HAL
open science

Visualisation interactive de grands bâtiments

Maxime Maria, Sébastien Horna, Lilian Aveneau

To cite this version:

Maxime Maria, Sébastien Horna, Lilian Aveneau. Visualisation interactive de grands bâtiments. AFIG 2013 - 26èmes journées de l'Association Française d'Informatique Graphique et du chapitre français d'Eurographics, Nov 2013, Limoges, France. hal-00913193v1

HAL Id: hal-00913193

https://hal.science/hal-00913193v1

Submitted on 3 Dec 2013 (v1), last revised 6 Dec 2013 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Visualisation interactive de grands bâtiments

Maxime Maria, Sébastien Horna, Lilian Aveneau[1]

[1]Laboratoire XLIM/SIC - Université de Poitiers

Résumé

Les performances des algorithmes de lancer de rayons sont directement liées à la structure accélératrice utilisée. En ce qui concerne les environnements architecturaux, plusieurs travaux ont précédemment démontré que la structure accélératrice la plus efficace est la structure cellules-et-passages*. Dans cet article, nous proposons une nouvelle structure accélératrice qui consiste en une extension des structures cellules-et-passages classiques par une description topologique complète de la scène. La structure de données est décrite par un graphe dont le parcours, utilisant l'ensemble des propriétés topologiques de notre modèle, est particulièrement simple et rapide. Nous montrons dans cet article que notre structure permet un rendu interactif même pour de grands bâtiments composés de plusieurs centaines de pièces meublées en prenant en compte l'éclairage direct de plusieurs milliers de sources lumineuses ponctuelles.*

Performances of ray-tracing algorithm are directly linked with the acceleration structure used. As part of architectural environments, several works have previously demonstrated that the most efficient acceleration structure is the cells-and-portals *one. In this paper, we propose a new acceleration structure which consists in an extension of usual cells-and-portals with a full topological description of the scene. The data structure is described by a graph whose scan is particularly simple and fast using all topological properties of our model. We show in this article that our structure allows an interactive rendering even for large buildings composed of some hundreds of furnished rooms and taking into account direct lighting from some thousands of point light sources.*

Mots clé : Lancer de rayons, lancer de rayons GPU, structure accélératrice, modélisation à base topologique.

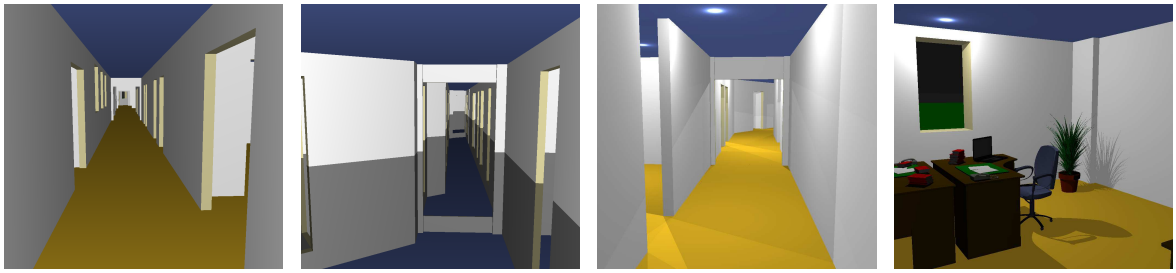
1. Introduction

Le lancer de rayons est une technique de rendu connue pour sa capacité à produire des images de haute qualité. Elle permet de prendre compte n'importe quel phénomène visuel tel que la réflexion, la réfraction, l'éclairement direct ou global. En contrepartie, elle se révèle coûteuse à mettre en œuvre. En effet, afin de déterminer la couleur de chaque pixel de l'image à calculer, un ou plusieurs rayons sont lancés, rebondissant à travers la scène. Ceci a pour effet de simuler le trajet inverse de la lumière à travers la scène, *i.e.* depuis le point de vue (la caméra) jusqu'aux sources lumineuses. La problématique du lancer de rayons consiste donc à trouver efficacement la plus proche intersection d'un rayon donné au sein de l'environnement. Depuis les travaux de Whitted [Whi80], beaucoup de méthodes sont proposées pour accélérer le lancer de rayons. Le plus souvent, une structure accélératrice est utilisée afin de réduire le nombre de tests d'intersection inutiles.

Pour les environnements architecturaux, les structures accélératrices générales telles que les arbres kd [Ben75], les arbres BSP [FKN80], les grilles régulières [FTI88] ou encore les hiérarchies de volumes englobants (BVH) [RW80, KK86] ne sont pas totalement adaptées. Alors qu'elles sont utilisées pour n'importe quel type de scène, elle peuvent devenir lentes, spécialement pour des scènes architecturales. Généralement, les environnements architecturaux sont rendus à l'aide d'une structure accélératrice particulière, appelée *cellules-et-passages* (ou structure CeP). Elle a pour principe de découper le bâtiment en volumes se rapprochant au maximum des pièces. Elle est stockée sous la forme d'un graphe dont le parcours s'effectue en utilisant uniquement les relations de voisinage entre volumes. L'efficacité de ce type de structure n'est pas dépendante de la taille de la scène, étant donné que la complexité de l'algorithme de parcours est locale. Cette efficacité est démontrée dans plusieurs travaux précédents [ARB90, TS91, LG95, MMB98, FMH05].

Dans cet article, nous proposons une nouvelle structure accélératrice pour le lancer de rayons dans des scènes architecturales. Cette structure résulte directement d'un modèle topologique dédié aux environnements architecturaux correspondant à une partition de l'espace 3D [HMDB09]. Elle consiste en fait en une structure CeP, étendue par une utilisation complète de la topologie. En effet, notre idée principale est de tirer avantage de l'ensemble des propriétés to-

© AFIG 2013, Association Française d'Informatique Graphique (http://afig.fr)



(a) (b) (c) (d)

Figure 1: *Exemples d'images et performances en images par seconde (IPS) : (a) Rendu lambertien basique : 978,24 IPS - 1025,76 Mrays/s ; (b) Le sol est un miroir : 925,62 IPS - 1406,66 Mrays/s (471 millions de rayons réfléchis) ; (c) Éclairement direct : 134,94 IPS - 1226,92 Mrays/s (8,04 millions de rayons lumineux) ; (d) Pièce meublée et éclairée : 58,2 IPS - 256 Mrays/s (3,35 millions de rayons lumineux).*

pologiques disponibles afin d'ordonnancer et donc accélérer son parcours. Ainsi, l'algorithme de parcours proposé utilise non seulement les relations de voisinage entre volumes, mais aussi celles entre faces, arêtes et sommets.

Contribution Notre principale contribution est une nouvelle structure accélératrice dédiée aux environnements architecturaux qui possède de nombreux avantages :

- Sa construction est rapide, car elle résulte directement de la modélisation à base topologique.
- Son parcours est très rapide, en utilisant l'ensemble des informations topologiques disponibles.
- Son parcours est possible pour un simple rayon, un paquet ou un faisceau.
- Elle est adaptée à la programmation parallèle à la fois sur CPU et sur GPU.

Un exemple d'utilisation est proposé dans la figure 1.

Ce papier est organisé comme suit : la section 2 présente les principaux travaux précédents visant à accélérer le lancer de rayons en général ainsi qu'en environnement architectural. La section 3 introduit notre structure accélératrice. La section 4 détaille son parcours pour un unique rayon. La section 5 décrit quelques extensions comme le lancer de rayons par paquets, les rayons secondaires (réflexions, éclairement direct), ou encore l'ameublement des scènes. La section 6 présente et analyse les performances de nos différents algorithmes, à la fois sur CPU et sur GPU. Enfin, la section 7 conclut et donne quelques perspectives de travaux futurs.

2. Travaux précédents

Depuis 1968 et les travaux d'Appel [App68], le lancer de rayons est utilisé afin de calculer des images hautement réalistes [Whi80, Gla89], tenant compte des phénomènes de réflexion, de réfraction ou encore de l'éclairement global. Une approche naïve du lancer de rayons teste l'intersection d'un rayon avec l'ensemble des polygones constituant la scène. Beaucoup de travaux précédents visent à réduire cette complexité linéaire par rayon. Dans cette section, nous présentons les principales techniques d'accélération (section 2.1), et discutons des spécificités des environnements architecturaux (section 2.2).

2.1. Techniques d'accélération du lancer de rayons

Dans cette partie, nous rappelons non exhaustivement les principales techniques d'accélération du lancer de rayons. La section 2.1.1 présente les structures accélératrices les plus communes. La section 2.1.2 rappelle les techniques regroupant les rayons en paquets. Enfin, la section 2.1.3 traite de la parallélisation du lancer de rayons sur GPU.

