Maintenance de la visibilité d'un point mobile, et applications

Samuel Hornus

Université Grenoble I – Joseph Fourier

Maintenance de la visibilité depuis un point mobile, et applications

Samuel HORNUS

Thèse présentée pour l'obtention du titre de Docteur de l'Université Joseph Fourier, spécialité informatique. Arrêté ministériel du 5 juillet 1984 et du 30 mars 1992.
Préparée au sein du laboratoire ARTIS-GRAVIR/IMAG-INRIA, UMR CNRS C5527.
Soutenue le 22 mai 2006.

Composition du jury :

Georges-Pierre BONNEAU	–	président
Frédo DURAND	–	rapporteur
Hazel EVERETT	–	rapporteur
John C. HART	–	examinateur
Michel POCCHIOLA	–	examinateur
Claude PUECH	–	directeur de thèse

Introduction

La synthèse d'images et la géométrie algorithmique sont deux disciplines récentes. La première est née dans les années 60 avec les travaux de Sutherland [136] sur le dessin assisté par ordinateur. La deuxième naît au milieu des années 70, quand Shamos et Hoey [127, 128] présentent un algorithme de construction du diagramme de Voronoi et améliorent, en l'utilisant, les algorithmes de résolution de nombreux problèmes connus. Dès le début en synthèse d'images, et un peu plus tard en géométrie algorithmique, des questions liées à la notion de visibilité apparaissent.

La notion de visibilité est indissociable de la synthèse d'images. De pair avec la modélisation des propriétés optiques de la matière, ce sera pendant de nombreuses années un des domaines de la synthèse d'images les plus étudiés, par exemple à travers l'étude des problèmes de lancer de rayons, du calcul de facteurs de forme, ou bien du calcul rapide des parties cachées/visibles d'une scène en 3D ; pour pouvoir enfin générer sur un ordinateur une image indistinguable d'une photographie. Très généralement, les calculs de visibilité consistent à déterminer l'ensemble des « rayons lumineux » émis par un objet et reçus par un autre ensemble d'objets. On cherche souvent seulement à déterminer l'existence de tels rayons, plutôt que leur distribution. Les variations sur ce problème tournent autour de la forme et la dimension des objets en « relation de visibilité », ou bien sur le type de résultat désiré : approximatif, conservatif, exact, *etc.*Un des problèmes les plus étudiés est la détermination des surfaces *visibles* depuis un point de l'espace donné. Souvent aussi, on cherche à déterminer l'ensemble des objets visibles depuis une région de l'espace, comme un parallélépipède. Ce dernier type de problème est d'une importance cruciale pour les applications interactives, dont les jeux vidéo sont le représentant principal.

En géométrie algorithmique, les chercheurs motivés par les questions que l'on se pose en synthèse d'images, attaquent d'autres problèmes liés à la visibilité, notamment dans le plan (quelle partie de ce polygone est visible par cette arête ? ou par ce point ?). La littérature sur les problèmes de visibilité abonde dans les deux domaines. De fait, la notion de visibilité est omniprésente en géométrie, on la retrouve donc également dans d'autres domaines scientifiques. Nous référons le lecteur à l'état de l'art écrit par Frédo Durand [43].

Examinons un problème classique. Nous souhaitons visualiser sur notre écran les images générées par une caméra « virtuelle » placée dans une scène également virtuelle, comme si nous regardions un film enregistré sur une cassette vidéo. On supposera que la scène est définie par un ensemble de polygones dans l'espace. Il nous faut donc trouver un algorithme capable, une fois la position de la caméra et les polygones de la scène connus, de déterminer l'ensemble des parties de la scène qui sont visibles sur la « pellicule » virtuelle de la caméra, et de dessiner sur l'écran ces morceaux de polygones. Nous souhaitons de plus avoir l'illusion d'un mouvement fluide et requérons donc que l'image soit mise à jour trente fois par seconde. Nous devons alors imaginer

un algorithme capable de tirer parti de toutes les informations connues pour pouvoir supporter une telle cadence.

Une observation importante a été faite depuis longtemps : lorsque la caméra (ou les objets de la scène) bouge à une vitesse raisonnable, deux images générées successivement se ressemblent fortement. Par exemple, un objet est un peu agrandi sur l'image ; l'autre s'est légèrement déplacé vers la droite, mais la relation entre les deux images est évidente. On parle alors de la notion de *cohérence temporelle*, qui désigne le haut degré de ressemblance de deux états rapprochés dans le temps.

L'existence de cette cohérence temporelle doit être exploitée pour améliorer la performance de notre caméra virtuelle. (Par abus de langage, nous identifions la caméra à l'algorithme utilisé pour calculer une image.) Un des moyens de l'exploiter est induit par l'observation suivante. Si la scène visible change peu quand la caméra bouge peu, alors l'ensemble des objets visibles depuis n'importe quel point d'un voisinage de la caméra devrait être seulement un peu plus grand que l'ensemble des objets visibles depuis la caméra. On exploite généralement cette observation en partitionnant l'espace des positions que peut prendre la caméra en cellules de tailles variables, et en précalculant pour chaque cellule l'ensemble des objets *potentiellement visibles*, c'est-à-dire une surestimation la plus juste possible, de l'ensemble des objets visibles depuis au moins un point de la cellule [139, 142].

L'autre manière d'exploiter la cohérence temporelle, plus directe, est de structurer la scène que nous souhaitons « filmer » de telle sorte qu'il devienne possible, étant données la scène visible $V(t)$ au temps t et la nouvelle position de la caméra au temps $t + \delta t$, de déterminer rapidement les changements à effectuer pour obtenir la scène visible $V(t + \delta t)$ au temps $t + \delta t$. En supposant que notre caméra fonctionne ainsi, elle sera optimale si le temps nécessaire pour calculer $V(t + \delta t)$ à partir de $V(t)$ est proportionnel (à un facteur logarithmique près) au nombre de changements de visibilité qui ont lieu lors du déplacement de la caméra dans l'intervalle de temps $[t, t + \delta t]$. L'exploitation de la cohérence temporelle de cette manière est très difficile. Ainsi, aucune caméra « optimale » n'est connue pour le cas d'une scène en 3D dans laquelle la caméra n'a pas de contrainte de mouvement. De plus, aucune caméra « optimale » n'est connue en 2D si les objets de la scène peuvent bouger.

En 2D, le *Corner Arc Algorithm* de Hall-Holt [67] implémente une telle caméra virtuelle optimale, dans une scène composée d'objets convexes simples, disjoints et statiques. Notons que cet algorithme fonctionne également avec des objets en mouvement, mais il est alors nécessaire de maintenir une certaine partition de l'espace libre, dont le coût de maintenance n'est pas directement lié au nombre de changements de visibilité, et peut lui être supérieur. Pour illustrer la complexité d'un tel algorithme, examinons son « historique ». Le *Corner Arc Algorithm* est fondé sur deux grandes idées géométriques apparues récemment et qui forment également le cadre de cette thèse.

Citons d'abord la considération de l'ensemble des rayons « lumineux » dans la scène comme objet d'« études ». En particulier, cela a abouti à la notion de *complexe de visibilité* [119] qui décrit, dans le plan, la structure interne d'un tel ensemble. Le complexe de visibilité fournit un cadre dans lequel on peut exprimer le fonctionnement d'algorithmes de calculs de visibilité.

Ensuite, le *Corner Arc Algorithm* est présenté sous la forme d'une *structure de données cinétique*. Les structures de données cinétiques (*kinetic data structures*, ou KDS) sont un formalisme pour la conception et l'analyse de la maintenance continue

d'attributs géométriques définis sur un ensemble d'objets *mobiles*, c'est-à-dire de primitives simples (points, triangles, polygones, *etc.*) animées d'un mouvement continu dans le temps. Une KDS pourra être utilisée, par exemple, si l'on souhaite maintenir l'enveloppe convexe d'un ensemble de points mobiles dans le plan, dont on connaît les trajectoires, au moins à court terme.

Le complexe de visibilité dans le plan a prouvé son utilité pour la mise au point de nouveaux algorithmes de calcul de visibilité [9, 67, 119, 125]. De plus, les pseudo-triangles et pseudo-triangulations utilisés d'abord pour la construction du complexe de visibilité, se sont vus promus au rang d'objets d'investigation de premier plan, en géométrie combinatoire (voir les travaux d'Ileana Streinu [134] [1]), ou pour la détection de collisions dans le plan (voir les travaux de Bettina Speckmann, par exemple [85]). Il semblait donc naturel d'explorer plus avant le complexe de visibilité d'un ensemble de convexes dans l'espace, introduit par Frédo Durand [44] puis étudié plus en détail par Xavier Goaoc [60]. J'étudierai dans ce manuscrit certaines propriétés topologiques simples du complexe de visibilité 3D, permettant la description d'un algorithme de construction de ce dernier, dans le cas où la scène est composée de polytopes convexes disjoints. Alors que l'algorithme de construction du complexe proposé par Goaoc manipule des ensembles algébriques, celui que je propose est une construction géométrique plus classique, utilisant l'algorithme de construction du squelette de visibilité par balayage, proposé également par Goaoc.

Contributions de cette thèse

Le manuscrit commence par un survol rapide de certaines structures utilisées pour les calculs de visibilité (Chapitre 1) et de divers problèmes liés à la visibilité (ainsi que leur résolution, Chapitre 2). J'y rends compte de problèmes « théoriques » et également de problèmes pratiques (où l'on souhaite dessiner une scène en 3D sur un écran d'ordinateur, à une fréquence élevée).

Je me suis intéressé, pour l'élaboration de cette thèse, au problème de maintenance exacte de la visibilité d'un point de vue mobile dans l'espace. L'objectif à long terme (et non atteint) de ces travaux est la conception d'un tel algorithme optimal dans la lignée d'un autre algorithme dû à Hall-Holt pour la maintenance de la visibilité d'un point mobile dans le plan, c'est-à-dire sa généralisation à un espace 3D.

Au Chapitre 3, j'aborde un problème de visibilité très simple : la maintenance du tri radial d'un ensemble de n points autour d'un point « de vue » mobile dans le plan. Ce travail a été effectué en collaboration avec Olivier Devillers, Vida Dujmović, Hazel Everett, Sue Whitesides et Steve Wismath. Il propose un algorithme optimal, coûtant $O(\log n)$ en temps à chaque changement dans l'ordre des points, et le même coût lorsque la trajectoire du point de vue change.

L'algorithme de Hall-Holt cité plus haut — le *visible zone algorithm* — est décrit en détail au Chapitre 4. Ce pour deux raisons. Premièrement, l'algorithme est relativement complexe à appréhender, de part la quantité de détails qu'il comprend, et la difficulté de représensation visuelle du complexe de visibilité 2D dans lequel l'algorithme se décrit naturellement. J'ai donc voulu réexpliquer cet algorithme dans un but pédagogique, en incluant donc une figure pour chaque notion importante. Deuxièmement, la preuve du lemme principal de l'algorithme, présentée par Hall-Holt est longue et

[1] http://maven.smith.edu/~streinu/Research/PseudoTriang/pseudotriang.html

difficile. Je propose au Chapitre 4 une preuve plus simple d'un lemme (lemme 4.12, page 54) qui implique le lemme principal.

Le Chapitre 5 aborde le problème de la maintenance du polyèdre de visibilité d'un point de vue mobile dans une scène en 3D composée de polytopes disjoints. J'y décris une certaine partition de l'espace libre (l'espace non occupé par un objet). appelée *subdivision* ou *décomposition radiale*. Les faces de cette décomposition seront projetées sur la surface d'une « sphère des directions » puis triangulées pour pouvoir être maintenues lors du mouvement du point de vue ou des objets eux-mêmes. La construction de la décomposition radiale en 3D fait appel à une structure cinétique de localisation d'un point dans le plan décrite au Chapitre 6.

Le Chapitre 7 présente un algorithme de construction du complexe de visibilité d'un ensemble de polytopes disjoints disposés dans l'espace. Il utilise une propriété de connexité de la frontière des cellules du complexe qui est démontrée dans ce même Chapitre. Ce travail a été fait en collaboration avec Maxime Wolff, et fût initié en collaboration avec Florent Duguet et Frédo Durand.

Les chapitres ci-dessus forment l'ensemble de mes contributions traitant de calculs géométriques exacts, et donc plutôt théoriques. Dans la troisième partie de ce manuscrit, je présente deux travaux reliés plus directement à la *pratique* de la génération d'images d'un monde virtuel tridimensionnel en temps interactif. Au Chapitre 8, je propose un nouvel algorithme pour dessiner rapidement les ombres dures portées sur une scène par des objets polyédriques éclairés par une source lumineuse ponctuelle. Il reprend un algorithme connu depuis 1991 (*stencil shadow volume*) et montre comment on peut le rendre robuste à tous les cas de figure. Ce travail a été effectué en collaboration avec Jared Hoberock, Sylvain Lefebvre et John C. Hart.

Dans le cas d'une scène architecturale, il existe une méthode très populaire pour résoudre le problème classique présenté plus haut de la caméra virtuelle mobile. Elle consiste à découper préalablement la scène en un graphe de cellules et de portails. Les cellules correspondant grossièrement aux pièces du bâtiment, et les portails, aux portes et fenêtres reliant ces pièces. Les portails peuvent alors être utilisés au moment de dessiner la scène pour déterminer rapidement quelles parties de la scène doivent être dessinées. Un des problèmes que pose cette méthode est la création de ce graphe, qui est le plus souvent construit « à la main » par les infographistes. Je propose au Chapitre 9 une méthode automatique de création d'un tel graphe pour une scène quelconque. Ce travail a été effectué en collaboration avec Sylvain Lefebvre.

Remerciements

Mes remerciements vont en premier lieu à Claude Puech, qui, après une brève entrevue dans « le 5ème », m'a permis de venir passer quelques années à iMAGIS[2] sous sa direction. Claude m'a encouragé à voyager partout à la découverte du monde de la recherche en informatique théorique. Pour paraphraser mon prédécesseur, j'ai beaucoup apprécié les discussions que nous avons eu entre deux voyages à Paris et à Urbana-Champaign.

Je suis honoré que John C. Hart et Michel Pocchiola aient accepté de participer à mon jury de thèse. Que Hazel Everett et Frédo Durand reçoivent mes sincères remerciements pour avoir participé à mon jury en tant que rapporteurs. Merci à George-Pierre Bonneau pour avoir accepté et joué à merveille son rôle de président du jury.

[2] Équipe dédoublée en ARTIS et ÉVASION depuis 2003.

Je remercie chaleureusement John C. Hart pour m'avoir accueilli par deux fois dans la belle contrée du maïs, à UIUC. Merci à Jared et à l'International Illini pour les ballades à Chicago et tout le reste. Je garde un souvenir tout particulier du *Chicago-trip con frenchies*. Frédo Durand m'a plusieurs fois accueilli dans son fief, au MIT. À chaque fois avec un peu de temps pour discuter, et de très sympathiques soirées sur le toit de son immeuble. Un grand merci s'impose, que j'adresse également à tous les *students* du *Graphics Group* et ses visiteurs. Merci à Sylvain Lazard et Hazel Everett, qui m'ont invité régulièrement aux *workshops* McGill/INRIA ainsi qu'à Nancy, où l'on s'est rendu compte que le problème Olaf-3D est vraiment compliqué. Merci également à Xavier Goaoc pour avoir partagé sa science du complexe.

J'ai toujours en tête (mais pas en détail) les discussions que j'ai eues avec Marie-Paule Cani et Jean-Dominique Gascuel, à Boisset. J'ai grâce à eux découvert que mes « démos graphiques du lycée » pouvaient devenir des sujets de recherche plus sérieux. À propos de démos, je salue au passage Florent Duguet et Maxime Wolff. `mov ax, 13h; int 10h;` *Amen*. Florent m'a tout appris sur le *texture mapping* sans correction de perspective. Maxime m'a fait découvrir l'ensemble de Mandelbrot qu'il cherchait à dessiner rapidement sur son PC, et plus tard, m'a fait ouvrir un œil sur la topologie algébrique.

Mes co-thésards ont formé une équipe formidable. Ils n'ont malheureusement pas voulu m'attendre pour soutenir leur thèse et sont déjà partis vers de brillantes aventures. Merci à Sylvain Lefebvre, Sylvain Paris et Stéphane Grabli, pour les soirées code/cafés/clopes, les 3h du mat' à l'Archange et les soirées jeux/pizza/mousse et rigolades (dans le désordre, bien entendu).

Pour la préparation de ce manuscrit, je remercie vivement mes relecteurs valeureux, Joëlle Thollot, Xavier Décoret, Elmar Eisemann, Michel Pocchiola et particulièrement Mélissa Rondet qui a lu plusieurs fois le manuscrit pour y débusquer coquilles et autres étourderies. Mélissa m'a apporté une aide inestimable pour surmonter ma paresse et mes déprimes passagères.

Je profite de ces remerciements pour saluer Caroline Larboulette, Matthieu Cunzi et le 17, Laure Heïgéas, Emmanuel Turquin, Stéphane Guy, Alexis Angelidis, Gilles Debunne, Joëlle, Philippe Decaudin, Fabrice Neyret ainsi que *the boss*, François Sillion, ARTIS et ÉVASION. Merci pour tous ces bons moments passés ensemble.

Last but not least, rien de tout cela n'aurait pu arriver sans le soutien sans faille de ma famille, Lætitia, Henri, Timothée, Jérémie et Quentin. Je leur en suis profondément reconnaissant.

Dans la version électronique de ce document, la plupart des références sont « cliquables » : citations, références aux Figures, Sections et Chapitres, notes de bas de pages, etc.

Table des matières

Première partie

État de l’art

Chapitre 1

Objets manipulés

Ce chapitre sert à l'introduction des notions géométriques que nous utiliserons dans la suite du manuscrit. On commence par définir la relation de visibilité existant entre deux objets dans l'espace, ainsi que la notion d'observateur et de carte de visibilité. Suit ensuite l'exposition de différentes structures de données construites autour de la notion de visibilité : le graphe de visibilité et le complexe de visibilité. Nous introduisons enfin un cadre de travail pour la conception et l'analyse d'algorithmes opérant sur des données géométriques mobiles : les structures de données cinétiques.

1.1 Rayons, observateurs, carte de visibilité

Nous commençons par définir la notion de visibilité entre deux objets. Plaçons quelques objets dans l'espace euclidien. Nous dirons que les objets A et B se voient, ou sont *mutuellement visibles*, si il existe un segment $[a, b]$ de l'espace tel que a soit un point de A, b soit un point de B et l'intérieur (a, b) du segment ne touche aucun autre objet de l'espace. Si aucune paire de points a, b vérifiant ces propriétés ne peut être trouvée, alors A et B ne se voient pas, ou sont *mutuellement invisibles*. Cette notion basique est illustrée sur la figure 1.1.

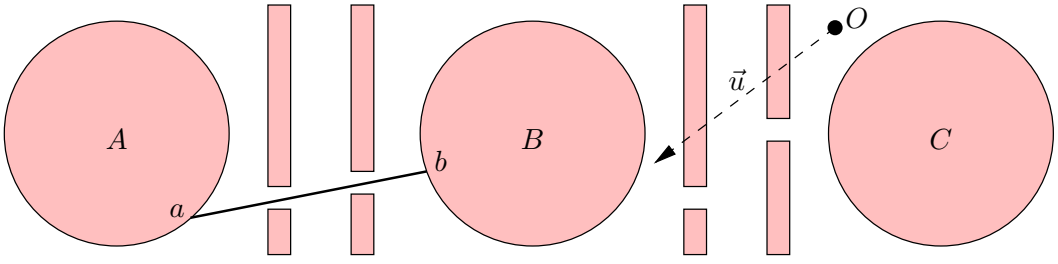


FIG. 1.1. A et B sont mutuellement visibles tandis que B et C sont mutuellement invisibles.

Formellement, nous définissons un **rayon** comme un segment de droite décrit par la paire $(O, \vec{u})$ où $O \in \mathbb{R}^n$ est l'origine du rayon et $\vec{u} \in \mathbb{S}^{n-1}$ est sa direction de propagation. Le rayon est l'ensemble des points P de l'espace tels que le segment ouvert OP n'intersecte aucun objet de la scène.

Un tel rayon n'est pas tout à fait l'analogue d'un rayon lumineux puisqu'un rayon *lumineux* prend physiquement source là où il y a de la matière et non en n'importe quel point de l'espace. Nous verrons plus tard la notion de *segment libre maximal*,

reliant deux points de la surface des objets, qui correspond davantage à la notion de rayon lumineux.

Nous plaçons ensuite un observateur dans notre scène. Un **observateur** est simplement un ensemble de points de l'espace. Nous considérerons en général que l'observateur se trouve à l'intérieur de l'**espace libre** $\mathcal{F}$ défini comme le complémentaire de l'union des objets introduits dans l'espace (observateur exclu !). Lorsque nous souhaiterons calculer l'ensemble des parties d'objets visibles par un observateur, nous considérerons que ce dernier est « simple » : un point (un *point de vue*), un segment de droite, un triangle, *etc.*

1.1.1 Vue, carte de visibilité.

De manière informelle, la **vue** d'un observateur est l'ensemble des rayons dont l'origine est un point de l'observateur, **ainsi** que la donnée, pour chacun de ces rayons, de l'objet qu'il touche. Si un rayon ne touche aucun objet (c'est donc une demi-droite), on dit qu'il touche l'*infini*, qu'il *part à l'infini*, qu'il *voit l'infini* ou le *ciel bleu* (qu'on notera ∞).

Cette définition est bien sûr peu pratique et il nous faut resteindre la forme de l'observateur pour pouvoir structurer sa vue, puis la calculer.

Point de vue dans le plan. Posons un point de vue V dans l'espace libre du plan euclidien peuplé d'un nombre fini d'objets (segments, convexes, polygones). Les rayons de la vue de V peuvent être paramétrés par leur direction de propagation $\theta \in [0, 2\pi)$ et leur union forme un sous-ensemble étoilé du plan : $\mathcal{V}_V$. La frontière de $\mathcal{V}_V$ alterne des portions de la frontière d'objets (les objets visibles par V) et des segments « alignés » avec V, qui correspondent à des changements de visibilité. Un tel changement de visibilité correspond à un rayon d'angle θ tangent à l'un des deux objets visibles selon des angles $\theta \pm \varepsilon$, pour ε arbitrairement petit.

Nous pouvons ainsi structurer la vue d'un point dans le plan à l'aide d'une liste circulaire alternant le nom de l'objet visible et l'identification de la tangente responsable du changement de visibilité. Voir la figure 4.1 page 40.

Le calcul de la visibilité d'un point dans le plan peuplé d'un polygone simple a été l'un des premiers problèmes abordés en géométrie algorithmique [24] et fait toujours l'objet de recherches [149, pour des polygones à trous].

Point de vue dans l'espace. Considérons par exemple un ensemble de polygones dans l'espace euclidien $\mathbb{E}^3$, et plaçons un point de vue V dans l'espace libre. À nouveau, l'ensemble des rayons de la vue V peut se paramétrer simplement par leur direction $u \in \mathbb{S}^2$ puisque leur origine est commune. La vue de V pourra donc être modélisée par un graphe planaire sur la sphère $\mathbb{S}^2$, pour lequel chaque arête est un arc d'équateur correspondant à des rayons de $\mathcal{V}_V$ tangents à un polygone. Dans ce graphe, chaque face correspond à un ensemble maximalement connexe de directions dans lesquelles V voit le même polygone.

En pratique, on a plus souvent besoin d'afficher une vue de la scène sur l'écran, et seule une partie de la vue de V est nécessaire : il s'agit du « frustum de vue », un ensemble de rayons contenus dans une pyramide à base rectangulaire et de sommet V. Ainsi, on définit le plus souvent la carte de visibilité comme une partition d'un rec-

tangle « image » (correspondant à l'écran d'affichage) en faces polygonales à l'intérieur desquelles la visibilité est constante.

