive the cameras and the OpenCV[DHF+] library is used for camera calibration.

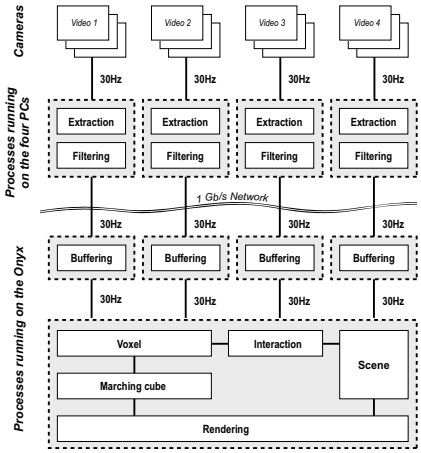


Figure 3: *The different modules involved (dashed rectangles correspond to processes.)*

The whole system comprises nine processes running in parallel (see Figure 3). Four identical processes run on the PCs (one on each): they repeatedly wait for a complete image sent by the camera, acquire it, perform background subtraction and filtering, and send the result to the Onyx by the network via an UDP port (see pseudocode bellow). As we shall see in Section 6 a constant sustained data flow of 30Hz is maintained.

```
// Pseudocode running on each PC
while(true) {
  AcquireImage(*img);
  BackgroundExtraction(*img, *BWBitmap);
  Filter(*BWBitmap);
```

```
  sendUDP(*BWBitmap);
}
```

Five processes run in parallel on the Onyx. Four of them are buffering silhouette images received through the network. Silhouettes are overwritten as they are received, allowing for a good synchronization as discussed in Section 4.2.3. The ninth process runs an infinite loop: it first sets a mutual exclusion to take data at the same time from each buffer (see Pseudocode bellow). The process then fills a voxel area (see Section 4.2.1), applies the marching cubes algorithm to smooth the geometry of the actor (see Section 4.2.2), tests for possible interaction and performs the integration with the virtual world (see Section 5).

```
// Pseudocode of the main process run-
ning on the Onyx
while(true) {
  Mutex.lock();
  ReadAllBuffers(# of camera, *buff);
  Mutex.unlock();
  updateVoxelArea(buff, *voxel);
  updateInteractionData(voxel);
  processMarchingCube(voxel, *polygons)
  render(polygons);
}
```

4. Reconstruction

The reconstruction of the actor model consists of finding the silhouette of the actor viewed by each camera and using these to estimate the 3D shape of the full body. We describe these steps and discuss the issue of synchronization in order to obtain a consistent reconstruction.

4.1. Silhouette Extraction

Cameras must first be calibrated. Estimating cameras parameters from coplanar points with known positions is a well known problem, see [Tsa86, Zha00] for instance. In practice, we use a large (1m) checkerboard pattern and the OpenCV library [DHF+] to calibrate the cameras.

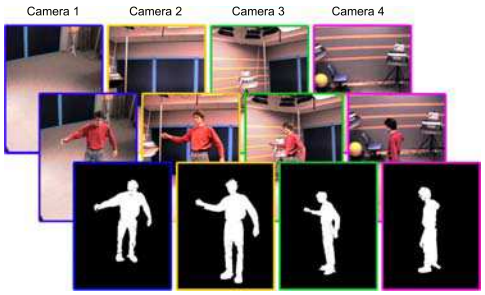


Figure 4: *Background subtraction.*

In our system, the background is static and the actor moves. To determine the silhouette viewed by each camera, we first acquire the background (without the actor) and when the actor is in the field of view of a camera we detect pixels whose value has changed. Indeed a background pixel value should be constant over time while a foreground pixel value can vary. Following this principle, several approaches exist that check the intensity functions at each pixel. Robust ones use temporal filters to detect changes as well as spatial filters to cluster pixels and eliminate false detection (see [TKBM99] for a comparative study). However, in our context the critical issue is not robustness but rather speed. Furthermore, even if some artifacts exist in an image, they are unlikely to be consistent in the whole set of images, and will thus be removed by the reconstruction step. Therefore we use a simple but fast method. Background pixels are assumed to follow Gaussian distributions, possibly correlated, in YUV space. Such distributions are learned *a priori* from a set of background images. The fact that a pixel belongs to the silhouette or the background is then checked by thresholding its Mahalanobis distance to the background mean position in the YUV space. Note that in order to take into account shadows during the subtraction, constraints on the intensity (the Y parameter) values could easily be relaxed. As a result, we obtain a flow of black and white pictures representing the silhouettes as viewed by the different cameras (see Figure 4). This operation takes an average time of 22ms per image, fully consistent with our 30 fps acquisition rate.

4.2. Shape reconstruction

The shape that can be estimated from the different silhouettes is the visual hull[Lau94] of the objects under construction. The visual hull is in fact the maximal solid shape consistent with the object silhouettes. Several approaches have been proposed to compute this visual hull, which we group into the following two categories: surface based approaches and volume based approaches.

Surface based approaches [CG99, MBM01] are not well suited to our application primarily because of the complexity of the underlying geometric calculations. Incomplete or corrupted surface models can be created, and ill-shaped polygons can produce rendering artifacts.

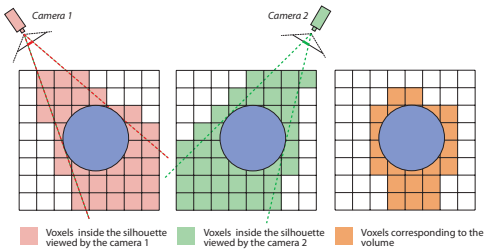


Figure 5: *Principle of voxel carving with two cameras.*

Space carving approaches[SCMS01, Dye01] operate on a discrete space (typically a voxel cube) and mark each space element according to its projection in the images from different viewpoints. Voxels that project outside the object's silhouette in one of the images cannot belong to the object (see Figure 5). These techniques are quite robust, easy to implement, and guaranteed to produce an approximation to the result, commensurate with the chosen resolution. Further as discussed below they can be accelerated using graphics hardware.

4.2.1. Hardware-assisted voxel reconstruction

We observe that the voxel carving approach is essentially a boolean operation on a number of silhouette volumes, computing their intersection. Such boolean operations can be computed on images during a texture mapping step by graphics hardware. The classical N^3 voxel cube is considered as a stack of N images (of resolution N^2), and the stacking direction is chosen to be the closest to the optical axes of the four cameras. The silhouette image for each camera view is projected on each of these slices using the proper perspective projection as texture transform [Lok01]. Note that by using a silhouette image resolution greater than the voxel cube resolution, together with texture filtering, a continuous gray-level image is obtained in each stack. The appropriate blending mode is used to compute the logical AND operation of the four camera views at each slice (Figure 6). The result of this operation is a voxel cube where each non-zero pixel in an image corresponds to a full voxel. This hardware-assisted voxel carving approach is described in more detail in [HLGB03].

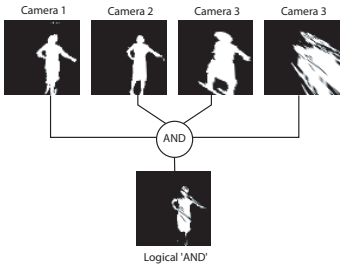


Figure 6: *Logical* AND *operation on four textures projected to the same slice.*

4.2.2. Surface generation

Viewing the reconstructed geometry as points or as cubes is not satisfactory for most applications (Figure 8, left). We apply a "marching cubes" algorithm to obtain a smooth, continuous surface while maintaining real-time.

The marching cubes process allows a good rendering but becomes costly if applied to the entire voxel space. We have tested different approaches to accelerate this process by reducing the number of parsed voxels: the scene can be limited to a bounding box, or divided into hierarchical/regular sub-regions, or into "Bounding Slices" (per-slice bounding boxes). This information can be computed at low cost while transferring graphics card results to our data structure. Best results were obtained with Bounding Slices (see Figure 7), which in most cases permit to only parse between 10% and 14% of the voxel space depending on the scene complexity. This saving accelerates the marching cubes process, but also filtering and collision detection: the smooth surface generation then can be performed in 9ms, instead of more than 30ms when operating on the whole voxel space.

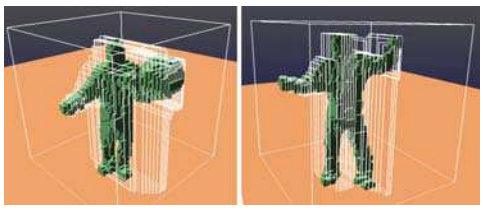


Figure 7: *Bounding box of each slice of the voxel region (the entire voxel region and the minimal bounding box are outlined in white).*

As mentioned before, we retrieve silhouettes as black and white images. If we simply transfer them as binary values in our voxel data structure, the marching cubes algorithm only produces 45 degrees oriented facets (Figure 8, center).

Applying a filter on voxel values before generating the marching cubes allows us to obtain a smooth geometry. This filtering can be done by using the "Imaging Subset" facilities of OpenGL: before reading pixels, we can activate a separable convolution filter (`glSeparableFilter2D`). However we observed that since we only work on a small part of the voxel space, better performance is obtained using software filtering (less than 1ms instead of 3ms). We have tested different filters, the best results being obtained with a simple gaussian filter ($[0.25, 0.5, 0.25]$ kernel).

Now that we have continuous values, we can select a threshold for the marching cubes, so that the isosurface geometry can be generated at a tunable position around the voxels. This threshold choice will act as a dilation (large values) or an erosion (small values) 3D filter. Its value depends on the geometry that is reconstructed. We generally use the default value of 0.5 to correctly smooth geometry without making thin objects disappear (Figure 8, right). As a result we obtain our reconstructed geometry with a smooth surface in real-time.

4.2.3. Synchronization

In theory, synchronization of the cameras should be crucial to obtain an exact reconstruction: if images are not acquired

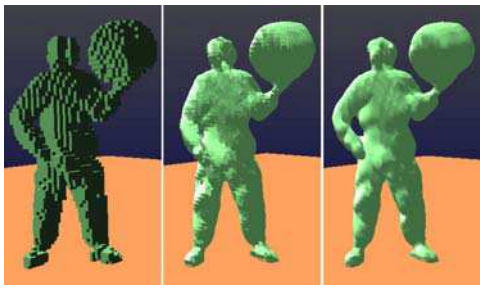


Figure 8: *From left to right: 3D object representation by cubes, pure marching cubes surface, and smoothed surface*

exactly at the same time the voxel carving approach would only reconstruct a part of the object (see Figure 9).

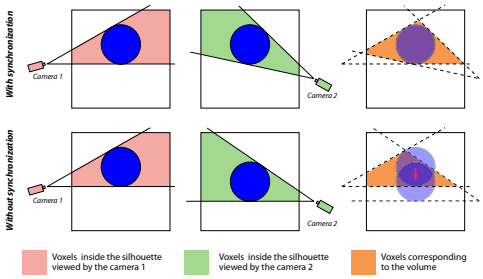


Figure 9: *In the upper row, the two cameras are synchronized, in the lower row, the blue ball had time to move between the two image captures, the final reconstruction (on the right) is only partial.*

To maintain real-time, we must have 30 silhouettes per second. Our cameras nonsustain this rate, but are limited to 15Hz in triggered mode. To bypass this limitation we implemented a software synchronization upon *reception* of the silhouettes (see Section 3.2), which lets us obtain the smoothest possible reconstruction. When shooting a scene, we can only be sure that the worst time shift between any two frames is strictly less than 33ms (see Figure 10). Should this lack of hardware synchronization lead to reconstruction inconsistencies? When the scene is almost static there is obviously no problem, but we could imagine artifacts would occur in the case of fast moving objects. In fact, we did not observe such problems, and instead always obtain realistic results: if we take into account the camera exposure time (independent of framerate), a moving object produces motion blur and thus generates a larger silhouette than its real size. If cameras were synchronized, the 3D reconstructed geometry would be much larger than the original object. Since our silhouettes are slightly time shifted, and only synchronized when used for reconstruction, we finally obtain a realistic

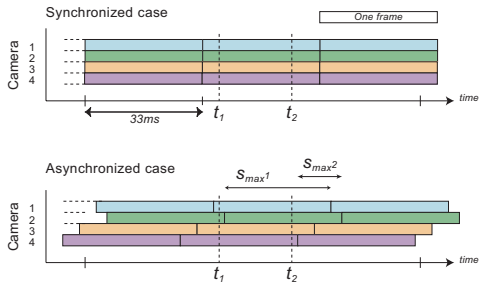


Figure 10: *Best and worst case of synchronization. Let D_i be the delay between t_k and the last frame received from camera i ($0 \leq D_i < 33$). The time shift $S_{i,j}$ between cameras i and j is $\|D_i - D_j\|$. In synchronize case all silhouettes are received at the same time: $S_{i,j} = 0ms$ for any t_k. In asynchronous case, frames are shifted. At t_1 the maximum shift is $S_{max^1} = S_{1,2} \approx 30ms$, at t_2 the maximum shift is $S_{max} = S_{2,4} \ll 33ms$: S_{max^k} is always smaller than 33ms.*

shape, which can be seen as the ”3D average” of the moving object. Finally this configuration allows us to obtain both fluid and consistent reconstructed geometry.

5. Interaction

Our interaction model is quite simple, based on “active” regions within the voxel space. Actions are triggered when a tunable percentage of these regions is filled by voxels, which is both a fast and robust test. There are no constraint on the shape of these regions, and the volume approach lets us use any body parts, or objects manipulated by the actor, to perform actions. On the other hand, of course, we cannot detect which part of the body (arm, foot, etc.) has triggered actions. Nevertheless it allows the actor to use 3D regions as buttons (which can be switched when voxels enter/exit the region as in Figure 11), or even 3D sliders whose value can be interpolated according to the filled voxels. Dynamic virtual objects can also react according to the actor’s position. For example falling balls will correctly bounce off the reconstructed body of the actor (or other objects), since we have normal vectors at each point of its geometry (Figure 12). These interactions are made possible thanks to the combination of different factors: instant feedback (low latency) and immersion sensation.
More results about experiments are described in next section.