2.1.1. Structures accélératrices générales

Contrairement à une version naïve, l'efficacité d'un lancer de rayons repose sur une structure accélératrice. Une telle structure permet d'organiser la géométrie afin de limiter le nombre de tests d'intersection pour un rayon donné.

L'arbre kd [Ben75] est une structure accélératrice connue pour son efficacité, au moins pour des environnements statiques [Hav00, RSH05]. Toutefois, sa construction se révèle très coûteuse en terme de temps de calcul. D'autres structures accélératrices proposent un coût de construction moins élevé, telles que : les hiérarchies de volumes englobants (BVH) [RW80, KK86], les grilles régulières [FTI88] ou encore les hiérarchies d'intervalles englobants (BIH) [WK06]. Si ces structures sont plus rapides à reconstruire qu'un arbre kd, elles sont cependant moins efficaces à parcourir.

Ces structures sont générales, au sens où elles peuvent être utilisées pour n'importe quel type de scènes virtuelles. Néanmoins, elles sont généralement mises en défaut avec des scènes architecturales et ce pour deux raisons. La première est que ces dernières contiennent généralement de petits objets très détaillés à l'intérieur de grands espaces vides, ralentissant le parcours de structures régulières ; ce type de configuration est connu comme le problème du *teapot in the stadium*. La seconde raison est que les murs des bâtiments ne sont pas nécessairement alignés aux axes du repère ; les structures accélératrices utilisant des plans de découpe alignés aux axes ne respectent pas la position de ces murs, et donc créent des schémas de découpe complexes, réduisant d'autant les performances.

2.1.2. Lancer de rayons par paquets

Une autre méthode d'accélération du lancer de rayons profite de la cohérence spatiale des rayons en les groupant

© AFIG 2013.

dans des faisceaux [HH84], des cônes [Ama84] ou encore des pinceaux [STN87]. Ces méthodes sont particulièrement utiles pour les rayons primaires ainsi que pour calculer des ombres franches, ou des ombres douces avec des sources lumineuses d'aire assez petite [BW09].

Ce type de méthodes révèle son potentiel avec la parution des instructions à virgule flottante compactée SIMD (AVX, SSE, 3DNow !) des processeurs modernes. Ces instructions traitent au moins 4 réels simultanément. Ainsi, il est possible de regrouper les rayons en paquets délimités par 4 rayons extrêmes. Si ces 4 rayons croisent la même face, alors tout les rayons internes aussi. Sinon, le paquet doit être découpé, et le processus doit être relancé pour chacun des nouveaux paquets. Par conséquent, en cas d'objets très détaillés, un paquet peut être découpé un grand nombre de fois et les performances peuvent chuter. Le lancer de rayons par paquets utilisant les instructions SIMD est introduit par Wald *et al.* [WSBW01] pour des arbres kd. Il est par la suite adapté pour des grilles [WIK*06] ainsi que pour des BVH [MW06].

2.1.3. Lancer de rayons sur GPU

Cette dernière décennie, le calcul générique sur processeur graphique (GPGPU) a considérablement évolué. Dans le cadre du lancer de rayons, Purcell *et al.* [PBMH02] montrent comment calquer l'algorithme de lancer de rayons sur un modèle de programmation de flux en utilisant une grille régulière comme structure accélératrice.

Depuis lors, de nombreux travaux se sont succédés sur le sujet. Dans [FS05], ils s'affranchissent de la pile nécessaire au parcours d'un arbre kd, qui se révèle problématique sur GPU en raison d'une mémoire limitée pour chaque multiprocesseur. Ils proposent deux nouveaux algorithmes de parcours sans pile, nommés *kd-restart* et *kd-backtrack*. Cette méthode est étendue et améliorée dans [PGSS07] ou [HSHH07].

Concernant les hiérarchies de volumes englobants, Günther *et al.* présentent un algorithme de parcours de BVH utilisant une pile partagée, soulignant que ce type de structure accélératrice utilise moins de registres et de mémoire globale qu'un arbre kd. Récemment, Aila *et al.* [ALK12] montrent comment se rapprocher au maximum des performances théoriques d'un lancer de rayons sur un GPU donné. Ainsi, un algorithme de parcours de BVH qui utilise au mieux l'architecture d'un GPU est proposé. Celui-ci surpasse alors tout les travaux précédents en terme de temps de parcours.

2.2. Considérations architecturales

Les scènes architecturales sont composées de larges surfaces occlusives : les murs, les sols et les plafonds. Ainsi, en un point de vue donné, seule une petite partie de la scène est potentiellement visible. La structure appelée *cellules-et-passages* (ou structure CeP) tire partie de cette organisation.

2.2.1. Structure cellules-et-passages

Dans ce type de structure, une cellule représente un volume dans lequel la lumière peut se propager (pièce, ouverture, *etc.*), alors qu'un passage est une face partagée par deux cellules (*cf.* figure 2). Idéalement, à toute pièce correspond une unique cellule ; cependant, et pour simplifier le parcours d'une structure CeP, les cellules sont généralement rendues convexes.

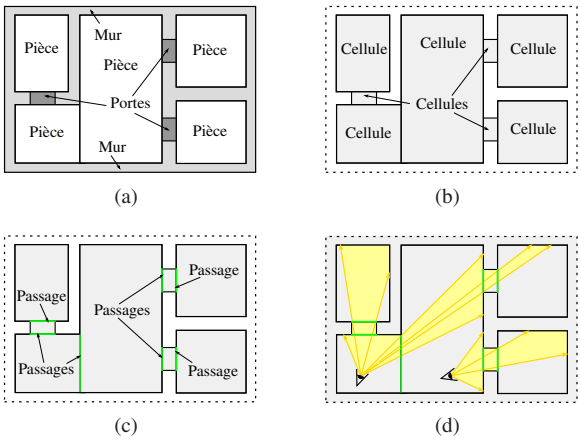


Figure 2: *Représentation et parcours d'une structure CeP en dimension 2 : (a) un bâtiment ; (b) les pièces et les portes correspondent aux cellules ; (c) les passages sont les faces partagées par deux cellules ; (d) un rayon traversant un passage est propagé dans la cellule voisine.*

Une structure CeP est stockée dans un graphe dans lequel un nœud correspond à une cellule et une arête à un passage. Depuis une cellule (convexe), la propagation d'un rayon est la suivante : si la face de sortie est de type passage, alors le processus continue avec le volume voisin ; sinon il est stoppé (*cf.* Figure 2(d)). Un tel algorithme de parcours a donc une complexité locale, et reste efficace même pour de très grandes scènes architecturales.

2.2.2. Construction d'une structure cellules-et-passages

La littérature propose plusieurs méthodes de construction de structure CeP. Les premières cherchent à extraire les informations de voisinage entre volumes depuis un ensemble de polygones afin de reconstruire au mieux une structure CeP [ARB90, TS91, MMB98]. Si la première fonctionne pour des murs alignés aux axes, la seconde traite des cellules convexes quelconques. Leur principal problème réside dans le fait que les cellules sont souvent inutilement découpées, même si Meneveaux *et al.* minimisent le nombre de cellules par pièce en utilisant un système à base de règles.

Une seconde approche extrait directement la structure CeP depuis un modeleur dédié aux environnements architecturaux [FMH05]. Les informations de voisinage entre volumes sont directement issues de la topologie, et les bords des cellules sont en parfaite adéquation avec les murs. Cette méthode produit une structure CeP optimale, dans laquelle une pièce correspond à une cellule unique. Le résultat est directement utilisable pour des simulations d'éclairage [FMH05]. Néanmoins, la construction de la scène est entièrement manuelle, et donc longue et fastidieuse. De plus, le modèle ne peut être validé étant donné qu'il ne contient aucune contrainte architecturale.

© AFIG 2013.

3. Nouvelle structure accélératrice

Notre structure repose sur le modèle topologique dédié aux environnements architecturaux introduit par Horna *et al.* [HMDB09]. Ce modèle, basé sur les cartes généralisées (G-cartes) [Lie94], est implémenté en utilisant le noyau libre Moka [Mok]. Dans cette section, nous présentons succinctement les G-cartes ; ensuite, nous décrivons le modèle topologique utilisé ; enfin, nous introduisons notre structure accélératrice.

3.1. Cartes généralisées

Les environnements architecturaux contiennent uniquement des objets du monde réel. Ainsi, il est inutile de pouvoir représenter des objets tels que les bandes de Möbius, les bouteilles de Klein ou encore les surfaces de Boy [GO10]. De plus, deux éléments ne peuvent occuper le même espace et l'espace entier est rempli. De tels environnements correspondent donc à des partitions de l'espace. Par conséquent, un bâtiment correspond à une quasi-variété 3D orientée. Dans [Lie94], il est démontré que les quasi-variétés 3D orientées sont équivalentes aux cartes généralisées 3D.