Le problème du calcul de la carte de visibilité est un problème fondamental en synthèse d'images et a été abordé dès l'apparition de moyens techniques d'affichage (voir le Chapitre 2).

Que ce soit en deux ou trois dimensions, la vue d'un « point de vue » est assez facile à visualiser. Dans le plan, et si les objets sont linéaires, elle correspond à un polygone étoilé, dit « de visibilité ». En 3D, dans un espace peuplé d'objets polyédriques, la vue d'un point peut se visualiser sous la forme d'un polyèdre « de visibilité », égal à l'union de ses rayons.

Lorsque la dimensionnalité de l'observateur augmente, il en va de même pour celle de sa vue. Ainsi, la vue d'une courbe sera décrite à l'aide de 3 paramètres, et celle d'un observateur de dimension 2 ou 3, à l'aide de 4 paramètres. La vue d'un tel objet est bien plus complexe à visualiser, ce qui a pour conséquence d'accroître la difficulté de création d'algorithmes pour le calcul de telles vues. Pour exemple, le lecteur est invité à essayer de « structurer » visuellement, dans l'espace, la vue d'un observateur triangulaire, parmi d'autres triangles.

Il est alors fondamental de réaliser que la « matière » que l'on travaille, dans les problèmes de visibilité, est l'ensemble des droites de l'espace, ou bien l'ensemble des rayons, tels que définis plus haut. Nous préciserons cette notion plus loin.

Après paramétrisation de l'espace des droites (certaines paramétrisations sont proches d'un véritable espace dual, on parlera donc souvent de dualisation), on pourra alors exprimer l'ensemble des rayons, ou des droites, passant par un certain objet, comme un ensemble de points dans cet espace paramétrique. Nous référons le lecteur au chapitre suivant pour illustrer l'utilisation de la dualité pour résoudre de nombreux problèmes de visibilité (ou plus généralement des problèmes géométriques).

Les questions de visibilité n'interviennent pas seulement dans les problèmes de calcul d'une vue. Par exemple, le calcul efficace d'un plus court chemin entre deux points, dans un espace peuplé d'obstacles, fait un usage essentiel des relations de visibilité mutuelle entre les obstacles. Dans le domaine de la synthèse d'images, le calcul d'images réalistes nécessite de coûteux calculs de visibilité entre les points visibles par la caméra et les sources lumineuses présentes dans la scène, ou bien entre les éléments constitutifs de la scène lorsque les inter-réflections de la lumière sont prises en compte [116].

Nous décrivons ci-dessous deux structurations des relations de visibilité existant entre des objets disposés dans l'espace : le graphe et le complexe de visibilité.

1.2 Graphe de visibilité – planification de mouvement

Planifier le mouvement d'un point pour le faire aller d'un point à un autre tout en évitant les obstacles (connus et fixes) est un problème de base en robotique. Lorsque les obstacles sont polygonaux (dans le plan), le plus court chemin entre a et b est une ligne brisée dont les sommets internes sont des sommets d'obstacles [36, Chapitre 15].

Il est avantageux de précalculer un *graphe de visibilité* (voir figure 1.2) dont les sommets sont ceux des obstacles polygonaux, et les arêtes relient toutes les paires de sommets mutuellement visibles. Les arêtes sont ensuite étiquetées par leur longueur.

Muni de ce graphe, le calcul de la plus grosse partie du plus court chemin s'effectue alors à l'aide d'un calcul de plus court chemin sur un graphe [34].

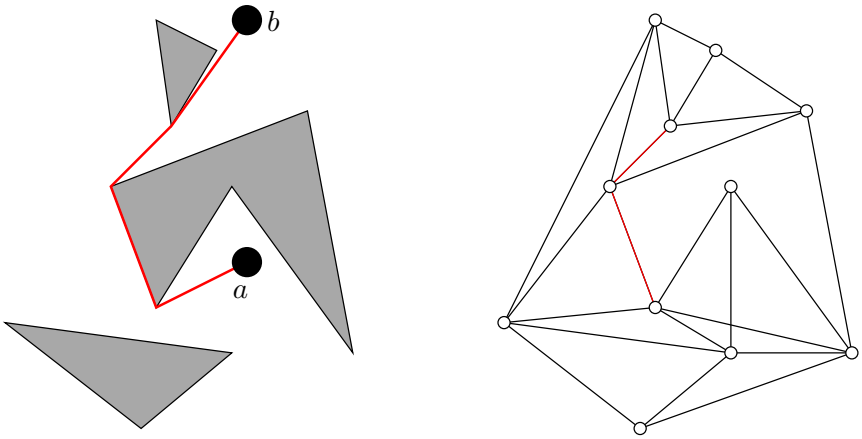


FIG. 1.2. À gauche, en rouge, le plus court chemin entre les points a et b. **À droite**, le graphe de visibilité de la scène décrite à gauche. Remarquer que le plus court chemin contient deux arêtes du graphe de visibilité.

Pour construire un tel graphe de visibilité, on procède généralement sommet par sommet. Pour chaque sommet s, on calcule l'ensemble des sommets visibles, à l'aide d'un balayage radial autour de s. La construction du graphe de visibilité peut ainsi se faire en temps $O(n^2 \log n)$ [36, Théorème 15.4].

Pour calculer le plus court chemin entre deux points a et b de l'espace libre, ces deux points sont d'abord ajoutés comme sommet dans le graphe de visibilité (en temps $O(n \log n)$), puis un algorithme de calcul du plus court chemin entre a et b sur le graphe est lancé (par exemple l'algorithme de Dijkstra).

1.2.1 Cas des objets lisses convexes

La notion de graphe de visibilité peut aussi être généralisée au cas d'obstacles lisses, pour lesquels tout point du bord admet une tangente unique. Il n'y a plus de sommet, mais nous observons que certains segments de droites tangents à deux obstacles peuvent jouer le rôle d'arêtes du graphe.

Une « bitangente libre » est un segment de droite tangent à un obstacle en chacune de ses extrémités, et contenu dans l'espace libre. Chaque bitangente libre induit deux sommets du graphe (ses extrémités, points de tangence). L'ensemble de ces sommets décompose les bords de chaque obstacle en une série de segments de courbes, que nous ajoutons aux bitangentes libres pour former l'ensemble des arêtes du graphe (voir Figure 1.3).

Pocchiola et Vergter [118] proposent un algorithme (le Greedy Flip Algorithm) de construction de tels graphes de visibilité en temps $O(n \log n + m)$ où n est le nombre d'objets et m la taille finale du graphe ($m = \Omega(n)$ et $m = O(n^2)$).

Cet algorithme a été implémenté dans le cadre de la bibliothèque d'algorithmes géométriques CGAL [22] par Pierre Angelier [8]. Notons que cette implémentation permet également le calcul du graphe de visibilité de scènes comportant aussi des segments disjoints (sauf éventuellement en leurs sommets).

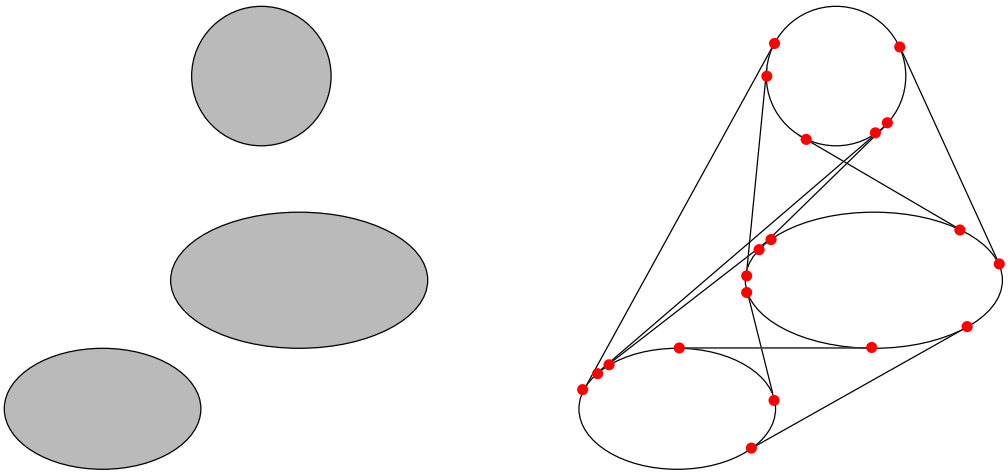


FIG. 1.3. **À gauche**, trois objets lisses dont le graphe de visibilité est représenté **à droite**.

Éléments structurants de dimension supérieure. Nous avons vu dans la définition du polygone de visibilité d'un point dans le plan que les rayons tangents jouent un rôle important, comme lieu de changement de visibilité. En ignorant le point d'origine de chaque rayon, nous pouvons remarquer que l'ensemble des rayons tangents à un même objet est de dimension 1 : chaque point du bord d'un objet lisse définit une unique droite tangente, support de rayons tangents.

Pourquoi ignorer l'origine des rayons ? Si deux rayons partagent la même droite support, la même direction, et sont tels que leurs origines soient mutuellement visibles, alors il est clair que chacun voit les deux mêmes objets *devant* (dans la direction du rayon) et *derrière* (dans la direction opposée). Cette observation a mené Pocchiola et Vegter [119] à considérer l'ensemble des rayons du plan (ensemble de dimension $3 = \dim(\mathbb{R}^2 \times \mathbb{S}^1)$) comme un objet d'étude à part entière et ont consacré certains de leurs travaux à structurer cet ensemble à l'aide des rayons tangents et bitangents. Ces travaux ont abouti à la notion importante de *complexe de visibilité*.

1.3 Le complexe de visibilité

Informellement, le complexe de visibilité considère les rayons « quasi-identiques » comme équivalents, et partitionne l'ensemble de ces « classes d'équivalence » en cellules connexes de classes de rayons ayant la même visibilité. Cette structure a d'abord été définie dans le plan [119], puis a été décrite dans l'espace par Durand *et al.* [44], qui ont également prouvé l'utilité du squelette de visibilité 3D, un sous-ensemble du complexe de visibilité 3D.

1.3.1 Le complexe dans le plan

Le complexe de visibilité de Pocchiola et Vegter est défini pour un ensemble d'objets convexes, lisses et deux à deux disjoints. Topologiquement, c'est un complexe cellulaire [144] qui structure l'ensemble des rayons libres du plan. Ces *rayons libres* sont simplement les rayons du plan dont l'origine appartient à l'espace libre.

Tout d'abord, une relation d'équivalence sur l'ensemble $\mathfrak{R}$ des rayons est introduite pour identifier les rayons quasi-identiques : soient $a = (A \in \mathbb{R}^2, \vec{u} \in \mathbb{S}^1)$ et $b = (B, \vec{v})$

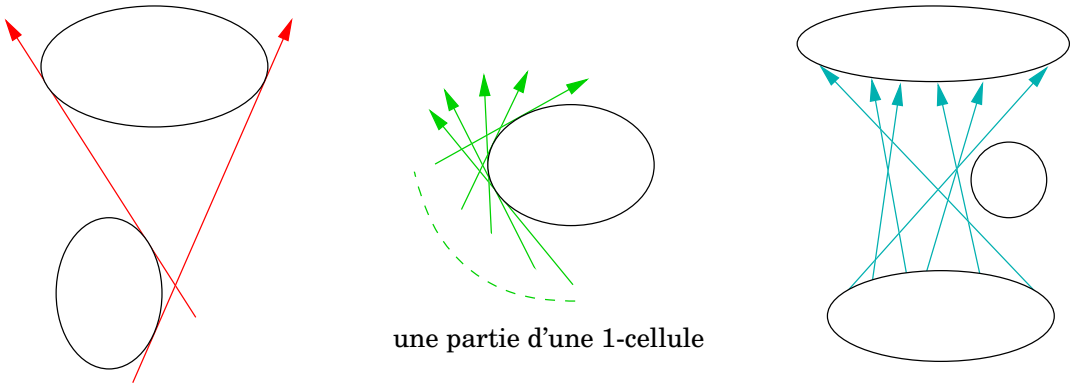


FIG. 1.4. Illustration des cellules de dimensions 0, 1 et 2 du complexe de visibilité d'objets lisses dans le plan. Pour chaque cellule, on a dessiné quelques segments libres maximaux qu'elle contient.

deux rayons libres. Nous définissons la relation d'équivalence suivante :

$$a \equiv b \iff \begin{cases} \vec{u} = \vec{v}, \\ \exists t \in \mathbb{R} \mid B = A + t\vec{u} \text{ et} \\ A \text{ et } B \text{ sont mutuellement visibles.} \end{cases}$$

En prenant le quotient de l'ensemble des rayons libres par la relation $\equiv$, nous obtenons l'espace $\mathfrak{V} = \mathfrak{R}/\equiv$ sur lequel est structuré le complexe de visibilité : $\mathfrak{V}$ est structuré en un complexe cellulaire en regroupant les rayons libres ayant la même visibilité avant et arrière en cellules maximalement connexes.

Segments libres maximaux. Pour mieux visualiser les classes d'équivalence définies ci-dessus, nous utiliserons la notion de segment libre maximal. Un segment de droite (fini, semi-infini ou infini) est libre s'il est contenu dans l'espace libre ; il est maximal si aucun autre segment libre ne le contient. De manière équivalente, un segment libre est maximal s'il ne peut être prolongé à une de ses extrémités sans entrer en intersection avec un objet.

Un segment libre maximal représente ainsi toutes les origines des rayons d'une même classe d'équivalence pour la relation $\equiv$. Nous pouvons donc caractériser complètement une classe d'équivalence en adjoignant une orientation à un segment libre maximal. Ainsi, l'espace sur lequel est structuré le complexe de visibilité dans le plan est celui des segments libres maximaux orientés. Nous écrirons souvent « segment libre » ou même « segment » pour parler de segment libre maximal. Nous définissons enfin la visibilité d'un segment libre $\mathrm{vis}(s)$ comme la paire d'objets touchés par s. $\mathrm{vis}(s)$ peut être égale à (∞, ∞).

Éléments du complexe. Le complexe de visibilité est un complexe cellullaire de dimension deux. Nous avons vu que la visibilité d'un rayon mobile change lorsque celui-ci devient tangent à un objet. Les segments tangents viennent donc naturellement partitionner l'espace des segments libres orientés $\mathfrak{V}$. Les segments tangents à un objet forment deux courbes dans $\mathfrak{V}$ (une pour chaque orientation). L'arrangement de toutes les courbes de segments tangents induit naturellement la partition de $\mathfrak{V}$ en un complexe cellulaire appelé le *complexe de visibilité*. La Figure 1.4 illustre les différents types de cellules apparaissant dans le complexe de visibilité.

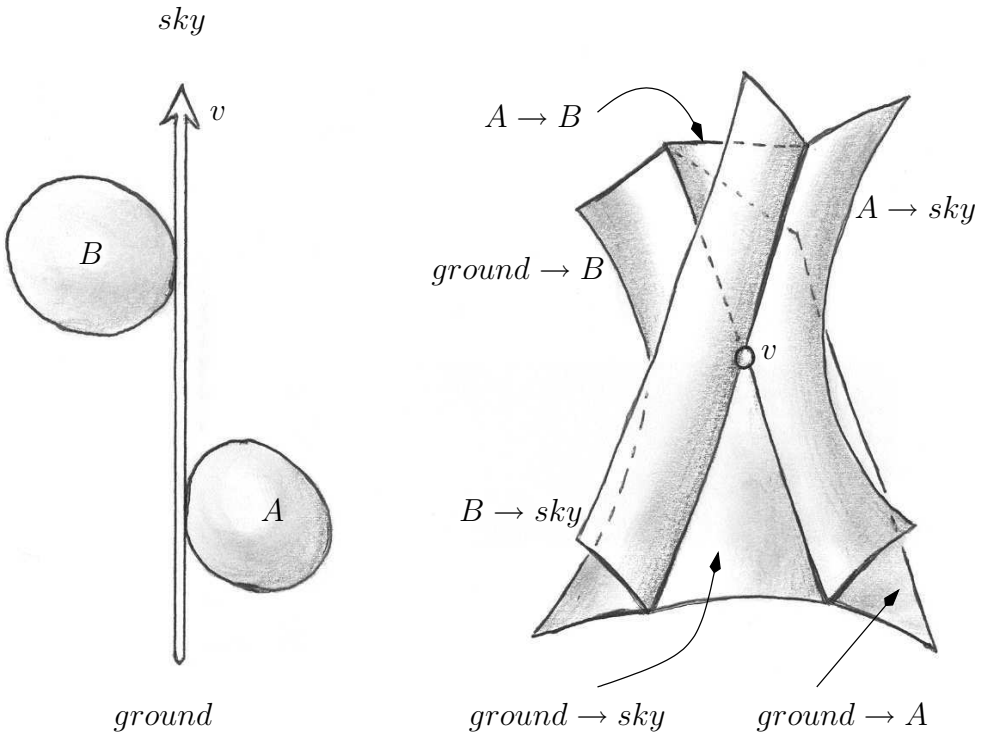


FIG. 1.5. Illustration d'un sommet du complexe de visibilité dans le plan. On voit **à gauche** deux objets et un segment libre v orienté bitangent. Ce même segment est représenté **à droite** comme un point v dans l'espace dual des segments libres maximaux. On voit les 6 2-cellules du complexe adjacentes à v.

Les cellules du complexe de visibilité de dimension 0 (ses *0-cellules*) sont les segments libres maximaux orientés et bitangents. Ils apparaissent comme l'intersection de deux courbes de segments tangents. Comme chaque paire d'objets de la scène peut engendrer au plus 8 segments bitangents libres, le nombre de 0-cellules est en $O(n^2)$ où n est le nombre d'objets dans la scène. La Figure 1.5 illustre la topologie d'un sommet du complexe de visibilité tangent à un premier objet par la gauche et à un deuxième objet par la droite. Les cellules de dimension 2 (ou *faces*) sont des ensembles maximalement connexes de segments libres orientés ayant la même visibilité. Chaque face du complexe entre deux objets de la scène est homéomorphe à une boule ouverte de $\mathbb{R}^2$. Enfin, les *1-cellules*, encore appelées *arêtes* sont les sous-ensembles d'une courbe de segments tangents compris entre deux segments bitangents.

Il est important de remarquer que le complexe de visibilité ne peut pas être représenté complètement sur un plan. En effet, si la « dimension » du complexe est 2, chaque arête du complexe est adjacente à 3 faces, et chaque segment bitangent est adjacent à 6 faces (et 4 arêtes). On peut, en revanche, plonger le complexe de visibilité dans $\mathbb{R}^3$.

Algorithmes, complexité et applications

Dès que $n > 1$, chaque 2-cellule possède exactement une 0-cellule dans sa frontière, de pente minimale ; ceci à l'exception de l'unique 2-cellule comprenant les segments libres maximaux ne touchant pas l'enveloppe convexe de l'ensemble des objets. De plus, chaque 0-cellule est minimale pour une 2-cellule. On en déduit immédiatement que la taille du complexe de visibilité de n objets convexes disjoints est en $O(n^2)$. Plus

précisément, elle est bornée par le nombre de bitangentes libres présentes, qui peut varier entre $O(n)$ et $O(n^2)$.

Expérimentalement, Everett *et al.* [53] ont observé que la taille du complexe de visibilité d'un ensemble de disques unités uniformément répartis dans le plan est asymptotiquement linéaire — à comparer à la borne quadratique théorique.

Rivière [124, 125] adapte la technique de balayage topologique d'un arrangement de droites du plan [48] pour obtenir un algorithme de construction du complexe de visibilité d'une scène polygonale en temps optimal $O(n \log n + k)$. Ici, n indique le nombre de segments composant la scène.

En supposant le complexe de visibilité construit en mémoire, Rivière propose un nouvel algorithme de calcul du polygone de visibilité d'un point de vue en temps $O(v \log n)$ où n est la taille de la scène (le nombre d'arêtes) et v la taille du polygone de visibilité. La complexité de cet algorithme est donc sensible à la taille de sa sortie. Cependant, un prétraitement doit être effectué et le complexe de visibilité stocké en mémoire ; or Rivière remarque que le complexe de visibilité d'une scène de 600 triangles peut occuper jusqu'à 100Mo ! Le problème du stockage doit donc être pris en considération.

Rivière montre également comment maintenir, à l'aide du complexe de visibilité, la vue d'un point de vue mobile, pour un coût logarithmique en le nombre de changements de visibilité. Il propose également un algorithme de maintenance du complexe lui-même, lorsque les objets de la scène sont en mouvement. Là encore, le coût de gestion de chaque changement combinatoire du complexe est logarithmique.

L'algorithme de Pocchiola et Vegter, le Greedy Flip Algorithm (GFA), permet de construire le complexe de visibilité d'un ensemble d'objets lisses et convexes en temps optimal $O(n \log n + k)$ où n est le nombre d'objets et k est la taille du complexe de visibilité. Il est supposé ici que chaque objet est de complexité constante et que les bitangentes à deux objets peuvent être calculées en temps constant (ou du moins peuvent-t-elles être comparées en temps constant). Le GFA a plus tard été simplifié par Angelier et Pocchiola [9], qui ont également montré comment prendre en compte des scènes de convexes lisses augmentées de bitagentes, ce qui permet de calculer le complexe de visibilité de scènes polygonales. Ce GFA simplifié a été implémenté par Pierre Angelier [7, 8].

1.3.2 Le complexe dans l'espace

La description du complexe de visibilité d'un ensemble de n objets convexes disjoints dans l'espace a été donnée par Durand *et al.* [44, 46]. Puisque l'orientation des segments libres maximaux fait apparaître une redondance dans le complexe de visibilité 2D (chaque face a une « sœur » géométriquement identique, mais dont l'« orientation » est inversée), le complexe 3D est décrit comme une structuration $\mathcal{VC}$ de l'ensemble $\mathcal{S}$ des segments libres maximaux *non-orientés*.

Mis à part l'oubli de l'orientation, la définition du complexe de visibilité 3D suit celle du complexe 2D. L'espace des segments libres est de dimension 4 partout sauf au voisinage des segments tangents. Les faces du complexe (ou *4-cellules*) sont les composantes maximalement connexes de segments libres ayant la même visibilité. Les segments libres tangents à chaque objet forment n hyper-surfaces (de dimension 3) dans l'espace $\mathcal{S}$. L'arrangement de ces hyper-surfaces est précisément la structure du complexe de visibilité $\mathcal{VC}$ des n objets. Remarquons qu'en 3D le complexe $\mathcal{VC}$ n'est

pas à proprement parlé un complexe cellulaire, puisque chaque face de dimension supérieure à 0 n'est pas forcément homéomorphe à une boule de même dimension. La Figure 1.6 (à droite) donne un exemple de face de dimension 2 comportant un trou. De même, si nous plaçons plusieurs petits convexes entre deux convexes plus grands A et B, certaines faces de dimension 4 dont la visibilité est (A, B), ont toutes les chances d'être perçées d'autant de « tunnels » que de petits convexes (voir Figure 1.6). Notons toutefois qu'une 4-face ne possède pas de « cavité interne » : la frontière d'une 4-face est connexe ; c'est un résultat que nous démontrerons et exploiterons au Chapitre 7.

Nous verrons cependant (Chapitre 7, page 93) que $\mathcal{S}$ peut bien être muni d'une structure de complexe cellulaire (ou CW-complexe) en rafinant le complexe de visibilité $\mathcal{VC}$.