Low latency The time between capture and rendering is kept minimal. We have measured that this delay is constant and does not exceed 4 frames: one is due to the transfer from the IEEE cameras to their PC, and the other 3 are due to network transfer, CPU (background subtraction, filtering,

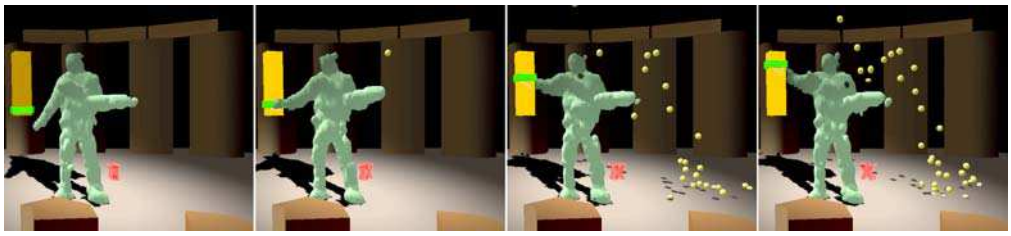


Figure 12: *Example of interaction with balls. Moving the slider on the left changes the flow of balls. Balls are bouncing according to normals of the body or objects.*

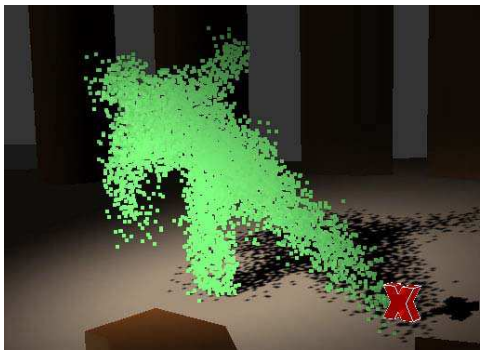


Figure 11: *Example of interaction with an on/off button. Touching the red cross starts the "disintegration" process of the actor!*

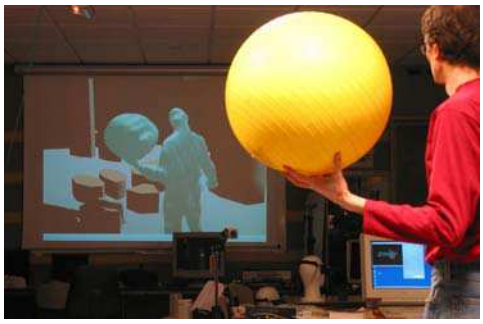


Figure 13: *The actor can visualize himself in the virtual world on a projected screen*

marching cubes) and GPU (conversion to texture, graphical reconstruction and geometry rendering) processing.

Feedback This small latency allows a very comfortable feedback: the user can see him/herself on a projected screen (see Figure 13) and react in real-time within the virtual world. To improve visual feedback we added projected shadows that give important cues for the relative position of the actor with surrounding objects in the virtual scene (see [HLHS03]). Feedback can also be auditive, as some actions generate sound, and allow for example playing on a virtual synthesizer.

6. Results and discussion

6.1. Real-time

As shown in the timeline of Figure 14 we maintain an output image at video framerate and with low latency: treating new silhouettes is done in less than 2ms, transforming them to voxels in 13ms, and conversion to smooth surface and interaction in less than 15 ms: we generate images as fast as silhouettes are coming with no loss. The performance of the last step varies according to scene complexity, for both marching cubes step and most of rendering. Note that body geometry is generated at each frame and can not be memorized on the graphics card as a display list, unlike the others elements of the virtual world.

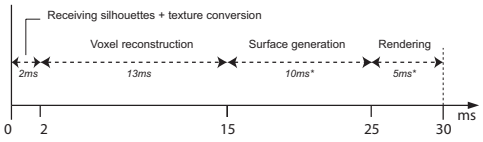


Figure 14: *Time line of the different processes. (*) time spent in surface generation and rendering depends on the complexity of the surface – shown value are typical bounds.*

6.2. Interaction

We have shown in Section 5 that these results allow real-time interaction in complex dynamic virtual scenes. Note that the use of cameras, as opposed to body trackers, allows any person (or object) to enter the scene at any time without any special apparatus, and be instantly embedded in the virtual

world.
Different experiments were performed with external users during demo sessions, on several platforms. In all cases the user can watch his avatar inserted in a virtual scene on a 2m-wide screen.
The first experiment is a virtual synthesizer where a user can play music by hitting 3D virtual coloured parallepipeds: in this case feedback is both auditive and visual. After a few seconds the user understands where he takes place within the virtual world and can play easily a desired melody, with his arms and feet. The impression is quite good, but only a lack of touch sense is felt by the users when hitting an active zone.
An other experimental application was developped around possibilities to create a virtual appartment, where the user can change furniture via a 3D menu, select its colour (by hitting corresponding colour boxes), and choose lighting and music environments. Time needed to feel immersed was also quite short and users could define their environment in less than two minutes.
The global feeling of users is that it is a funny and potentially powerful device: for the first time they could interact with their whole body in 3D (instead of moving their hand in 2D like with a mouse), with no constraints (no apparatus) and at high speed. The restricted zone of capture (limited to 2m) was not perceived as a strong constraint.
To conclude on interaction possibilities we observe that the immersion feeling is easily perceived by users, by quickly identifying them to their avatar even if not realistic (no textures, simple lighting effects). This plaform is therefore adapted to any application not requiring a very high precision but fast and pleasant feedback, and usable without any training.

6.3. Scalability

It is particularly interesting to study the scalability of our system, given the fast-moving pace of technology progress. We briefly discuss here the addition of more cameras, and resolution changes for the voxel space or the output image.

Number of cameras: Using more cameras will improve the carving process. Adding cameras would increase the network traffic, increase reconstruction time, but reduce rendering time (since each new silhouette removes voxels, less geometry should be processed).

- Network overhead: each silhouette is only 10Kb, thus it produces a flow of 300Kb/s. The network - and the Ethernet cards - which can transfer 125Mb/s are far from being saturated.

- Receiving overhead: the processes on the server that are receiving silhouettes all work in parallel and are not very time consuming: adding a few cameras would not slow down this step.

- Reconstruction overhead: as seen in Section 4.2.1, reconstruction is performed on the graphics card. The time to transfer graphics card results to memory is constant. Thus the only additional cost for using a new camera is to convert the silhouette to texture and to render n (voxel size) quads of $n.n$ pixels with this texture projected on it. This step takes under 2ms for each camera.

Thus adding new cameras is not a major bottleneck, adding about 2ms per cameras to our timeline.

Output image resolution Since we are working with standard 3D geometry, there is no specific limitation on the size of the output image. In fact its size hardly changes the rendering times: 4ms are needed to render a reconstructed scene in a 512x512 output window, and 5ms in a 1280x1024 window. Furthermore, as we only use standard graphics card facilities for reconstruction, we can take advantage of its additional features for adding effects like complex shadows to the final rendering (Figure 11 and 12).

Voxel space size The main bottleneck in our configuration is the time spent transferring the final silhouette rendering (corresponding to the results of the intersection of silhouettes) from the graphics card to main memory. This operation is quite slow (80Mpixels/sec) and would slow down the whole process if working with a higher resolution voxel space.

7. Conclusions and future work

We have presented a complete solution that allows one or several live actors to be reconstructed and embedded in a virtual scene at video framerate with reduced latency. Thanks to the availability of 3D information, virtual objects can be interactively manipulated, real-time lighting computation can be performed, and the whole body can be used to generate a new kind of performance. We plan to combine this technique with real textures as it will surely improve the immersion feeling, and body parts recognition would allow more advanced interactions. We can also now imagine sharing the same virtual world with other actors by connecting this system with distant ones, or combining it with augmented reality setups. We also plan to investigate if it could be possible to replace the Onyx server by an on-the-shelf PC, that should obtain equivalent performance, and would allow more up-to-date graphics effects.

References

[BD00] E. Borovikov and L. Davis. A distributed system for real-time volume reconstruction. In *Computer Architectures for Machine Perception*, pages 183–190, 2000.

[CG99] R. Cipolla and P.J. Giblin. *Visual Motion of Curves and Surfaces*. Cambridge University Press, 1999.

[CKBH00] Kong Man Cheung, Takeo Kanade, J.-Y. Bouguet, and M. Holler. A real time system for robust 3d voxel reconstruction of human motions. *Proceedings of the 2000 IEEE Conference on Computer Vision and Pattern Recognition*, 2:714–720, June 2000.

[CMSS03] C.Theobalt, M. Magnor, P. Schueler, and H.P. Seidel. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. *International Journal of Image and Graphics - Special issue on Combining Images amd Graphics*, 2003.

[CTMS03] J. Carranza, C. Theobalt, M Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *ACM Trans. on Computer Graphics*, 22(3), July 2003.

[DHF+] Bob Davies, Chris Halsall, Frank Fritze, Gen-Nan Chen, Valery Mosyagin, Dr. Neurosurgus, Stelian Persa, Victor Eruhimov, Sergey Molinov, and Vadim Pisarevsky. Open computer vision library. http://sourceforge.net/projects/opencvlibrary/.

[Dye01] Charles Dyer. Volumetric scene reconstruction from multiple views. In L. S. Davis, editor, *Foundations of Image Understanding*, pages 469–489. Kluwer, 2001.

[GM03a] B. Goldlücke and M. Magnor. Real-time, free-viewpoint video rendering from volumetric geometry. *Proc. SPIE Conference on Visual Communications and Image Processing*, 5150(2):1152–1158, June 2003.

[GM03b] B. Goldlücke and M. Magnor. Real-time microfacet billboarding for free-viewpoint video rendering. *Proc. IEEE International Conference on Image Processing*, 3:713–716, September 2003.

[HLGB03] Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, and Edmond Boyer. Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics*, pages 49–56. Eurographics, Elsevier, 2003.

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, and François Sillion. A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4), 2003. State-of-the-Art Report.

[Lau94] A. Laurentini. The visual hull concept for silhouette-based image understanding. *PAMI*, 16(2):150–162, February 1994.

[LMS03a] Ming Li, Marcus Magnor, and Hans-Peter Seidel. Hardware-accelerated visual hull reconstruction and rendering. In *Proceedings of Graphics Interface*, pages 65–71, June 2003.

[LMS03b] Ming Li, Marcus Magnor, and Hans-Peter Seidel. Improved hardware-accelerated visual hull rendering. In *Proceedings of Vision, Modeling, and Visualization*, pages 151–158, Nov 2003.

[LMS03c] Ming Li, Marcus Magnor, and Hans-Peter Seidel. Online accelerated rendering of visual hulls in real scenes. *WSCG 2003*, 2(11):290–297, Feb 2003.

[Lok01] Benjamin Lok. Online model reconstruction for interactive virtual environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 69–72. ACM Press, 2001.

[LSMS02] Ming Li, Hartmut Schirmacher, Marcus Magnor, and Hans-Peter Seidel. Combining stereo and visual hull information for on-line reconstruction and rendering of dynamic scenes. In *Proc. 5th Conf. on Multimedia Signal Processing*, 2002.

[MBM01] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Eurographics Workshop on Rendering*, pages 115–126, 2001.

[MBR+00] Wojciech Matusik, Chris Buehler, Ramesh Raskar, Steven J. Gortler, and Leonard McMillan. Image-based visual hulls. In *Siggraph 2000, Computer Graphics Proceedings*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.

[MT02] Takashi Matsuyama and Takeshi Takai. Generation, visualization, and editing of 3d video. In *1st International Symposium of 3D Data Processing Visualization and Transmission*, pages 234–245, 2002.

[PCF+02] Simon Prince, Adrian David Cheok, Farzam Farbiz, Todd Williamson, Nik Johnson, Mark Billinghurst, and Hirokazu Kato. 3-d live: real time interaction for mixed reality. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, pages 364–371. ACM Press, 2002.

[SCMS01] G. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafe. A survey of methods for volumetric scene reconstruction from photographs. In *International Workshop on Volume Graphics*, 2001.

[SH03] J. Starck and A. Hilton. Towards a 3d virtual studio for human appearance capture. In *Vision, Video and Graphics*, pages 17–24. Eurographics, Elsevier, 2003.

[TCM+03] C. Theobalt, J. Carranza, M. Magnor, J. Lang, and H.-P. Seidel. Enhancing silhouette-based human motion capture with 3d motion fields. *Proc. IEEE Pacific Graphics 2003*, pages 185–193, Oct 2003.

[TCMS03] C. Theobalt, J. Carranza, M. Magnor, and H.-P. Seidel. A parallel framework for silhouette-based human motion capture. *Proc. Vision, Modeling, and Visualization*, pages 207–214, Nov 2003.

[TKBM99] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and Practice of Background Maintenance. In *iccv99*, pages 255–261, 1999.

[TLMS03] C. Theobalt, M. Li, M Magnor, and H.P. Seidel. A flexible and versatile studio for synchronized multi-view video recording. In *Vision, Video and Graphics*, pages 9–16, 2003.

[TMSS02] C. Theobalt, M. Magnor, P. Schueler, and H.P. Seidel. Multi-layer skeleton fitting for online human motion capture. In *7th International Fall Workshop on Vision, Modeling and Visualization*, pages 471–478, 2002.

[Tsa86] Roger Y. Tsai. An Efficient and Accurate Camera Calibration Technique for 3d Machine Vision. In *Proceedings of IEEE Coonference on Computer Vision and Pattern Recognition*, pages 364–374, 1986.

[TSS02] Christian Theobalt, Marcus Magnor Pascal Shüler, and Hans-Peter Seidel. Combining 2d feature tracking and volume reconstruction for online video-based human motion capture. In *Proc. IEEE Pacific Graphics 2002*, pages 96–103, 2002.

[UBNN] Iwan Ulrich, Christopher Baker, Bart Nabbe, and Illah Nourbakhsh. Ieee-1394 digital camera windows driver. http://www-2.cs.cmu.edu/∼iwan/1394/.

[WLG04] S. Würmlin, E. Lamboray, and M. Gross. 3d video fragments: Dynamic point samples for real-time free-viewpoint video. In *Computers & Graphics*, pages 3–14, 2004.