En utilisant les G-cartes, un bâtiment est défini comme une partition de l'espace 3D fermée et orientée, subdivisée en volumes, faces, arêtes et sommets. Chaque élément est représenté par ses bords dans chaque dimension. Un volume est donc décrit par ses faces, une face par ses arêtes et une arête par ses sommets. La figure 3 montre l'organisation de ces informations topologiques (pour simplifier l'illustration, les liens de dimension 0 ne sont pas représentés).

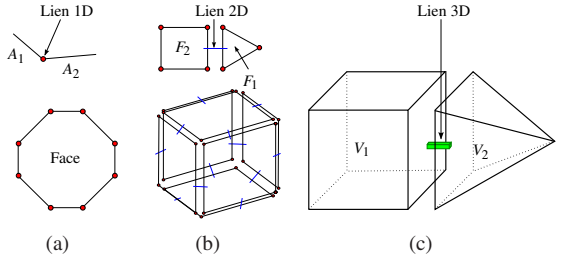


Figure 3: *Relations topologiques : (a) en dimension* 1*, les arêtes sont liées pour former des faces ; (b) en dimension* 2*, les faces sont liées pour former des volumes ; (c) une liaison de dimension* 3 *lie deux volumes.*

3.2. Modèle topologique

Afin de valider la scène dès sa modélisation, le modèle topologique de Horna *et al.* étend les G-cartes par un ensemble de contraintes topologiques, géométriques et sémantiques. Le modeleur associé génère automatiquement une scène 3D à partir de plans architecturaux 2D [HDMB07]. Ainsi, la construction d'une scène devient à la fois facile et rapide.

Le modèle topologique utilisé représente une partition fermée et orientée de l'espace 3D dans laquelle chaque volume a une épaisseur significative. À chaque volume est associé une unique sémantique définissant sa nature (PIÈCE, MUR, SOL, EXTÉRIEUR, *etc.*). La figure 4(a) présente un bâtiment en 2D respectant ces propriétés.

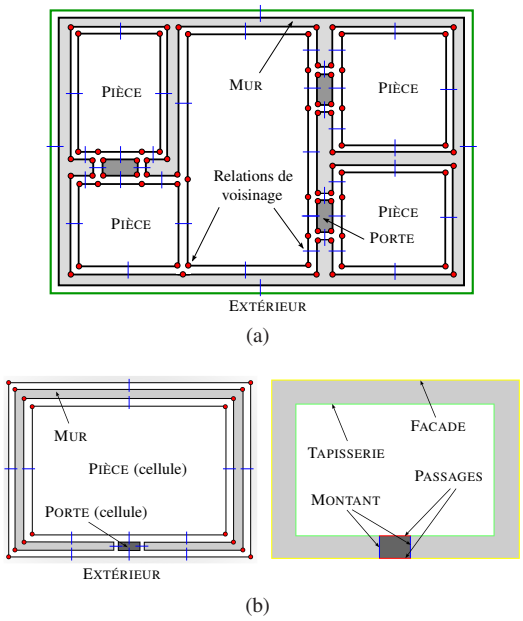


Figure 4: *Topologie et sémantique : (a) représentation topologique 2D d'un bâtiment ; chaque face (en 3D, chaque volume) a une épaisseur significative et correspond à un unique élément identifié par une sémantique ; (b) une arête (en 3D, une face) a une sémantique* PASSAGE *lorsqu'elle est incidente à deux cellules, et spécifique à son voisinage sinon.*

La structure CeP est construite depuis cette partition de l'espace. Les cellules et les passages sont déduits de la sémantique des volumes, et correspondent aux éléments permettant la propagation de la lumière. Les cellules sont ainsi les volumes de sémantique PIÈCE, PORTE, FENÊTRE, *etc.* Les passages correspondent alors aux faces liant deux volumes préalablement identifiés comme cellules. De la même manière, les propriétés photométriques des autres faces sont automatiquement déterminées et utilisées pour le rendu. Ce processus permet d'associer à chaque face une unique sémantique (*cf.* figure 4(b)).

3.3. Structure accélératrice topologique

Un bâtiment est en majeure partie composé de pièces délimitées par des murs verticaux, par un sol et un plafond. Ainsi, chaque volume de la partition de l'espace est représenté par un ensemble de faces verticales et deux faces horizontales. Afin de représenter des portes ou des fenêtres, une face verticale peut être subdivisée (*cf.* figure 5).

Notre structure accélératrice consiste donc en une partition de l'espace 3D représentant une structure CeP dans laquelle une pièce correspond à une unique cellule. Néanmoins, pour des questions d'efficacité de rendu, chaque cellule concave est automatiquement convexifiée, donc divisée en plusieurs cellules séparées par des passages. La structure de données est stockée sous la forme d'un graphe dont un nœud correspond à une cellule, et une arête est un passage.

© AFIG 2013.

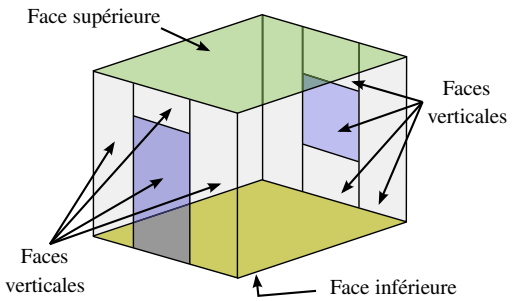


Figure 5: *Un volume est délimité par un ensemble de faces verticales et par deux faces horizontales.*

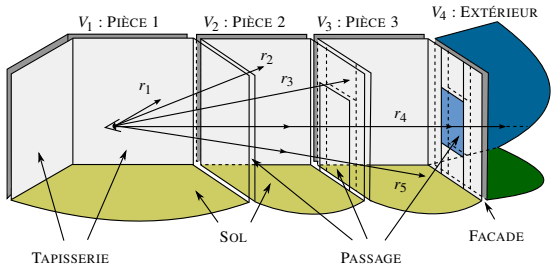


Figure 6: *Les rayons sont propagés lorsqu'ils traversent un passage, et sont stoppés sinon. Exemple avec 5 rayons, tous partant du volume V_1 : r_1 est stoppé par une face de V_1 de sémantique* TAPISSERIE*, alors que r_4 traverse trois passages et continue dans le volume V_4 de sémantique* EXTÉRIEUR.

4. Utilisation de notre structure

Dans cette section, nous présentons une version simplifiée du parcours de notre structure pour chercher la plus proche intersection le long d'un unique rayon. Ce parcours ressemble fort à un parcours classique de cellules-et-passages, bien qu'il soit optimisé par l'utilisation de toutes les propriétés topologiques disponibles.

Suivant une approche didactique, la présentation de notre algorithme part d'une vision globale du parcours de la structure CeP, avant de s'intéresser en particulier à la recherche du passage de sortie depuis une cellule.

4.1. Lancer un rayon

Le parcours de notre structure est similaire à celui d'une structure CeP classique. Le traitement débute en une cellule donnée, associée au point de départ du rayon (*e.g.* la caméra). Depuis une cellule quelconque l'algorithme recherche la face de sortie, puis continue de façon itérative s'il s'agit d'un passage, ou retourne la face en question sinon (*cf.* algorithme 1). Une illustration du fonctionnement de cet algorithme est proposée dans la figure 6.

Algorithm 1 Traversée d'un rayon dans notre structure

Require: r : rayon ; C : cellule initiale ;
1: **loop**
2: {Calcul de la face de sortie F de r dans C}
3: $F \leftarrow getFaceSortie(r, C)$;
4: **if** F.semantique $\neq$ PASSAGE **then**
5: **return** F ;
6: **end if**
7: $C \leftarrow F$.voisin ;
8: **end loop**

Cet algorithme cache en réalité deux points clés. Le premier concerne la cellule de départ du rayon, autrement dit celle contenant la caméra pour un lancer de rayons, ou le point de départ d'un rayon secondaire. Plusieurs solutions à ce problème sont envisageables, comme un BVH, un BSP ou encore un arbre kd. Dans une application interactive, ceci est néanmoins un problème mineur car il existe uniquement en amont du premier rendu. Lorsque la caméra est déplacée par l'utilisateur, il s'agit toujours de mouvement de type translation, et donc une version à peine modifiée (prenant en compte une distance maximale) de l'algorithme 1 permet de résoudre très rapidement la question. Même sur CPU, nos expérimentations montrent que le nombre d'images rendues par seconde n'est pas impacté par un éventuel déplacement de la caméra. Notons que ce problème est trivial pour les rayons secondaires (spéculaires ou d'éclairement).