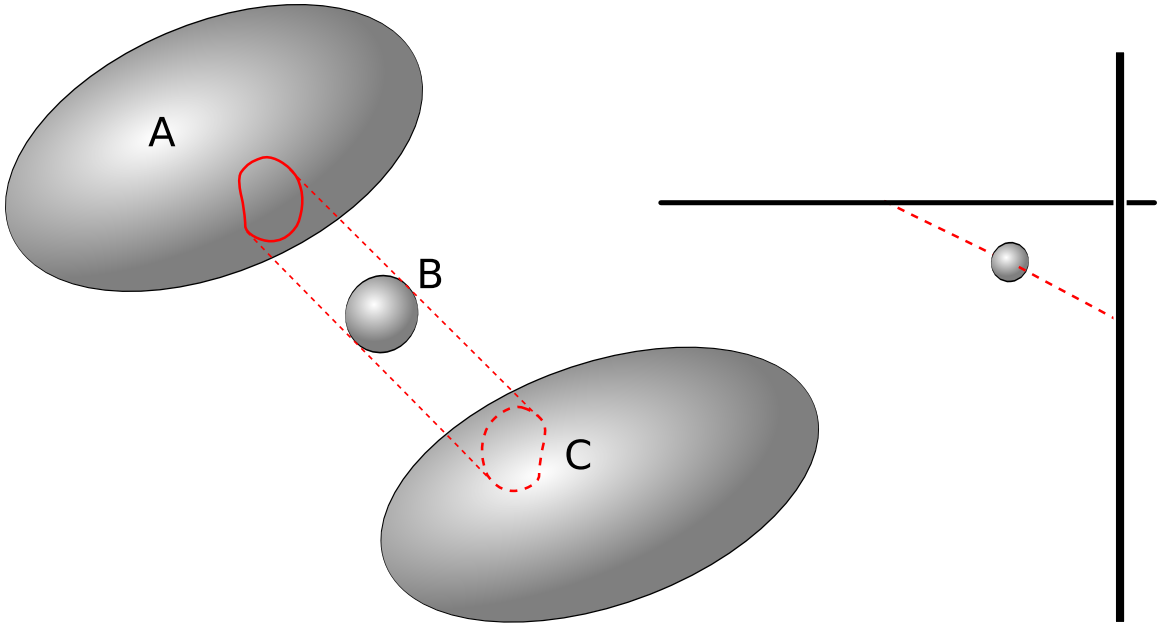


FIG. 1.6. À gauche, l'ensemble des segments libres maximaux voyant (touchant) les objets A et C est connexe, mais comporte un tunnel « perçé » par la présence de l'objet B entre A et C. **À droite**, les segments libres bitangents aux deux segments forment une 2-cellule $\mathcal{C}$ du complexe de visibilité. La petite sphère au milieu crée un trou dans $\mathcal{C}$. $\mathcal{C}$ a donc la topologie d'un anneau du plan.

Complexe de visibilité d'un ensemble de polytopes

Pour pouvoir fournir des solutions algorithmiques aux problèmes liés au complexe de visibilité (le plus évident étant sa construction), il nous faut un moyen pratique de décrire la combinatoire du complexe sous forme d'une structure de données. Nous explicitons une telle structure dans le cas où la scène est composée de polytopes disjoints. La frontière d'un polytope est une surface polyédrique formée de polygones convexes. Aussi, un segment libre maximal s est tangent à un polytope P selon différents types d'adjacences. On dira que s est tangent en un sommet v du polytope si il contient ce sommet et n'intersecte pas l'intérieur de P ; v est appelé un *générateur* de s. On dira que s est tangent en une arête e de P si s intersecte l'intérieur de e et n'intersecte pas l'intérieur de P ; e est également appelée un générateur de s.

Nous commençons par considérer un unique polytope P et décrivons comment

nous structurons l'ensemble des segments libres tangents à P en un complexe cellulaire T_P de dimension 3.

Les sommets (les 0-cellules de T_P) sont les segments libres possédant deux sommets générateurs. On notera c_{v_1,v_2} le segment libre tangent à P en les sommets v_1 et v_2 de P. On dit que ce sont des 0-cellules de type VV.

Les 1-cellules de T_P sont les ensembles maximalement connexes de segments libres dont les générateurs sont un sommet v et une arête e donnés, notés $c_{v,e}$. On dit que ce sont des 1-cellules de type VE. On peut vérifier que de tels ensembles sont bien de dimension 1. Une 1-cellule de T_P peut être paramétrée sur l'intervalle $(0,1)$ par la position du point d'intersection de ses segments libres avec e. Soient une 1-cellule $c_{v_1,e}$ et v_2 et v_3 les sommets de l'arête e. Alors, les 0-cellules c_{v_1,v_2} et c_{v_1,v_3} forment le bord de $c_{v_1,e}$.

Nous distinguons ensuite deux types de 2-cellules dans T_P. D'une part, les ensembles maximalement connexes de segments libres possédant un seul sommet v comme générateur. Une telle cellule est notée c_v et est dite de type V. D'autre part, les ensembles maximalement connexes de segments libres générés par une paire d'arêtes de P donnée, e_1 et e_2. Ces cellules seront notées c_{e_1,e_2} et dites de type EE. On paramètre les 2-cellules de type V sur l'espace $\mathbb{RP}^2$ des directions non orientées. Une 2-cellule c_{e_1,e_2} de type EE pourrait être paramétrée sur $(0,1)^2$ en encodant la position des points d'intersection d'un segment de c_{e_1,e_2} avec les arêtes génératrices e_1 et e_2. Mais nous verrons, au Chapitre 7 que — si e_1 et e_2 partagent un sommet — cela pose des problèmes pour représenter les adjacences de c_{e_1,e_2} au niveau des segments de paramètres $(\varepsilon_1,\varepsilon_2)$ pour ε_1 et ε_2 infiniment petits. Aussi, il sera plus aisé de paramétrer c_{e_1,e_2} comme un polygone convexe dans l'espace dual du plan support de e_1 et e_2 dans lequel les droites (mises en bijection avec les segments libres de c_{e_1,e_2}) deviennent des points. Nous référons le lecteur au Chapitre 7 pour de plus amples détails sur la paramétrisation des 2-cellules de T_P.

Les 3-cellules de T_P sont les composantes maximalement connexes de segments libres admettant une unique arête comme générateur. Si l'arête e est génératrice d'une telle cellule, cette dernière sera notée c_e. On pourra paramétrer une 3-cellule c_e comme un « cylindre » 3D : si q est un point de e, l'ensemble de segments libres de c_e passant par q forme une région c_p (connexe) de $\mathbb{RP}^2$. Ainsi, c_e peut être vu comme un cylindre égal à $\bigcup_{p\in e} c_p$ et représenté par sa frontière union d'un nombre fini de 2-, 1- et 0-cellules.

Considérons maintenant que la scène $\mathcal{O}$ comprend n polytopes disjoints : $|\mathcal{O}| = n$. Le complexe de visibilité de $\mathcal{O}$ est la partition de $\mathcal{S}$ résultant de l'arrangement des $T_P, P \in \mathcal{O}$. On distinguera alors, dans cet arrangement, d'autres types de cellules de dimensions 0, 1, et 2 :

- des 0-cellules de type VV dont les deux sommets générateurs appartiennent à deux polytopes distincts.
- des 0-cellules (dites de type VEE) générées par un sommet d'un polytope A et deux arêtes de deux polytopes B et C ; A, B et C étant distincts deux à deux.
- des 0-cellules (dites de type E4) générées par quatre arêtes de polytopes.
- des 1-cellules de type VE dont les deux générateurs appartiennent à deux polytopes distincts.
- des 1-cellules de type EEE générées par trois arêtes de trois polytopes distincts.
- des 2-cellules de type EE générées par deux arêtes de deux polytopes distincts.

Surfaces d'événements visuels. Le *1-squelette de visibilité* est l'union des cellules de dimensions 0 et 1 du complexe de visibilité. Une partie du 1-squelette du complexe de visibilité est d'une importance particulière pour la maintenance de la visibilité d'un point de vue mobile. Considérons le sous-ensemble du 1-squelette contenant, d'une part, l'ensemble des segments libres maximaux tangents à 3 convexes (de type $T + T + T$ et EEE) et, d'autre part, l'ensemble des segments libres tangents à deux convexes dans un plan bitangent à ces deux convexes (de type T ++ T et VE).[1] Si nous considérons l'ensemble de ces segments comme un ensemble de points dans l'espace libre, nous pouvons observer qu'ils forment un ensemble de surfaces telles que la topologie du polyèdre de visibilité d'un point de vue mobile P change uniquement lorsque P passe à travers l'une de ces surfaces – d'où leur nom de *surfaces d'événements visuels*.

L'arrangement de ces surfaces dans l'espace libre, $\mathcal{F}$, induit donc une partition $\mathcal{P}$ de $\mathcal{F}$ telle que le polyèdre de visibilité de P reste combinatoirement constant lorsque P reste à l'intérieur d'une cellule de $\mathcal{P}$. On pourrait penser construire explicitement cette partition dans le but de maintenir la visibilité de P. Mais, dans le cas où les convexes sont des polytopes, ces surfaces peuvent être quadratiques, et leur arrangement avoir une très grande taille. Il serait irréaliste de penser pouvoir stocker une telle partition en mémoire, même pour une scène relativement simple (quelques centaines de polytopes).

Nous référons enfin le lecteur aux Chapitres 4 et 5, ainsi qu'aux Appendices B et C de la thèse de Xavier Goaoc [60] pour une description plus détaillée des éléments mentionnés ci-dessus.

Le 1-squelette de visibilité a été utilisé en pratique pour améliorer la qualité de calculs de radiosité [45] et pour le calcul géométrique d'ombres dures complexes [42].

Complexité, construction. Si n est la complexité de l'ensemble des polytopes de la scène, la complexité du complexe de visibilité est en $O(n^4)$ [46]. Nous donnons d'autres bornes sur la taille du complexe au Chapitre 7, où sont également introduits les algorithmes connus de construction du complexe de visibilité 3D.

1.4 Structures de données cinétiques

La plupart des algorithmes et structures de données créés en géométrie algorithmique portent sur des données d'entrée fixes. Ainsi, par exemple, certains algorithmes de construction d'une triangulation de Delaunay d'un ensemble de points supposent que ces points sont donnés en entrée une fois pour toutes ; ils sont inamovibles. Il en va de même pour la construction d'une multitude d'autres structures géométriques, comme les arrangements de droites, la superposition de cartes planaires, l'enveloppe convexe d'un ensemble de points, les polygones ou polyèdres de visibilité, *etc.*Ces algorithmes sont qualifiés de *statiques*.

Soit N un ensemble de n éléments géométriques (par exemple des segments du plan). Et soit $H(N)$ une structure géométrique complètement dépendante de N que l'on souhaite construire, par exemple une décomposition trapézoïdale. Certains algorithmes statiques sont dits *incrémentaux* si la structure $H(N)$ est construite en

[1] Voir l'article de Durand *et al.* [46] pour de plus amples détails.

considérant les éléments de N consécutivement, c'est-à-dire en construisant successivement les structures $H(N^i), i = 0, \ldots, n$, où $N^0 = \emptyset, N^n = N$ et N^{i+1} est obtenu en ajoutant à N^i un élément de $N \setminus N^i$ [107][2].

D'autres algorithmes permettent, après construction de $H(N)$, d'ajouter un nouvel élément géométrique à N (pour obtenir N') puis de calculer $H(N')$ efficacement à partir de $H(N)$; ces algorithmes sont dits *semi-dynamiques*. S'ils permettent également de supprimer un élément e de N et d'obtenir efficacement $H(N \setminus \{e\})$, on parle alors d'algorithmes *[complètement] dynamiques*.

La caractéristique de tous ces types d'algorithmes est d'opérer dans un espace à temps *discret* : l'ensemble des données d'entrée ne change qu'à des points discrets dans le temps (un temps fictif).

En 1997, Basch, Guibas et Hershberger [12, 13] ont introduit la notion de *structure de données cinétique* (ou KDS, pour *Kinetic Data Structure*). Le modèle des KDS reprend ceux décrits ci-dessus mais rend la notion de temps *explicite* et *continue* en permettant de spécifier le mouvement de chaque élément géométrique de N. Dans la suite de cette section, nous présentons le modèle des KDS plus en détail, pour en introduire le vocabulaire spécifique et expliquer les façons nouvelles d'analyser les performances théoriques de telles structures de données.

1.4.1 Définitions

Soit N un ensemble d'éléments géométriques mobiles (souvent appelés *items*). Soit $A(N)$ une structure de données géométrique complètement dépendante de N. $A(N)$ est appelé un *attribut* de N. De plus, nous sont donnés la position des *items* au temps $t = 0$ et leur mouvement à court terme. Ces mouvements sont supposés définis par des équations algébriques de degré borné, et sont susceptibles de changer à n'importe quel moment via certains événements externes (action d'un utilisateur, déclenchement d'un capteur, décision d'un programme externe, *etc.*)

Le problème que l'on se donne est de maintenir l'attribut $A(N)$ continûment lors du mouvement des *items* de N. Il est entendu par là que les données numériques (coordonnées géométriques) de $A(N)$ dépendent directement et simplement de celles de N et n'ont donc pas à être maintenues explicitement, ce qui est du reste impossible puisque ces données évoluent continûment au cours du temps. C'est en revanche la combinatoire de la structure $A(N)$ que l'on souhaite maintenir au cours du temps, ce qui est effectivement possible si elle change de manière discrète dans le temps. Pour s'en assurer, nous considérons un modèle théorique dans lequel les mouvements sont spécifiés de manière à garantir (en général) que les zéros des différentes équations manipulées soient en nombre fini — des fractions rationnelles polynômiales non dégénérées feront l'affaire.

L'idée sous-jacente à l'acte de maintenance de $A(N)$ est la notion de *preuve animée*. Considérons un algorithme *statique* de construction de $A(N(0))$ où $N(t)$ représente l'ensemble des *items* au temps t. Il met en œuvre un ensemble de procédures et de prédicats qui, ensemble, fournissent une *preuve* de la validité de la construction de $A(N(0))$. Les données numériques de $N(0)$ apparaissent en certains endroits du déroulement de l'algorithme sous forme de prédicats (test du signe d'une expression numérique).

[2] On parle d'algorithme incrémental *randomisé* si l'élément ajouté à N^i est choisi aléatoirement dans $N \setminus N^i$.

Si les positions des *items* à un certain $t \in (0, \varepsilon)$ (pour ε petit) changent sans modifier pour autant le résultat des prédicats apparaissant dans le déroulement de l'algorithme sur l'entrée $N(0)$, alors il est clair que la combinatoire de $A(N(t))$ est la même que $A(N(0))$ et aucune modification de $A(N(0))$ n'est à apporter pour connaître $A(N(t))$.

Un changement combinatoire de A ne peut survenir qu'en cas de changement du résultat d'un des prédicats mis en œuvre.

Nous définissons alors la notion de *certificat*. C'est une entité « abstraite » en charge d'un prédicat, et qui s'assure que ce dernier reste correct. Dès lors qu'un prédicat change de signe, il nous faut procéder à la mise à jour de l'attribut A et à la création des nouveaux certificats.

Toute structure de données cinétique possède donc une file d'attente (implémentée, par exemple, comme une queue de priorité) qui stocke les certificats en cours, et permet de rapidement accéder au premier élément, et insérer ou retirer un certificat de la file d'attente. Dans cette file, la clé de tri d'un certificat est la date à laquelle son prédicat changera de signe.

Exemple. Nous souhaitons maintenir trié un ensemble de n points $P_1, \ldots, P_n$ mobiles sur la droite réelle. L'attribut A maintenu sera une liste doublement chaînée contenant les points en question. Nous insérons $n-1$ certificats dans la file d'attente. Si les points P_j et P_k sont consécutifs, le certificat (j,k) s'assure qu'ils restent ainsi ordonnés. Un certificat perd sa validité quand les deux points qu'il surveille se croisent. Quand le temps t dépasse la date de validité du premier certificat de la file, les deux points concernés sont échangés dans la liste A :

$$\cdots P_i, P_j, P_k, P_l \cdots \longrightarrow \cdots P_i, P_k, P_j, P_l \cdots$$

Nous devons alors supprimer de la file d'attente les 3 certificats (i,j), (j,k), (k,l), et y insérer 3 nouveaux certificats (i,k), (k,j), (j,l) ; opérations s'effectuant en temps $O(\log n)$. Notons qu'un nouveau certificat n'est à insérer que si les deux points concernés sont amenés à se croiser dans le futur.

1.4.2 Analyse d'une structure de données cinétique

L'exemple ci-dessus ne fait pas apparaître de grandes nouveautés « techniques ». De fait, le modèle des KDS apporte surtout une manière cohérente de mesurer les performances de telles structures de données. Comme la maintenance de l'attribut $A(N)$ peut s'étaler aussi longtemps que souhaité dans la durée, exprimer sa complexité en temps n'a pas grand sens.

Reprenons les définitions proposées dans [13] :

Une fonction $f(n)$ croissante et positive est dite *petite* si $f(n) = O(\mathrm{Polylog}\, n)$ ou $f(n) = O(n^{\varepsilon})$, pour ε petit et positif.

- Une KDS est dite *répondante (responsive)* si le coût associé à la gestion d'un certificat quand il perd sa validité est petit.
- Certains événements (invalidation d'un certificat) n'entraînent pas de modifications de l'attribut $A(N)$. Ce sont des événements *internes* à la structure de données considérée. Les autres événements modifient effectivement $A(N)$ et sont appelés *externes*. Une KDS est dite *efficace* si le rapport du nombre total d'événements sur le nombre d'événements externes est petit.

– La *taille* d'une KDS est définie comme le nombre maximum d'événements potentiellement présents dans la file d'attente. Une KDS est *compacte* si sa taille est à peu près linéaire en le nombre d'items mobiles.
– Enfin, une KDS est dite *locale* si chaque item est impliqué dans un petit nombre de certificats présents dans la file d'attente.

L'exemple ci-dessus décrit une KDS répondante, efficace, compacte et locale. En particulier, cette KDS permet trivialement de maintenir le point le plus à droite (dont la coordonnée est la plus grande). Cependant, si l'attribut A qui nous intéresse est seulement le point le plus à droite, alors la KDS décrite n'est plus efficace. Nous l'expliquons dans le cas où le mouvement des points est linéaire (leur vitesse est constante et n'est pas modifiée par les événements extérieurs). Il est alors clair que l'attribut A ne peut changer qu'au plus $n - 1 = O(n)$ fois (le nombre d'événements externes est $O(n)$). Or, le nombre de croisements de deux points au cours de la simulation est $O(n^2)$. Ainsi, le rapport du nombre total d'événements sur le nombre d'événements externes peut être en $O(n)$ et n'est donc pas petit. Dans [13], Basch *et al.* introduisent une KDS appelée *un tournoi cinétique* qui maintient le point le plus à droite, et est répondante, compacte, locale et efficace.

Chapitre 2

Visibilité : premiers pas et récents développements

Nous présentons dans ce chapitre un aperçu des techniques développées en géométrie algorithmique et en synthèse d'images pour résoudre le problème de la « caméra virtuelle », ou comment calculer ou dessiner rapidement la partie d'une scène visible depuis un point de vue donné.

Nous commençons par donner un rapide historique du problème puis décrivons les algorithmes conçus pour la résolution de ce problème dans le plan (section 2.2) et dans l'espace (section 2.3). Nous donnons ensuite un aperçu des techniques imaginées pour mettre à jour rapidement la vue d'un point de vue mobile (section 2.4).

En 1951, l'utilisation de *vectorscope* sur un ordinateur *Whirlwind* pour l'affichage d'images est présenté pour la première fois publiquement. L'idée d'utiliser de tels afficheurs vectoriels pour la représentation d'objets tridimensionnels apparaît alors très rapidement.

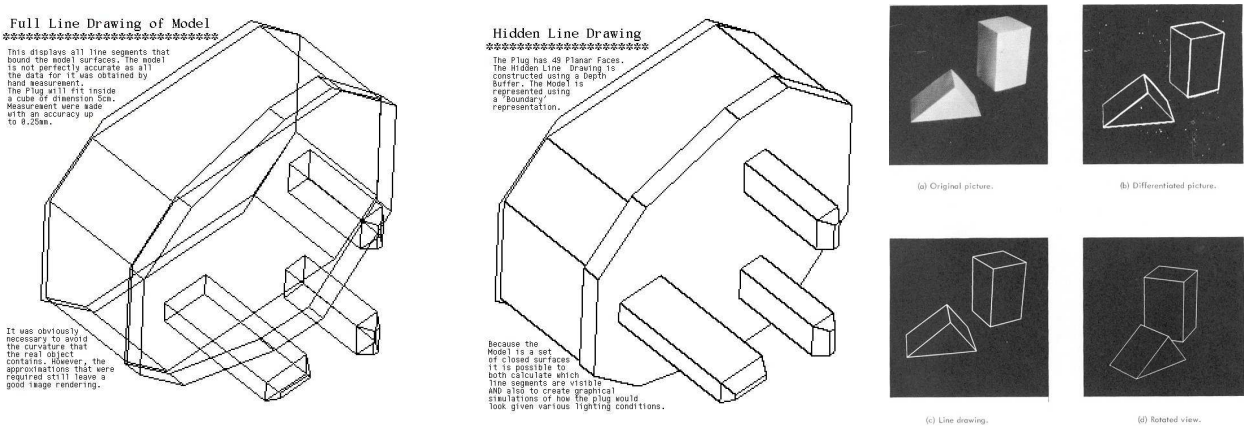
1963 est une année faste pour la *synthèse d'images* [1] : S. A. Coons invente un nouveau modèle de représentation d'un morceau d'une surface lisse, Ivan E. Sutherland créé *SketchPad* [136], le premier système de dessin sur ordinateur, et Lawrence G. Roberts [126] présente un algorithme effectuant l'affichage en fils de fer d'objets 3D polygonaux, avec élimination des lignes cachées.[2]

2.1 Élimination des lignes/surfaces cachées

L'élimination des lignes cachées est le processus par lequel sont déterminées les parties des arêtes d'une scène polygonale qui sont visibles depuis un point de vue donné. Bien que cette définition soit toujours valide pour des objets à surface courbe (en considérant les arcs silhouettes), le problème a été d'abord attaqué dans le cas de modèles polyédriques, et était en partie motivé par le matériel disponible dans les années soixante, en particulier les afficheurs vectoriels. Ces derniers étant incapables de « remplir » des régions 2D de l'écran, les modèles 3D devaient être représentés en « mode fil-de-fer ». La Figure 2.1(a) montre la représentation d'un modèle polygonal

[1] Le terme anglais *computer graphics* a été proposé en 1960 par William Fetter, alors employé chez Boeing.

[2] Cette page web présente une chronologie de l'histoire de l'image de synthèse :
http://accad.osu.edu/~waynec/history/ID797.html



(a) (b) (c)

FIG. 2.1. (a) Rendu en fils de fer d'un modèle d'une prise de courant. Il est difficile d'appréhender la forme globale de l'objet. (b) L'élimination des lignes cachées rend le dessin bien plus lisible. Ces deux images proviennent de http://perso.club-internet.fr/alib/gall05.htm. (c) Les travaux de Roberts (126) décrivent une méthode de reconstruction d'objets 3D à partir de photographies, et introduisent le premier algorithme d'élimination des lignes cachées sur un ensemble de polytopes.

3D d'une prise de courant (américaine) en fil-de-fer (mais pas sur un afficheur vectoriel), et avec élimination des lignes cachées (b). Il est clair que seule l'élimination des lignes cachées nous permet d'apprécier correctement la forme de l'objet 3D représenté.

Parmi les travaux de Roberts [126] figure le premier algorithme pour l'élimination des lignes cachées sur un ensemble de polytopes (polyèdres convexes) 3D (voir Figure 2.1(c)). Tout d'abord, les arêtes sont projetées dans l'espace écran, et deviennent des segments 2D qui sont coupés sur les bords de l'écran. Puis, chaque segment est examiné et rejeté si les deux polygones 3D adjacents à son arête correspondante *tournent le dos au point de vue*. Enfin, chaque segment restant est testé par rapport aux polytopes auxquels il n'appartient pas, ce qui permet d'éliminer les parties du segment cachées par les autres polytopes. L'algorithme tire profit de la convexité des polytopes, et un soin particulier est donné à la suppression des arêtes de contact entre deux polytopes, si les deux polygones visibles adjacents à une telle arête sont coplanaires.