[WM03] X. Wu and T. Matsuyama. Real-time active 3d shape reconstruction for 3d video. In *Proc. of 3rd International Symposium on Image and Signal Processing and Analysis*, pages 186–191, 2003.

[Zha00] Zhengyou Zhang. A Flexible New Technique for Camera Calibration. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 22(11):1330–1334, 2000.

6 Omnidirectional texturing of human actors from multiple view video seq.

[OH05] Alexandrina Orzan, Jean-Marc Hasenfratz, "Omnidirectional texturing of human actors from multiple view video sequences", Proceedings of the Conference on Computer-Human Interaction, Sept. 2005

Omnidirectional texturing of human actors from multiple view video sequences

Alexandrina Orzan[*], Jean-Marc Hasenfratz[†]

Artis[‡], GRAVIR/IMAG - INRIA

Abstract

In 3D video, recorded object behaviors can be observed from any viewpoint, because the 3D video registers the object's 3D shape and color. However, the real-world views are limited to the views from a number of cameras, so only a coarse model of the object can be recovered in real-time. It becomes then necessary to judiciously texture the object with images recovered from the cameras. One of the problems in multi-texturing is to decide what portion of the 3D model is visible from what camera. We propose a texture-mapping algorithm that tries to bypass the problem of exactly deciding if a point is visible or not from a certain camera. Given more than two color values for each pixel, a statistical test allows to exclude outlying color data before blending.

1 Introduction

Currently, visual media such as television and motion pictures only present a 2D impression of the real world. In the last few years, increasingly more research activity has been devoted to investigate 3D video from multiple camera views. The goal is to obtain a free-viewpoint video, where the user is able to watch a scene from an arbitrary viewpoint chosen interactively.

The possible applications are manifold. A free-viewpoint system can increase the visual realism of telepresence technology [1] , thus enabling users in different locations to collaborate in a shared, simulated environment as if they were in the same physical room. Also, special effects used by the movie industry, such as *freeze-and-rotate* camera, would be made accessible to all users.

For free-viewpoint video, a scene is typically captured by N cameras. From the views obtained by the cameras a 3D video object, with its shape and appearance, is created. The shape can be described by polygon meshes, point samples or voxels. In order to make the model more realistic, appearance is typically described by the textures captured from the video streams. Appearance data is mapped onto the 3D shape, thus completing the virtual representation of the real object. The 3D video object can be seamlessly blended into existing content, where it can be interactively viewed from different directions, or under different illumination.

Since people are central to most visual media content, research has been dedicated in particular to the extraction and reconstruction of human actors.However, the system used in this article is not restricted to human actors, as [2]. Moreover, it allows the acquisition of multiple objects present in the scene.

The rest of the paper proceeds with a review of related work in section 2. Section 3 will be dedicated to describing the proposed method of texture-mapping, after which results and future tasks are discussed.

2 Previous Work

Over the last few years, several systems with different model reconstruction and different ways of texturing the 3D model have been proposed.

[*]ENS de Cachan - France

[†]University Pierre Mendès France - Grenoble II

[‡]Artis is a team of the GRAVIR/IMAG laboratory, a joint research unit of CNRS, INPG, INRIA, UJF

[1]"Telepresence technology" enables people to feel as if they are actually present in a different place or time (S. Fisher & B. Laurel, 1991) or enables objects from a different place to feel as if they are actually present (T. Lacey & W. Chapin, 1994).

2.1 3D Model reconstruction

Two different approaches of model reconstruction have been studied in the recent years: model-free and model-based reconstruction.

Model-free reconstruction makes no a priori assumptions on scene geometry, allowing the reconstruction of complex dynamic scenes. In human modeling it allows the reproduction of detailed dynamics for hair and loose clothing.

Most model-free methods aim to estimate the visual hull, an approximate shell that envelopes the true geometry of the object [10]. To achieve this, object silhouettes are extracted from each camera image by detecting the pixels not belonging to the background.

The visual hull can then be reconstructed either by voxel-based or polyhedron-based approaches. The first approach discretizes a confined 3D space in voxels and carves away those voxels whose projection fall outside the silhouette of any reference view [7]. Polyhedron-based approaches represent each visual cone as a polyhedral object and computes the intersection of all visual cones [11, 14, 13].

The visual hull allows real-time reconstruction and rendering, yet it needs a large number of views to accurately represent a scene, otherwise the obtained model is not very exact.

Model-based reconstruction assumes that the real object is a human body and uses a generic humanoid model, which is deformed to fit the observed silhouettes [2, 8, 9]. Although it results in a more accurate model and permits motion tracking over time, this approach is restricted to a simple model and does not allow complex clothing movements. Moreover, it places a severe limitation on what can be captured (i.e. a single human body) and it is not real-time.

In this paper, the 3D model used is the one created in the context of CYBER-II project[2], a polyhedron-based model obtained in real-time.

2.2 Multi-view texture mapping

Original images from multiple viewpoint are often mapped onto recovered 3D geometry in order to achieve realistic rendering results [3]. Proposed methods for multi-texture mapping are either view-dependent or view-independent.

View-dependent texture mapping considers only the camera views closest to the current viewpoint. In between camera views, two to four textures are blended together in order to obtain the current view image [4, 5]. This method exhibits noticeable blending artifacts in parts where the model geometry does not exactly correspond to the observed shape. What's more, the result is usually blurred and the passing from one camera view to another does not always go unnoticed.

View-independent texture mapping selects the most appropriate camera for each triangle of the 3D model, independently of the viewer's viewpoint [2, 8, 13]. The advantage of this method is that it does not change the triangle texture when the user changes the viewpoint. Moreover, the blurred effect is less noticeable. However, the problem is that the best camera is not the same from patch to patch, even if they are neighboring. Here also, blending between visible views is necessary in order to reduce the abrupt change in texture at triangle edges.

Blending is done using various formulas that depend of:

- the angle between the surface normal and the vector towards the considered camera

- the angle between the surface normal and the vector towards the viewpoint

- the angle the vector towards a camera and the vector towards the viewpoint

Blending weights can be computed per vertex or per polygon [2, 4, 5]. Matsuyama [13] proposes using this method for determining each vertex color and then paints the triangles with the colors obtained by linearly interpolating the RGB values of its vertices. However, for large triangles, small details like creases in the clothes are lost.

Li and Magnor [12] compute the blending for each rasterized fragment, which results in a more accurate blending.

2.3 Visibility

Visibilities with respect to reference views are very important for multi-view texture mapping. For those parts that are invisible in a reference view, the corresponding color information should be ignored when blending multiple textures.

[2] http://artis.imag.fr/Projects/Cyber-II/

Debevec et al. [3] splits the object triangles so that they are either fully visible or fully invisible to any source view. This process takes a long time even for a moderately complex object and is not suitable for real-time applications. Matusik [14] proposes computing the vertex visibility at the same time that the visual hull is generated. Magnor et al. [12] solves the visibility problem per fragment, using shadow mapping. However, they require rendering the scene from each input camera viewpoint and is not real-time even with a hardware-accelerated implementation.

We propose a per pixel method that checks only polygon visibility and eliminates the wrong colors by considering only those colors that are close to a computed average.

3 Texture mapping algorithm

3.1 Model constraints

The polyhedron-based model-free method recreates the geometrical object at each frame. The number of polygons, their form and position in space vary greatly in time, so we cannot track vertices from one frame to another.

This means that it is impossible to decide the color of the polygons only once, at the beginning of the video. Color values have to be computed in real-time, for each frame.

3.2 Algorithm description

To achieve realistic rendering results, we use the projective texture mapping, a method introduced by Segal [15] and included in the OpenGL graphics standard. But the current hardware implementation of projective texture mapping in OpenGL lets the texture pass through the geometry and be mapped onto all back-facing and occluded polygons. Thus it is necessary to perform visibility check so that only polygons visible to a particular camera are texture mapped with the corresponding image.

A point p on the object's surface is visible from a camera c_i if (1) the triangle t_j to which the point belongs faces the camera and (2) the point is not occluded by any other triangles.

The first condition can be fast determined by checking the equation $n_{t_j} \cdot v_{c_i \to t_j} < 0$, where n_{t_j} is the triangle normal vector and $v_{c_i \to t_j}$ is the viewing direction from c_i towards the centroid of t_j.

Still, in a per-pixel approach, we do not have the geometrical data. We solve this problem by an additional rendering of the object from the current viewpoint, where we use the polygon ID as its color. Thus, we can determine what polygons are visible from the viewpoint and exactly which pixel of the current image view belongs to which triangle.

Determining if a point viewed by the viewer is occluded or not to the cameras is a less obvious problem. Methods to determine what points are occluded were briefly presented in the previous section. We propose to bypass the occlusion checking by doing a basic statistical test. The strong condition that has to be fulfilled is that for each pixel at least three cameras have to pass the first visibility test, and the majority has to see the correct color. Still, this is usually the case with a system having an evenly distributed camera configuration.

As all the cameras are calibrated prior to use and the images are acquired at the same time and in the same lighting conditions, we can compare colors and calculate distances in the RGB space [1, 6].

If a sufficient number of color values are available for a pixel, we compute the mean (μ) and the standard deviation (τ) for each of the R, G, B channels. Individual colors falling outside the range $\mu \pm \beta \cdot \tau$ for at least one channel are excluded. The factor β permits us to modify the confidence interval for which the colors are accepted. The classical normal deviation test considers β is 1. We experimentally concluded that it was best to set it at 1.17, to allow for slight errors in manipulating the color-values.

If less than three possible colors are available for a pixel, we do not exclude any of them.

A weighted mean of all contributing images is finally used for texturing each particular pixel. The blending weight is computed using the value of the $\cos(angle(n_{t_j}, v_{c_i \to t_j}))$.

If the pixel is invisible for all cameras, we compute its color using the color values of the neighbours whose color was already decided.

The algorithm runs as follows:

```
1: for all polygons in the 3D model do
2:     check if they are at least partially visible from the
       current view
3: end for
4: for all pixels in the image view do
5:     for all cameras do
6:         if the polygon that colored the pixel faces the
           camera then
7:             retain the corresponding color
8:         end if
9:         if there are three or more colors then
10:            compute the mean and standard deviation
11:            for all colors do
12:                if they are not in the allowed interval
                   then
13:                    exclude
14:                end if
15:            end for
16:            compute the weighted mean
17:        else if there are two colors then
18:            compute the weighted mean
19:        else if there is no color then
20:            compute the color using neighbouring col-
               ors
21:        end if
22:    end for
23: end for
24: draw
```

4 Results

We tested this algorithm with the system used by the CYBER-II project. The system has 6 cameras, 4 in the front and 2 in the back, as seen in Figure 1[3].

For the front views, the algorithm succeeded in eliminating the wrong colors and in seamlessly mixing data from various cameras. Moreover, the pixel color doesn't change with the change of viewpoint. Images comparing view-dependent and view-independent algorithms, without occlusion checking, and our method can be seen in Figure 2.

However, for the back views, where the object is seen by at most 2 cameras, the algorithm does only a weighted average, without color elimination.

[3] video sequences were acquired with the Grimage platform of Inria Rhône-Alpes

Figure 1: Camera setting

Figure 2: a) View dependent, b) View independent, c) Our method

We set the resolution of the rendered novel view to 512x512, and we tested the algorithm for a model of about 5000 polygons. On a Intel 2.40GHz CPU and a GeForce4 Ti 4800 graphic card, the frame rate is of 17 fps.

5 Conclusions and Future work

A per-pixel algorithm for multi-view texture mapping has been implemented. It succeeds in eliminating wrong colors for pixels viewed by more that 2 cameras, without doing a time-consuming occlusion checking.

Yet, further enhancements are both necessary and feasible. Thus, a hardware-implementation should be considered, since the mane time-consuming task in our algorithm is transferring information from the frame-buffer to the CPU. Moreover, we would like to consider a continuity in time of the computed pixel colors and a dynamic deactivation of the unused cameras.

References

[1] A. Agathos and R. Fishe. Colour texture fusion of multiple range images. In *Proceedings of the 4th International Conference on 3-D Digital Imaging and Modeling*, pages 139– 146, 2003.

[2] Joel Carranza, Christian Theobalt, Marcus Magnor, and Hans-Peter Seidel. Free-viewpoint video of human actors. *ACM Trans. on Computer Graphics*, 22(3):569 – 577, July 2003.

[3] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. *Computer Graphics*, 30(Annual Conference Series):11–20, 1996.

[4] Paul E. Debevec, Yizhou Yu, and George D. Borshukov. Efficient view-dependent image-based rendering with projective texture-mapping. In *9th Eurographics Workshop on Rendering*, 1998.

[5] Bastian Goldlücke and Marcus Magnor. Real-time microfacet billboarding for free-viewpoint video rendering. In *Proceedings of ICIP 2003, IEEE Computer Society*, volume 3, pages 713–716, 2003.

[6] L. Grammatikopoulos, I. Kalisperakis, G. Karras, T. Kokkinos, and E. Petsa. On automatic orthoprojection and texture-mapping of 3d surface models. In *ISPRS Congress - Geo-Imagery Bridging Continents*, 2004.

[7] Jean-Marc Hasenfratz, Marc Lapierre, and François Sillion. A real-time system for full body interaction. *Virtual Environments*, pages 147–156, 2004.

[8] Adrian Hilton and Jonathan Starck. Model-based multiple view reconstruction of people. In *IEEE International Conference on Computer Vision*, pages 915–922, 2003.

[9] Adrian Hilton and Jonathan Starck. Multiple view reconstruction of people. In *3D Data Processing, Visualization, and Transmission*, pages 357–364, 2004.

[10] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(2):150–162, 1994.

[11] Ming Li, Marcus Magnor, and Hans-Peter Seidel. Online accelerated rendering of visual hulls in real scenes. In *Journal of WSCG*, 2003.

[12] Ming Li, Marcus Magnor, and Hans-Peter Seidel. A hybrid hardware-accelerated algorithm for high quality rendering of visual hulls. In *Proceedings of the 2004 conference on Graphics interface*, pages 41–48, 2004.

[13] Takashi Matsuyama and Takeshi Takai. Generation, visualization, and editing of 3d video. In *3D Data Processing, Visualization, and Transmission*, page 234, 2002.