Le second point important de cet algorithme réside dans la recherche de la face de sortie d'une cellule. En effet, la complexité globale de l'algorithme 1 est en nombre de cellules traversées, soit bien moins que le nombre de cellules total. Cette complexité est cependant modulée par le coût de la traversée d'une cellule. Une approche naïve consiste à tester l'intersection du rayon avec toutes les faces orientées de la cellule. Cette méthode fonctionne avec des volumes concaves en retournant la plus proche des intersections calculées. Par contre, ceci implique de nombreux tests d'intersection avec des faces polygonales, et donc un coût important. Dans le but d'obtenir un parcours le plus rapide possible de notre structure, nous proposons une recherche optimisée de la face de sortie. Elle repose notamment sur l'orientation de deux droites connues par leurs coordonnées de Plücker (*cf.* annexe A).

4.2. Optimisation de la recherche de la face de sortie

Afin d'accélérer la recherche de la face de sortie d'une cellule, nous proposons une stratégie d'ordonnancement des traitements utilisant toutes les propriétés topologiques de la scène. Ainsi, et alors que les travaux antérieurs [ARB90, LG95, MMB98, FMH05] n'utilisent que les relations topologiques de dimension 3 (relations de voisinages entre volumes), nous exploitons les relations de dimensions inférieures entre faces (dimension 2), arêtes (dimension 1) et sommets (dimension 0).

Rappelons que notre structure ne contient que des cellules convexes. Ainsi, un rayon ne traverse qu'une seule face orientée d'une cellule donnée : soit elle est verticale, soit horizontale (le sol ou le plafond). Notre approche est alors divisée en deux étapes (*cf.* algorithme 2).

Dans un premier temps, nous considérons le volume comme un prisme droit infini, ignorant les deux faces in-

© AFIG 2013.

Algorithm 2 Recherche de la face de sortie d'un rayon r dans une cellule C

Require: C : cellule ; r : rayon ;
1: {Étape 1 : section verticale de sortie}
2: $i \leftarrow 0$;
3: **while** (sideVertical(r, C.vSec[i].left) ≤ 0
    $\|$ sideVertical(r, C.vSec[i].right) > 0) **do**
4: $i \leftarrow i+1$;
5: **end while**
6: {Étape 2 : face de sortie dans section verticale i}
7: $h \leftarrow C$.vSec[i].hSec[] ;
8: **if** sideHorizontal(r, $h[0]$) < 0 **then**
9: **return** $h[0]$; {traverse le sol}
10: **end if**
11: **for** $k = 1$ to C.vSec[i].nb_hSec-1 **do**
12: **if** sideHorizontal(r, $e[k]$) < 0 **then**
13: **return** $e[k]$;
14: **end if**
15: **end for**
16: {Aucune face verticale : la face de sortie est le plafond !}
17: **return** C.plafond ;

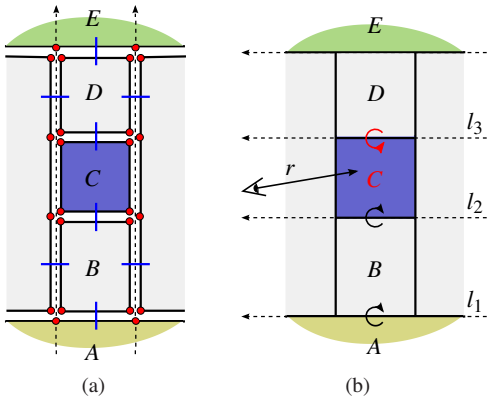


Figure 8: *Recherche de la face de sortie : (a) une section verticale est divisée en plusieurs faces verticales, incluant le sol et le plafond ; (b) chaque face verticale est horizontalement délimitée par deux droites orientées de droite à gauche ; la face de sortie de r est C (r passant au dessus de l_1 et l_2 et en dessous de l_3).*

férieure et supérieure. Ceci est une conséquence de la façon dont est générée notre structure de données, par extrusion verticale de plan architectural 2D. La première étape de l'algorithme consiste donc à rechercher la section verticale de sortie (algorithme 2 - ligne 2-5). Cette recherche est effectuée en dimension 2 ; elle consiste en un parcours des arêtes de la face inférieure du volume en utilisant les relations topologiques de dimensions 0 et 1 (*i.e.* les liens entre les sommets et les arêtes). Cette recherche est effectuée de façon linéaire (*cf.* figure 7(a)). Considérant le nombre restreint de sections verticales par volume (en moyenne 4,37), une recherche plus complexe, basée par exemple sur une dichotomie est inutile.

Arbitrairement, nous orientons les droites délimitant une section verticale infinie de bas en haut. En utilisant les coordonnées de Plücker, le test d'intersection d'un rayon avec une section verticale est très rapide : un rayon traverse une

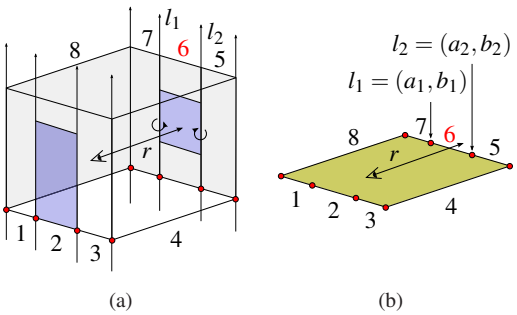


Figure 7: *Recherche de la section verticale de sortie : (a) la cellule est considérée comme un prisme droit infini dont les côtés sont délimités par deux droites orientées de bas en haut ; à chaque arête de la face inférieure correspond une unique section verticale ; la section verticale de sortie est la 6 (r passe à gauche de l_2 et à droite de l_1) ; (b) cette recherche est effectuée strictement en 2D.*

section verticale si et seulement si il passe à droite (*resp.* à gauche) de la droite de gauche (*resp.* de droite) (*cf.* figure 7(a)) ; ainsi, deux tests d'orientation entre le rayon et les droites verticales délimitant la section verticale suffisent. Bien que nous utilisions des coordonnées de Plücker, notons que cette recherche correspond exactement à un calcul en dimension 2 (*cf.* figure 7(b)). Ainsi, si a et b sont les coordonnées 2D d'une droite verticale, et pour un rayon défini par les coordonnées de Plücker $\Pi_r = \{\pi_0, \pi_1, \pi_2, \pi_3, \pi_4, \pi_5\}$, alors l'opérateur *side* est réduit à : $b \times \pi_0 - a \times \pi_1 + \pi_5$.

Partant de la section verticale de sortie, la seconde étape de l'algorithme 2 consiste à parcourir les faces qu'elle contient, en incluant le sol et le plafond. Chaque face horizontale est délimitée par deux droites horizontales orientées arbitrairement de droite à gauche (sens trigonométrique depuis l'intérieur du volume) (*cf.* figure 8). Le parcours des faces d'une section verticale utilise les propriétés topologiques de dimension 2, *i.e.* les relations de voisinage entre les faces. Une fois encore, une stratégie de recherche non linéaire est inutile en raison du peu de faces contenues dans une section verticale (en moyenne 2,11). Nous utilisons donc un parcours linéaire des faces, de bas en haut. Nous commençons avec l'arête inférieure (algorithme 2 - ligne 7-10). Si le rayon passe en dessous l'arête (*i.e.* l'orientation est négative) alors il traverse le sol, qui est donc retourné. Sinon, ce processus est répété avec les arêtes supérieures en parcourant les faces de la section verticale (algorithme 2 - ligne 11-15). Si aucune face n'est identifiée par ce processus, alors le plafond est retourné (algorithme 2 - ligne 16). Notons qu'étant donné que les droites délimitant les faces sont toutes horizontales, le calcul d'orientation se réduit à 4 multiplications et 5 additions.

4.3. Bilan sur le lancer de rayons

L'efficacité du parcours de notre structure accélératrice repose sur deux points clés. D'une part, il est optimisé par l'utilisation des relations topologiques en toutes dimensions.

© AFIG 2013.

– Dimensions 0 (sommets) et 1 (arêtes) pour trouver la section verticale de sortie.
– Dimension 2 (faces) pour parcourir les faces verticales d'une section verticale et déterminer celle de sortie.
– Dimension 3 (volumes) pour propager un rayon depuis une cellule vers sa voisine.

D'autre part, les calculs d'intersection sont supprimés, et remplacés par de simples tests d'orientation reposant sur les coordonnées de Plücker.