Les travaux de Appel [10] introduisirent une terminologie qui est toujours utilisée aujourd'hui — que nous introduisons ci-dessous — ainsi que le premier algorithme permettant l'élimination des lignes cachées sur des polyèdres non nécessairement convexes.

Un polyèdre étant constitué de plusieurs faces planes, peut donc servir d'approximation linéaire de la surface d'un objet. En général, l'objet est fermé et son intérieur peut être défini. Alors, la **normale** n_f d'une face f est définie comme un vecteur non nul (souvent normalisé) orthogonal à f et pointant « hors » de l'objet, c'est-à-dire vers le demi-espace localement à l'extérieur de l'objet.

Quand un point de vue V est donné, nous pouvons définir une ligne de vue (V, l) pour la face f comme un rayon orienté d'origine le point de vue et d'orientation $l \in S^2$ allant vers un point de f. Alors, f est une **face-arrière** si $n_f \cdot l \geq 0$; sinon, f est une **face-avant** (f fait « face » au point de vue v).

Les **arêtes de contour** ou **lignes de contour** sont définies comme les arêtes du

polyèdre ayant exactement une face-adjacente-avant. Nous supposons ici que la surface du polyèdre est une surface bornée et sans bord : chaque arête a donc exactement deux faces adjacentes. Les arêtes de contour incluent la silhouette de l'objet, mais le contraire n'est pas vrai.

Arthur Appel définit l'*invisibilité quantitative* le long d'une arête comme une fonction entière (à valeur dans $\mathbb{N}$) constante par morceau, qui correspond au nombre d'« objets » ocultant un point sur l'arête. L'invisibilité quantitative change lorsque l'arête passe visuellement (depuis V) au travers — et derrière — une ligne de contour. Dès lors, dessiner l'ensemble des objets avec élimination des lignes cachées se réduit à dessiner l'ensemble des morceaux d'arêtes dont l'invisibilité quantitative est égale à 0. De plus, l'invisibilité quantitative permet d'obtenir des rendus du style « dessin technique » dans lesquels les morceaux d'arêtes invisibles sont dessinés en pointillés, par exemple. L'invisibilité quantitative connaît un regain d'intérêt dans le domaine du rendu non-photoréaliste (NPR) de scènes 3D [63].

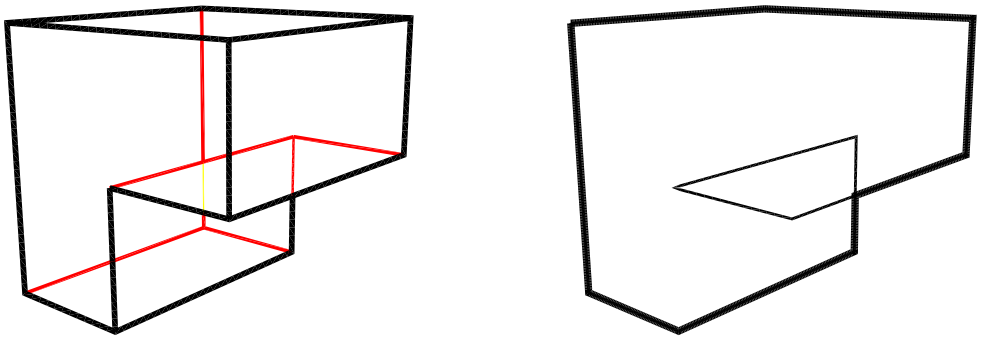


FIG. 2.2. Illustration de l'invisibilité quantitative. **À gauche**, les segments sont colorés en fonction de leur invisibilité quantitative : en noir pour 0, en rouge pour 1 et en jaune pour 2. **À droite**, seuls les contours sont dessinés et la silhouette est dessinée en gras. *Image produite avec le système* FREESTYLE *de Stéphane Grabli* et al.

Philippe Loutrel propose un algorithme d'élimination des lignes cachées pour un polytope (polyèdre convexe) [95] puis pour un polyèdre quelconque [96]. Son algorithme se révèle plus efficace que les précédents grâce à une propriété de cohérence spatiale dont il tire parti. La visibilité quantitative est d'abord calculée à un sommet P du polyèdre. Cette valeur est ensuite propagée aux sommets voisins en utilisant le concept d'un point mobile le long des arêtes adjacentes à P. Le changement d'arête support ainsi que la présence d'arête contour *devant* l'arête en cours de traitement constituent des événements (en nombre fini) modifiant l'invisibilité quantitative de ± 1. La propagation de l'information ne se fait que le long des arêtes potentiellement visibles et doit éventuellement être faite sur plusieurs composantes connexes.

Citons enfin la méthode de Galimberti et Montarani pour l'élimination des lignes cachées [56] qui est essentiellement la même que celle d'Appel [10]. Toutes ces méthodes ont un temps de calcul quadratique car chaque arête est comparée à toutes les autres. Notons qu'une comparaison de ces algorithmes est disponible dans [137], qui compare également des algorithmes discrets (c'est-à-dire par pixel) pour le même problème (par exemple le fameux algorithme de Warnock [145]).

Au début des années soixante-dix, les premiers écrans *raster* (à pixels ou matriciels) apparaissent, comme le système *SuperPaint* de Richard Shoup évoqué dans [131]. Les surfaces peuvent désormais être « peintes » sur un afficheur graphique,

et les chercheurs se penchent avec un intérêt renouvelé sur une variante de l'élimination des lignes cachées : l'élimination des surfaces cachées (le fameux *hidden-surface removal* ou HSR). Bien sûr, les chercheurs n'avaient pas attendu l'apparition des premier écrans matriciels pour inventer des méthodes d'élimination des surfaces cachées.

2.2 Visibilité dans le plan

Graham publie son algorithme de calcul de l'enveloppe convexe d'un ensemble de points dans le plan en 1972. La géométrie algorithmique apparaît aux alentours de 1975 [127, 128]. Les problèmes de visibilité dans le plan (en deux dimensions, donc) sont abordés par la communauté de la géométrie algorithmique, motivés déjà par les problèmes rencontrés en synthèse d'images.

En 1982, Bernard Chazelle [24] démontre qu'il est possible de séparer un polygone simple [3] de taille n en deux sous-polygones bien balancés, c'est-à-dire dont le nombre de sommets est inférieur ou égal à $2n/3$, ouvrant ainsi la voie à des méthodes de type « diviser pour régner » appliquées à des polygones simples.

Étant donné un polygone simple P ainsi qu'une triangulation de P (qui peut être calculée en temps $O(n \log n)$ avec la méthode décrite dans ce même papier) et un point V à l'intérieur de P, il est possible de calculer le polygone de visibilité $\mathcal{V}(V)$ de V en temps $O(n)$, c'est-à-dire en temps optimal. Nous rappelons que le polygone de visibilité $\mathcal{V}(V)$ de V dans P est l'ensemble des points de P visibles depuis V (les bords de P étant considérés opaques). Une fois le triangle t contenant V localisé, $\mathcal{V}(V)$ est calculé en trois parties, correspondant chacune à la partie de P visible au travers d'une arête de t. Le calcul de chaque partie se fait de manière récursive en traversant l'arbre (T, E) constitué des triangles T de la triangulation de P connectés par leurs adjacences géométriques E. Il est intéressant de remarquer que cet algorithme de calcul du polygone de visibilité est fort ressemblant aux calculs de géodésiques (plus courts chemins) entre deux points dans un polygone simple ou sur un maillage 3D [104, 135]. Cette ressemblance est mise à profit par Guibas *et al.* [64] pour résoudre divers problèmes de visibilité ou de plus court chemin dans un polygone simple, en temps linéaire si la triangulation du polygone simple est connue. Ces algorithmes deviennent donc « vraiment » linéaires en 1990 avec l'algorithme de Chazelle [25] permettant de trianguler un polygone simple en temps linéaire.

En 1985, le papier de Chazelle et Guibas [26] démontre la puissance de la notion de dualité point/droite en l'appliquant à plusieurs problèmes de visibilité dans le plan. Étant donné un polygone simple dont une arête e est marquée comme *lumineuse*, des méthodes sont proposées pour :

- précalculer une structure en temps $O(n \log n)$ et espace $O(n)$ pour résoudre le problème du lancer de rayon depuis un point de e en temps $O(\log n)$.
- calculer la partie de P illuminée par e, en temps $O(n \log n)$ et espace $O(n)$.
- calculer les points de e visibles depuis un point de la frontière de P en temps $O(\log n)$.

En fait, tout ces problèmes sont résolus à l'aide de la même structure de données (précalculée au premier point), que nous détaillons maintenant. Dans l'espace dual [65], l'ensemble des droites orientées passant par e forment un polygone convexe. Ce

[3] Un polygone simple est un polygone homéomorphe à un disque (il ne contient pas de trou).

polygone est partitionné en sous-polygones convexes correspondant chacun, dans le primal, à l'ensemble des droites perçant e et une autre arête de P. Cette subdivision est ensuite augmentée d'une structure de recherche rapide [49]. Le tout peut être construit en temps $O(n \log n)$ et espace $O(n)$.

Plutôt que d'utiliser la dualité point/ligne, Pocchiola [117] propose d'utiliser une paramétrisation des droites *orientées* par les paramètres $\theta \in [0, 2\pi)$ et $u \in \mathbb{R}$. $\ell = (\theta, u)$ est la droite orientée d'orientation θ et u est la distance de ℓ à l'origine. Lorsque la scène est composée d'objets convexes disjoints, l'ensemble des droites orientées tangentes aux objets de la scène forment un ensemble de courbes dans l'espace des paramètres (espace dual). Leur arrangement peut être calculé efficacement et il en découle une série d'algorithmes permettant de résoudre les problèmes de *ray-shooting*, de calcul du polygone de visibilité d'un point et sa maintenance lors du mouvement du point de vue en espace $O(n^2)$ et temps quasi-optimal (avec prétraitement). Ce sont les prémices des travaux de Pocchiola et Vegter sur le complexe de visibilité [119].

Aronov *et al.* [11] explorent plus avant la connection entre plus courts chemins et polygone de visibilité d'un point à l'intérieur d'un polygone simple P de taille n. Ils donnent un algorithme permettant, après un précalcul en temps $O(n^2 \log n)$ et espace $O(n^2)$ de calculer le polygone de visibilité $V(p)$ de n'importe quel point p à l'intérieur de P en temps $(\log n + |V(p)|)$. Dans le même article, ils mettent au point une structure de données cinétique optimale pour la maintenance continue du polygone de visibilité d'un point p mobile à l'intérieur de P : chaque changement combinatoire de $V(p)$ est détecté et traité en temps $O(\log |V(p)|)$.

Zarei et Ghodsi [149] ont récemment montré que l'on pouvait calculer le polygone de visibilité $V(q)$ d'un point q à l'intérieur d'un polygone P de taille n *comportant* h *trous* en temps $O((1 + h') \log n + |V(q)|)$, où $h' \leq \min(h, |V(q)|)$.

2.3 Visibilité dans l'espace

En 1976, les travaux de Clark [29] sont parmi les premiers à montrer comment la notion de scène hiérarchisée permet d'améliorer l'efficacité d'un grand nombre d'étapes lors de la détermination des parties visibles d'une scène.

En 1980, les travaux de Fuchs *et al.* [55] décrivent des solutions efficaces pour le dessin de polyèdres avec élimination des faces cachées. Ils décrivent comment l'espace 3D contenant un ensemble de polygones peut être décomposé en un arbre de partition binaire (un arbre *BSP*) et comment cet arbre de décomposition peut ensuite être traversé de telle sorte que les polygones soient dessinés d'arrière en avant (ou l'inverse) par rapport au point de vue. Ils mettent également en avant un des défauts de cette approche : lors de la construction de l'arbre *BSP*, les polygones peuvent être coupés en deux (ou plusieurs) parties. Si la scène comprend n polygones, l'arbre *BSP* résultant peut contenir $O(n^3)$ feuilles, ce qui peut être prohibitif.

Plus tard, en 1989, Paterson et Yao [114] montrent qu'étant donnés n polygones sans intersection intérieure,[4] on peut construire un arbre *BSP* de taille $O(n^2)$. Si les polygones sont orthogonaux deux à deux, on peut construire une partition binaire de l'espace de taille optimale $O(n^{3/2})$ [115].

Notons que la méthode de dessin de la scène présentée par Fuchs *et al.* utilisant l'arbre *BSP* ne permet pas, telle quelle, de calculer les parties des polygones visibles

[4] Les polygones ne peuvent s'intersecter que le long d'une arête.

depuis le point de vue, c'est-à-dire la carte de visibilité. Lors du parcours de l'arbre, chaque polygone est dessiné entièrement (il est *rasterisé* en un ensemble de pixels affichés à l'écran).

Rappelons le problème de l'élimination des faces cachées : étant donné un ensemble de polygones dans l'espace, ainsi qu'un point de vue et une pyramide de vue (correspondant à l'écran d'affichage, par exemple), on souhaite calculer l'ensemble des morceaux de polygones visibles depuis le point de vue dans la pyramide de vue. Suite à de nombreux travaux sur ce problème au début des années 80, McKenna propose en 1987 [101] un algorithme optimal dans le cas le pire, tournant donc en temps $O(n^2)$, alors que les travaux précédents atteignent des temps en $O(n^2 \log n)$ ou $O((n+k)\log n)$ où k est le nombre d'intersections parmi les arêtes de la scène projetées sur le plan image. La carte de visibilité est calculée en projetant les arêtes de la scène sur le plan image, puis en étendant chaque arête en sa droite support. L'arrangement de ces droites est calculé puis balayé de gauche à droite par une ligne verticale « topologique » [48] tout en calculant pour chaque face de l'arrangement dont le balayage se termine, le polygone le plus proche du point de vue.

Plusieurs travaux donnent des solutions au *hidden surface removal (HSR) problem* pour des configurations particulières de la scène. Citons par exemple Preparata *et al.* [121] qui montrent comment calculer en temps $O(n \log^2 n + d \log n)$ la vue axiale d'un ensemble de n parallélépipèdes rectangles, où d est le nombre de segments dans la carte de visibilité calculée. Citons encore Reif et Sen [122] qui proposent un algorithme de calcul de la carte de visibilité en perspective d'un terrain polygonal (exprimable comme une fonction $z = f(x, y)$ linéaire par morceaux). Leur méthode tourne en temps sensible à la complexité du résultat, et est parallélisable.

Ketan Mulmuley [105] présente un algorithme randomisé pour résoudre le *HSR problem*, dans lequel les polygones sont insérés un par un (et aléatoirement) dans une décomposition planaire du plan de projection de l'image. La complexité en temps de l'algorithme est en $O(n \log n + A)$ où A est un terme presque proportionnel à la taille de la carte de visibilité finale. C'est une avancée significative par rapport aux résultats précédents.

Si l'on s'autorise à effectuer un prétraitement sur les n polygones, on peut obtenir de meilleurs temps de calcul du polyèdre de visibilité d'un point dans l'espace. Mulmuley [106] imagine une décomposition cylindrique H de $\mathbb{R}^3$ de taille $O(B + n)$ (où $B \leq n^2$), munie d'une structure de localisation spatiale, qui permet, pour tout point P de l'espace libre, de calculer le polyèdre de visibilité de P en temps $O(\log^2 n + \sigma(P) \log n)$ où $\sigma(P)$ est la taille de l'intersection de H avec le polyèdre de visibilité de P.

Goodrich [61] présente deux algorithmes de calcul des lignes et surfaces cachées, dont le temps de calcul est sensible en le nombre d'intersections polygone/polygone et segment/segment dans le plan de projection de la caméra. Considérons l'ensemble Γ des n polygones déjà projetés sur le plan image. L'algorithme utilise un certain « arrangement des polygones de Γ » défini de telle sorte que toute paire de polygones donnant lieu à une relation de visibilité soit connectée dans l'arrangement. Par exemple, si le polygone p_1 contient proprement le polygone p_2, alors l'arrangement contient au moins une arête reliant les frontières de p_1 et p_2. Alors, le calcul des lignes ou surfaces cachées peut se faire par un parcours en profondeur de chacune des composantes connexes de l'arrangement des polygones. Soit k le nombre d'intersections segment/segment dans le plan image, et t le nombre d'intersections polygone/polygone dans le même plan. Les deux algorithmes tournent en temps $O(n \log n + k + t)$.

Supposons que notre scène comprenne n *triangles*, et que, après projection perspective, l'ordre partiel des polygones selon la relation d'occlusion partielle par rapport au point de vue soit connue. Supposons enfin que cet ordre ne contienne pas de cycle (qu'il faudrait alors « casser »). Ces hypothèses étant vérifiées, Sharir et Overmars [129] donnent un algorithme relativement simple qui calcule la carte de visibilité en temps $O(n\sqrt{k}\log n)$, qui est donc sensible en la taille de la sortie. Leur idée est de traiter les triangles par groupes, et dans l'ordre d'avant en arrière. L'ajout d'un groupe consiste en (1) le calcul de la carte de visibilité de ce groupe de triangles, (2) le calcul de l'intersection de cette carte avec le contour de la carte précédente, (3) la fusion des cartes de visibilité. La taille d'un nouveau groupe est choisie comme la racine carrée de la taille du contour de la carte de visibilité courante. Alors, la borne en temps s'obtient facilement. C'est un algorithme très simple mais dont la mise en œuvre et l'analyse sont astucieuses.

Erickson [52] propose un algorithme pour le calcul des parties visibles de n triangles dans $\mathbb{R}^3$. Cette méthode est intéressante pour deux raisons. D'abord, son temps d'exécution est, comme pour la méthode de Sharir et Overmars, sensible en la taille de la sortie. Ensuite, c'est une méthode hybride en ce qu'elle prend également en entrée un ensemble P de points du plan de projection : la sortie de l'algorithme est l'ensemble des trapèzes de la décomposition trapézoïdale de la carte de visibilité, qui contiennent au moins un point de P. C'est donc un sous-ensemble de la décomposition trapézoïdale de la carte de visibilité. Soit p le nombre de points dans P et t la taille de la sortie (le nombre de trapèzes calculés) ; l'algorithme tourne en temps $O(n^{1+\varepsilon}+n^{2/3+\varepsilon}t^{2/3}+p)$. Une borne meilleure est obtenue si l'ensemble des points de P forment les sommets d'une grille régulière. L'algorithme est intéressant car sa complexité est (presque) optimale dans le cas le pire et dépend linéairement du nombre de « pixels ». Si $t=\Theta(n^2)$ alors le temps de calcul devient $O(n^{2+\varepsilon}+p)$, et si $t=\Theta(1)$, le temps de calcul devient $O(n^{1+\varepsilon}+p)$. Il serait instructif d'implémenter cet algorithme pour le comparer aux algorithmes de *ray-casting* rapides connus qui semblent *a priori* difficiles à battre (voir par exemple l'article de Reshetov *et al.* [123] (et les références incluses), dans lequel est décrit un algorithme permettant d'afficher une scène d'environ 300000 triangles dans une fenêtre de 1024*1024 pixels à la cadence de 35 images par seconde, en lançant 1 rayon par pixel !).

2.4 Visibilité et mouvement

Les travaux de Matsushita [99, 1972] semblent être parmi les premiers à essayer d'exploiter la cohérence temporelle des relations de visibilité entre des polygones sujets un à mouvement continu (dans le repère spatial de l'observateur). Matsushita présente un algorithme permettant de maintenir relativement efficacement l'ensemble des lignes visibles et invisibles d'un ensemble de polytopes (polyèdres convexes) tournant autour d'un axe fixé. À chaque paire de polygones convexes est associée une « table de rotation » qui indique la relation de visibilité entre ces deux polygones en fonction de l'angle selon lequel ils sont vus (avec une projection orthogonale, et un axe de rotation parallèle au plan de projection de l'image). Le gain de temps de mise à jour de la vue est notable, mais le coût en espace, $O(n^2)$, est élevé.

Dans cette section, nous proposons un rapide aperçu des algorithmes permettant de maintenir la vue d'un point mobile dans une scène 2D ou 3D. C'est un problème

d'importance capitale si l'on considère d'une part, l'utilisation de plus en plus courante d'affichage numérique tri-dimensionnel de l'information (jeux vidéo bien-sûr, mais aussi création virtuelle de son logement, navigation GPS ou encore réalité augmentée lors de visites de sites archéologiques), et d'autre part, l'accroissement continu de la complexité des modèles 3D créés, qu'il faut dessiner à un taux interactif (une trentaine d'images par seconde).

Nous préférons regrouper ici les travaux les plus récents, tant du domaine de la géométrie algorithmique que de celui de la synthèse d'images. Nous référons donc également le lecteur au cours SIGGRAPH de Durand [43] et au *survey* de Cohen-Or *et al.* [30].

2.4.1 Maintenance de la vue d'un point mobile dans le plan

Nous considérons une scène planaire dont les objets sont polygonaux. Un point de vue V est autorisé à bouger continûment dans l'espace libre (il ne peut traverser un obstacle). Le problème est de maintenir la description combinatoire du polygone de visibilité de V au cours de son mouvement.

Chen et Daescu [27] proposent un algorithme de calcul des changements combinatoires du polygone de visibilité d'un point mobile le long d'un segment à l'intérieur d'un polygone simple. Leur méthode utilise les caractéristiques des chemins les plus courts d'un point du polygone à ses sommets [64].

Ghali et Stewart [58, 59] proposent une solution dans le cas où la scène S consiste en un ensemble de n segments disjoints deux à deux. Leur algorithme travaille dans l'espace dual classique qui transforme un point (a, b) en la droite $ax+by = 1$. Le graphe de visibilité de S est construit et chacune de ses droites (bitangentes aux segments de la scène) est représentée par son point dual. Le point de vue V devient une droite mobile V^* dans le plan dual. Le croisement par V d'une ligne de discontinuité visuelle correspond alors au croisement de la droite V^* et d'un point dual. Pour détecter ces croisements efficacement, les deux enveloppes convexes des points duaux de part et d'autre de la droite duale V^* sont calculées initialement et maintenues au cours du mouvement de V^*. Soit m la taille du graphe de visibilité. La méthode de Ghali et Stewart prend un espace $O(m)$ et un temps $O(\log^2 n)$ par mise à jour.

Nous référons le lecteur au Chapitre 1, où nous présentons brièvement les travaux de Rivière [125] concernant le calcul et la maintenance de la vue d'un point de vue mobile dans une scène polygonale, utilisant le complexe de visibilité, qui doit être calculé au préalable.

Nechvíle et Tobola [108, 109] améliorent l'occupation mémoire de l'algorithme précédent en utilisant la décomposition radiale de la scène (centrée sur le point de vue). En effet, l'ensemble des droites de discontinuité de visibilité que le point de vue peut croiser à un moment donné peut être restreint à un ensemble de taille linéaire trivialement calculable à partir de la décomposition radiale (voir le Chapitre 4). Par cette simple astuce, ils réduisent l'espace utilisé à $\Theta(n)$ au lieu d'un espace situé entre $\Omega(n)$ et $O(n^2)$.

Nous avons vu précédemment les travaux de Aronov *et al.* [11] qui donnent un algorithme optimal pour la maintenance du polygone de visibilité d'un point de vue se déplaçant à l'intérieur d'un polygone simple.

Hall-Holt [67] est le premier à fournir des algorithmes presque optimaux, et utilisant un espace mémoire de taille linéaire pour le problème de la maintenance du poly-

gone de visibilité d'un point de vue mobile dans une scène composée d'objets convexes, polygonaux ou lisses. Nous revenons plus en détail sur ces travaux au Chapitre 4.