[14] Wojciech Matusik, Chris Buehler, and Leonard McMillan. Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 115–126, 2001.

[15] Mark Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadows and lighting effects using texture mapping. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 249–252, 1992.

7 Fast Voxel Carving Using OpenGL Facilities

[HLS05] Jean-Marc Hasenfratz, Marc Lapierre, François Sillion, "Fast Voxel Carving Using OpenGL Facilities", **Rejected** to Journal of Graphics Tools, ?(?), 2005

Journal Of Graphics Tools (2004), pp. 1–5
N.N. and N.N. (Editors)

Fast Voxel Carving using standard OpenGL facilities

Jean-Marc Hasenfratz, Marc Lapierre, François Sillion

Abstract

In this paper we present a fast method to generate voxel data from a set of silhouettes of any dynamic object. Our goal is to generate 3D data, and not only a view of the geometry like in other many papers. The method takes advantage and makes extensive use of standard OpenGL facilities: by only combining projected textures and blending operations we obtain an entire plane of voxels at a time without any precomputation by using the GPU. Our method is much less sensible to fillrate limitations than others methods[Lok01, LMS03] that render reconstructed geometry to scene view. Moreover we'll see that retrieving data from framebuffer can be optimized by combining several slices per pixel.

Since it is based on bitmap silhouettes, the method is robust and easily tunable. The time needed to generate the voxel representation is independant of the scene's geometric complexity. Its speed - a few milliseconds with 4 silhouettes - is adapted for scenes requiring a low latency, such as mixed reality applications.

The generated data can be used for collision detection, and rendered either as a fake or real geometry using common techniques such as splatting, LDIs or marching cubes.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation – Bitmap and framebuffer operations I.4.5 [Image Processing and Computer Vision]: Reconstruction

1. Introduction

More and more applications need to reconstruct dynamic geometry from video data in real-time. The extracted 3D information is vital to allow for a real actor interaction and realistic lighting computation within a virtual scene.

Our algorithm is based on voxel carving, a discretized solution of visual hull[Lau94] construction which can generate a volume from a set of its silhouettes. In this paper we show how to use efficiently basic capabilities of graphic cards (no 3D texture, no stencil buffer nor pixel shaders) to make such computation entirely in image space. We show that reconstruction time is independant of geometric complexity so that the method can be used for time-critical applications.

The remainder of this paper is structured as follows. In section 2, we present previous work on 3D reconstruction, and then discuss our algorithm in section 3. Next we discuss results and limitations of the algorithm using our implementation. Finally we conclude and explore ways to optimize our method.

2. Previous work

The concept of visual hull of an object has been introduced by Laurentini[Lau94] who explains how the closest approximation of a 3D shape can be obtained from its silhouettes.

A very complete survey from Dyer[Dye01] lists methods to generate voxels from images, and in particular shape-from-silhouettes approaches[KR01]. Most of these techniques check the existence of a voxel by back-projecting it into each silhouette. This step may be accelerated by precomputing projections and using octrees[SA90] instead of a regular voxel space. In this case a (very) large memory should be used for lookup tables.

Silhouettes can be used "as is" as bitmaps, or as a 2D shape after being polygonized: either in image space where the volume is discretized into voxels[SBS02], or in 3D space where each silhouette contour is extruded along camera axes[MBR*00, MBM01]. This latter technique gives fast results[LMS03] for low resolution geometry (≈500 polygons).

Another idea to generate a volume from binary data is to use hardware 3D texture functionalities[DKC*98], but this

technique has limitations in terms of bandwith (to transfer silhouettes to GPU), texture memory space and pixel fill rate.
A hardware accelerated reconstruction method[SBS02] based on voxel carving gives good results, as reconstructed geometry is accurate and colored, but it doesn't fit to real-time applications (reconstruction is done in seconds to minutes).

3. Description of the algorithm

The algorithm is based on voxel carving and performs all computation in the frame buffer: silhouettes projection by using projective texturing[HS93], and their intersection by using blending functions. Since we do not retrieve color information, we do not need a stencil buffer (like [Lok01]) and simply take advantage of fast hardware textured geometry rasterization which leads to fast reconstruction.
In the following parts we consider working with n silhouettes (from calibrated cameras) of resolution RxR, and generating a regular N^3 voxel space. The algorithm is divided in 3 steps: acquisition of silhouettes, voxel computation and voxel reading.

3.1. Aquisition of silhouettes

Since silhouettes - retrieved directly or from remote cameras[HLGB03, LMS03] - are binary (Figure 1), they can be converted to standard one component textures at a negligible cost: necessary bandwidth to tranfer them to GPU is only $n.R^2$ bytes per frame. However, in the case of a large number of cameras, texture size issue should be taken into account. Using index textures would not be useful (indices are also encoded in one byte) but texture compression facilities should optimize this step if necessary.

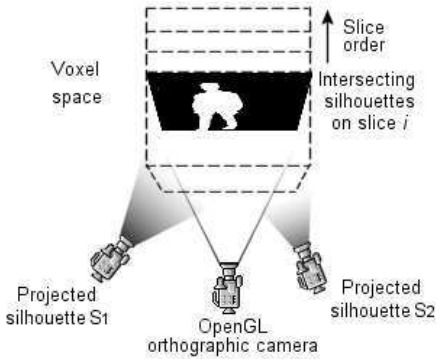


Figure 1: *We combine projections of all binary silhouettes onto each slice of voxel space.*

3.2. Voxel computation

All voxel computation is done in the frame buffer: each silhouette is projected onto each slice of the parsed voxel space (Figure 1) and intersection is computed on the fly thanks to blending operations. Remaining white pixels at the end of this process generate voxels.

3.2.1. Projection

Our voxel space is discretized in N slices of NxN voxels. Let's consider each slice as a NxN pixels quad in which each pixel corresponds to a voxel. The GPU can render a quad at a very high speed, even if it is textured. If a texture could be projected onto this quad according to camera position we would obtain lit voxels slice by slice in a single render. In OpenGL by correctly setting an orthographic camera in front of voxel space and texture matrix from calibration parameters, we can rasterize a quad with silhouette texture projected on it (see Figure 1). We thus obtain on screen the resulting voxels hit by the projection of our silhouette onto a slice.

So the frame buffer now contains - in only one operation per slice - the result of the projection of R^2 pixels of the silhouette onto the quad. Repeating this process N times (rendering N quads) will parse all the voxel space. Instead of rendering these quads in their real place (the ones behind the others), we render them side by side so that we obtain a Nx(NxN) pixels frame containing all the information we want: a "planar" view of our voxel space where each white pixel is a voxel set by its silhouette (Figure 2).

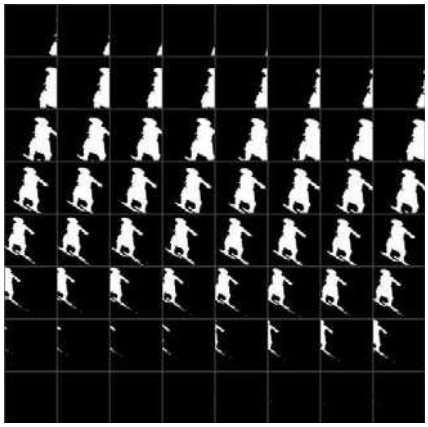


Figure 2: *Result of the projection of one silhouette onto each slice of a* 64^3 *voxel space (i.e. onto 64 quads of* 64^2 *pixels).*

3.2.2. Intersection

Now we know which voxels are set by a silhouette: let us consider the problem of the intersecting projections. This

step can also be done efficiently slice by slice on GPU, not in 3D but in 2D for each slice.
The voxel carving process consists in only keeping voxels wich back project to ***all*** silhouettes. We can express this condition by saying that all pixels projected in black on a slice will turn off previous ones. Thus the remaining white pixels will correspond to the intersection of the projections. This is accomplished using a logical or blending operator: in OpenGL by setting the blend operator to `glBlendFunc(GL_DST_COLOR, GL_ZERO)`, a pixel from the frame buffer will turn black if a new pixel is drawn in black, or otherwise will not change. Obviously in the first pass quads are rendered "as is" without this test. Note that each slice is rendered without depth test to ensure pixel value updates for each projected texture.

For performance reasons, since changing texture is costly it is much faster to draw all the quads for one texture than for each quad projecting all the textures.

3.3. Data reception

Now we have a frame buffer containing all the information concerning the voxels, we just have to retrieve it back to main memory. This can be done fast for a small voxel space, but can become costly for higher resolutions (see Table 1).

Voxel space (N)	16^3	25^3	40^3	64^3	100^3
$\Leftrightarrow$ Pixel block size	64^2	128^2	256^2	512^2	1024^2
Time in ms	0.32	0.47	0.88	2.4	8.2

Table 1. *Time to retrieve data from the frame buffer for different voxel space sizes, i.e. pixel read back performance (measured on sgi Onyx2 IR3).*

Let's see how this step could be optimized in different ways. First, we have to take into account that such a transfer includes a fixed cost; as an example reading a single block of 512x512 pixels is much faster than reading 64 times a block of 64x64 (which is the same amount of pixels). So we avoid reading frame buffer multiple times, this is the reason why in previous step quads are rendered side by side and we only read the whole image once.
Another point is that we can take advantage of the fact that silhouettes are black and white, thus information needed is only a binary value. In the case of RGBA frame buffer the same information is repeated for each component, so we could get all information by reading a single component per pixel. Unfortunately time to read a single component is not four times faster than to read the four (see Table 2). Then the idea is to store voxels not per pixel but per component. By using `glColorMask` function while rendering quads, we mask pixels so that first slice only affects red component, second only green, etc. Thus by reading pixels in such a buffer we obtain values of 4 voxel slices at a time.

Another approach is to work not in RGBA but in color index mode. Reading such a buffer is faster than in RGBA case. Even more, the principle to mask pixels can be done not for each component but for each bit. So in this case by reading a N^2 region in a 12 bits/pixel frame buffer we'll get values of 12 voxel slices by reading one pixel. Time comparison (see Table 2) clearly shows that this latter technique is the most efficient. It only requires a buffer to be created while other method can use back buffer.

	RGBA mode		ColorIndex mode	
Read bytes	Red	RGBA	BYTE	SHORT
Time(ms)	2.4	6.3	2.34	2.41
Mvox/s (1 vox/pix)	109	42	**112**	108
Mvox/s (n vox/pix)	109	168	896	**1300**

Table 2. *Performance without and with multiple voxels/pixel use for retrieving a 512^2 pixels block (in bold, fastest speed for each approach).*

The contiguous buffer containing all voxel data can now be used to fill any data structure, be filtered and processed. The corresponding geometry can then be rendered or used for any specific application such as computing shadows, gesture recognition, movement analysis, collision detection, etc.

4. Results

To test this algorithm we have implementated it into an application that reconstructs a scene from 4 cameras sending 128x128 silhouettes. Performance measures for the complete reconstruction process are given in Table 3. Performance were measured using following buffer optimization: as color index modes are no more available on recent cards, 3 voxels are stored per pixel (on successive R, G and B channel), and buffer is read back in `GL_UNSIGNED_BYTE_3_3_2` mode. Even if the speed-up is less than the one described previously, it allows a gain factor of x2.5 on read back step. Last line gives an example of reachable framerate for a complete reconstruction process, from silhouettes acquisition to point rendering for an actor within a virtual scene. An example of reconstruction is given on Figure 3.

Platform	PIV+GF3ti500		Bi-PIV+FX5600	
Voxel space	64^3	96^3	64^3	96^3
DrawQuads	1.3ms	2.0ms	1.1ms	1.4ms
ReadPixels	4.7ms	13.6ms	5.1ms	12.2ms
Total	10.5ms	27.7ms	8.3ms	19.0ms
Framerate	85Hz	35Hz	**>120Hz**	>55Hz

Table 3. *Performance of whole reconstruction process.*

Note that on a more powerful configuration (Bi-Xeon with GeForce4 ti4800), the framerate varies between 150fps and 180fps (depending on window size) for complete reconstruction process rendered as points.

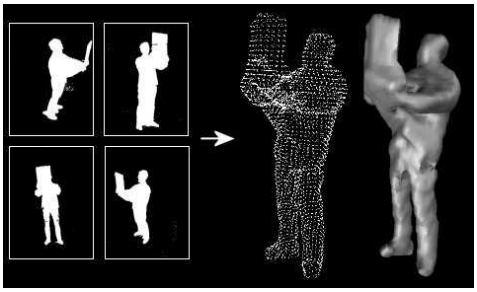


Figure 3: *View of generated voxel space from 4 silhouettes, reconstruction as points and as Marching Cubes.*

4.1. Algorithm parameters

This method is tunable in the way that the size of voxel space can be tuned dynamically: according to desired accuracy or time constraints, the size of our voxel space N can be easily changed by simply resizing the size of the rendered quads (its size does not need to be a power of 2).

4.2. Scalar voxels from binary silhouettes?

With binary values we can only generate "blocky" voxels. To be able to generate a smooth surface it would be interesting to have not binary but scalar values. Let's see how hardware can be used for this process. A great functionality of openGL texturing is that it can render geometry with interpolated textures at the same speed. Thus by specifying linear interpolation for our textures, projected silhouettes now contains (in RGBA mode) scalar values for each pixel: by retrieving these values we can associate scalar values to voxels at no additional cost.
Another idea is to apply a convolution filter onto the rendered quads. Applying such a filter on the frame would give a blurred image which could be used to generate "soft" voxels. Unfortunately this extension is quite costly, and it is much more efficient to do it on CPU specifically on the region containing voxels than doing a brute force filter on the GPU for the whole frame.

5. Conclusions

We have shown that it is possible to obtain 3D information from 2D silhouettes at a cost independant of the geometric complexity, using frame buffer operations, and by only using very basic functionalities of OpenGL. We have taken care to optimize readback, and not to affect other rendering capabilities that could be used on the GPU: stencil buffer is kept untouched (so it can be used for standard effect like shadow volumes, mirrors...) and texture units use is minimal.
Using this technique should easily be distributed: by cutting and distributing image space on a grid of PCs it should linearly save reconstruction time.
The appearance of the new high-speed PCI express bus should improve these results, and more generally lead to re-use this type of interaction between CPU and GPU to other fields.