5. Utilisations avancées

Dans cette section, nous présentons des utilisations avancées de notre structure accélératrice : le lancer de rayons par paquets, l'éclairement direct par des sources lumineuses ponctuelles, et l'ajout de petits et gros mobiliers dans les bâtiments.

5.1. Lancer de rayons par paquets

Une méthode d'optimisation du lancer de rayons consiste à regrouper les rayons dans des paquets, et à propager des paquets dans la structure accélératrice (*cf.* section 2.1.2). Notre structure peut bénéficier d'une telle stratégie.

Nous utilisons des paquets contenant un nombre quelconque de rayons, et délimités par quatre rayons extrêmes. Pour chacun de rayons extrêmes, la face de sortie est déterminée selon l'algorithme 2 présenté dans la section 4.2. Lorsque ces quatre rayons sortent par la même face, alors il en est de même de l'ensemble des rayons du paquet. Sinon, les rayons du paquet n'ont pas un comportement cohérent. Le paquet est alors divisé en 2 ou 4 selon l'incohérence rencontrée. L'algorithme se poursuit alors avec les nouveaux paquets, directement depuis la dernière cellule rencontrée.

En pratique, cette stratégie est très efficace. Elle utilise les instructions SIMD à l'image du lancer de rayons par paquets proposé par Wald *et al.* [WSBW01]. De plus, elle s'utilise facilement en environnement multi-cœurs. Le nombre de paquets à lancer dépend directement du nombre de processus utilisés, noté t_h. Pour équilibrer la charge, nous divisons l'image en $2 \times t_h$ paquets. Dans nos tests et avec $t_h = 8$, cela correspond à 16 paquets pour une image de 1024×1024 pixels.

5.2. Calcul d'éclairement direct

L'éclairement direct permet de tester notre structure accélératrice avec des rayons spatialement incohérents. Cela implique que de nouveaux rayons (ou rayons lumineux) sont lancés afin de déterminer la visibilité entre chaque point d'intersection des rayons primaires et chaque source lumineuse ponctuelle. Ces rayons sont lancés selon une variante des algorithmes 1 et 2. L'éclairement est ensuite calculé en fonction de la visibilité et de l'intensité chromatique de chaque source.

La principale difficulté dans ce contexte réside dans le grand nombre de sources lumineuses à prendre en compte. En effet ceci augmente considérablement le nombre de rayons à lancer, et par conséquent fait chuter les performances de rendu. Étant donné que les bâtiments sont composés de larges surfaces occlusives, la plupart des tests de visibilité effectués est inutile (*cf.* figure 9(a)). Pour pallier ce problème, nous calculons l'ensemble des sources lumineuses potentiellement visibles depuis chaque volume (PVS) [ARB90, TS91, LG95, MMB98].

Calcul du PVS Nous effectuons un pré-traitement déterminant l'ensemble des sources lumineuses potentiellement visibles en chaque cellule. Cette étape est effectuée en inversant le problème, afin de réduire les temps de calcul à quelques secondes. Ainsi et depuis chaque source lumineuse, un faisceau continu est lancé à travers chaque passage de la cellule la contenant (*cf.* figure 9(b)) ; ce faisceau est ensuite propagé dans la scène, de sorte à ajouter la source lumineuse au PVS de chaque cellule traversée.

Lors du rendu, les temps de calcul sont divisés par le nombre des lumières potentiellement visibles. En effet, pour chacune d'elles, notre structure doit être parcourue. Afin d'améliorer les performances nous réutilisons les calculs effectués durant les calculs de PVS.

Arbre de lumière Durant le calcul du PVS et pour chaque source lumineuse, nous sauvegardons le parcours de ces faisceaux lumineux sous la forme d'un arbre binaire contenant l'ensemble des faces PASSAGE traversées. La racine est la source lumineuse, et chaque nœud interne contient un passage. Une branche correspond à un faisceau de plus en plus subdivisé à mesure de son parcours. Les feuilles sont des cellules où le faisceau qui y conduit s'est arrêté.

Cette structure que nous nommons *arbre de lumière*, permet d'accélérer le calcul de visibilité (*cf.* tableau 3 - section 6). En effet, il optimise le parcours classique de notre structure (selon les algorithmes 1 et 2) car il ne contient que les informations de type passage, et donc simplifie les traitements à effectuer. Le calcul de visibilité entre un point et une source lumineuse se résume alors à un simple parcours d'un arbre binaire de recherche.

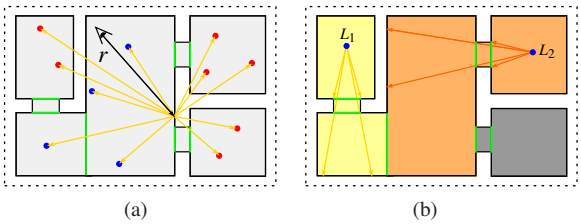


Figure 9: *Éclairement direct : (a) pour le rayon r, 11 sources lumineuses sont testées alors que seulement 4 sont visibles (en bleu) ; (b) calcul du PVS : la source est ajoutée au PVS des cellules traversées par le faisceau lumineux ; les sources l_1 et l_2 ne seront plus utilisées que dans 2 cellules sur 5.*

5.3. Prise en compte du mobilier

Notre structure accélératrice est directement issue du modeleur de bâtiments (basé sur le modèle de [HMDB09]). Les

© AFIG 2013.

Scenes	Nb. pièces	Nb. cellules (vide)	Nb. cellules (meublée)	Nb. polygones (vide)	Nb. polygones (meublée)	Nb. lumières	Mémoire (Mo) (vide)	Mémoire (Mo) (meublée)
MAISON	8	510	577	1.166	1.005.454	8	0,46	72,7
BAT-1	46	1.923	2.510	8.557	8.956.708	55	2,03	20,8
BAT-2	135	7.243	6.722	29.937	21.491.046	200	6,82	54,1

Table 1: *Caractéristiques de nos scènes : nombres de pièces, cellules et polygones.*

rendus, par exemple pour des visites de bâtiments virtuels, restent assez pauvres en raison de l'absence de petit ou gros mobilier. Nous avons donc étendu le modeleur pour permettre un ameublement automatique. Il utilise un système basé sur des règles de probabilité de placement pour trouver la position la plus adéquate de l'objet en considérant :

- la taille et le type de l'objet (bureau, plante, *etc.*) ;
- la pièce dans laquelle il est placé ;
- les objets déjà présents dans la pièce.

Notre modeleur propose en outre une édition des objets par l'utilisateur.

Cette étape d'ameublement est effectuée avant la génération de la structure accélératrice. Le volume associé à la boîte englobante de chaque objet est inclus dans la structure, avec une sémantique OBJET. Le volume initial qui reçoit le nouvel objet doit être découpé afin de respecter la contrainte de convexité de notre structure. Les volumes résultants conservent la sémantique originale, et sont séparés par des faces de sémantique PASSAGE (*cf.* figure 10).

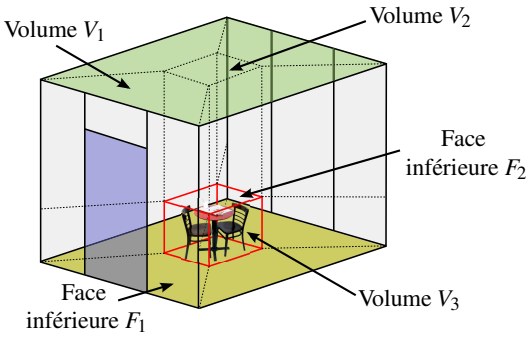


Figure 10: *Inclusion d'un objet dans notre structure : le volume contenant l'objet est découpé afin de respecter la contrainte de convexité ; le volume V_3 correspond à la boite englobante de l'objet ; F_1 et F_2 sont respectivement les faces inférieures des volumes V_1 et V_2.*

Un même objet pouvant apparaître plusieurs fois dans une scène, nous utilisons un système de clones afin de limiter la consommation mémoire (notamment sur GPU). Chaque volume de sémantique OBJET fait référence à un seul clone, mais avec une matrice de transformation spécifique décrivant sa position et son orientation.

Pour un rendu rapide, le parcours de l'objet peut être optimisé avec n'importe quelle structure accélératrice. Notre structure accélératrice est donc multi-niveaux. Lorsqu'un rayon arrive dans un volume de sémantique OBJET, nous procédons à un changement de repère (selon sa matrice de transformation) puis nous parcourons la structure accélératrice du clone. Si le rayon ne traverse pas l'objet, il est de nouveau propagé dans notre structure CeP jusqu'à qu'il touche un polygone de la scène.

6. Mise en œuvre et résultats

Nous discutons dans cette section quelques résultats de rendus effectués à partir de notre structure accélératrice.