2.4.2 Maintenance de la vue d'un point mobile dans l'espace

Maintenance de la carte de visibilité

Lenhof et Smid [92] proposent un algorithme pour maintenir la carte de visibilité d'un point de vue se déplaçant sur un « cercle à l'infini » autour d'un ensemble de sphères. Leur technique est assez directe et met à profit l'éloignement du point de vue à l'infini : la carte de visibilité devient alors planaire (comme une projection orthographique) et chaque sphère devient un cercle de rayon constant se déplaçant uniquement dans une « bande verticale » du plan de la carte de visibilité.

Examinons le cas d'une scène composée d'un ensemble S de n polygones dans $\mathbb{R}^3$. Soit v un point de vue. Plaçons une sphère *imaginaire* centrée sur v. Nous définissons la projection d'un point p de l'espace sur la sphère comme l'intersection du rayon $(v, \vec{vp})$ avec la sphère. Nous pouvons alors projeter toutes les arêtes des polygones de S sur la sphère. L'arrangement des arêtes projetées constitue un plongement sphérique d'un graphe planaire $\mathcal{G}$. La projection sur la sphère de l'ensemble des arêtes visibles depuis v forme un sous-graphe $\mathcal{V}$ de $\mathcal{G}$.

Nous utiliserons la variable t pour dénoter le temps. Si v bouge, on peut exprimer sa position comme une fonction $v(t)$ de t. Lorsque le point de vue v se meut dans l'espace libre ($\mathbb{R}^3$ privé des polygones de S), toutes les arêtes de $\mathcal{G}$ bougent sur la sphère (on peut imaginer que la sphère « suit » le point de vue). Notons que les graphes $\mathcal{G}$ et $\mathcal{V}$ changent continûment lorsque la vitesse de v n'est pas nulle. Cependant, la vue de v restera topologiquement la même tant que le graphe $\mathcal{V}$ ou le graphe $\mathcal{G}$ restent combinatoirement les mêmes. On notera $\mathcal{G}(t)$ (*resp.* $\mathcal{V}(t)$) le graphe $\mathcal{G}$ (*resp.* $\mathcal{V}$) au temps t. Ces deux graphes planaires sont plongés dans la sphère à l'aide de segments géodésiques. Ainsi, ils ne changent combinatoirement au temps t que lorsque trois arêtes comportent un point en commun au temps t. On dit que les 3 arêtes se « croisent » au temps t. Lors d'un tel croisement, les graphes $\mathcal{G}(t-\varepsilon)$ et $\mathcal{G}(t+\varepsilon)$ ne sont pas isomorphes : leur combinatoire est différente.

Nous distinguons 3 catégories d'événements : soient e_1, e_2 et e_3 trois arêtes de polygones non nécessairement différents deux à deux, dont les projections se croisent au temps t, induisant un changement combinatoire du graphe $\mathcal{G}$. Il existe donc, au temps t un segment s_1s_3 de $\mathbb{R}^3$ dont la droite support contient $v(t)$ et qui intersecte les trois arêtes e_1 e_2 et e_3 aux points s_1, s_2 et s_3 dans cet ordre (quitte à renuméroter).

- Si le segment s_1s_3 intersecte l'intérieur d'un polygone de S, alors l'événement $e_1e_2e_3$ est dit **transparent** (il aurait effectivement lieu du point de vue de v si les polygones étaient transparents).
- Si s_1 et s_3 sont mutuellement visibles, alors l'événement $e_1e_2e_3$ est dit **semi-opaque**.
- Si le graphe $\mathcal{V}$ subit également un changement combinatoire au temps t, on dira que l'événement $e_1e_2e_3$ est **opaque**. $e_1e_2e_3$ est un événement opaque si il est un événement semi-opaque et que s_1 est visible par le point de vue v.

Les événements opaques sont de la première importance. Ce sont eux, en effet, qui induisent un changement réel sur le polyèdre de visibilité d'un point de vue mobile. Il n'existe pas d'algorithme capable de calculer efficacement le seul ensemble

des événements opaques apparaissant lorsque v suit une trajectoire donnée. En revanche il n'est pas nécessaire de calculer tous les événements transparents pour en extraire les événements opaques : on peut en effet calculer efficacement l'ensemble des événements semi-opaques le long d'une trajectoire du point de vue.

Bern *et al.* [15] examinent le problème du calcul des événements transparents (au nombre de k_t) et opaques (au nombre de k_o). Pour une scène polygonale et une trajectoire rectiligne du point de vue, ils proposent un algorithme de calcul des événements transparents en temps $O((n^2+k_t)\log n)$ ou $O(n^2\log n+k_t)$ randomisé. La trajectoire du point de vue est un segment rectiligne $p : [0,1] \mapsto \mathbb{R}^3$. Pour chaque arête e d'un polygone de la scène, l'espace bidimensionnel des droites traversant e et p est balayé pour calculer l'ensemble des événements opaques (qui apparaissent comme des sommets lors du balayage).

Ils proposent également un algorithme pour le cas où la scène est un terrain polygonal, et dont le temps d'exécution est en $O((n+k_o)\lambda_3(n)\log n)$ où $\lambda_3(n)$ est une fonction légèrement surlinéaire ($\lambda_3(n) = \Theta(n\alpha(n))$ où α est la pseudo-inverse de la fonction d'Ackermann).

Le premier algorithme est également appliqué au problème du lancer de rayon de la façon suivante : après avoir calculé l'ensemble des événements transparents ayant lieu le long d'une trajectoire rectiligne, le polyèdre de visibilité de $p(0)$ est calculé et maintenu lors du déplacement du point de vue le long de sa trajectoire, grâce aux événements opaques extraits de l'ensemble des événements transparents. Une structure de localisation d'un point dans une partition de la sphère des directions orientées correspondant au polyèdre de visibilité est également maintenue.

Cette maintenance est effectuée dans une structure de données persistante, ce qui permet par la suite de répondre à une requête de lancer d'un rayon ayant pour origine un point de la trajectoire p et une direction quelconque en temps $O(\log^2 n)$.

Mulmuley [106] montre comment décomposer $\mathbb{R}^3$ en une partition dépendante du point de vue pour maintenir le polyèdre de visibilité d'un point de vue se déplaçant sur une trajectoire rectiligne. Si la scène comporte n polygones, alors l'algorithme de Mulmuley calcule l'ensemble des événements semi-opaques en temps $O(k_s \log n + n^2\alpha(n)\log n)$ et k_s est le nombre d'événements semi-opaques.

Efrat *et al.* [50] s'intéressent au problème de la maintenance du polyèdre de visibilité d'un point de vue mobile dans une scène composée de k polytopes de taille totale n. Ils exhibent plusieurs bornes sur l'arrangement de la projection des arêtes silhouettes de la scène, et sur certaines sous-structures. Ils montrent en particulier que la complexité de la vue d'un point de vue statique est $\Theta(kn)$ (on ne considère que les arêtes silhouettes), et que le nombre de changements combinatoires de cette vue est en $\Theta(k^2n)$ lorsque le point de vue est en translation, et en $\Theta(kn^2)$ lorsque le point de vue suit une trajectoire donnée par une équation algébrique. Dans le cas d'une trajectoire rectiligne (translation du point de vue), leur résultat se traduit directement en un algorithme de calcul de ces changements en temps $O(k^2n\log n)$.

Pop *et al.* [120] montrent comment on peut maintenir efficacement l'ensemble des arêtes silhouettes d'un polyèdre incrémentalement, c'est-à-dire quand la position du point de vue assume un ensemble discret de positions $\{p_0, p_1, p_2, \cdots\}$. Leur algorithme fonctionne dans un espace dual où le point de vue devient un plan et les plans support des faces du polyèdre deviennent des points. Deux positions successives du point de vue deviennent deux plans dans l'espace dual. Les points entre ces plans sont déterminés efficacement avec un algorithme de *range searching* puis l'ensemble des arêtes

silhouettes est mis à jour.

Maintenance d'un sur-ensemble des objets visibles

On examine maintenant un problème un peu différent des précédents. Il ne s'agit plus de maintenir la carte de visibilité du point de vue mobile, mais plutôt d'estimer rapidement et de manière conservative, au temps $t + \delta t$, l'ensemble des objets visibles par le point de vue, en s'aidant des informations connues au temps t (comme l'ensemble des objets visibles au temps t et la vitesse de chaque objet s'ils sont en mouvement).

Les méthodes décrites ci-après sont *a priori* mieux adaptées pour l'affichage d'une scène en temps-réel à l'aide des cartes graphiques, disponibles dans tous les PC actuels.

Coorg et Teller [31–33] proposent un tel algorithme, qui fonctionne comme suit. Les polygones de la scène sont insérés dans un *octree*. On suppose qu'au moment de dessiner la scène, on connaît un ensemble (pas trop grand) de polygones (appelés *oculteurs*) proches du point de vue et occupant une grande surface sur le plan image. Ces oculteurs sont alors utilisés lors d'un parcours de l'*octree* pour déterminer hiérarchiquement l'état de visibilité d'un nœud de l'*octree* (un nœud sera visible, partiellement visible ou invisible). Les structures utilisées pour calculer l'état de visibilité d'un nœud (une liste de plans supports et séparateurs) sont stockées dans ce nœud et réutilisées pour calculer les objets visibles au temps suivant. Enfin, dans chaque feuille de l'*octree*, on précalcule une liste de bons oculteurs qui seront utilisés comme oculteurs si le point de vue se trouve dans cette feuille.

2.5 Observateur de plus grande dimension

Comme nous l'avons vu dans l'introduction de cette thèse, les algorithmes décrits ci-dessus tirent parti de la notion de cohérence temporelle pour calculer rapidement la visibilité $V(t+\delta t)$ d'un point de vue mobile à partir de sa visibilité $V(t)$ au temps t. Un autre moyen de tirer parti de la cohérence temporelle est d'observer qu'étant donné un voisinage spatial $W(P)$ proche du point de vue P, l'ensemble des objets visibles depuis un point de $W(P)$ est en général seulement légèrement plus grand que l'ensemble des objets visibles depuis P. On peut donc subdiviser l'espace accessible par le point de vue en cellules (par exemple des parallélépipèdes rectangles) et précalculer pour chaque cellule C l'ensemble $\mathrm{PVS}'(C)$ des objets visibles depuis au moins un point de C [4]. En général, pour accélérer les calculs, on déterminera plutôt un ensemble $\mathrm{PVS}(C)$ qui sera une sur-estimation (au plus juste, si possible) de l'ensemble $\mathrm{PVS}'(C)$.

Cette idée générale a donné naissance à un grand nombre de méthodes adaptées à différents types de scènes, et est particulièrement adaptée au dessin d'une scène en temps interactif pour lequel on compte sur le *z-buffer* [21] des cartes graphiques pour déterminer précisément quel triangle est visible en chaque pixel de l'écran.

2.5.1 Calcul du PVS d'un ensemble de cellules

Lorsqu'un ensemble de cellules à été défini, chacune d'elles se voit attribuer une liste des polygones qu'elle contient. On effectue ensuite, toujours en précalcul, le calcul, pour chaque cellule C, de l'ensemble $\mathrm{PVS}(C)$ des cellules visibles depuis au moins

un point de C. Par la suite, si l'on connaît la cellule C' dans laquelle se trouve le point de vue, il suffit de parcourir les cellules de $\mathrm{PVS}(C')$ et de dessiner les polygones qu'elles contiennent avec l'aide de la carte graphique.

Teller [138] propose une telle approche dans le cas où la scène est de type architectural et a déjà été décomposée en un ensemble de cellules correspondant approximativement aux pièces du bâtiment (par exemple en calculant un arbre de décomposition binaire de l'espace (BSP)).

Durand *et al.* [47] résolvent le problème général à l'aide d'un balayage de la scène par un plan discrétisé (et géré par la carte graphique). Pour chaque face de chaque cellule (supposée rectangulaire 3D), le plan balaye la scène en partant de cette face et en s'éloignant de la cellule. On suppose que la face de départ émet de la lumière et induit donc des parties ombrées dans l'espace. Au fur et à mesure du balayage, l'ombre des objets balayés est accumulée sur le plan de balayage (en tenant compte de leur éventuelle « disparition » avec la distance). L'ombre ainsi maintenue sert à déterminer la visibilité des objets ou des cellules rencontrées lors du balayage. Les performances de cet algorithme sont remarquables puisque dans des scènes très denses, il est capable de déclarer (conservativement) que plus de 96% de la géométrie est invisible.

Nirenstein *et al.* [111] résolvent le même problème, mais de manière exacte (avec des calculs utilisant des `float`s, cependant). Les ensembles de droites traversant un polygone convexe (les faces bordant chaque cellule, et les triangles de la scène) sont représentés sous la forme de polytopes dans un espace 5D. Leur technique est alors fondée sur l'utilisation de soustractions booléennes dans cet espace 5D pour déterminer si une paire de polygones convexes sont en relation de visibilité ou pas. Leur méthode est très efficace puisqu'elle calcule des PVS exacts et donc non surestimés, cependant les temps de calcul sont assez longs (une scène représentant un village et contenant environ un million de triangles est prétraitée en environ 3 jours, ce qui est quand même très impressionnant !). Plus tard Nirenstein *et al.* [110] ont proposé une autre approche pour résoudre ce problème. Elle est de type *agressive*, c'est-à-dire qu'elle sous-estime toujours les PVS et peut donc entraîner la disparition d'objets qui auraient dû être dessinés. Cependant, si la sous-estimation reste très proche du PVS exact, les résultats à l'écran peuvent être acceptables (c'est notamment le cas si la scène est très détaillée, comme une forêt). Leur méthode détermine ainsi automatiquement l'ensemble des cellules : une cellule C sera subdivisée si les ensembles de polygones visibles (déterminés avec la carte graphique) aux coins de C diffèrent trop. L'avantage de cette méthode est qu'elle est très rapide : ils montrent l'exemple d'une forêt de 5 millions de triangles traitée en 1 heure et 19 minutes. Après ce prétraitement, 91.3% de la scène est estimée comme invisible, en moyenne, et donc non-dessinée, et l'erreur moyenne mesurée à l'écran n'est que de 0,3% (0,3% de la surface de l'image est fausse).

Haumont *et al.* [72] reprennent le problème du calcul exact de la visibilité mutuelle entre deux polygones. Il simplifient l'algorithme de Nirenstein *et al.* [111] en montrant que l'on peut se contenter de faire les opérations booléennes sur le 1-squelette des polytopes 5D. Ils apportent également plusieurs heuristiques très efficaces pour réduire les temps de calcul. Les temps de calcul obtenus sont en général meilleurs que ceux obtenus précédemment [111].

Laine [86] montre comment utiliser la disposition spatiale des cellules dont on veut calculer les PVS pour obtenir un algorithme dont le temps d'exécution est sensible à la taille des PVS. Son algorithme est indépendant de la méthode choisie pour

le calcul de visibilité mutuelle entre deux polygones (exact, conservatif ou agressif). Beaucoup d'autres travaux ont été effectués dans ce domaine. Nous référons le lecteur aux états de l'art écrits par Durand [45] et Cohen-Or *et al.* [30] pour plus d'informations.

2.5.2 Cellules et portails

Pour une scène de type architectural, on peut mettre en œuvre une décomposition de la scène en cellules et portails. Les cellules sont des polyèdres dont les bords contiennent une partie des polygones de la scène et des *portails*. Les portails sont des polygones « transparents » qui séparent chaque cellule de ses cellules voisines. Cette décomposition permet de déterminer rapidement une surestimation de l'ensemble des polygones visibles de la scène, sans précalcul de PVS (voir les travaux de Luebke *et al.* [97]). Le Chapitre 9 (page 137) expose une nouvelle méthode de construction d'une telle décomposition ; aussi nous référons le lecteur aux deux premières sections de ce chapitre pour de plus amples détails sur les cellules, les portails et leurs diverses utilisations.

Deuxième partie

Maintenance de la vue d'un point mobile

Chapitre 3

Visibilité sur des points dans le plan

Dans ce chapitre, nous abordons le problème de la maintenance de la vue d'un point mobile dans le plan, dans le cas le plus simple où les objets peuplant $\mathbb{R}^2$ sont des points. Ces derniers sont donc visibles par le point de vue à tout instant. Aussi, nous nous intéressons à la maintenance efficace de l'ordre radial de ces points autour du point de vue. L'algorithme résultant est également applicable à la maintenance de ces points ordonnés selon leur distance au point de vue. Ce travail a été effectué en collaboration avec Olivier Devillers, Vida Dujmović, Hazel Everett, Sue Whitesides, et Steve Wismath et présenté à la *Canadian Conference on Computational Geometry* [39].

3.1 Introduction

Soit P un ensemble de n points de $\mathbb{R}^2$. Par souci de simplicité, nous ferons l'hypothèse que trois points quelconques de P ne sont pas alignés. V sera, à tout moment, un point du plan, appelé *point de vue*, qui peut se déplacer continûment dans le plan, en évitant toutefois l'ensemble de points P.

Pour toutes les positions possibles de V, sauf un ensemble de mesure nulle, tous les points de P sont visibles par V, c'est-à-dire que le segment $[Vp]$ ne contient aucun autre point de P. Aussi, la seule structure « de visibilité » que nous pouvons apporter au couple (P, V) est l'ordre circulaire des points de P autour de V. Nous souhaitons maintenir cet ordre circulaire tandis que le point de vue bouge.

Le scénario est le suivant : l'ensemble P nous est donné avec la position initiale du point de vue V. À ce stade, nous pouvons procéder à un pré-traitement des données. Ensuite, la trajectoire de V nous est donnée « en ligne », c'est-à-dire par morceaux, au bon vouloir d'un *utilisateur*. Nous supposerons pour le moment que chaque morceau de la trajectoire de V est un segment de droite.

Soit Ord_V la liste circulaire des points de P autour de V. Lors du déplacement de V entre deux instants donnés, l'ordre circulaire des points de P autour de V ne change qu'un nombre fini de fois. Nous allons montrer le théorème suivant :

Théorème 3.1. *Avec un pré-traitement en temps $O(n \log n)$ et linéaire en espace, nous pouvons calculer les changements de Ord_V lors du déplacement de V le long d'un segment σ en temps $O(k \log n + s)$ amorti ou $O(k \log^2 n + s)$ au pire, où k est le nombre d'états différents de Ord_V le long de σ et s est la taille de la sortie.*[1]

[1] Voir la « Remarque sur s » page 38.

La complexité amortie provient en fait d'un théorème de Brodal et Jacob :

Théorème 3.2 ([17]). *Il existe une structure de données pour le problème dynamique de l'enveloppe convexe dans le plan, qui supporte les opérations d'insertion et de suppression en temps $O(\log n)$ amorti, où n est le nombre de points avant l'opération. La structure est linéaire en espace.*

La complexité dans le cas le pire provient d'un résultat de Overmars et van Leeuwen [113].

La structure de données utilisée pour démontrer le théorème 3.2 ne construit pas explicitement l'enveloppe convexe des points du plan, mais permet, entre autre, de calculer en temps $O(\log n)$ les deux segments de l'enveloppe convexe intersectés par une droite. C'est cette propriété que nous utiliserons.

Une première approche. Voici une première approche simple pour répondre au problème donné dans le théorème 3.1, qui n'atteint pas, cependant, les complexités en temps annoncées. En pré-traitement, nous trions les points de P circulairement autour de V pour construire Ord_V. Ensuite, pour chaque paire de points consécutifs dans Ord_V, insérons un événement-futur dans une queue de priorité, qui indique à quel moment la permutation de ces deux points aura lieu (si jamais elle a lieu). La mise à jour de Ord_V procède alors comme suit :

1. *Pop* le premier événement de la queue.
2. Permuter les deux points concernés dans Ord_V.
3. Supprimer deux événements de la queue et en insérer au plus 3 nouveaux.

Il faudra faire spécialement attention lorsque le point de vue V sort de l'enveloppe convexe de P, mais cela n'augmente pas la complexité totale. En tout, pour une trajectoire en segment de droite donnée, la complexité en temps est de $O(n \log n)$ pour le pré-traitement, puis $O(n \log n + k \log n + s)$ en temps. Le terme $O(n \log n)$ est nécessaire puisque, alors que la nouvelle trajectoire vient à être connue, nous devons vider la queue des événements, puis calculer et insérer $O(n)$ nouveaux événements futurs dans la queue.

Dans la section suivante, nous montrons comment se débarrasser du terme en $O(n \log n)$, au prix d'une nouvelle complexité $O(k \log n + s)$ en *temps amorti*, ou bien $O(k \log^2 n + s)$ dans le cas le pire.

3.2 Le CL-arrangement

Examinons notre problème plus avant. Supposons les points de P déjà triés autour de V. Cet ordre change lorsque V croise certaines demi-droites du plan. Plus précisément, nous avons le lemme suivant :

Lemme 3.3. *Soit p, q deux points consécutifs de Ord_V. Si $n = 2$, alors cet ordre ne change jamais. Si $n > 2$ alors l'ordre de p et q dans Ord_V est inversé lorsque V croise l'une des deux demi-droites constituant $(pq) \setminus [pq]$. Voir la figure 3.1. Nous appelons ces deux demi-droites une* Chipped Line *d'où le terme de CL-arrangement.*

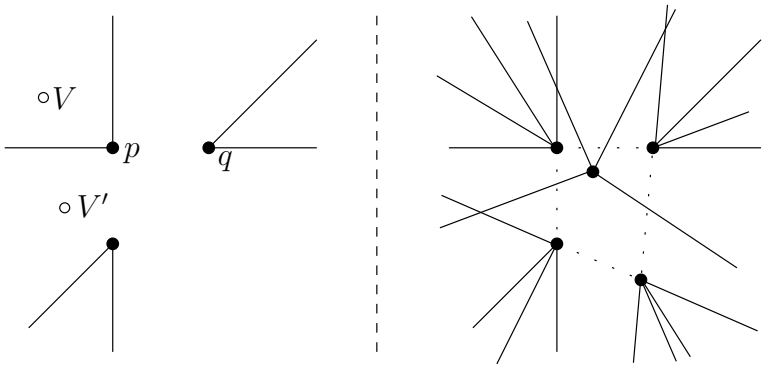


FIG. 3.1. À gauche, lorsque V se déplace en ligne droite vers V', les points p et q permutent dans Ord_V. **À droite**, le CL-arrangement de cinq points. En pointillés, l'enveloppe convexe de P.

Nous définissons le **CL-arrangement** de P comme l'arrangement induit par les ensembles
$\{(pq) \setminus [pq], p \neq q \in P\}$. Voir la figure 3.1.

Lemme 3.4. *Ord_V est constant à l'intérieur d'une cellule de dimension 2 du CL-arrangement de P.*

Preuve : C'est une conséquence directe du lemme 3.3. □

Comme illustré sur la figure 3.1, certaines cellules du CL-arrangement ne sont pas convexes. Or, la convexité est une propriété intéressante à exploiter. Nous pouvons en fait ajouter quelques segments de droites au CL-arrangement pour rendre toutes ses 2-cellules convexes.

Une cellule est non-convexe si un de ses sommets n'est pas convexe (l'angle interne en ce sommet dépasse π). Soit v un tel sommet. Tout d'abord, v est un point de P, car tous les autres sommets du CL-arrangement sont des intersections de demi-droites et sont donc convexes. Puisque v n'est pas convexe, l'ensemble des $n-1$ demi-droites qui émanent de v dans le CL-arrangement sont contenues dans un demi-plan $\mathcal{H}^+$ défini par une droite passant par v. Puisque chacune de ces demi-droites sont définies par la position des $n-1$ autres points de P, il s'ensuit que ces points sont dans $\mathcal{H}^-$. Ainsi, par définition, v est un sommet de l'enveloppe convexe de P. En ajoutant les arêtes de l'enveloppe convexe de P au CL-arrangement, nous éliminons donc toutes les 2-cellules non-convexes. Voir la figure 3.1.