References

[DKC*98] DACHILLE F., KREEGER K., CHEN B., BITTER I., KAUFMAN A.: High-quality volume rendering using texture mapping hardware. In *Proc. SIGGRAPH '98* (Aug 1998), pp. 69–76. 1

[Dye01] DYER C.: Volumetric scene reconstruction from multiple views. In *Foundations of Image Understanding*, Davis L. S., (Ed.). Kluwer, 2001, pp. 469–489. 1

[HLGB03] HASENFRATZ J.-M., LAPIERRE M., GASCUEL J.-D., BOYER E.: Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics* (2003), Eurographics, Elsevier, pp. 49–56. 2

[HS93] HAEBERLI P., SEGAL M.: Texture mapping as A fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering* (1993), Cohen M. F., Puech C., Sillion F., (Eds.), pp. 259–266. 2

[KR01] KUZU Y., RODEHORST V.: Volumetric modeling using shape from silhouette. In *Fourth Turkish-German Joint Geodetic Days* (2001), pp. 469–476. 1

[Lau94] LAURENTINI A.: The Visual Hull Concept for Silhouette-Based Image Understanding. *pami 16*, 2 (Feb. 1994), 150–162. 1

[LMS03] LI M., MAGNOR M., SEIDEL H.: Online accelerated rendering of visual hulls in real scenes, 2003. 1, 2

[Lok01] LOK B.: Online model reconstruction for interactive virtual environments. In *Proceedings of the 2001 symposium on Interactive 3D graphics* (2001), ACM Press, pp. 69–72. 1, 2

[MBM01] MATUSIK W., BUEHLER C., MCMILLAN L.:

Polyhedral visual hulls for real-time rendering. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 115–126. 1

[MBR*00] MATUSIK W., BUEHLER C., RASKAR R., GORTLER S. J., MCMILLAN L.: Image-based visual hulls. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), ACM Press/Addison-Wesley Publishing Co., pp. 369–374. 1

[SA90] SRIVASTAVA S. K., AHUJA N.: Octree generation from object silhouettes in perspective views. In *Computer Vision, Graphics and Image Processing* (1990), vol. 49, pp. 68–84. 1

[SBS02] SAINZ M., BAGHERZADEH N., SUSIN A.: Hardware accelerated voxel carving. In *Proc. 1st Iberoamerican Symposium in Computer Graphics* (2002), M.P. dos Santos, L. Velho, X. Pueyo edit., pp. 289–297. 1, 2

8 Soft Shadow Maps: Efficient Sampling of Light Source Visibility

[ALHHHS05]Lionel Atty, Marc Lapierre, Nicolas Holzschuch, Chuck Hansen, J-Marc Hasenfratz, Francois Sillion , "Soft Shadow Maps: Efficient Sampling of Light Source Visibility", **Rejected** to Eurographics Symposium on Rendering (2005)

Eurographics Symposium on Rendering (2005), pp. 1–12
Kavita Bala, Philip Dutré (Editors)

Soft Shadow Maps: Efficient Sampling of Light Source Visibility

paper1024

Abstract

Shadows, particularly soft shadows, play an important role in the visual perception of a scene by providing visual cues about the shape and position of objects. Recent results have produced interactive methods for soft shadows but fail to scale well for models with modestly large number of polygons. In this paper, we present a new algorithm for computing interactive soft shadows on the GPU. This novel approach overcomes several limiting factors in existing methods while providing interactive frame-rates for models with tens of thousands of polygons.

Our technique is based on a sampled image of the occluders, as in shadow map techniques. However these shadow samples are used in a novel manner, by computing their effect on a second projective shadow texture using fragment programs. In essence, the fraction of the light source area hidden by each sample is accumulated at each texel position of this soft shadow map. *We discuss the underlying approximation due to the combination of independent samples, and show that it remains very modest in most practical cases.*

Categories and Subject Descriptors (according to ACM CCS): I.3.1 [Computer Graphics]: Graphics processors I.3.7 [Computer Graphics]: Color, shading, shadowing, and texture

1. Introduction

Shadows add important visual information to computer-generated images. The perception of spatial relationships between objects can be altered or enhanced simply by modifying the shadow shape, orientation, or position [WFG92, Wan92, KMK97]. Soft shadows, in particular, provide robust contact cues by the hardening of the shadow due to proximity resulting in a hard shadow upon contact. The advent of powerful graphics hardware on low-cost computers has led to the emergence of many real-time shadow algorithms [HLHS03].

In this paper, we introduce a novel method based on shadow maps to interactively render soft shadows. Our method computes a projective shadow texture that incorporates soft shadows based on light source visibility from receiver objects. This texture is then projected into the scene to provide interactive soft shadows of dynamic objects and dynamic area light sources.

There are several advantages to our technique. First, it is not necessary to compute silhouette edges. Second, the algorithm is not fill-bound, unlike methods based on shadow volumes. These properties provide better scaling for occluding geometry than other GPU based soft shadow techniques [WH03, CD03, AAM03]. Third, unlike some other shadow map based soft shadow techniques, our algorithm does not dramatically overestimate the umbra region [WH03, CD03]. Fourth, while other methods have relied on an interpolation from the umbra to the non-shadowed region to approximate the penumbra for soft shadows [WH03, CD03, BS02], our method computes the visibility of an area light source for receivers in the penumbra regions.

Since our algorithm splits scene geometry into occluders and receivers, self shadowing is not accounted for and must be applied with another method such as standard shadow maps. Since our algorithm uses shadow maps to approximate occluder geometry, it inherits the well known issues with aliasing from shadow map techniques. For large area light sources, the soft shadows tend to blur such artefacts but for smaller area light sources, such aliasing is apparent.

Our method provides the ability to interactively render scenes with soft shadows that have more complex occuders than was possible with previous interactive soft shadow techniques. Our algorithm is particularly useful for applications with dynamic scenes such as computer games or immersive environments. Additionally, these are precisely the type of applications that benefit from the proximity cues of soft shadows.

In the following section, we review previous work on computing soft shadows interactively. Then in section 3, we present our algorithm. In section 4, we present our results. In section 5, we discuss the limitations and costs of our algorithm. In section 6, we present several simple improvements to our algorithm. Finally, in section 7, we conclude and expose possible future directions for research.

2. Previous Work

Researchers have investigated shadow algorithms for computer-generated images for nearly three decades. The reader is refered to a recent state-of-the art report by Hasenfratz *et al.* [HLHS03], the overview by Woo *et al.* [WPF90] and the book by Akenine-Möller and Haines [AMH02].

The two most common methods for interactively producing shadows are shadow maps [Wil78] and shadow volumes [Cro77]. Both of these techniques have been extended for soft shadows. Chan and Durand [CD03] and Wyman and Hansen [WH03] both employed a technique which uses the standard shadow map method for the umbra region and builds a map containing an approximate penumbra region that can be used at run-time to give the appearance, including hard shadows at contact, of soft shadows. While these methods provide interactive rendering, over estimating the umbra can lead to incorrect soft shadows in certain cases such as large area light sources and thin objects. Their methods depend on computing the silhouette edges in object space for each frame and this requirement limits the scalability for occluders with large numbers of polygons.

Brabec and Seidel [BS02] and Kirsch and Doellner [KD03] use a shadow map to compute soft shadows, by searching at each pixel of the shadow map for the nearest boundary pixel, then interpolating between illumination and shadow as a function of the distance between this pixel and the boundary pixel and the distances betwen the light source, the occluder and the receiver. Their algorithm requires scanning the shadow map to look for boundary pixels, a potentially costly step; in practical implementations they limit the search radius, thus limiting the size of the penumbra region.

Soler and Sillion [SS98] compute a soft shadow map as the convolution of two images representing the source and blocker. Their technique is only accurate for planar and parallel objects, although it can be extended using an object hierarchy. Our technique can be seen as an extension of this approach, where the convolution is computed for each sample of an occlusion map, and the results are then combined.

Assarsson and Akenine-Möller [AAM03] used penumbra wedges in a technique based on shadow volumes to produce soft shadows. Their method depends on locating silhouette edges to form the penumbra wedges. While providing good soft shadows without an overestimate of the umbra, the algorithm is fill-limited, particularly when zoomed in on a soft shadow region. Since it is necessary to compute the silhouette edges at every frame, the algorithm also suffers from scalability issues when rendering occluders with large numbers of polygons.

Figure 1: *Applying our algorithm (*200,000 *polygons, occluder map* 256 × 256*, displayed at 32 fps).*

3. Algorithm

3.1. Presentation of the algorithm

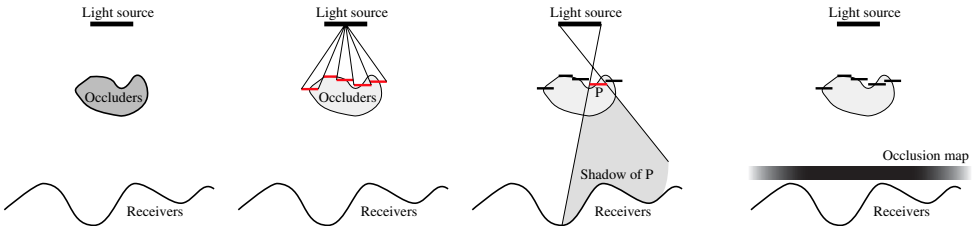
Our algorithm assumes a rectangular light source and starts by separating potential occluders (such as moving characters) from potential receivers (such as the background in a scene) (Fig. 2(a)). We will compute the *soft shadows* only from the occluders onto the receivers.

Our algorithm computes a soft shadow map map for each light source: a texture containing the percentage of occlusion from the light source. This soft shadow map is then projected onto the scene from the position of the light source, to give soft shadows (Fig. 1).

Our algorithm is an extension of the shadow map algorithm: we start by computing shadow map depth buffers of the scene. Unlike the standard shadow map method, we will need two shadow map depth buffers: one for the occluders and the other for the receivers.

The first shadow map depth buffer is used to discretize the set of occluders (Fig. 2(b)): each pixel in this occluder map is converted into a micro-patch that covers the same image area but is located in a plane parallel to the light source, at a distance corresponding to the pixel depth. Pixels that are close to the light source are converted into small rectangles and pixels that are far from the light source are converted into larger rectangles. At the end of this step, we have a discrete representation of the occluders.

We then compute the soft shadow of each of the micropatches constituting the discrete representation of the occluders (Fig. 2(c)), and sum them into the soft shadow map



(a) Scene view (b) Discretizing occluders (c) Soft shadows from one micro-patch (d) Summing the soft shadows

Figure 2: *The main steps of our algorithm*

```
Compute depth map of receivers
Compute depth map of occluders
for all pixels in occluder map
    Retrieve depth of occluder at this pixel
    Compute micro-patch associated with this pixel
    Compute extent of penumbra for this micro-patch
    for all pixels in penumbra extent for micro-patch
        Retrieve receiver depth at this pixel
        Compute percentage of occlusion for this pixel
        Add to current percentage of occlusion in the soft shadow map
    end
end
Project soft shadow map on the scene
```

Figure 3: *Our algorithm*

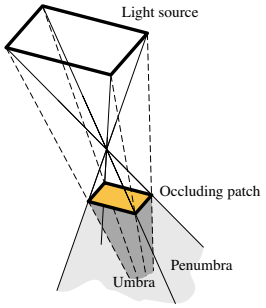


Figure 5: *The penumbra extent of a micro-patch is a rectangular pyramid*

(SSM) (Fig. 2(d)). This step would be potentially costly, but we achieve it in a reasonnable amount of time with two key points: 1) the micro-patches are parallel to the light source, so computing their penumbra extent and their percentage of occlusion only requires a small number of operations, and 2) these operations are computed on the graphics card, exploiting the parallelism of the GPU engine. The percentage of occlusion from each micro-patch takes into account the relative distances between the occluders, the receiver and the light source. Note that the sum of the occlusion percentages at a given texel is not always equal to the actual global occlusion percentage. This approximation is discussed in Section 5.2.

The pseudo-code for our algorithm is given in Fig. 3. In the following subsections, we will review in detail the individual steps of the algorithm: discretizing the occluders (section 3.2), computing penumbra extent for each micro-patch (section 3.3) and computing percentage of occlusion for each pixel in the soft shadow map (section 3.4).

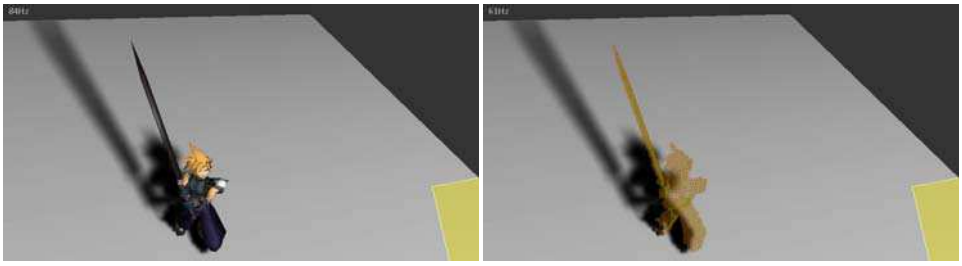
3.2. Discretizing the occluders

The first step in our algorithm is a discretization of the occluders. We compute a depth buffer of the occluders, as seen from the light source, then convert each pixel in this *occluder map* into the equivalent polygonal micro-patch that lies in a plane parallel to the light source, at the appropriate depth and occupies the same image plane extent (Fig. 4).

The occluder map is axis-aligned with the rectangular light source and has the same aspect ratio: all micro-patches created in this step are also axis-aligned with the light source and have the same aspect ratio.

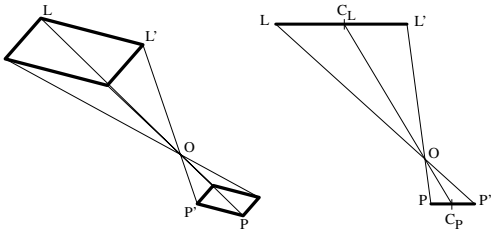
3.3. Computing penumbra extents

Each micro-patch in the discretized occluder should block some light between the light source and some portion of the receiver. To reduce the amount of computations, we compute the penumbra extent of the micro-facets before computing the exact occlusion quantity.