6.1. Mise en œuvre

Nous avons implanté notre structure accélératrice sur CPU et GPU. Cela concerne d'une part, un rendu de base, avec ou sans rayons spéculaires et éclairement direct non optimisé ; d'autre part, les différentes utilisations avancées (lancer de rayons par paquets, éclairement direct par PVS et arbres de lumière ainsi que les objets). L'évaluation des performances est donnée en termes de :

- IPS : nombre d'images calculées par seconde (incluant une *brdf* de type Lambert).
- Mrays/s : nombre de millions de rayons lancés par seconde pour calculer une image.

Les rendus sont toujours effectués avec une taille d'image de 1024×1024 pixels, et sans traitement de l'aliassage. Afin de garder les mêmes points de vue pour chaque algorithme testé, nous sauvegardons un ensemble de points de vue lors d'une visite interactive. Ainsi le nombre de points de vue peut différer en fonction de la taille de la scène.

Nous présentons ici trois scènes, plus ou moins complexes en terme de taille, de nombre de polygones et de sources lumineuses. Leurs caractéristiques sont résumées dans le tableau 1. La plus petite d'entre elles (MAISON) correspond au premier étage d'une maison composée de 8 pièces ; elle contient 1 million de polygones une fois meublée. Les scènes BAT-1 et BAT-2 représentent deux étages différents d'un grand bâtiment administratif (le SP2MI, au Futuroscope). Elles sont respectivement composées de 46 et 135 pièces, pour 9 et 21 millions de polygones en versions meublées.

Tous les rendus sont effectués sur un ordinateur équipé d'un processeur Intel® Core™ i7 CPU 960 @ 3.20GHz, de 12 Go de mémoire RAM et d'un carte graphique NVidia® GeForce® GTX 680 fonctionnant sous un système d'exploitation Linux 64 bits.

6.2. Résultats et discussion

Le tableau 2 récapitule les résultats de mesures obtenus avec 8 algorithmes. Les 3 premiers fonctionnent sur CPU, et sont parallélisés avec OpenMP sur 8 threads. Le premier est un lancer de rayons classique. Les deux autres sont optimisés par paquets en utilisant des instructions SIMD SSE,

© AFIG 2013.

	Scènes	MAISON		BAT-1		BAT-2	
		IPS	Mrays/s	IPS	Mrays/s	IPS	Mrays/s
CPU	Lambert	65.09	68.26	51.28	53.78	51.35	53.85
	Paquets	388.16	407.01	384.13	370.68	375.13	393.35
	Paquets & Lumières	128.71	419.75	66.64	367.05	60.48	352.09
GPU	Lambert	1389.16	1445.66	1064.71	1116.43	1125.19	1179.84
	Sol spéculaire	1035.71	1294.62	723.81	988.3	730.58	1048.53
	Lumières	439,47	1438,59	156,77	988,88	163,35	1124,92
	Objets	191.39	200.69	245.77	257.7	230.47	241.67
	Objets & Lumières	58.28	163.48	28.13	183.124	27.43	174.67

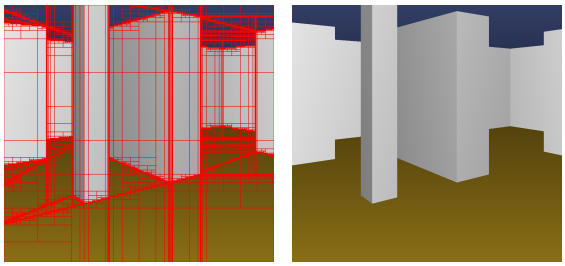
Table 2: *Résultats : nombre moyen d'images calculées par seconde (IPS), et nombre de rayons lancés par seconde (Mrays/s)*

respectivement sans et avec éclairage direct par des sources lumineuses ponctuelles (optimisé par les arbres de lumière).

Les 5 algorithmes suivants sont parallélisés sur GPU en utilisant CUDA. Le premier est un lancer de rayons classique. Le deuxième prend en compte les surfaces spéculaires (ici le sol est un miroir). Le troisième tient compte de l'éclairage direct par des sources lumineuses ponctuelles (optimisé par les arbres de lumière). Le quatrième correspond à un lancer de rayons dans des scènes meublées. Enfin, le dernier prend à la fois compte du mobilier et de l'éclairage avec PVS (sans arbre de lumière).

6.2.1. Lancer de rayons

Le tableau 2 indique que notre structure accélératrice permet le calcul d'un lancer de rayons (avec une *brdf.* de type Lambert) très efficace autant sur CPU que sur GPU. La version CPU travaillant rayon par rayon calcule plus de 50 images par secondes. En version par paquets, ce nombre augmente à 382,47 en moyenne (*cf.* figure 11(a)). Enfin, la première version du lancer de rayons sur GPU calcule facilement plus de 1000 IPS (*cf.* figure 11(b)). Pour comparaison, en utilisant le logiciel de lancer de rayons sur GPU d'Aila *et al.* [ALK12] et toujours avec une NVidia® GeForce® GTX 680, ces mêmes scènes sont rendues à approximativement 100 IPS.



(a) (b)

Figure 11: *Images et performances : (a) lancer de rayons par paquets sur CPU : 370,86 IPS ; les lignes rouges représentent les paquets ; (b) lancer de rayons sur GPU : 1536,9 IPS.*

Étant donné la complexité locale de notre algorithme de parcours, la taille globale de la scène n'affecte que peu les performances de rendu. La complexité de cet algorithme pour un rayon dépend du nombre de volumes traversés ainsi que du nombre de polygones constituant ces volumes. En fait, les performances sont réduites en cas de cellules divisées un grand nombre de fois ou en cas de volumes contenant trop de polygones. C'est pourquoi, il est nécessaire d'adopter une stratégie de convexification adaptée.

Nous utilisons une stratégie assez proche de l'heuristique de surface SAH [MB90]. Cette dernière est une approche probabiliste généralement utilisée lors de la division d'un volume donné pour répartir sa géométrie en deux sous-ensembles moins coûteux à parcourir. L'heuristique SAH cherche ainsi un découpage ayant localement un coût de traversée le plus faible possible. Lors de la convexification d'une cellule concave, notre approche cherche à placer des faces de découpe en minimisant la somme des coûts SAH au sein de cette cellule.

Néanmoins, la convexification avec des murs courbés reste problématique (*cf.* figure 12). La figure 12(a) montre une pièce convexifiée avec la méthode présentée dans [FTCP08] (les lignes rouges correspondent aux plans de découpe créés). À cause des murs courbés, beaucoup de découpes sont effectuées. Le rendu d'une image nécessite alors la traversée d'un grand nombre de volumes, réduisant les performances.

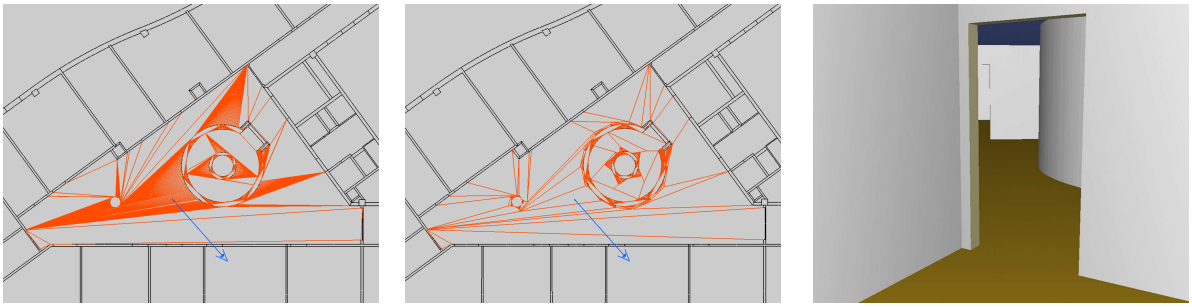
Notre méthode de convexification est illustrée sur la figure 12(b). Elle utilise des boîtes englobantes autour des murs courbés. Ainsi, la sur-découpe engendrée par ceux-ci est restreinte dans un espace réduit ; le coût SAH total diminue donc et les performances sont améliorées (*cf.* figure 12(c)).

6.2.2. Éclairage direct par sources ponctuelles

L'éclairage direct par des sources ponctuelles nécessite de lancer en un point donné autant de rayons qu'il existe de sources lumineuses. Pour une scène contenant une seule lumière, le nombre de rayons nécessaires pour calculer l'image est simplement doublé. Mais pour x lumières, les performances chutent rapidement. Comme expliqué dans la section 5.2, afin de pallier ce problème et de réduire le nombre de tests de visibilité inutiles, nous calculons le PVS pour chaque cellule de notre structure.