Dans la suite, le CL-arrangement contient les arêtes de l'enveloppe convexe de P.

Posons V dans une 2-cellule C (une cellule de dimension 2) du CL-arrangement de P. Soit e une arête de la frontière de C. e est un segment dont la droite support est engendrée par deux points p et q de P. Rapprochons V infiniment près du milieu de e tout en restant dans C[2].

- Si e n'est pas une arête de l'enveloppe convexe de P. Puisque nous avons fait l'hypothèse que tout triplet de points dans P est linéairement indépendant, nous pouvons en déduire que p et q sont consécutifs dans Ord_V, puisqu'ils vont permuter si on bouge V infinitésimalement à travers la droite (pq).
- Si e est une arête de l'enveloppe convexe de P. Alors p et q sont diamétralement opposés (par rapport à V), et, puisque $[pq]$ est une arête de l'enveloppe convexe, un des deux demi-plans définis par la droite (pq) est vide de tout point de P. Par conséquent, p et q sont consécutifs dans Ord_V.

[2] Si e est une demi-droite, il suffit d'éloigner un peu V de l'unique sommet de e.

Nous avons montré le lemme suivant.

Lemme 3.5. *Les arêtes d'une 2-cellule C du CL-arrangement de P sont déterminées par les paires de points consécutifs dans Ord_V pour un point V quelconque dans C.*

Ce dernier lemme est encourageant. Il nous indique que la donnée de la liste circulaire Ord_V des points de P autour de V suffit à construire la cellule du CL-arrangement de P dans laquelle se trouve V. C'est une propriété somme toute assez intuitive.

Soit $Ord_V = v_0, v_1, \ldots, v_{n-1}$. Notons $\mathcal{H}_i$ le demi-plan défini par la droite (v_i, v_{i+1}) et contenant V.[3] L'intersection de ces n demi-plans est la cellule C_V du CL-arrangement, contenant V. Nous pouvons maintenant décrire l'algorithme qui satisfait les complexités annoncées dans le théorème 3.1.

Pré-traitement. En temps $O(n \log n)$, les points de P sont triés radialement autour du point de vue V, et les n demi-espaces sont insérés dans la structure de données $\mathcal{S}$ dont l'existence est montrée par le théorème 3.2 [17]. Nous utilisons ici un résultat classique de dualité point/droite qui rend équivalent la maintenance d'une enveloppe convexe de n points avec celle de l'intersection de n demi-espaces [113].

Répondre à une requête de mouvement de V. Une trajectoire (un segment de droite orienté) σ est donnée. Alors, nous utilisons $\mathcal{S}$ pour lancer un rayon depuis V dans la direction de σ et trouver l'arête e de V_C intersectée par le rayon. Si le point d'intersection est plus loin que l'extrémité finale de σ, on s'arrête. Sinon, on met à jour Ord_V en y permutant les deux points q, r de P contenus dans la droite support de e,

$$\ldots p, q, r, s \ldots \quad \Rightarrow \quad \ldots p, r, q, s \ldots,$$

et on met à jour $\mathcal{S}$ en supprimant les demi-plans définis par (p, q) et (r, s), puis en insérant ceux définis par (p, r) et (q, s). La traversée d'une cellule de l'arrangement coûte donc $O(\log n)$ en temps amorti, ou $O(n \log^2 n)$ en temps dans le cas le pire.

Trajectoire quelconque. Notons que l'algorithme décrit ci-dessus marche identiquement pour une trajectoire quelconque du point de vue. Il faut bien entendu que la première intersection entre une trajectoire donnée σ et la frontière de C_V soit calculable, en temps $\tau(n)$. Selon le type de trajectoire envisagée, la complexité de l'algorithme sera augmentée (puisqu'on peut raisonnablement supposer que $\tau(n) = \Omega(\log n)$).

3.3 Autre tri, même algorithme

Il est intéressant de remarquer que l'algorithme que nous venons de décrire peut être utilisé tel quel pour répondre à un autre problème similaire. Au lieu de maintenir la liste des points de P dans un ordre circulaire autour de V, nous souhaitons maintenant les maintenir triés en fonction de leur distance à V. Soit Dis_V la suite des points de P triés en fonction de leur distance à V.

[3] On définira $v_n = v_0$.

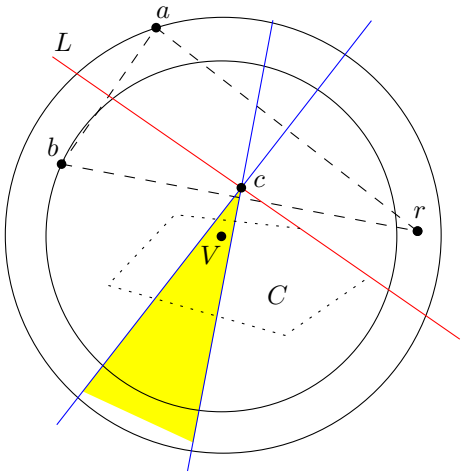


FIG. 3.2. Illustration de la preuve du lemme 3.6.

Que deviennent les divers éléments que nous avons décrits à la section précédente ?

Tout d'abord, le CL-arrangement devient l'arrangement $\mathcal{D}$ des droites (entières) bissectrices de chaque paire de points de P. Il y a $n(n-1)$ telles droites, dont l'arrangement a une taille en $O(n^4)$ comme précédemment. Notons que toutes les cellules de $\mathcal{D}$ sont convexes.

On remarque facilement que dans chaque 2-cellule de $\mathcal{D}$, l'ordre des points de P, triés selon leur distance à V, est constant. On obtient également que lorsque V passe d'une cellule à une autre, deux points de P permutent dans Dis_V, et ce sont précisément les deux points qui définissent la bissectrice que V vient de croiser, comme le montre le lemme suivant :

Lemme 3.6. *Les arêtes d'une 2-cellule C de $\mathcal{D}$ sont déterminées par les paires de points consécutifs dans Dis_V pour un point V quelconque dans C.*

Preuve : Soit C une cellule de l'arrangement $\mathcal{D}$, et posons V à l'intérieur de C. Soit L la bissectrice définie par les points a et b et supportant une arête du bord de C. Nous allons démontrer par l'absurde que a et b sont consécutifs dans Dis_V. Supposons le contraire, alors il existe un point r de P dans la couronne définie par V, a et b (voir Figure 3.2). c, le centre du cercle circonscrit au triangle abr est sur la droite L, et les deux autres bissectrices (définies par ar et br) passent par c et partagent le demi-plan dans lequel est V en trois régions. Comme $|V-b| < |V-r| < |V-a|$, V doit être dans la région du milieu (en jaune sur la figure), ce qui contredit l'hypothèse que L supporte une arête du bord de C. □

L'algorithme pour maintenir Dis_V s'ensuit — identique au précédent :

Théorème 3.7. *Avec un pré-traitement en temps $O(n \log n)$ et linéaire en espace, nous pouvons calculer les changements de Dis_V lors du déplacement de V le long d'un segment σ en temps $O(k \log n + s)$ amorti ou $O(k \log^2 n + s)$ dans le cas le pire, où k est le nombre d'états différents de Dis_V le long de σ et s est la taille de la sortie.*

Remarque sur s. Dans les théorèmes 3.1 et 3.7, s est la taille de la sortie de l'algorithme. Si nous souhaitons seulement connaître Ord_V ou Dis_V après que V soit arrivé à la fin de sa trajectoire, alors $s = \Theta(n)$. Si nous souhaitons connaître complètement tous les états de Ord_V ou Dis_V pendant le mouvement de V, alors $s = \Theta(kn)$. Enfin, si l'on souhaite connaître tous les changements de Ord_V ou Dis_V au cours du mouvement de V, alors $s = \Theta(k)$, puisque chaque événement induit un changement de taille $O(1)$.

Chapitre 4

Maintenance de la vue d'un point mobile dans le plan

Nous avons pu mesurer, à la Section 2.4, l'ampleur des travaux accomplis sur le problème de la maintenance de la visibilité d'un point de vue mobile dans le plan. Dans sa thèse [67], Olaf Hall-Holt a étudié ce problème pour des scènes composées de convexes deux-à-deux disjoints. Il propose deux algorithmes qui le résolvent relativement efficacement et surtout en utilisant un espace linéaire : le *visible zone algorithm* et le *corner-arc algorithm*. Le premier considère l'ensemble des segments libres alignés avec le point de vue V comme une courbe $\mathcal{C}$ dans le complexe de visibilité. En relâchant une certaine contrainte sur les parties invisibles de $\mathcal{C}$, il montre comment cette dernière peut être maintenue efficacement d'une manière évoquant le balayage topologique d'un arrangement de droites [48] ou du complexe de visibilité [118]. La partie de $\mathcal{C}$ formée de segments libres touchant V nous donne alors, à n'importe quel moment, le polygone de visibilité de V. Le second (le *corner arc algorithm*) fonctionne directement dans l'espace primal. D'abord, l'espace libre est pseudo-triangulé (en incluant le point de vue comme un sommet de la pseudo-triangulation). Ensuite, cette partition est ajustée et maintenue de telle sorte qu'elle contienne à tout moment le polygone de visibilité de V.

Dans ce chapitre, nous ne décrirons pas plus avant le *corner-arc algorithm*, mais nous concentrons sur le *visible zone algorithm*, car ce dernier apparaît comme un bon candidat pour une extension en 3D. Bien que ce but n'ait pas été atteint, nous souhaitons rendre compte de manière détaillée de la preuve de l'algorithme en 2D : ce chapitre est ainsi consacré à l'étude du *visible zone algorithm* développé par Hall-Holt. Un effort particulier a été apporté à l'illustration de tous les concepts introduits pour la description de cet algorithme. De plus, nous donnons une preuve plus simple de sa correction : la preuve du Lemme 4.12 est nouvelle et simplifie considérablement l'exposé de l'Appendice A de la thèse de Hall-Holt. Aussi, la Section 4.8 donne des détails d'implémentation absents de la présentation originale de l'algorithme ainsi que quelques observations nouvelles sur les structures géométriques que nous allons manipuler.

La Section 4.1 introduit plus formellement le problème que résout le *visible zone algorithm*. Dans la Section 4.2, nous étudions un algorithme simple mais relativement inefficace pour la maintenance du polygone de visibilité d'un point mobile autour de k convexes disjoints du plan. À partir de la Section 4.3, nous réexaminons cet algorithme comme un balayage « paresseux » du complexe de visibilité, puis exposons

le développement du *visible zone algorithm*.

4.1 Maintenance du polygone de visibilité

Soit $\mathcal{O} = \{O_1, O_2, \ldots, O_k\}$ un ensemble de k régions convexes disjointes du plan. Chaque région est appelée un **obstacle**, un **objet** ou un **oculteur**. Le complémentaire de l'union des objets de $\mathcal{O}$ est appelé l'**espace libre** et est noté $\mathcal{F}$. L'espace libre est la région du plan où un point P est libre de se mouvoir. Voir la Figure 4.1 à gauche.

Soit P un point (de vue) dans l'espace libre. Le **polygone de visibilité** $\mathcal{V}_P$ de P est défini comme l'union de tous les points de $\mathcal{F}$ visibles depuis P, ou comme l'union des segments ayant un sommet en P, une direction quelconque (dans $\mathbb{S}^1$), et s'étendant jusqu'au premier objet rencontré. Quand aucun objet de la sorte n'est rencontré, alors le segment s'étend à l'infini. $\mathcal{V}_P$ est une région connexe et étoilée du plan. Voir la Figure 4.1 à droite.

Observons que la frontière de $\mathcal{V}_P$ nous donne explicitement la liste circulaire des objets de $\mathcal{O}$ visibles par P. Plus précisément, la frontière de $\mathcal{V}_P$ peut être décrite combinatoirement comme une liste circulaire $v_0, v_1, ..., v_n$ où $v_i \in \{1, 2, ..., k\} \cup \{\infty\}$ indique quel objet est vu dans le $i^{\text{ème}}$ intervalle angulaire autour de P, et ∞ indique un intervalle angulaire où P ne voit aucun objet de $\mathcal{O}$ (P voit l'infini, ou le « ciel bleu »).

L'algorithme de calcul du polygone de visibilité de P est linéaire en espace et prend $O(k \log k)$ en temps, si on suppose que les tangentes à un objet, passant par P, peuvent être calculées en temps constant. La construction de $\mathcal{V}_P$ peut se faire en balayant le plan radialement avec une demi-droite dont P est l'origine, et en maintenant l'ordre des objets intersectés par cette demi-droite au cours du balayage [36, Chapitre 15].

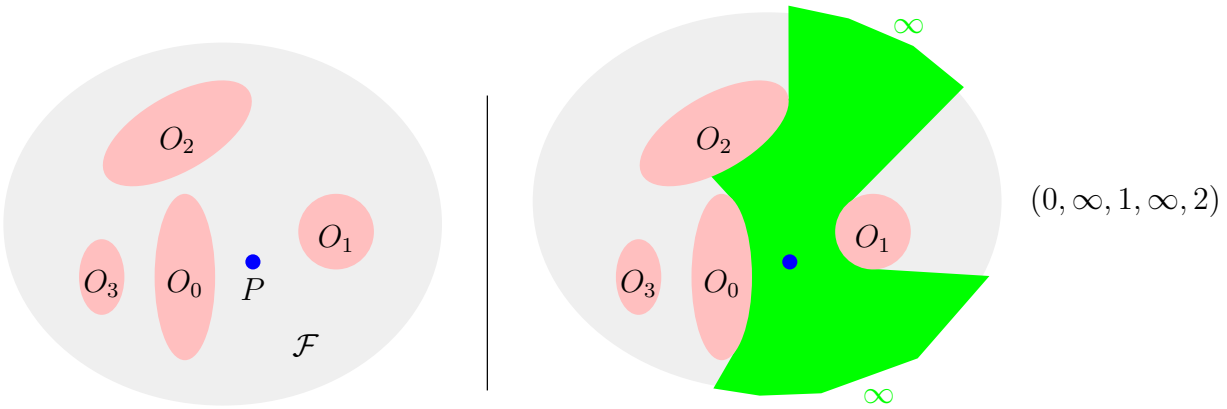


FIG. 4.1. La figure de **gauche** montre quatre obstacles et le point de vue P baignant dans l'espace libre. **À droite**, le polygone de visibilité est dessiné en vert, et la liste circulaire des objets visibles est donnée.

Supposons maintenant que P bouge dans l'espace libre, et que l'on connaisse le mouvement de P à court terme, exprimé par exemple à l'aide d'une équation algébrique. Nous souhaitons maintenir $\mathcal{V}_P$ continûment alors que le temps passe.

4.2 La décomposition radiale

Commençons par observer que le polygone de visibilité en lui-même ne suffit pas pour sa maintenance au cours du temps : si nous connaissons uniquement $\mathcal{V}_P$ et la

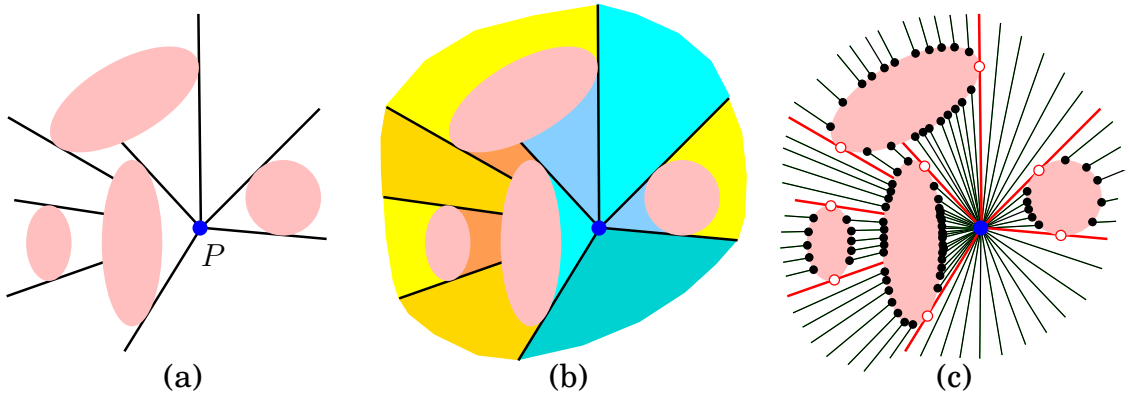


FIG. 4.2. (a) Les murs radials (tangentes) de la décomposition radiale. (b) Les faces de la décomposition radiale. (c) Nous interprétons $\mathcal{D}_P$ non plus comme une décomposition de l'espace libre, mais comme une courbe unidimensionnelle dans l'espace des segments libres maximaux orientés.

trajectoire de P à court terme, il est impossible de trouver le prochain événement de visibilité (le prochain changement combinatoire de $\mathcal{V}_P$) plus rapidement qu'une approche « en force » par recherche linéaire parmi les $O(k^2)$ bitangentes de $\mathcal{O}$. En effet, le prochain événement de visibilité peut être l'apparition d'un objet H précédemment caché. Puisque H n'apparaît pas dans la liste circulaire décrivant $\mathcal{V}_P$, il ne sera pas facile de détecter qu'il est impliqué dans le prochain événement de visibilité pour la trajectoire de P connue.

Définition 4.1. *Une* **tangente à droite** *à un objet O est une tangente orientée, et localement à droite de O. Une* **tangente à gauche** *est définie de manière similaire, et une* **tangente** *est définie comme une tangente (orientée) droite ou gauche. La structure de données pour une tangente contient donc : l'objet de tangence, le côté de tangence (gauche ou droite), l'***oculteur avant** *(le premier objet rencontré en suivant la tangente dans le sens de son orientation) et l'***oculteur arrière** *(le premier objet rencontré en suivant la tangente dans le sens opposé). Un oculteur peut éventuellement être l'infini ou le point de vue P lui-même.*

Étant donnée la position du point P dans l'espace libre, il existe exactement deux tangentes (une gauche, une droite) dont la droite support passe par P.

Définition 4.2. *Nous pouvons alors définir la* **description combinatoire du polygone de visibilité** *$\mathcal{V}_P$ comme étant la liste, triée circulairement, des tangentes qui partagent le point P en tant qu'oculteur arrière.*

Par exemple, le polygone de visibilité de la Figure 4.1 consiste en 5 tangentes ordonnées circulairement. On peut directement récupérer la liste des objets visibles à partir de cette liste de tangentes. Quand le point de vue bouge, nous pouvons nous attendre à ce que son polygone de visibilité change plusieurs fois.

Définition 4.3. *Un* **événement de visibilité** *est un changement de la description combinatoire de $\mathcal{V}_P$.*

Étant donnée une trajectoire du point de vue $P(t), t \in [0,1]$, les événements de visibilité se produisent à des moments discrets de l'intervalle $[0,1]$.

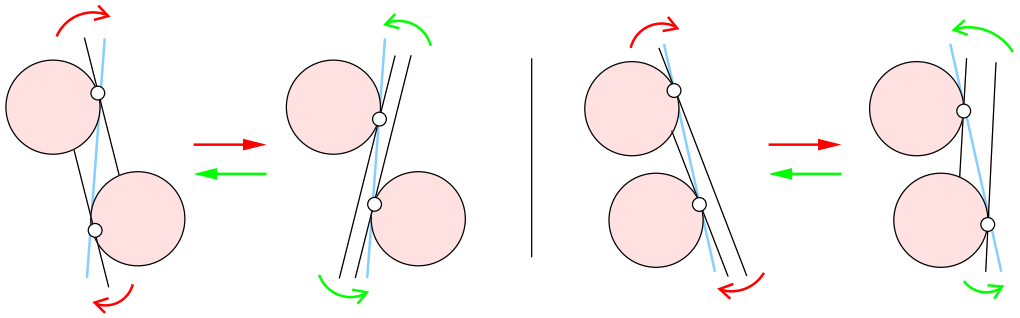


FIG. 4.3. Les événements modifiant $\mathcal{D}_P$. Deux murs radials (tangentes, en noir) se croisent sur une bitangente (en bleue). Les flèches courbes indiquent le sens de glissement des murs radials. Les flèches droites indiquent le sens de lecture de la figure correspondant aux flèches courbes de même couleur.

Nous décrivons maintenant la **décomposition radiale**, une structure simple contenant le polygone de visibilité et facilement maintenable au cours du temps. C'est une décomposition de l'espace libre très semblable à une décomposition verticale. Elle dépend de la position du point de vue P et nous la noterons $\mathcal{D}_P$. En fait, quand P est poussé infiniment loin des objets de $\mathcal{O}$ dans une direction ω, $\mathcal{D}_P$ coïncide avec la décomposition verticale, où ω est considérée comme la verticale. $\mathcal{D}_P$ est définie ainsi : pour chaque objet O de $\mathcal{O}$, nous construisons deux segments libres maximaux (appelés *murs radials*) tangents à O et alignés avec P, de chaque côté de O. Si on les oriente dans la direction $P \rightarrow \infty$, l'un des deux murs, $t_{O,left}$ est une tangente à gauche, tandis que l'autre, $t_{O,right}$ est une tangente à droite. Voir Figure 4.2(a). Notons que ces murs radials s'étendent vers et s'éloignent de P jusqu'à rencontrer un objet (un *bloqueur*) qui peut être P, ∞ ou appartenir à $\mathcal{O}$.

Les **faces de la décomposition radiale** sont les sous-ensembles connexes de l'espace libre induits par $\mathcal{O}$ et les murs radials, comme illustré sur la Figure 4.2(b). Elles sont décrites de manière simple :

- Chaque face possède un mur gauche et un mur droit, ainsi qu'un bloqueur arrière (dans $\mathcal{O} \cup \{P\}$) et un bloqueur avant (dans $\mathcal{O} \cup \{\infty\}$).
- Chaque mur est adjacent à trois faces.

Notons que le nombre de faces dans $\mathcal{D}_P$ est constant, égal à $3k$. Clairement, la taille de $\mathcal{D}_P$ est $O(k)$. Enfin, on peut simplement retrouver le polygone de visibilité en extrayant les faces de $\mathcal{D}_P$ dont le bloqueur avant est P (voir les faces bleues sur la Figure 4.2(b)).

4.2.1 Maintenance de $\mathcal{D}_P$

Si nous pouvons maintenir la décomposition radiale tandis que P bouge, alors nous pouvons aussi maintenir le polygone de visibilité puisque $\mathcal{V}_P \subset \mathcal{D}_P$. Nous montrons que la structure de la décomposition radiale peut être maintenue au cours du temps sans information supplémentaire autre que la décomposition radiale elle-même.

Lorsque P bouge dans l'espace libre, les $2k$ murs radials glissent le long du bord de leur objet de tangence pour rester alignés avec P. Nous observons qu'aussi longtemps que chaque mur possède les mêmes bloqueurs avant et arrière, la combinatoire de $\mathcal{D}_P$ ne change pas. Autrement dit, les événements accompagnant un changement conbinatoire de $\mathcal{D}_P$ correspondent au fait qu'un mur radial devient bitangent à une

paire d'objets de $\mathcal{O}$. On peut facilement vérifier que dans un tel cas, il y a en fait deux murs radials qui se croisent simultanément en une même bitangente. La Figure 4.3 illustre ces événements qui modifient la combinatoire de $\mathcal{D}_P$ quand P bouge. Quand un tel événement a lieu, $\mathcal{D}_P$ peut être facilement mise à jour en temps constant. Notons enfin que chaque événement correspond aussi à l'écrasement (ou la disparition) d'une face de $\mathcal{D}_P$: les deux murs qui se croisent sont les murs gauche et droit d'une même face de $\mathcal{D}_P$. Nous pouvons donc prédire quel sera l'événement prochain en calculant pour chaque face le moment auquel ses deux murs radials se croiseront.[1]

4.2.2 Algorithme de maintenance de $\mathcal{D}_P$

Initialisation.