(a) The original model (b) The discretized occluder

Figure 4: *Discretizing the occluders*



(a) (b)

Figure 6: *Finding the apex of the pyramid is reduced to a 2D problem*

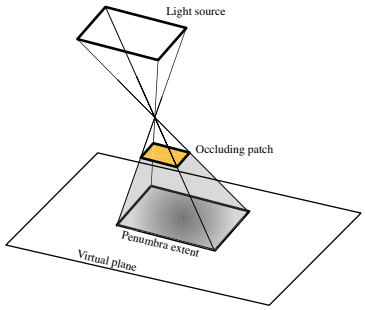


Figure 7: *The intersection between the pyramid and the virtual plane is an axis-aligned rectangle*

Since the micro-patches are parallel, axis-aligned with the light source and have the same aspect ratio, the penumbra extent of each micro-patch is a rectangular pyramid (Fig. 5). Finding the penumbra extent of the light source is equivalent to finding the apex O of the pyramid (Fig. 6(a)). This reduces to a 2D problem, considering parallel edges (LL') and (PP') on both polygons (Fig. 6(b)). Since (LL') and (PP') are parallel lines, we have:

$$\frac{OL}{OP} = \frac{OL'}{OP'} = \frac{LL'}{PP'}$$

This ratio is the same if we consider the center of each line segment:

$$\frac{OC_L}{OC_P} = \frac{LL'}{PP'}$$

Since the micro-patch and the light source have the same aspect ratio, the ratio $r = \frac{LL'}{PP'}$ is the same for both sides of the micro-patch: the penumbra of the micro-patch is indeed a pyramid.

We find the apex of the pyramid by applying a scaling to the center of the micro-patch (C_P), with respect to the center of the light source (C_L):

$$\overrightarrow{C_L O} = \frac{r}{1+r}\overrightarrow{C_L C_P}$$

We now use this pyramid to compute occlusion in the soft shadow map. We use a virtual plane, parallel to the light source, to represent this map (which will be projected onto the scene). The intersection of the penumbra pyramid with this virtual plane is an axis-aligned rectangle (Fig. 7). We only have to compute the percentage of occlusion inside this rectangle.

Computing the position and size of the penumbra rectangle uses the same formulas as for computing the apex of the

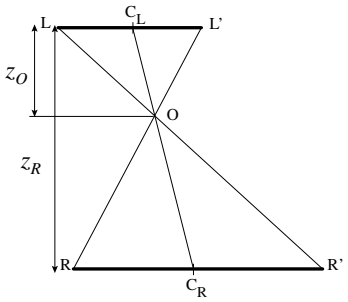


Figure 8: *Computing the position and extent of the penumbra rectangle for each micro-patch.*

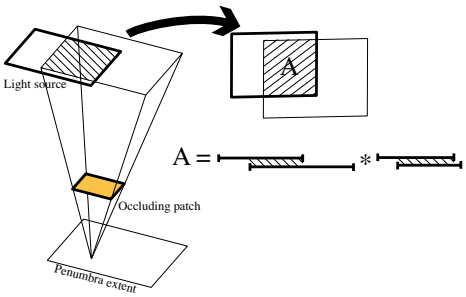


Figure 9: *We reproject the occluding micro-patch onto the light source and compute the percentage of occlusion.*

pyramid (Fig. 8):

$$\overrightarrow{C_L C_R} = \frac{z_R}{z_R - z_O}\overrightarrow{C_L O}$$
$$RR' = LL' \frac{z_R - z_O}{z_O}$$

3.4. Computing the soft shadow map

For all the pixels of the SSM lying inside this penumbra extent, we compute the percentage of the light source that is occluded by this micro-patch. This percentage of occlusion depends on the relative positions of the light source, the occluders and the receivers. To compute it, for each pixel on the receiver inside this extent, we project the occluding micro-facet back onto the light source [DF94] (Fig. 9). The result of this projection is an axis-aligned rectangle; we need to compute the intersection between this rectangle and the light source.

Computing this intersection is equivalent to computing the two intersections between the respective intervals on both axes. This part of the computation is done on the GPU, using a fragment program: the penumbra extent is converted into an axis-aligned quad, which we draw in a float buffer. For each pixel inside this quad, the fragment program computes the percentage of occlusion. These percentages are summed using the blending capability of the graphics card.

To further optimize the computations, we use the `SAT` instructions in the fragment program assembly language: without loss of generality, we can convert the rectangle corresponding to the light source to $[0,1] \times [0,1]$. Each interval intersection becomes the intersection between one $[a,b]$ interval and $[0,1]$. Exploiting the `SAT` instruction and swizzling, computing the area of the intersection between the projection of the occluder $[a,b] \times [c,d]$ and the light source $[0,1] \times [0,1]$ only requires three instructions:

```
MOV_SAT rs,{a,b,c,d}
SUB rs, rs, rs.yxwz
MUL result.color, rs.x, rs.z
```

Computing the $[a,b] \times [c,d]$ intervals requires projecting the micro-patch onto the light source and scaling the projection. This uses 11 instructions: 9 basic operations (`ADD`, `MUL`, `SUB`), one reciprocal (`RCP`) and one texture lookup to get the depth of the receiver.

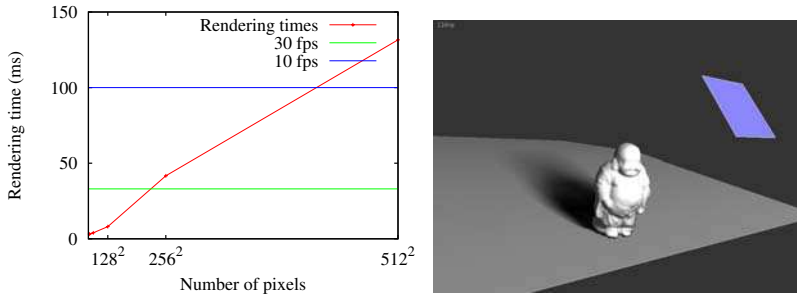
4. Experiments and Comparison

The main advantages of our algorithm are its rendering speed and its scalability. With a typical setup (an occluder map of 128×128 pixels, a scene between $50,000$ polygons and $300,000$ polygons), we get framerates between 30 and 150 fps. In this section, we study the behaviour of our algorithm and the sensitivity to parameter changes, such as the number of polygons in the scene, the size of the occluder map, the size of the light source. We would also like to identify reasonnable values for these parameters, values for which our algorithm will give interactive framerates.

All measurement in this section were conducted on a 2.4 GHz Pentium4 PC with a GeForce 6800 Ultra graphics card. All framerates and rendering times correspond to *observed* framerates, that is the framerate for a user manipulating our system. We are therefore measuring the time it takes to display the scene *and* to compute soft shadows, not just the time it takes to compute soft shadows.

4.1. Size of occluder map

The most obvious parameter is the size of the occluder map. Fig. 10(a) shows the observed rendering times (in ms) of our algorithm, on a scene with 24,000 polygons (Fig. 10(b)), when the size of the occluder map changes. We plotted the rendering time as a function of the number of pixels in the occluder map (that is, the *square* of the size of the occluder map) to illustrate the linear variation of rendering time with respect to the total number of pixels.



(a) Rendering times (in ms)

(b) Test scene (24,000 polygons)

Figure 10: *Influence of the size the occluder map*

An occluder map of 512×512 pixels gives a rendering time of 150 ms – or 7 fps, too slow for interactive rendering. An occluder map of 128×128 or 256×256 pixels gives a rendering time of 10 to 50 ms, or 20 to 100 fps, fast enough for real-time rendering. For a large penumbra region, an occluder map of 128×128 pixels qualitatively gives a reasonnable approximation, as in Fig. 10(b). For a small penumbra region, our algorithm behaves like the classical shadow mapping algorithm and artifacts can appear with an occluder map of 128×128 pixels; it is better to use 256×256 pixels.

4.2. Number of polygons

We also studied the influence of the polygon count. Fig. 11(a) shows the observed rendering time (in ms) as a function of the polygon count, with a constant occluder map size of 128×128 pixels. The first thing we note is the speed of our algorithm: even on a large scene of $340,000$ polygons, we achieve real-time framerates (more than 30 frames per second). Second, we observe that the rendering time varies linearly with respect to the number of polygons. That was to be expected, as we must render the scene twice (once for the occluder map and once for the actual display), and the time it takes for the graphics card to display a scene varies linearly with respect to the number of polygons. For smaller scenes (less than 10,000 polygons, rendering time below 10 ms), some factors other than the polygon count play a more important role.

Our algorithm exhibits good scaling, and can handle significatively large scenes without incurring a high performance cost. The maximum size of the scene depends on the requirements of the user. If the user needs real-time rendering (above 30 fps), then the maximum size of the scene is approximately 350,000 polygons. If the user is ready to accept only interactive framerates (say, 10 fps), then the maximum size of the scene would be 10^6 polygons.

4.3. Light source size

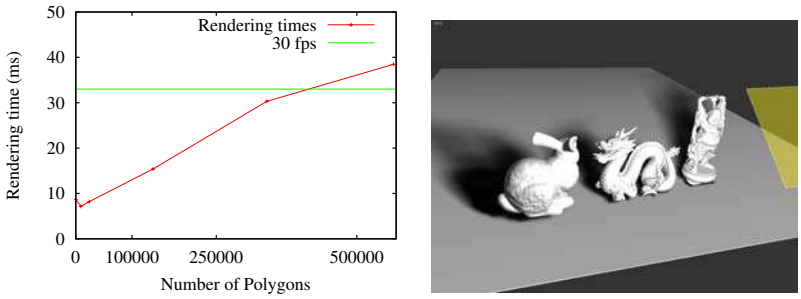
Another important parameter is the size of the light source, compared to the size of the scene itself. A large light source results in a large penumbra region for each micro-patch, resulting in more pixels of the soft shadow map covered, and a larger computational cost. Fig. 12(a) shows the observed framerate as a function of the size of the light source. The scene used for the tests has a dimension of 1, so a size of 0.5 means a light source that is 50 % of the dimension of the scene — a very large light source.

As expected, changing the size of the light source changes the observed framerate, causing it to vary between 140 fps for a very small light source to 60 fps for a light source that is about as large as the scene. For this parameter, a more practical range would be between 0 and 0.4. Fig. 13 shows the visual effects of changing the size of the light source. As can be seen, our algorithm computes both the inner- and the outer- penumbra.

When the light source is very small, the penumbra extent of each micro-patch is also very small, but we are still computing one penumbra extent for each occluded pixel in the SMDB, so the framerate is limited — in this case to 140 fps.

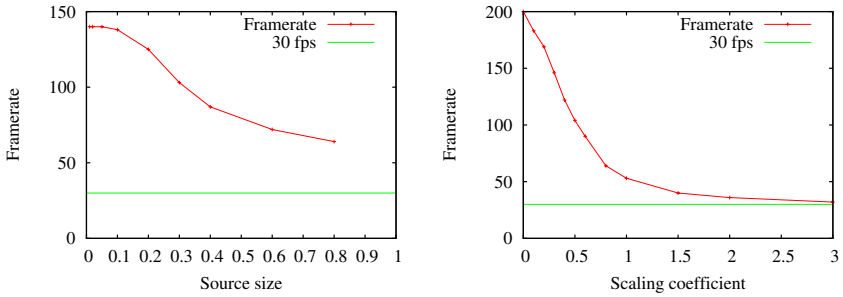
4.4. occluder map coverage

Another important parameter is the percentage of the occluder map that is actually covered by occluders. Our algorithm is only doing work for pixels in the occluder map that correspond to an occluder. Fig. 12(b) shows the observed framerate as a function of a scaling coefficient applied to the occluder.



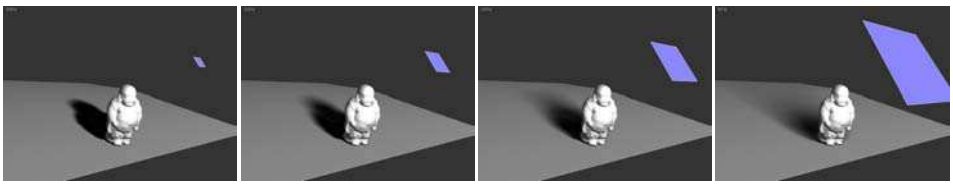
(a) Rendering times (in ms) (b) Our largest test scene (565,203 polygons)

Figure 11: *Influence of polygon count*



(a) Changing the size of the light source (b) Changing scene coverage

Figure 12: *The effects of varying parameters on the framerate of our algorithm*



(a) 0.05 (b) 0.1 (c) 0.2 (d) 0.4

Figure 13: *Changing the size of the light source*

As expected, the framerate decreases when the occluder gets larger and occupies more pixels in the occluder map. When no occluders are present, the framerate is 200 fps. This corresponds to the time it takes to compute the occluder map, read it back into memory and scan its pixels. As the occluder gets larger, it covers more pixels in the occluder map. There is a limit, as the occluder can not cover more than 100 % of the occluder map

5. Discussion

In this section, we review the complexity of our algorithm (Section 5.1) and discuss the approximations we made and their effect on the accuracy of the algorithm (Section 5.2).

5.1. Complexity and cost estimates

The main step in our algorithm is a loop over all the pixels in the occluder map. For each of these pixels, we compute the corresponding micro-patch, then its soft shadow. If the occluder map is $n \times n$ pixels, the body of the loop is executed $O(n^2)$ times.

Computing the penumbra extent of a micro-patch is done in constant time; computing the percentage of occlusion for all pixels in this penumbra extent requires drawing a quad and using the fragment program in the GPU. The actual complexity of the algorithm depends on the number of pixels covered by each quad: for a small light source, the penumbra region for each micro-patch is small: the amount of work for each micro-patch is small. On the other hand, for a large light source, each quad has a large penumbra extent, requiring a large amount of work. It is possible that each quad has to access a significative portion of the pixels in the percentage occclusion map, giving this step a theoretical complexity of $O(m^2)$, where m is the width of the soft shadow map. Usually, m and n are of the same order of magnitude, so $O(m^2)$ is equivalent to $O(n^2)$, putting the global complexity of the algorithm at $O(n^4)$.