En pratique, le PVS permet de réduire considérablement le nombre de sources lumineuses testée. Par exemple, dans la scène BAT-2 contenant 200 sources lumineuses, le nombre moyen de rayons lumineux par image est réduit de 209,71 millions à 10,14 millions (soit -95,16%). Le tableau 3 montre l'influence du PVS en terme d'IPS par rapport à une

© AFIG 2013.



(a) Convexification selon [FTCP08] ; 9.139.670 volumes sont traversés pour calculer l'image (c).

(b) Convexification avec boîtes englobantes et topologie ; 3.422.570 volumes sont traversés pour calculer l'image (c).

(c) Image associée, de taille 1024 × 1024, à 499,56 IPS avec la convexification (a) et 977,86 IPS avec la (b).

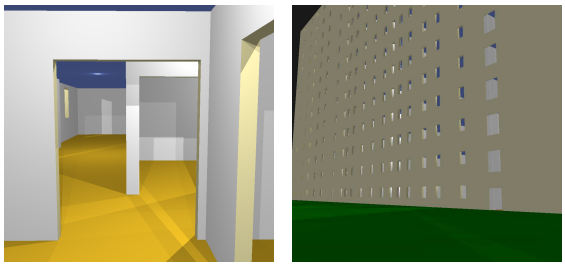
Figure 12: *Convexification d'une pièce.*

méthode naïve. Dans la section 5.2, nous introduisons les arbres de lumière afin d'accélérer une fois encore les temps de calcul. Le tableau 3 montre l'importante amélioration apportée par l'utilisation de ces arbres. En effet, les images sont en moyenne calculées 70,85% plus rapidement qu'en utilisant un PVS classique.

Scènes	Classique	PVS	Arbres de lumière
MAISON	134.45	274,36	439,47
BAT-1	12.39	83,24	156,77
BAT-2	4.4	99,36	163,35

Table 3: *Comparaison des performances (en IPS) sans, avec PVS et avec arbres de lumière sur GPU.*

Les résultats présentés dans le tableau 2 montrent que le nombre d'images calculées par seconde (IPS) chute lorsque les sources lumineuses sont prises en compte. Néanmoins, nous observons que les nombres de rayons lancés par seconde sont peu impactés (*cf.* figure 13).



(a) (b)

Figure 13: *L'augmentation du nombre de sources lumineuses implique la réduction du nombre d'IPS ; en revanche, le nombre de rayons lancés par seconde reste stable ; (a) 124,33 IPS ; 959,64 Mrays/s (6,7 millions de rayons lumineux) ; (b) 7,64 IPS ; 1076,08 Mrays/s (139,8 millions de rayons lumineux) ; à l'extérieur, une grande partie des 2500 lumières est potentiellement visible.*

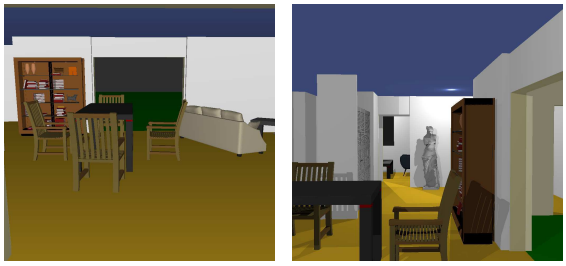
La robustesse de notre structure accélératrice en terme de nombre de rayons lancés par seconde nous laisse penser qu'elle peut être utilisée de manière efficace pour calculer l'éclairement global en environnement architectural.

6.2.3. Prise en compte du mobilier

L'ameublement d'une scène est présenté dans la section 5.3. Après l'ajout d'objets, notre structure accélératrice devient une structure multi-niveaux. Pour traiter efficacement un objet, nous pouvons utiliser n'importe quelle structure accélératrice annexe.

Dans nos expériences, nous utilisons un BVH [RW80, KK86] construit selon l'heuristique SAH [MB90]. Sur l'ensemble de nos scènes nous calculons en moyenne 222,54 IPS (*cf.* figure 14(a)). Logiquement, étant donné que le lancer de rayons proposé par Aila *et al.* [ALK12] n'est pas prévu pour utiliser des clones, seule la plus petite de nos scènes (composées d'environ 1 million de polygones) a pu être comparée. Sur cette scène, approximativement 100 images sont calculées par seconde.

Concernant la prise en compte conjointe du mobilier et de l'éclairage, nous avons mesuré une moyenne de 37,95 IPS (soit 173,76 Mrays/s) sur l'ensemble de nos scènes (*cf.* figure 14(b)). Notons cependant que le rendu avec éclairage direct est alors effectué en utilisant le PVS mais sans l'optimisation apportée par les arbres de lumière. L'implantation des arbres de lumière pour des scènes meublées étant en cours, nous espérons une amélioration des performances à l'image des résultats présentés pour des scènes vides dans le tableau 3.



(a) (b)

Figure 14: *Images meublées et performances : (a) 181,92 IPS ; 190,76 Mrays/s ; (b) prise en compte de l'éclairage ; 40,61 IPS ; 135,54 Mrays/s (2,29 millions de rayons lumineux) ;*

© AFIG 2013.

7. Conclusion et travaux futurs

Dans cet article, nous avons proposé une nouvelle structure accélératrice dédiée au lancer de rayons en environnement architectural. Cette structure est déduite automatiquement à partir d'un modèle topologique [HMDB09]. Les bâtiments sont décrits par une partition de l'espace 3D composée d'éléments d'une épaisseur significative.

Notre structure est une amélioration des cellules-et-passages, ajoutant deux avantages principaux. D'une part, l'utilisation d'un modèle topologique permet de construire automatiquement une structure CeP dont les plans de découpes respectent parfaitement la position des murs. D'autre part, alors que les algorithmes de parcours d'une structure CeP classique n'utilisent que les relations de voisinage entre volumes, nous proposons d'utiliser l'ensemble des relations topologiques disponibles. Ces informations nous permettent d'optimiser la recherche de la face de sortie d'un rayon au sein d'une cellule donnée, et par conséquent d'accélérer l'ensemble du calcul de rendu.

Plusieurs algorithmes de parcours de notre structure accélératrice sont implantés sur CPU et sur GPU. Nos résultats expérimentaux montrent qu'ils sont très efficaces, y compris pour des scènes meublées composées de plusieurs millions de polygones. Ils permettent ainsi un rendu interactif avec éclairement direct au sein de scènes contenant des milliers de sources lumineuses ponctuelles.

Compte tenu des performances en terme de nombre de rayons lancés par seconde, nous pensons que notre structure peut être utilisée pour calculer l'éclairement global d'environnements architecturaux, ou encore dans d'autres types d'applications par exemple en simulation de propagation d'ondes.

Le principal inconvénient de notre structure accélératrice est que les scènes manipulées doivent obligatoirement provenir d'un modeleur à base topologique. Si cela est problématique pour des scènes quelconques, cet aspect pose aussi problème pour le rendu du mobilier. C'est pourquoi nous prévoyons d'appliquer nos idées à n'importe quel type d'environnement, architectural ou non, et décrit uniquement par sa géométrie. Nous pensons utiliser les cartes généralisées pour connecter les éléments géométriques entre eux, et redéfinir la topologie de l'objet. Une première application sera de pouvoir intégrer directement des objets quelconques dans nos scènes, puis de rendre des scènes quelconques.

Remerciements

Nous remercions la Région Poitou-Charentes pour son soutien financier.

Références

[ALK12] AILA T., LAINE S., KARRAS T. : *Understanding the Efficiency of Ray Traversal on GPUs – Kepler and Fermi Addendum*. NVIDIA Technical Report NVR-2012-02, juin 2012.

[Ama84] AMANATIDES J. : Ray tracing with cones. *SIGGRAPH Comput. Graph.* (1984), 129–135.

[App68] APPEL A. : Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference* (1968), AFIPS '68 (Spring), ACM, pp. 37–45.

[ARB90] AIREY J. M., ROHLF J. H., BROOKS JR. F. P. : Towards image realism with interactive update rates in complex virtual building environments. *SIGGRAPH Comput. Graph.. Vol. 24*, Num. 2 (Février 1990), 41–50.

[Ave09] AVENEAU L. : Les coordonnées de Plücker revisitées. *Revue Electronique Francophone de l'Informatique Graphique. Vol. 3*, Num. 2 (janvier 2009), 59–68.

[Ben75] BENTLEY J. L. : Multidimensional binary search trees used for associative searching. *Commun. ACM. Vol. 18*, Num. 9 (septembre 1975), 509–517.

[BW09] BENTHIN C., WALD I. : Efficient ray traced soft shadows using multi-frusta tracing. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 135–144.