- Premièrement, nous construisons $\mathcal{D}_{P(0)}$, la décomposition radiale induite par la position initiale $P(0)$ du point de vue. Cette phase prend un temps $O(k \log k)$ tout en occupant un espace linéaire.
- Deuxièmement, pour chaque face F de $\mathcal{D}_{P(0)}$, nous calculons le temps t_F auquel F va disparaître, à l'aide de la trajectoire courante de P. On insère t_F (ainsi qu'un pointeur sur la face F) dans une queue de priorité EVENT_QUEUE, qui permet un accès efficace au plus petit élément, c'est-à-dire au prochain événement. Cette phase prend un temps $O(k \log k)$ tout en occupant un espace linéaire.

Maintenance.

Tant que EVENT_QUEUE n'est pas vide, les opérations suivantes sont effectuées :

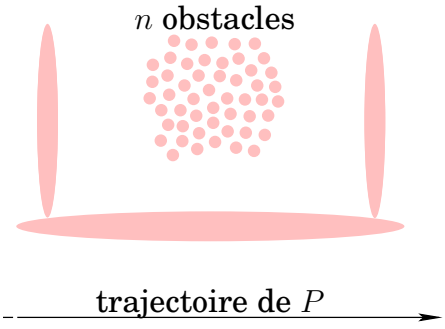
- On récupère le prochain événement dans la file EVENT_QUEUE pour un coût $O(\log k)$ en temps.
- On met à jour les cinq faces de $\mathcal{D}_P$ dont la description combinatoire change. Cette opération est effectuée en temps constant.
- On supprime les anciens et on calcule les nouveaux instants auxquels ces cinq faces vont disparaître. On insère ensuite ces événements dans EVENT_QUEUE, pour un coût total $O(\log k)$ en temps.

L'initialisation prend $O(k \log k)$ en temps, et la mise à jour de $\mathcal{D}_P$, pour chaque événement coûte $O(\log k)$. L'algorithme complet utilise un espace de travail de taille linéaire puisque $\mathcal{D}_P$ a une taille linéaire et pas plus de $2k$ événements sont en permanence présents dans la queue de priorité. Le problème principal de cet algorithme est que, si la trajectoire de P change, il nous faut vider EVENT_QUEUE et la remplir à nouveau comme à l'étape de l'initialisation, pour un coût linéaire, ce qui peut être prohibitif.

En un sens, cet algorithme est efficace, puisque son espace de travail est minimal (linéaire en la taille de la scène $\mathcal{O}$, et avec une petite constante), et la mise à jour de $\mathcal{D}_P$ pour chaque événement de visibilité prend un temps logarithmique. Cependant, pour une trajectoire donnée, le nombre total de changements combinatoires de $\mathcal{D}_P$ peut être bien supérieur au nombre d'événements de visibilité, qui sont les événements qui nous intéressent en premier lieu.

[1] Attention : deux murs radials d'une même face ne se croiseront pas forcément dans le futur.

Par exemple, si la scène est complètement occultée de P par un très gros objet, le nombre de changements de $\mathcal{D}_P$ peut être en $O(k^2)$ alors que le nombre de changements du polygone de visibilité $\mathcal{V}_P$ est constant. C'est par exemple le cas, dans la figure de droite, si P suit la trajectoire indiquée de $-\infty$ à $+\infty$. Retenons que maintenir la décomposition radiale peut coûter très cher par rapport au coût de maintenance du polygone de visibilité $\mathcal{V}_P$ proprement dit.



Nous avons vu que le polygone de visibilité n'est pas suffisant pour être maintenu seul lorsque son centre — le point de vue — bouge. La décomposition radiale apporte une solution qui n'est pas tout à fait satisfaisante. Peut-on faire mieux ?

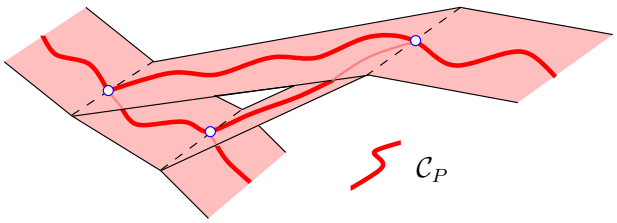
4.3 Le *visible zone algorithm*

Dans la suite de ce chapitre, nous décrivons le *visible zone algorithm* (VZA) conçu par Hall-Holt [67]. Ce dernier montre que pour une classe très générale de scènes (dites *non-alignées*), le coût du VZA associé à un événement de visibilité est constant (en négligeant le facteur logarithmique dû à la queue de priorité). De plus, ce sont les seuls événements à prendre en compte quand le point de vue se déplace sur une trajectoire connue. Aussi, quand le point de vue change de trajectoire, le coût associé est linéaire en la taille du polygone de visibilité au moment où la trajectoire change, ce qui est optimal. Nous devons cependant nuancer nos propos. En effet, la constante qui se cache derrière [2] le temps de mise-à-jour de la structure de données lorsqu'un événement de visibilité a lieu, est exponentielle par rapport à certaines quantités descriptives de la scène. Nous ne donnerons pas plus de détails dans ce chapitre et référons le lecteur à l'article de Hall-Holt [66]. Nous n'examinerons pas non plus, la question de la correction du VZA et référons le lecteur à la thèse de Hall-Holt [67] pour les détails concernant la terminaison de l'algorithme de mise-à-jour.

Pour décrire le *visible zone algorithm*, nous commençons par introduire une autre manière de voir la décomposition radiale $\mathcal{D}_P$. Regardons $\mathcal{D}_P$ sur la Figure 4.2(b). Nous observons qu'une face de $\mathcal{D}_P$ peut aussi être représentée par l'union d'un ensemble de segments libres maximaux alignés avec P. Ceci est illustré sur la Figure 4.2(c). Les segments libres alignés avec P, dont l'union (dans le plan) est une face F de $\mathcal{D}_P$, peuvent être vus comme une courbe unidimensionnelle à l'intérieur d'une 2-cellule du complexe de visibilité, qui relient les deux murs radials de F.

[2] Nous devrions écrire « devant ».

Plus généralement, $\mathcal{D}_P$ peut être vue comme une courbe $\mathcal{C}_P$ à l'intérieur du complexe de visibilité $\mathcal{VC}$ de $\mathcal{O}$. Lorsque P bouge, $\mathcal{C}_P$ se déforme et les événements qui modifient $\mathcal{D}_P$ ont lieu lorsque $\mathcal{C}_P$ croise un sommet de $\mathcal{VC}$. Notons que $\mathcal{C}_P$ est unidimensionnelle en tout point, sauf aux intersections de $\mathcal{C}_P$ avec une arête du complexe de visibilité $\mathcal{VC}$, où $\mathcal{C}_P$ a une valence de 3. Voir les points blancs/bleus sur la figure de droite. Soulignons les propriétés de $\mathcal{C}_P$ suivantes :



Vue schématique de $\mathcal{C}_P$ dans $\mathcal{VC}$.

P0 Tous les segments de $\mathcal{C}_P$ sont alignés avec le point de vue P.

P1 Soit v un segment de $\mathcal{C}_P$ tangent à un objet. Dans le complexe de visibilité, v a une valence de 3 : trois morceaux de $\mathcal{C}_P$ partent de v, un à l'intérieur de chacune des trois 2-cellules de $\mathcal{VC}$ adjacentes à v.

P2 Si $\mathcal{C}_P$ intersecte une 2-cellule f du complexe de visibilité, alors les deux chaînes de bordure de f sont intersectées exactement une fois par $\mathcal{C}_P$.

P3 Soit $O \in \mathcal{O}$. $\mathcal{C}_P$ contient exactement deux segments tangents à O (un à gauche, $t_{O,left}$ et un à droite, $t_{O,right}$) et il existe un cycle π dans $\mathcal{C}_P$ contenant $t_{O,left}$ et $t_{O,right}$ tel que tous les segments de π touchent la frontière ∂O de O.
L'inverse est aussi vrai : tous les segments de $\mathcal{C}_P$ touchant O forment un cycle dans $\mathcal{VC}$ qui contient exactement deux segments tangents à O.

La Figure 4.4 propose une manière commode de *dessiner* et *visualiser* $\mathcal{C}_P$ dans le plan (primal). Nous suggérons au lecteur de prendre quelques secondes pour examiner la figure ainsi que le schéma ci-dessus pour avoir une bonne intuition d'une telle courbe dans le complexe de visibilité.

La Figure 4.4(b) donne une bonne idée de ce qu'il se passe quand le point de vue bouge : dans le plan (primal), les segments tangents (ou murs radials de $\mathcal{D}_P$) tournent en glissant le long du bord de leur objet de tangence. Dans cette « vue topologique » de $\mathcal{C}_P$, les arcs « circulaires » noirs glissent les uns sur les autres, et on observe que la combinatoire du diagramme change quand deux sommets colorés se croisent. Cependant, il n'est pas vrai que deux sommets *adjacents* dans ce diagramme peuvent se croiser immédiatement.

Maintenir la décomposition radiale correspond à forcer tous les segments de $\mathcal{C}_P$ à bouger simultanément pour rester alignés avec P. Or, nous sommes seulement intéressés par la partie de $\mathcal{D}_P$ qui est égale au polygone de visibilité (c'est le grand cercle noir dans la Figure 4.4(b)).

Supposons que $\mathcal{C}_P$ soit connue, et que nous soyons capables de trouver le prochain *vrai* événement de visibilité, c'est-à-dire le prochain événement modifiant la combinatoire $\mathcal{V}_P$, et non une autre partie de $\mathcal{C}_P$. Cet événement est localisé sur un sommet v du complexe de visibilité que $\mathcal{C}_P$ **doit** croiser. Un tel croisement correspond, dans le plan, au croisement de deux segments tangents sur une bitangente.

Alors, au lieu de déformer toute la courbe $\mathcal{C}_P$ jusqu'à ce qu'elle croise v, nous aimerions ne « déformer » que la partie concernée de $\mathcal{C}_P$.[3] Cela sera possible si nous

[3] Cette partie devra contenir tous les segments de $\mathcal{V}_P$, puisque $\mathcal{V}_P$ est la structure que nous voulons maintenir complètement.

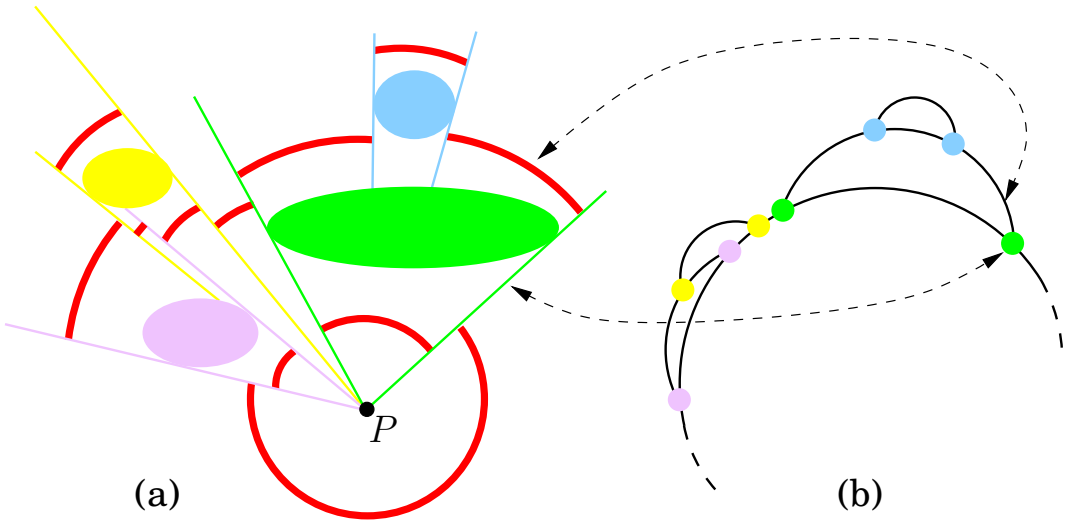


FIG. 4.4. Visualiser $\mathcal{C}_P$ comme une courbe unidimensionnelle. **(a)** : $\mathcal{C}_P$ est constituée des segments libres maximaux passant par un arc rouge et orthogonaux à cet arc. Les segments tangents (fins et colorés) font également partie de $\mathcal{C}_P$. On peut observer que la valence de ces segments tangents est bien 3. **(b)** : ce diagramme montre la topologie de $\mathcal{C}_P$. Les points colorés correspondent aux segments tangents de $\mathcal{C}_P$. Vus comme un graphe planaire, les bords d'une face de ce diagramme correspondent aux segments libres touchant un objet en particulier.

supprimons la contrainte **P0** des contraintes satisfaites par $\mathcal{C}_P$. C'est ce que fait le *visible zone algorithm*.

Définition 4.4. *Une* **courbe de segments**, *ou une* **courbe dans le complexe de visibilité** *est une courbe contenant $\mathcal{V}_P$ et satisfaisant les propriétés* **P1**, **P2**, *et* **P3** *mais pas — en général — la contrainte* **P0**. *Par exemple, $\mathcal{C}_P$ est une courbe de segments, qui satisfait aussi la contrainte* **P0**.

Avant de décrire le VZA plus avant, nous devons analyser un peu plus en détail les courbes de segments et certaines parties du complexe de visibilité dans lesquelles nous considérerons ces courbes de segments.

4.4 Zone et tour

Dans le *visible zone algorithm* que nous allons décrire bientôt, nous déformons une courbe de segments dans le complexe de visibilité $\mathcal{VC}$. En pratique bien sûr, nous maintiendrons sa combinatoire, c'est-à-dire ses intersections avec les arêtes de $\mathcal{VC}$. Si les contraintes **P1**, **P2** et **P3** sont satisfaites, les intersections de la courbe de segments avec les arêtes du complexe de visibilité sont au nombre de deux par objet de la scène ; nous devrons donc maintenir exactement deux tangentes à gauche et à droite de chaque objet de $\mathcal{O}$. Et nous ne parlerons de la courbe de segments $\mathcal{C}$ qui engendre ces $2k$ segments tangents, que pour raisonner.

Tandis qu'une courbe de segments $\mathcal{C}$ bouge dans $\mathcal{VC}$, la seule information préhensible à laquelle nous avons accès est sa description combinatoire. Et nous avons besoin d'un algorithme pour détecter les futurs changements combinatoires de $\mathcal{C}$, c'est-à-dire détecter à quels moments $\mathcal{C}$ croisera un sommet du complexe.

Si l'on maintient $\mathcal{D}_P$, le problème est facile parce que $\mathcal{D}_P$ est aussi une décomposition du plan, dont les faces sont décrites simplement ; et tout changement dans

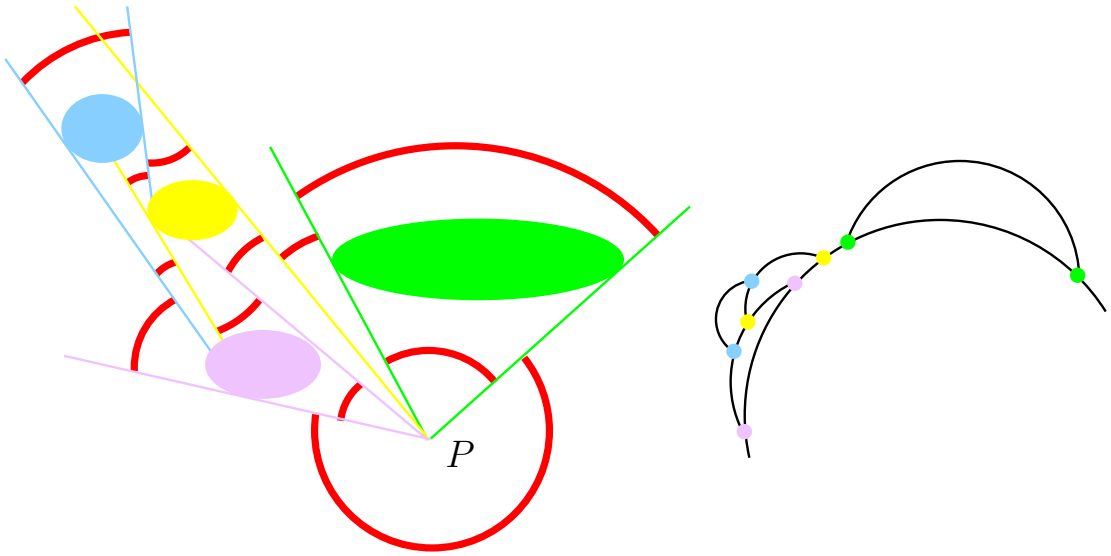


FIG. 4.5. Avec les mêmes conventions que pour la Figure 4.4, nous illustrons ici une *courbe de segments* qui ne peut pas être mise en correspondance avec une décomposition de l'espace libre. Elle satisfait cependant les propriétés **P1**, **P2** et **P3**.

cette décomposition correspond à la disparition (ou l'« écrasement ») d'une face. Mais lorsque nous supprimons la propriété **P0**, $\mathcal{C}$ ne peut plus être vue comme une décomposition du plan. La Figure 4.5 est une illustration d'une courbe de segments $\mathcal{C}$ dessinée dans le plan primal en utilisant la même convention de dessin que sur la Figure 4.4. On y aperçoit, entre autres, deux segments tangents qui se croisent (bleu et jaune à gauche de la figure). Ce sont bien deux segments tangents de $\mathcal{C}$: dans le complexe de visibilité $\mathcal{C}$ est une courbe et ces deux segments tangents sont deux points, intersections de $\mathcal{C}$ avec deux arêtes (ou 1-cellules) du complexe. Nous référons le lecteur à [68] pour quelques illustrations similaires d'une courbe de segments.

Le *visible zone algorithm* commence par créer une courbe de segments $\mathcal{C}$ exactement égale à la décomposition radiale. C'est un bon point de départ. Cependant, les structures de données utiles au stockage des faces de la décomposition radiale seront inutiles ici et ne sont donc pas calculées. Aussi, puisque nous souhaitons maintenir le polygone de visibilité $\mathcal{V}_P$ de P lorsque P bouge, le VZA maintient tous les segments tangents de $\mathcal{C}$ dont le bloqueur arrière est P, alignés avec P (nous les appellerons les segments tangents visibles). Comme ces segments forment la description combinatoire de $\mathcal{V}_P$, nous avons ainsi l'assurance que le polygone de visibilité est accessible à tout moment. Ceci implique que les événements de visibilité impliquant deux segments tangents visibles peuvent être détectés exactement de la même manière que lors de la maintenance de la décomposition radiale. Sur la Figure 4.5, les segments tangents verts et mauves ainsi qu'un des segments jaunes forment l'ensemble des segments tangents visibles : la description combinatoire de $\mathcal{V}_P$. Nous dirons qu'un segment tangent visible s *est un élément* de $\mathcal{V}_P$; ce que nous noterons $s \in \mathcal{V}_P \subset \mathcal{C}$ par abus de notation.

Nous allons voir qu'il existe une méthode efficace de recherche du prochain événement de visibilité impliquant un segment tangent visible et un segment tangent non-visible (c'est-à-dire dont le bloqueur arrière n'est pas P). En particulier, si le prochain événement de visibilité implique $s \in \mathcal{V}_P$ et un segment non-visible s', alors, les bloqueurs avant de s et s' sont identiques : s et s' partagent le même bloqueur avant. Soit $F \in \mathcal{O}$ le bloqueur avant de s ; il nous suffira donc de chercher le futur événe-

ment de visibilité impliquant s en parcourant l'ensemble des segments tangents de $\mathcal{C}$ dont le bloqueur avant est F. Cet ensemble sera avantageusement stocké dans une structure de données attachée à F.

Une fois le segment s' trouvé il ne nous reste plus qu'à effectuer la mise-à-jour de $\mathcal{C}$. Nous verrons que cette opération est moins simple que dans la maintenance de la décomposition radiale. En particulier, elle peut impliquer plusieurs « croisements de bitangentes ». La mise-à-jour sera un processus récursif qui effectuera plusieurs phases de recherche (la « descente ») avant de « remonter » en effectuant une série de croisements de deux segments tangents sur une bitangente, jusqu'au croisement correspondant à l'événement de visibilité détecté ($s \leftrightarrow s'$).

Définition 4.5. *Nous définissons la* **zone** *d'une courbe de segments $\mathcal{C}$ comme l'ensemble des 2-cellules du complexe de visibilité traversées par $\mathcal{C}$. $\mathcal{C}_P$ (la courbe de segments correspondant à la décomposition radiale $\mathcal{D}_P$) est une* **zone visible**. *Plus généralement, la zone d'une courbe de segments ayant été — avant déformation — la courbe $\mathcal{C}_P$, est aussi appelée une* **zone visible**. *Une* **2-cellule d'une zone** *est une 2-cellule du complexe qui appartient à cette zone.*

La notion de zone visible sera utilisée pour démontrer que l'on peut trouver efficacement la prochaine bitangente qui sera croisée par un segment tangent d'une courbe de segments lorsque celui-ci glisse le long du bord de son objet de tangence :

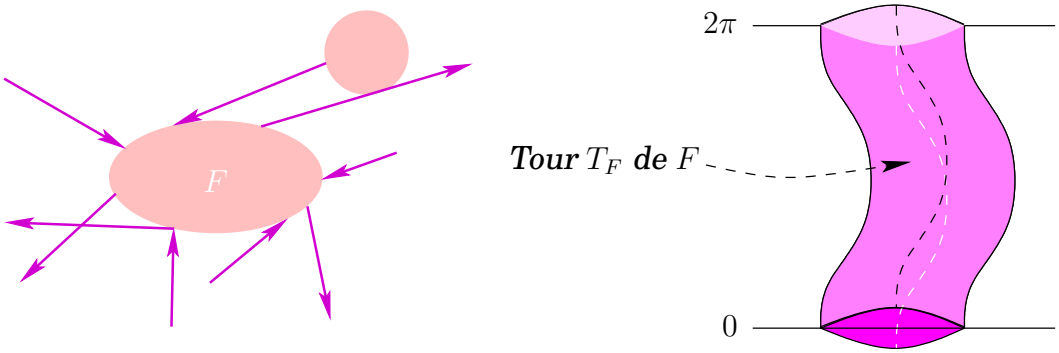
Soit $\mathcal{C}$ une courbe de segments dont la zone est une zone visible, et soit s un segment tangent de $\mathcal{C}$ (s fait donc partie de la description combinatoire de $\mathcal{C}$). s est orienté donc s a un bloqueur avant et un bloqueur arrière. Ces deux bloqueurs sont également bloqueurs pour d'autres segments tangents de $\mathcal{C}$, dont l'ensemble sera noté S dans ce paragraphe. Le **lemme de propagation de la zone** de Hall-Holt permet de montrer que le second segment tangent s' définissant avec s la bitangente que $\mathcal{C}$ va croiser *est un élément* de S. Essentiellement, le lemme de propagation de la zone restreint l'ensemble des tangentes parmi lesquelles nous devons chercher le prochain croisement : $s' \in S$.

Nous allons maintenant procéder à quelques études sur les relations entre les faces du complexe de visibilité et les zones. Elles nous mèneront au lemme de propagation de la zone. Nous pensons que le développement ci-dessous est particulièrement plus simple que celui présenté dans l'appendice A de la thèse de Hall-Holt [67].

Quand deux segments tangents sont sur le point de se croiser, ils partagent un bloqueur en commun. Il paraît dès lors intéressant d'étudier l'ensemble des segments libres maximaux partageant un même bloqueur F. En particulier, nous nous concentrons sur l'ensemble des segments (orientés) dont le bloqueur **avant** est le même (et noté F). Cet ensemble de segments a l'avantage d'être purement bidimensionnel.