In a previous CPU implementation of our algorithm, we indeed observed this $O(n^4)$ complexity for scenes with large penumbra regions, while we observed a more acceptable $O(n^2)$ cost for scenes with small penumbra regions.

Moving the computation of the percentage of occlusion on the GPU completely changes the cost of the algorithm: since this step is done on an efficient, parallel computer, its cost is significantly reduced, to the point where we can consider it as constant: the observed cost of our algorithm is $O(n^2)$.

5.2. Approximations and Error estimates

Our algorithm is replacing the occluder with a discretized version. This discretization ensures interactive framerates, but it can also be a source of inaccuracies. From a given point on the receiver, we are separately estimating occlusion from several micro-patches, and adding these occlusion values together.

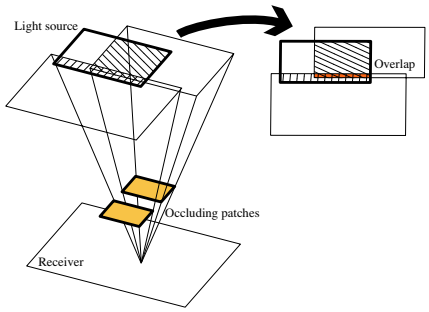


Figure 14: *The reprojection of two neighbouring micro-patches may overlap.*

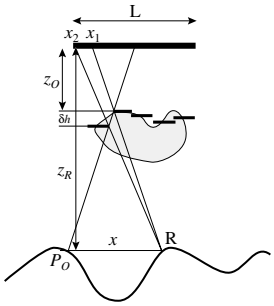


Figure 15: *Computing the extent of overlap or gap between two neighbouring micro-patches.*

At a point on the receiver, the parts of the light source that are occluded by two neighbouring micro-patches should be joined exactly for our algorithm to compute the exact percentage of occlusion on the light source. This is typically not the case, and these parts may overlap or there may be a gap between them (Fig. 14). The amount of overlap (or gap) between the occluded parts of the light source depends on the relative positions of the light source, the occluding micro-patches and the receiver

5.2.1. Error estimation

If we consider the 2D equivalent of this problem, it appears that there is a point P_0 on the receiver where there is no overlap between the occluded parts. As we move away from this point, the overlap increases (Fig. 15). For a point at a distance x from P_0, the boundaries of the occluding micro-patches project at abciss x_1 and x_2; as the occluding micro-

patches and the light source lie in parallel planes, we have:

$$\frac{x_1}{x} = \frac{z_O + \delta h}{z_R - z_O - \delta h}$$
$$\frac{x_2}{x} = \frac{z_O}{z_R - z_O}$$

The amount of overlap is therefore:

$$x_2 - x_1 = x\left(\frac{z_O}{z_R - z_O} - \frac{z_O + \delta h}{z_R - z_O - \delta h}\right)$$
$$= -x\frac{z_R \delta h}{(z_R - z_O)(z_R - z_O - \delta h)} \quad (1)$$

x itself is limited, as the occlusion area must fall inside the light source:

$$|x| < \frac{L}{2}\frac{z_R - z_O}{z_O} \quad (2)$$

The amount of overlap is therefore limited by:

$$|x_2 - x_1| < \frac{L}{2}\frac{z_R \delta h}{z_O(z_R - z_O - \delta h)} \quad (3)$$

5.2.2. Evolution with x and L:

The amount of overlap between neighbouring patches increases with x (Eq. 1), as we move away from P_O. Hence the error introduced by our algorithm becomes larger as the size of the light source increases. Increasing the size of the light source also contradicts other approximations we made in our algorithm, for example the fact that the visible portion of the object corresponds to what is present in the occluder map.

5.2.3. Evolution with δh:

According to Eq. 3, the error introduced by our algorithm is proportional to δh. Obviously, for an occluder that is planar and parallel to the light source, $\delta h = 0$ and our algorithm gives the correct answer. For an occluder that projects continuously on the occluder map, δh goes to 0 when we increase the resolution of the map, hence the overlap and the error both go to 0.

If the model exhibits discontinuities on the occluder map, our algorithm will result in false values near these discontinuities. This limitation of our algorithm may cause visible artifacts on non-convex models, when one part of the model masks other parts (Fig. 19(c), the soft shadow of the ear appears inside the shadow of the body of the bunny). In section 6.2, we present an extension of our algorithm that removes most of these artifacts — by computing two soft shadows, one for the front side of the occluders, one for the back side of the occluders.

5.2.4. Evolution with z_O:

If z_O goes towards 0, the amount of overlap goes toward infinity. Worse, the area where this overlap intersects the light source has a size that becomes infinitely large: our method should break down if the occluder is very close to the light source. Empirically, this is not the case (Fig. 16), and our algorithm still gives qualitatively good looking shadows even in this case.

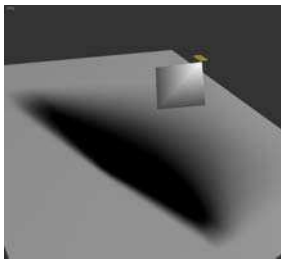
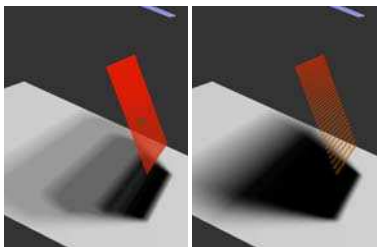


Figure 16: *Our method gives qualitatively good looking shadows even for occluders that are very close to the light source.*



(a) Occluder map: 512×512 pixels (b) Occluder map: 128×128 pixels

Figure 17: *Numerical issues with blending may result in qualitatively bad looking shadows for small micro-patches. We show here the discretized version of the occluders.*

5.2.5. Evolution with z_R:

Similarly, if z_R goes towards $z_O + \delta h$, the amount of overlap would go towards infinity. However in that case, the area where the overlap intersects with the light source goes towards 0 (Eq. 2). Our method therefore works correctly with occluders in contact with receivers (Fig. 10(b), 11(b), 13 and 19).

5.2.6. Blending accuracy

Another important point in our algorithm is the accuracy of the blending step. The contribution of each micro-patch is computed as a percentage of the area of the light source,

and these contributions are summed. For high-resolution occluder maps and large light sources, it is possible that the contribution of each patch is too small to be recorded, resulting in qualitatively bad looking shadows (Fig. 17). Reducing the size of the occluder map solves the problem. In our implementation, we used hardware blending on 16-bits floating point buffers. Hardware blending on 32-bits floating point buffers (not available on our configuration) would remove this limitation.

5.3. Hardware requirements

Our algorithm is surprisingly light on the GPU. The core of the algorithm is a small fragment program, of 14 instructions including one texture lookup.

The main hardware requirement is that we need floating point buffers for storing occlusion values: storing them on a 8 bit buffer results in visible sampling artifacts if the contribution from micro-patch is smaller than $1/256$. We also need *blending* abilities on these floating point buffers, to add the contributions from the different micro-patches. Currently, hardware blending on floating point buffers is supported on NV40 (GeForce 6) engines and on ATI Radeon X800 engines.

6. Extensions of our algorithm

6.1. Self-occlusion

An important limitation of our algorithm is that we are only computing the soft shadow cast from the selected occluders onto the selected receivers: there is no self-occlusion computed, either on the occluders or on the receivers.

However, since our algorithm requires computing a shadow map for both the occluders and the receivers, a trivial extension to our algorithm is to use these shadow maps to compute self-occlusion on the occluders and the receivers with a classical shadow mapping algorithm [Wil78]. With this extension, we would have hard shadows on all elements of the scene, combined with the soft shadows computed by our algorithm.

In some applications, such as gaming environments or virtual reality applications, the set of receivers correspond to static elements in the scene; in that case it is possible to precompute lighting effects between the static light sources and the receivers, including soft shadows. Also in that case we do not need to recompute the shadow map for the receivers at each frame.

6.2. Two-sided soft-shadow maps

As with many other soft shadow computation algorithms [HLHS03], our algorithm exhibits artifacts because we are computing soft shadows using a single view of the occluder. Shadow effects linked to parts of the occluder that are not directly visible from the light source are not visible. In Fig. 18(a), our algorithm only computes the soft shadow for the front part of the occluder, because the back part of the occluder does not appear in the occluder map. This limitation is common in real-time soft-shadow algorithms [HLHS03].

For our algorithm, we have devised an extension that solves this limitation: we compute two occluder maps. In the first, we discretize the closest, front-facing faces of the occluders (Fig. 18(b)). In the second, we discretize the furthest, back-facing faces of the occluders (Fig. 18(c)). Computation of both depth buffers is easily done using OpenGL.

We then compute a soft shadow map for each occluder map, and merge them. using the maximum of each occluder map. The resulting occlusion map has eliminated most artifacts (Fig. 18(d) and 19). Empirically, the cost of the two-passes algorithm is between 1.6 and 1.8 times the cost of the one-pass algorithm. Depending on the size of a model and the quality requirements of a given application, the second pass may be worth this extra cost. For example, if the size of an animated model is less than $100,000$ polygons, the one-pass algorithm renders at approximately 60 fps. Adding the second pass drops the framerate to 35 fps – which is still interactive.

6.3. Non-rectangular light sources

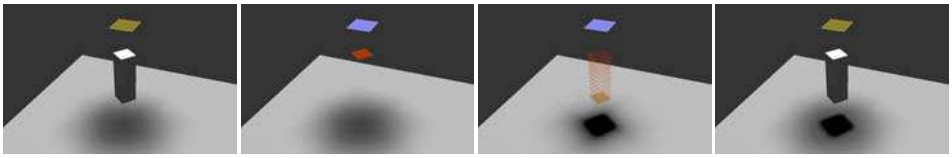
Another important limitation of our algorithm is that it only works with rectangular light sources. It is possible to overcome this limitation with Summed Area Tables [Cro84]. If we have a non-rectangular light source, we can convert it into a textured rectangular light source. We then compute a Summed-Area Table for this texture. When we are computing the percentage of occlusion for each micro-patch on the occluder depth map, we need to access the area of the light source that is covered by an axis-aligned rectangle, which is exactly what is given by Summed-Area Tables, at the cost of four texture lookups.

Using this technique, textured light sources are easily modeled, including animated textured light sources as in [AAM03].

7. Conclusion and Future Directions

In this paper, we have presented a new algorithm for computing soft shadows in real-time on animated scenes. Our algorithm is based on the shadow mapping algorithm, and is entirely image-based. As such, it benefits from the advantages of image-based algorithms, especially speed. It also suffers from the same problems, especially aliasing. For a small light source, our algorithm gives the same result as the standard shadow mapping algorithm, including all aliasing problems.

A possible solution to aliasing problems in the shadow mapping algorithm is perspective-corrected shadow



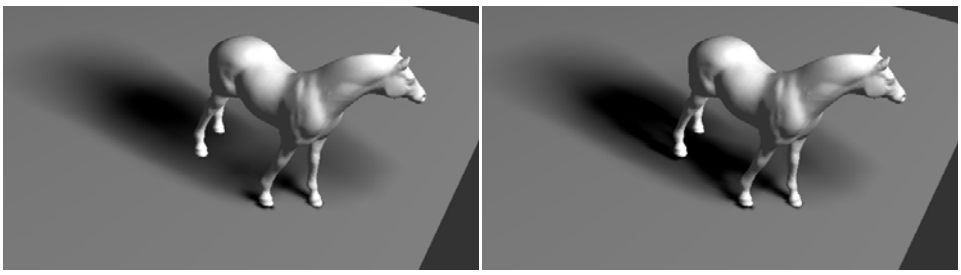
(a) Original algorithm

(b) Closest, front faces of the occluder discretized with their shadow

(c) Furthest, back faces of the occluder discretized with their shadow

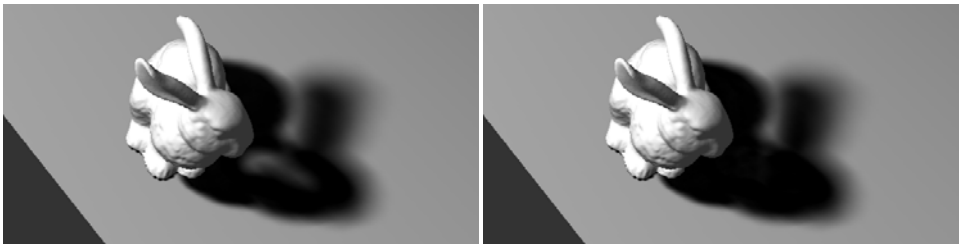
(d) Combining the two soft shadow maps

Figure 18: *The original algorithm fails for some geometry. The two-pass method gives the correct shadow.*



(a) One pass (94 fps)

(b) Two passes (57 fps)



(c) One pass (39 fps)

(d) Two passes (22 fps)

Figure 19: *Two-pass shadow computations enhance precision.*

maps [SD02, WSP04, MT04, CG04]. In our future work, we would like to combine our algorithm with perspective-corrected shadow maps; we think is is important to ensure the usability of our algorithm in complex scenes. This raises many interesting research problems because our algorithm relies on the parallelism between different planes used for discretization. Also, because we are computing a soft shadow, we can afford a coarser resolution for the occluder map in areas where there is a large penumbra. The optimal projection method for our algorithm might therefore be different from the methods identified in previous studies [SD02, WSP04, MT04, CG04].

In its current form, our algorithm still requires a transfer of the occluder map from the GPU to the main memory, and a loop, on the CPU, over all the pixels in the occluder map. We would like to design a full GPU implementation of our algorithm, using the future render-to-vertex capabilities (or superbuffers).

An important advantage in our algorithm is that we do not require any access to the geometry of the actual occluders. It is only necessary to be able to draw the occluders inside an occluder map. It should therefore be possible to use our algorithm to compute soft shadows for many alternate rendering methods, where access to the actual geometry is not easy, such as point-based rendering or volume-based rendering. The relative independance with polygon count also makes it an interesting method for large automatically generated models, such as the models generated from image and video analysis, e.g. in Virtual Reality and Augmented Reality applications.