[FKN80] FUCHS H., KEDEM Z. M., NAYLOR B. F. : On visible surface generation by a priori tree structures. *SIGGRAPH Comput. Graph.. Vol. 14*, Num. 3 (juillet 1980), 124–133.

[FMH05] FRADIN D., MENEVEAUX D., HORNA S. : Out-of-core Photon-Mapping for Large Buldings. In *Proceedings of EGSR* (June 2005).

[FS05] FOLEY T., SUGERMAN J. : KD-tree acceleration structures for a GPU raytracer. In *Proceedings of HWWS '05* (New-York, USA, 2005), ACM, pp. 15–22.

[FTCP08] FERNÁNDEZ J., TÓTH B., CÁNOVAS L., PELEGRÍN B. : A practical algorithm for decomposing polygonal domains into convex polygons by diagonals. *TOP. Vol. 16*, Num. 2 (2008), 367–387.

[FTI88] FUJIMOTO A., TANAKA T., IWATA K. : Arts : accelerated ray-tracing system. In *Tutorial : computer graphics ; image synthesis*, Grant C. W., Hatfield L., (Eds.). Computer Science Press, 1988, pp. 148–159.

[Gla89] GLASSNER A. S. (Ed.) : *An introduction to ray tracing*. Academic Press Ltd., London, UK, UK, 1989.

[GO10] GOODMAN J., O'ROURKE J. : *Handbook of Discrete and Computational Geometry, Second Edition*. Discrete Mathematics and Its Applications. Taylor & Francis, 2010.

[Hav00] HAVRAN V. : *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.

[HDMB07] HORNA S., DAMIAND G., MENEVEAUX D., BERTRAND Y. : Building 3d indoor scenes topology from 2d architectural plans. In *GRAPP'2007*. (March 2007).

[HH84] HECKBERT P. S., HANRAHAN P. : Beam tracing polygonal objects. *SIGGRAPH Comput. Graph.* (1984), 119–127.

[HMDB09] HORNA S., MENEVEAUX D., DAMIAND G., BERTRAND Y. : Consistency constraints and 3d building reconstruction. *Computer-Aided Design. Vol. 41*, Num. 1 (janvier 2009), 13–27.

© AFIG 2013.

[HSHH07] HORN D. R., SUGERMAN J., HOUSTON M., HANRAHAN P. : Interactive k-d tree gpu raytracing. In *Proceedings of the 2007 symposium on Interactive 3D graphics and games* (New York, NY, USA, 2007), I3D '07, ACM, pp. 167–174.

[KK86] KAY T. L., KAJIYA J. T. : Ray tracing complex scenes. *SIGGRAPH Comput. Graph.. Vol. 20*, Num. 4 (1986), 269–278.

[LG95] LUEBKE D., GEORGES C. : Portals and mirrors : simple, fast evaluation of potentially visible sets. In *Proceedings of the 1995 symposium on Interactive 3D graphics* (1995), I3D '95, pp. 105–ff.

[Lie94] LIENHARDT P. : N-dimensional generalized combinatorial maps and cellular quasi-manifolds. *International Journal on Computational Geometry and Applications. Vol. 4*, Num. 3 (1994), 275–324.

[MAAG12] MORA F., AVENEAU L., APOSTU O., GHAZANFARPOUR D. : Lazy visibility evaluation for exact soft shadows. *Comput. Graph. Forum. Vol. 31*, Num. 1 (Février 2012), 132–145.

[MB90] MACDONALD D. J., BOOTH K. S. : Heuristics for ray tracing using space subdivision. *Vis. Comput.. Vol. 6*, Num. 3 (mai 1990), 153–166.

[MMB98] MENEVEAUX D., MAISEL E., BOUATOUCH K. : A new partitioning method for architectural environments. *Journal of Vizualisation and Computer Animation. Vol. 9*, Num. 4 (Novembre 1998), 195–213.

[Mok] MOKA : Moka, 3d topological modeler. `http://www.sic.sp2mi.univ-poitiers.fr/moka/`.

[MW06] MAHOVSKY J., WYVILL B. : Memory-conserving bounding volume hierarchies with coherent raytracing. *Computer Graphics Forum. Vol. 25*, Num. 2 (2006), 173–182.

[PBMH02] PURCELL T. J., BUCK I., MARK W. R., HANRAHAN P. : Ray tracing on programmable graphics hardware. *ACM Trans. Graph.. Vol. 21*, Num. 3 (juillet 2002), 703–712.

[PGSS07] POPOV S., GÜNTHER J., SEIDEL H.-P., SLUSALLEK P. : Stackless kd-tree traversal for high performance gpu ray tracing. *Computer Graphics Forum. Vol. 26*, Num. 3 (2007), 415–424.

[RSH05] RESHETOV A., SOUPIKOV A., HURLEY J. : Multi-level ray tracing algorithm. *SIGGRAPH Comput. Graph.* (2005), 1176–1185.

[RW80] RUBIN S. M., WHITTED T. : A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH Comput. Graph.. Vol. 14*, Num. 3 (juillet 1980), 110–116.

[STN87] SHINYA M., TAKAHASHI T., NAITO S. : Principles and applications of pencil tracing. *SIGGRAPH Comput. Graph.* (1987), 45–54.

[TS91] TELLER S. J., SÉQUIN C. H. : Visibility preprocessing for interactive walkthroughs. *SIGGRAPH Comput. Graph.. Vol. 25*, Num. 4 (juillet 1991), 61–70.

[Whi80] WHITTED T. : An improved illumination model for shaded display. *Commun. ACM. Vol. 23*, Num. 6 (juin 1980), 343–349.

[WIK*06] WALD I., IZE T., KENSLER A., KNOLL A., PARKER S. G. : Ray tracing animated scenes using coherent grid traversal. *SIGGRAPH Comput. Graph.* (2006), 485–493.

[WK06] WÄCHTER C., KELLER A. : Instant ray tracing : The bounding interval hierarchy. In *Proceedings of the 17th Eurographics conference on Rendering Techniques* (2006), EGSR'06, pp. 139–149.

[WSBW01] WALD I., SLUSALLEK P., BENTHIN C., WAGNER M. : Interactive rendering with coherent ray tracing. In *Computer Graphics Forum* (2001), pp. 153–164.

Appendix A: Coordonnées de Plücker

Les coordonnées de Plücker ont été présentées par Julius Plücker en 1865. Elles permettent de représenter des sous-espaces linéaires de dimension k dans un espace projectif de dimension n. En informatique graphique, ces coordonnées sont utilisées pour représenter des droites orientées de l'espace affine 3D dans un espace projectif de dimension 5 $\mathbb{P}^5$.

Ainsi, une droite orientée l passant par deux points A et B est définie par un sextuplé de coordonnées $\Pi_l = \{u = b - a : v = a \times b\}$, où les vecteurs 3D a et b correspondent respectivement aux positions des points A et B. Ainsi, le vecteur u représente la direction de la droite l et v sa position (ou moment mécanique) [Ave09].

Les coordonnées de Plücker sont utilisées dans divers problèmes de visibilité [MAAG12]. Ici, nous les utilisons afin de déterminer l'orientation relative entre deux droites orientées via l'opérateur *side*. Cet opérateur est défini pour deux droites orientées l et l' tel que $side(\Pi_l, \Pi_{l'}) = u.v' + u'.v$. Si le résultat est négatif, l tourne autour de l' dans le sens des aiguilles d'une montre (l passe à gauche de l') ; s'il est positif, l tourne autour de l' dans le sens trigonométrique (l passe à droite de l') ; sinon, le résultat est nul, l et l' sont coplanaires *i.e.* elles se croisent au moins à l'infini (*cf.* figure 15).

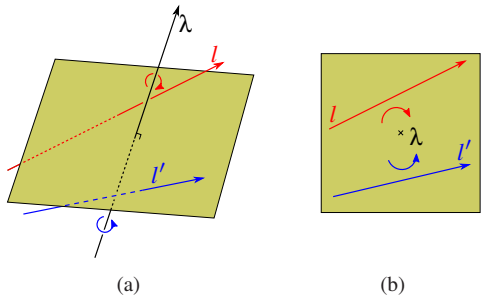


Figure 15: *Orientation relative de droites dans $\mathbb{R}^3$: l passe à gauche de λ (sens horaire), le résultat de $side(\Pi_l, \Pi_\lambda)$ est négatif ; l' passe à droite de λ (sens trigonométrique), le résultat de $side(\Pi_{l'}, \Pi_\lambda)$ est positif ; (a) représentation des droites λ, l et l' en 3D ; (b) projection des mêmes droites sur un plan orthogonal à λ.*

© AFIG 2013.