References

[AAM03] ASSARSSON U., AKENINE-MÖLLER T.: A geometry-based soft shadow volume algorithm using graphics hardware. *ACM Transactions on Graphics 22*, 3 (July 2003), 511–520. 1, 2, 10

[AMH02] AKENINE-MÖLLER T., HAINES E.: *Real-Time Rendering*, second edition ed. A K Peters, 2002. 2

[BS02] BRABEC S., SEIDEL H.-P.: Single sample soft shadows using depth maps. In *Graphics Interface* (2002). 1, 2

[CD03] CHAN E., DURAND F.: Rendering fake soft shadows with smoothies. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering* (June 2003), pp. 208–218. 1, 2

[CG04] CHONG H., GORTLER S. J.: A lixel for every pixel. In *Proceedings of the 2nd EG Symposium on Rendering* (2004), Springer Computer Science, Eurographics, Eurographics Association. 12

[Cro77] CROW F. C.: Shadow algorithms for computer graphics. *Computer Graphics (Proceedings of SIGGRAPH 77) 11*, 2 (July 1977), 242–248. 2

[Cro84] CROW F. C.: Summed-area tables for texture mapping. *SIGGRAPH Computer Graphics 18*, 3 (1984), 207–212. 10

[DF94] DRETTAKIS G., FIUME E.: A fast shadow algorithm for area light sources using backprojection. In *Computer Graphics (SIGGRAPH 1994)* (1994), Annual Conference Series, ACM SIGGRAPH, pp. 223–230. 5

[HLHS03] HASENFRATZ J.-M., LAPIERRE M., HOLZSCHUCH N., SILLION F.: A survey of real-time soft shadows algorithms. *Computer Graphics Forum 22*, 4 (Dec. 2003), 753–774. 1, 2, 10

[KD03] KIRSCH F., DOELLNER J.: Real-time soft shadows using a single light sample. *Journal of WSCG (Winter School on Computer Graphics 2003) 11*, 1 (2003). 2

[KMK97] KERSTEN D., MAMASSIAN P., KNILL D. C.: Moving cast shadows and the perception of relative depth. *Perception 26*, 2 (1997), 171–192. 1

[MT04] MARTIN T., TAN T.-S.: Anti-aliasing and continuity with trapezoidal shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering* (2004), Springer Computer Science, Eurographics, Eurographics Association. 12

[SD02] STAMMINGER M., DRETTAKIS G.: Perspective shadow maps. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002)* (2002), 557–562. 12

[SS98] SOLER C., SILLION F. X.: Fast calculation of soft shadow textures using convolution. In *Computer Graphics (SIGGRAPH 1998)* (1998), Annual Conference Series, ACM SIGGRAPH, pp. 321–332. 2

[Wan92] WANGER L.: The effect of shadow quality on the perception of spatial relationships in computer generated imagery. In *1992 Symposium on Interactive 3D Graphics* (Mar. 1992), vol. 25, pp. 39–42. 1

[WFG92] WANGER L., FERWERDA J. A., GREENBERG D. P.: Perceiving spatial relationships in computer-generated images. *IEEE Computer Graphics and Applications 12*, 3 (1992), 44–58. 1

[WH03] WYMAN C., HANSEN C.: Penumbra maps: Approximate soft shadows in real-time. In *Eurographics Symposium on Rendering: 14th Eurographics Workshop on Rendering* (June 2003), pp. 202–207. 1, 2

[Wil78] WILLIAMS L.: Casting curved shadows on curved surfaces. *Computer Graphics (Proceedings of SIGGRAPH 78) 12*, 3 (Aug. 1978), 270–274. 2, 10

[WPF90] WOO A., POULIN P., FOURNIER A.: A survey of shadow algorithms. *IEEE Computer Graphics & Applications 10*, 6 (Nov. 1990), 13–32. 2

[WSP04] WIMMER M., SCHERZER D., PURGATHOFER W.: Light space perspective shadow maps. In *Proceedings of the 2nd EG Symposium on Rendering* (2004), Springer Computer Science, Eurographics, Eurographics Association. 12

9 Bibliographie

[ABFM+04] Jérémie Allard, Edmond Boyer, Jean-Sébastien Franco, Clément Ménier, Bruno Raffin, "*Marker-less Real Time 3D Modeling for Virtual Reality*", Immersive Projection Technology (IPT'2004), May 2004.

[Assa03] Ulf Assarsson, "*A Real-Time Soft Shadow Volume Algorithm*", PhD thesis, Department of Computer Engineering, Chalmers University of Technology, October 2003.

[BF03] Edmond Boyer, Jean-Sebastien Franco, "*A Hybrid Approach for Computing Visual Hulls of Complex Objects*", Computer Vision and Pattern Recognition (CVPR'03), Volume. 1, pp. 695–701, June 2003

[BL99] F. van den Bergh and V. Lalioti, "*Software Chroma Keying in an Immersive Virtual Environment*", South African Computer Journal, (24), pp. 155–162, Nov. 1999.

[CTMS03] Joel Carranza, Christian Theobalt, Marcus Magnor, Hans Peter Seidel, "*Free viewpoint video of human actors*", Proceedings of ACM SIGGRAPH'03, 22(3), pp.569–577, 2003.

[CG99] R. Cipolla and P.J. Giblin, "*Visual Motion of Curves and Surfaces*", Cambridge University Press, 1999.

[CKBH00] German K.M. Cheung, Takeo Kanade, Jean-Yves Bouguet, Mark Holler, "*A Real Time System for Robust 3D Voxel Reconstruction of Human Motions*", Computer Vision and Pattern Recognition (CVPR'00), Vol. 2, pp. 714–720, 2000.

[CPP03] Rita Cucchiara, Massimo Piccardi, Andrea Prati, , "*Detecting Moving Objects, Ghosts, and Shadows in Video Streams*", IEEE Transactions on Pattern Analysis and Machine Intelligence, 25(10), pp. 1337–1342, Oct. 2003.

[CTMS03] Joel Carranza, Christian Theobalt, Marcus Magnor, Hans-Peter Seidel, "*Free-viewpoint video of human actors*", ACM Trans. on Computer Graphics, 22(3), pp. 569–577, July 2003.

[DFR04] Doug DeCarlo, Adam Finkelstein, Szymon Rusinkiewicz, "*Interactive Rendering of Suggestive Contours with Temporal Coherence"*, NPAR 2004, pp. 15–24, 2004.

[DR95] I. Debled-Rennesson, J.-P. Réveilles, "*A linear algorithm for segmentation of digital curves*", International Journal of Pattern Recognition and Artificial Intelligence, 9(6), pp. 635–662, Dec. 1995.

[DYB98] Paul E. Debevec, Yizhou Yu, George D. Borshukov, "*Efficient view-dependent image-based rendering with projective texture-mapping*", Eurographics Workshop on Rendering, pp. 105–116, 1998.

[Dye01] Charles Dyer, "*Volumetric scene reconstruction from multiple views*", In L. S. Davis, editor, Foundations of Image Understanding, pp. 469–489. Kluwer, 2001.

[Fai01] Sylvain Faisan, "*Incrustation temps réel d'un personnage dans un monde virtuel*", mémoire de stage de 3ième année, diplôme d'ingénieur de l'ENSPS, 2001.

[FB03] Jean-Sebastien Franco, Edmond Boyer, "*Exact Polyhedral Visual Hulls*", British Machine Vision Conference (BMVC'03), Vol. 1, pp. 329–338, September 2003.

[FMBR04] Jean-Sebastien Franco, Clement Menier, Edmond Boyer, Bruno Raffin, "*A Distrbuted Approach for Real-Time 3D Modeling*", CVPR Workshop on Real-Time 3D Sensors and their Applications, Washington DC, July 2004.

[GrImage] *GrImage (Grid and Image)*, GrImage is funded by INRIA and an Allocation Spécifique from the Ministère délégué à la Recherche et aux Nouvelles Technologies (via INPG), http://www.inrialpes.fr/sed/grimage/

[HLHS03] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion, "*A survey of Real-Time Soft Shadows Algorithms*", STAR Eurographics, pp. 1–20, 2003.

[HLHS03b] Jean-Marc Hasenfratz, Marc Lapierre, Nicolas Holzschuch, François Sillion, "*A survey of Real-Time Soft Shadows Algorithms*", Computer Graphics Forum, 22(4), pp. 753–774, 2003.

[HLGB03] Jean-Marc Hasenfratz, Marc Lapierre, Jean-Dominique Gascuel, Edmond Boyer, "*Real-Time Capture, Reconstruction and Insertion into Virtual World of Human Actors*", Vision, Video and Graphics, pp. 49–56, 2003.

[HLS04] Jean-Marc Hasenfratz, Marc Lapierre, François Sillion, "*A Real-Time System for Full Body Interaction with Virtual Environments*", Eurographics Symposium on Virtual Environments, pp. 147–156, 2004.

[HDSD99] Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, George Drettakis, "*A Practical Analysis of Clustering Strategies for Hierarchical Radiosity*", Computer Graphics Forum (Proc. of Eurographics '99), 18(3), pp. 221–232, 1999

[HM04] Bertrand Holveck, Hervé Mathieu, "*Infrastructure of the GrImage experimental platform: the video acquisition part*", INRIA RT-0301, 50 pages, Nov. 2004.

[KDMF03] Robert D. Kalnins, Philip L. Davidson, Lee Markosian, Adam Finkelstein, "*Coherent stylized silhouettes*", Proceedings of ACM SIGGRAPH 2003, 22(3), pp 856–861, 2003.

[KS00] K. Kutulakos and S. Seitz, "*A Theory of Shape by Space Carving*", International Journal of Computer Vision, 38(3), pp. 199–218, 2000.

[Kut00] K. Kutulakos, "*Approximate N-View Stereo*", Proceedings European Conference on Computer Vision, pp. 67–83, 2000.

[Lau94] A. Laurentini, "*Visual hull concept for silhouette-based image understanding*", IEEE Trans. on PAMI, 16(2), pp. 150–162, 1994.

[Lok01] Benjamin Lok, "*Online model reconstruction for interactive virtual environments*", Proceedings of the 2001 symposium on Interactive 3D graphics, pp. 69–72, ACM Press, 2001.

[MBM01] Wojciech Matusik, Chris Buehler, and Leonard McMillan, "*Polyhedral visual hulls for real-time rendering*", Eurographics Workshop on Rendering, pp. 115–126, 2001.

[OH05] Alexandrina Orzan, Jean-Marc Hasenfratz, "*Omnidirectional texturing of human actors from multiple view video sequences*", Conference on Computer-Human Interaction, 2005.

[OpenCV] "*Open Computer Vision Library*", http://sourceforge.net/projects/opencvlibrary/

[Par02] David Parrish, "*Inspired 3d Lighting & Compositing*", Muska & Lipman Publishing, ISBN 1931841497, 2002.

[SB96] Alvy Ray Smith and James F. Blinn, "*Blue screen matting*", In Proc. of the 23rd Annual Conf. on Computer Graphics and Interactive Techniques, ACM Press, New York, USA, pp. 259–268, 1996.

[SCMS01] A. Slabaugh, B. Culbertson, T. Malzbender, and R. Schafe, "*A survey of methods for volumetric scene reconstruction from photographs*", International Workshop on Volume Graphics, pp. 81–100, 2001.

[SH00] François Sillion, Jean-Marc Hasenfratz, "*Efficient Parallel Refinement for Hierarchical Radiosity on a DSM computer*", Third Eurographics Workshop on Parallel Graphics and Visualisation, Girona, pp. 61–74, Sep 2000.

[SP93] François Sillion, Claude Puech, "*Radiosity and Global Illumination*", Morgan Kaufmann Publishers, 1994.

[TCMS03] Christian Theobalt, Joel Carranza, Marcus Magnor, Hans Peter Seidel, "*Enhancing Silhouette-Based Human Motion Capture with 3D Motion Fields*", Proceedings of the 11th Pacific Conference on Computer Graphics and Applications, pp. 185–193, 2003.

[TMSS02] Christian Theobalt, Marcus Magnor, Pascal Schüler, Hans Peter Seidel, "*Combining 2D Feature Tracking and Volume Reconstruction for Online Video-Based Human Motion Capture*", Proceddings of Pacific Graphics, pp. 96–103, 2002.

[TKBM99] Kentaro Toyama, John Krumm, Barry Brumitt, Brian Meyers, "*Wallflower: Principles and Practice of Background Maintenance*", ICCV, pp. 255–261, 1999.

[TL04] Eric Tabellion, Arnauld Lamorlette, "*An Approximate Global Illumination System for Computer Generated Films*", Proceedings of Siggrpah 2004, 23(3), pp. 469–476, August 2004.

[Tsa86] Roger Y. Tsai, "*An Efficient and Accurate Camera Calibration Technique for 3d Machine Vision*", In Proceedings of IEEE Coonference on Computer Vision and Pattern Recognition, pp. 364–374, 1986.

[VBK05] Sundar Vedula, Simon Baker, Takeo Kanade, "I*mage-based spatio-temporal modeling and view interpolation of dynamic events*", ACM Transactions on Graphics (TOG), 24(2), pp. 240–261, April 2005.

[WFSM02] Dongsheng Wangy, Tao Fengz, Heung-Yeung, Shumz Songde May, "*A Novel Probability Model forBackground Maintenance and Subtraction*", 15th International Conference on Vision Interface (ICVI'02), pp. 109–117, 2002.

[WJVT+05] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville (Eddy) Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Marc Levoy, Mark Horowitz, "*High Performance Imaging Using Large Camera Arrays*", Proceeding of SIGGRAPH 2005.

[YPLL04] Tao Yang, Quan Pan, Stan Z. Li, Jing Li, "*Multiple Layer Based Background Maintenance in Complex Environment*", Third International Conference on Image and Graphics (ICIG'04), pp. 112–115, 2004.

[Zha00] Zhengyou Zhang. "*A Flexible New Technique for Camera Calibration*", IEEE Transcions on Pattern Analysis and Machine Intelligence, pp. 22(11), pp.1330–1334, 2000.