Les figures de la page suivante sont séparées en deux parties. La partie gauche « vit » dans l'espace primal (le plan) tandis que la partie droite vit dans l'espace dual (espace des droites). Les parties gauche et droite de ces figures représentent les mêmes éléments géométriques. L'espace dual est — dans cette section 4.4 — paramétrisé en (ρ, θ).

Définition 4.6. *La* **tour** T_F *d'un objet F est l'ensemble des segments libres maximaux orientés dont un des bloqueurs est F. T_F a la topologie d'un tore.*



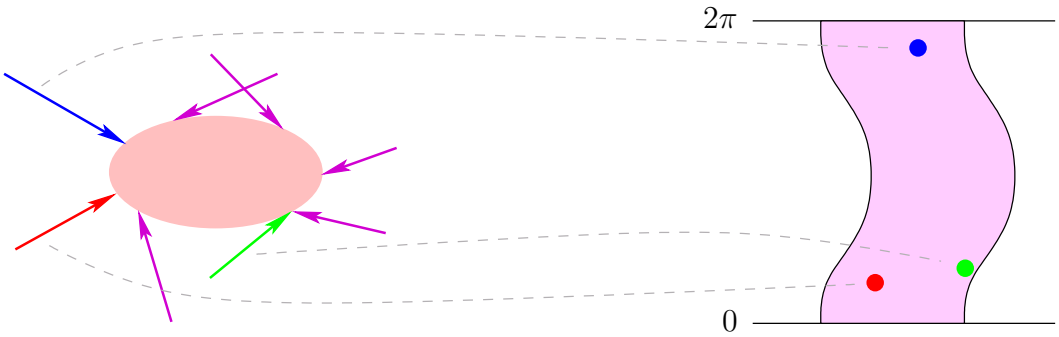
Les tangentes (orientées) à F ne sont **pas** dans T_F, mais en forment la frontière.

Définition 4.7. *La frontière de T_F est l'ensemble des segments libres maximaux tangents à F.*

◦ La **frontière gauche** de T_F est l'ensemble connexe des segments libres maximaux localement à gauche de F (en bleu ou gris clair sur la figure ci-dessus).

◦ La **frontière droite** de T_F est l'ensemble connexe des segments libres maximaux localement à droite de F (en violet ou gris foncé sur la figure ci-dessus).

Définition 4.8. *La* **tour avant** *de F est l'ensemble des segments libres maximaux orientés dont le bloqueur avant est F. Elle a la topologie d'un cylindre $\mathbb{S}^1 \times [0,1]$.*



Dorénavant, nous ne considérerons plus que les tours « avant » et omettrons donc ce qualificatif.

4.5 Paramétrisation d'une tour

Puisque nous ne considérerons que ce qu'il se passe dans la tour d'un objet, nous utiliserons une paramétrisation adaptée pour l'ensemble des segments de cette tour.

Soit F un objet et m un segment de la tour T_F de F. Nous représentons m par le point $m^* = (r(m), \theta(m))$ où $\theta(m) \in [0, 2\pi]$ est l'orientation du segment m et $r(m) \in [0, 1]$ est la distance normalisée entre m et la tangente à gauche à F parallèle à m :

$$r(m) = \frac{d(m_l, m)}{d(m_l, m_r)}$$

. Par exemple, si m est une tangente à droite à F alors $r(m) = 1$. Voir la Figure 4.6 (gauche et milieu).

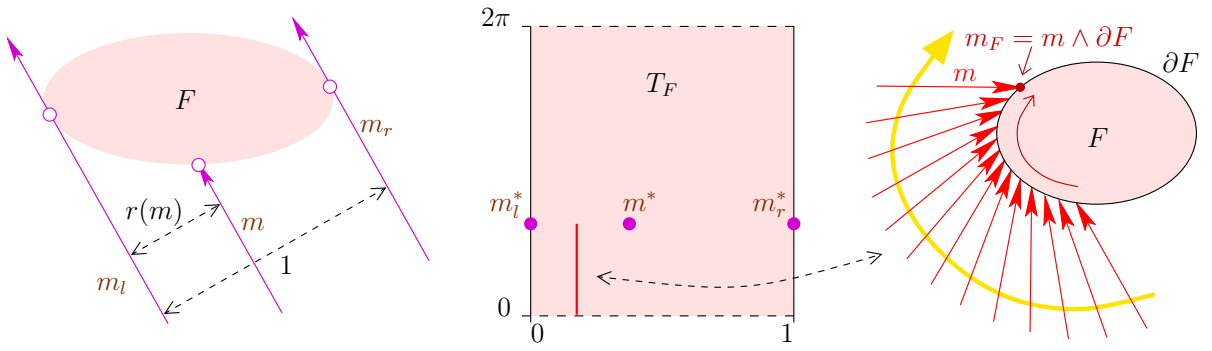


FIG. 4.6. Paramétrisation de la tour de F. L'espace des paramètres est un rectangle, ou plutôt, un cylindre de largeur 1, puisque la dimension angulaire est cyclique. $\theta(m_l) = \theta(m) = \theta(m_r)$, $r(m_l) = 0$, $r(m_r) = 1$.

Examinons quelques propriétés de cette paramétrisation.

Propriété 1 : Quand le point m^* de l'espace des segments se translate horizontalement, le segment libre maximal m se translate parallèlement à lui-même.

Propriété 2 : Quand m^* translate verticalement, m tourne autour de F en gardant la même distance normalisée à la tangente à gauche à F parallèle à m. Soit δF la frontière de F (dans le plan primal). Soit m_F le point d'intersection entre m et δF. Quand m^* bouge verticalement vers le haut ($\theta(m)$ est croissant) alors m_F tourne le long de δF dans le sens inverse des aiguilles d'une montre autour de F. Si m^* descend verticalement, le sens de rotation est inversé. Voir Figure 4.6 (à droite).

4.6 Structure d'une tour

Un sommet du complexe de visibilité est adjacent à 6 faces, pour deux desquelles il est un sommet extrémal dans la dimension angulaire (maximal pour l'une et minimal pour l'autre). Cependant, si l'on se restreint aux faces du complexe de visibilité dans la tour de F, alors un sommet du complexe est adjacent à seulement 3 faces (dans la tour) pour l'une desquelles il est extrémal (maximal ou minimal). Voir Figure 4.7. Les sommets de la frontière gauche de T_F ($r = 0$) sont adjacents à deux faces de T_F et sont tous maximaux. Les sommets de la frontière droite de T_F ($r = 1$) sont adjacents à deux faces de T_F et sont tous minimaux. Ainsi, chaque sommet appartenant à T_F ou à sa frontière, est sans ambiguïté maximal ou minimal, quand on le considère dans le cadre de la tour T_F.

Considérons la tour T_F de l'objet F. Dans l'espace des segments, nous avons vu que T_F est paramétrisée par un cylindre $[0, 1] \times \mathbb{S}$, ou un rectangle $[0, 1] \times [0, 2\pi]$ par

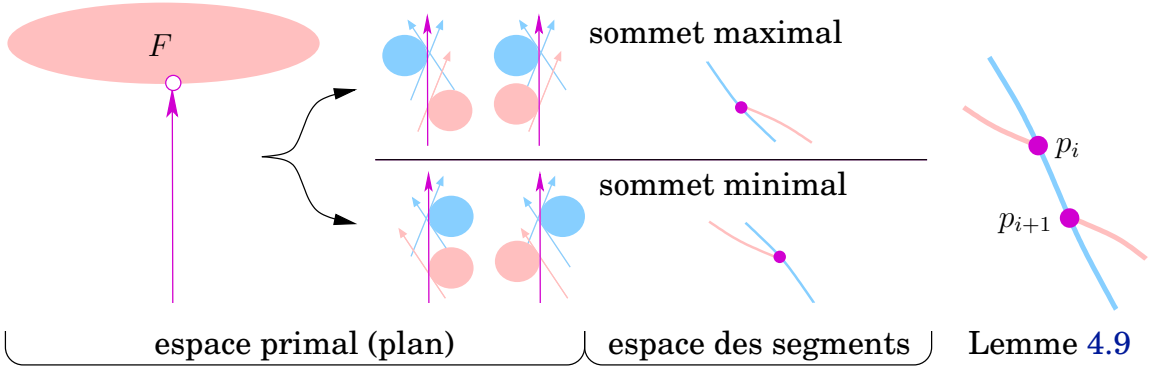


FIG. 4.7. Configuration d'un sommet dans une tour. Dans l'**espace primal**, on voit un sommet (bitangente) à l'intérieur de la tour T_F. On distingue, au bout des flèches, les quatres configurations possibles pour les deux objets tangents au sommet. Dans l'**espace des segments**, on montre la configuration d'un sommet dans la *paramétrisation* rectangulaire de la tour : les arêtes colorées sont les segments libres tangents à l'objet de même couleur. Le schéma de droite illustre une situation impossible qui apparaît dans la preuve du lemme 4.9.

commodité. Les faces de la tour sont les faces du complexe de visibilité contenues dans cette tour. Elles forment une subdivision en graphe planaire de cette tour.

Soit $\mathcal{C}$ une courbe de segments dans $\mathcal{VC}(\mathcal{O})$ (définition 4.4). Si on la restreint à la tour d'un objet $F \in \mathcal{O}$, $\mathcal{C}$ est purement unidimensionnelle, et croise le rectangle T_F (la tour paramétrisée) de sa frontière droite à sa frontière gauche. Cette propriété est déduite des propriétés **P2** et **P3** (page 45).

Les arêtes du complexe à l'intérieur de la tour T_F seront appelées les arêtes de la tour T_F. Ces dernières ont la propriété très importante d'être strictement descendantes en tant que fonctions de r : si une arête e dans T_F est paramétrée par $e : (r_1, r_2) \mapsto T_F$, alors $r < r' \Rightarrow \theta(e(r)) > \theta(e(r'))$ ($\theta \circ e : (r_1, r_2) \mapsto \mathbb{R}/2\pi\mathbb{R}$ est une « fonction » décroissante). En conséquence, les deux chaînes de chaque face f de la tour T_F peuvent non seulement être qualifiées de *gauche* et *droite* mais aussi de *haute* et *basse*. Autrement dit, les faces de T_F sont r-monotone et θ-monotone. Les deux chaînes d'une face f sont séparées par le sommet maximal f^{up} de f et son sommet minimal f_{down}. f^{up} est aussi le point le plus à gauche de f. f_{down} est aussi le point le plus à droite de f. Le graphe planaire formé dans T_F par les arêtes du complexe a une certaine structure :

Les arêtes intérieures à la tour T_F forment un graphe planaire dans la paramétrisation de T_F. Chaque arête est « descendante » lorsqu'on la suit de gauche à droite (en faisant croître le paramètre r : θ est une fonction décroissante de r). En suivant plusieurs arêtes consécutives, on peut former un chemin descendant quand on le parcours de gauche à droite. Soit $\phi = p_1, p_2, \ldots, p_n$ la séquence des sommets apparaissant le long d'un chemin monotone φ descendant en suivant certaines arêtes de T_F (voir, par exemple, la Figure 4.11). Les sommets sont ordonnés de gauche à droite (ou de haut en bas).

Lemme 4.9. *Il existe $k \in [0..n]$ tel que $\forall i \leq k, p_i$ est un sommet maximal, et $\forall i > k, p_i$ est un sommet minimal. Autrement dit, ϕ est la concaténation d'une séquence de sommets maximaux et d'une séquence de sommets minimaux.*

Preuve : Supposons que ce ne soit pas le cas. Alors, pour quelques $i < n$, p_i est minimal et p_{i+1} est maximal. Puisque p_i n'est pas maximal, il n'est pas sur la frontière

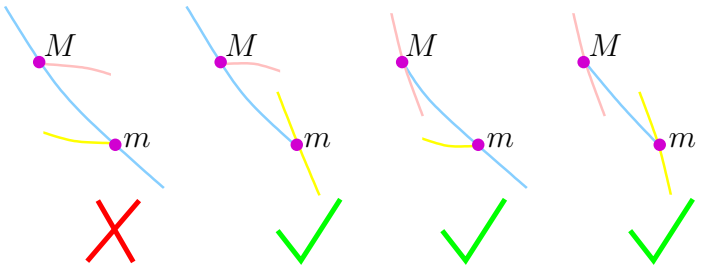


FIG. 4.8. La configuration de gauche est impossible. Les arêtes d'une même couleur représentent des segments libres tangents à un même objet.

gauche de T_F. De même, p_{i+1} n'est pas sur la frontière droite de T_F. Ainsi, ces deux sommets sont à l'intérieur de la tour T_F (c'est-à-dire qu'ils ne sont pas tangents à F).

Examinons le chemin φ autour des sommets p_i et p_{i+1} (voir la Figure 4.7 (à droite)). Le chemin bleu, qui est une partie du chemin φ est formé de segments tangents à un même objet O. Or, si nous examinons la configuration des sommets p_i et p_{i+1}, nous observons que ces segments tangents bleus sont tangents à droite au voisinage de p_i et tangents à gauche au voisinage de p_{i+1}. C'est une contradiction. □

Définition 4.10. *La* **branche-basse** *d'une face f d'une tour T_F est le chemin qui part du sommet minimal f_{down} de f et suit les arêtes du complexe vers le bas/droite jusqu'à la frontière droite de T_F. La* **branche-haute** *de f part du sommet maximal f^{up} de f, et suit les arêtes du complexe, dans T_F, vers le haut/gauche, jusqu'à la frontière gauche de T_F.*

La branche-basse de f est bien définie car, à partir de son sommet minimal f_{down}, le lemme 4.9 nous garantit l'unicité du chemin en allant vers le bas. Il en va de même pour la branche-haute.

Lemme 4.11. *Soit m un sommet minimal à l'intérieur de T_F juste en dessous d'un sommet maximal M à l'intérieur de T_F (M et m sont connectés par exactement une arête du complexe). Parmi les 4 configurations combinatoires pouvant exister (voir Figure 4.8), l'une est impossible.*

Preuve : La démonstration est analogue à celle du lemme 4.9 et utilise le fait que les deux objets de tangence à m et M les plus proches de F ne peuvent être identiques. □

Nous allons maintenant utiliser les Lemmes 4.9 et 4.11 pour donner une nouvelle preuve, plus simple, du lemme principal du *visible zone algorithm*, que Hall-Holt appelle le *branch face lemma*, dont dépend le lemme de propagation de la zone.

Lemme 4.12 (branch face lemma). *Soit f^{up} le sommet maximal d'une face f de T_F telle que f^{up} soit à l'intérieur de T_F. Soit g la face de T_F juste en dessous de f^{up}. Alors, la branche-basse de f (définition 4.10) contient le sommet minimal g_{down} de g.*

Preuve : La situation est illustrée sur la Figure 4.9(a). Voir aussi la Figure 4.11.

Si le sommet minimal de g — appelé g_{down} — est sur la frontière droite de T_F, le lemme est prouvé (voir l'exemple 1 de la Figure 4.11). Supposons maintenant que g_{down} est à l'intérieur de T_F, comme f^{up}.

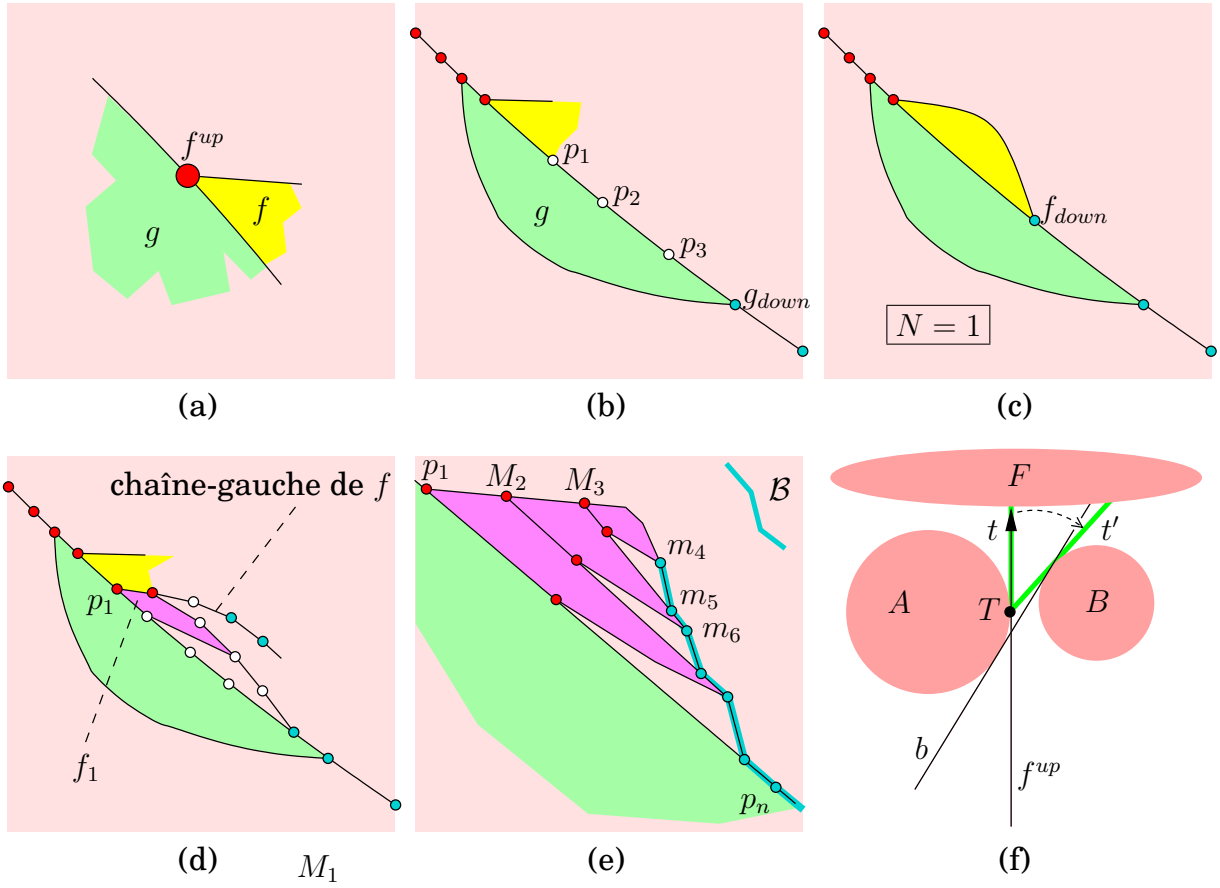


FIG. 4.9. Illustration de la preuve du lemme 4.12. Les sommets rouges sont maximaux, les bleus sont minimaux, les blancs sont indécidés. Tous les sommets ne sont pas représentés.

• En suivant la branche-haute de f jusqu'à la frontière gauche de T_F, nous croisons quelques sommets maximaux. Le sommet maximal de g est soit sur cette branche-haute de f [4], soit sur la frontière gauche. Ensuite, nous suivons la branche-basse de g et croisons quelques sommets minimaux, jusqu'à la frontière droite. Notre situation est illustrée sur la Figure 4.9(b).

Nous allons prouver le lemme par induction sur le nombre N de sommets dans la tour T_F, dont la coordonnée r est comprise entre $r(f^{up})$ et $r(g_{down})$:

• $N = 0$. Ce cas est impossible grâce au lemme 4.11, puisque f^{up} et g_{down} sont à l'intérieur de T_F.

• $N = 1$. Alors, ce seul sommet s doit être sur le chemin d'arêtes du complexe entre f^{up} et g_{down}, pour la même raison qu'au cas $N = 0$. Puisque s est le seul sommet dans cette région, il doit être le sommet minimal de f : f_{down} (Figure 4.9(c)). Le lemme est vérifié.

• $N > 1$. Soit $f^{up}, p_1, p_2, \ldots, p_n, g_{down}$ la séquence de sommets le long de la chaîne-droite de g, commençant en f^{up} et finissant en g_{down}.

– Si p_1 est minimal, alors tous les autres p_i le sont, et p_1 doit être le sommet minimal de f. Le lemme est vérifié comme pour $N = 1$. Ce cas est illustré sur l'exemple 2 de la Figure 4.11.

[4] C'est le cas que nous avons choisi d'illustrer sur la figure, mais ce choix n'a pas d'importance pour la preuve.

– p_1 est maximal. Remarquons que p_1 est le premier sommet en dessous de f^{up} le long de la chaîne-gauche de f. Soit f_1 la face dont p_1 est le sommet maximal. Comme p_1 est juste au-dessus de g, nous savons — par induction — que la branche-basse de f_1 contient g_{down}. La situation est décrite sur la Figure 4.9(d). Considérons maintenant la chaîne-gauche de la face f, dont nous baptisons les sommets

$$f^{up}, p_1, M_2, M_3, \ldots, M_k, m_{k+1}, m_{k+2}, \ldots, f_{down}.$$

Les sommets M_i sont maximaux tandis que les sommets m_j sont minimaux.
Soit f_2 la face dont le sommet maximal est M_2. M_2 est au-dessus de f_1 donc — par induction — la branche-basse de f_2 contient le sommet minimal de f_1. Nous en déduisons que la branche-basse de f_2 doit aussi contenir g_{down} !
Nous appliquons alors le même raisonnement successivement pour $M_3, \ldots, M_k$ et pouvons conclure que la branche-basse de la face f_k (dont le sommet maximal est M_k) contient également g_{down}. Appelons cette branche $\mathcal{B}$. Voir la Figure 4.9(e). Clairement, m_{k+1} est le sommet minimal de la face f_k. Ainsi, en appliquant le lemme 4.9, nous en déduisons que $m_{k+1}, m_{k+2}, \ldots, f_{down}, \ldots$ est un préfixe de $\mathcal{B}$. Cela signifie que $\mathcal{B}$ contient la branche-basse de f.
Nous avons également vu que $\mathcal{B}$ contient g_{down}. Il nous reste à prouver que g_{down} apparaît après f_{down} le long de $\mathcal{B}$.
Regardons la Figure 4.9(f). T est le point de tangence de f^{up} le plus proche de F. Soit t le segment de droite de f^{up} entre T et F. Tournons t dans le sens des aiguilles d'une montre (t descend dans la tour de F) jusqu'à ce qu'il soit bloqué en t' par B. B peut ne pas exister, auquel cas t est stoppé quand il devient tangent-à-droite à F. Maintenant, clairement, la bitangente b est une borne inférieure sur les pentes atteignables dans f. Mais t' a une pente plus basse que b et $t' \in g$. Donc g doit descendre plus bas que f dans T_F, autrement dit, f_{down} apparaît avant g_{down} le long de $\mathcal{B}$.

$\mathcal{B}$ contient f_{down} et g_{down}. De plus, f_{down} est situé plus haut que g_{down}. On en déduit que g_{down} est bien sur la branche basse de f. □

4.7 Le lemme de la propagation de la zone

Revenons au *visible zone algorithm*. Supposons que nous ayons détecté que deux tangentes, t et s, doivent se croiser au temps t_0, alors au temps t_0^-, les deux tangentes partagent un bloqueur en commun (le bloqueur avant ou arrière). Ainsi, l'intuition nous dit que la recherche de la prochaine tangente s que va croiser t pour une certaine direction de rotation, est à effectuer dans l'ensemble des tangentes bloquées par le bloqueur arrière ou avant de t. Preuve en est faite par le lemme suivant.

Lemme 4.13. Lemme de propagation de la zone *Soit f une face d'une zone visible. Soit T_F la tour du bloqueur avant F de f et v le sommet maximal de f. Alors, dans la tour T, la face g en dessous de v est aussi une face de la zone. Voir Figure 4.10 (à gauche).*

La Figure 4.11 (page 56) illustre le lemme. Sur cette figure, il est supposé que f est une face d'une zone (f est traversée par la courbe $\mathcal{C}_{sub}$, sur la partie droite de la figure). Le lemme prouve que g est aussi une face de la zone, c'est-à-dire que la courbe bleue traverse aussi g.

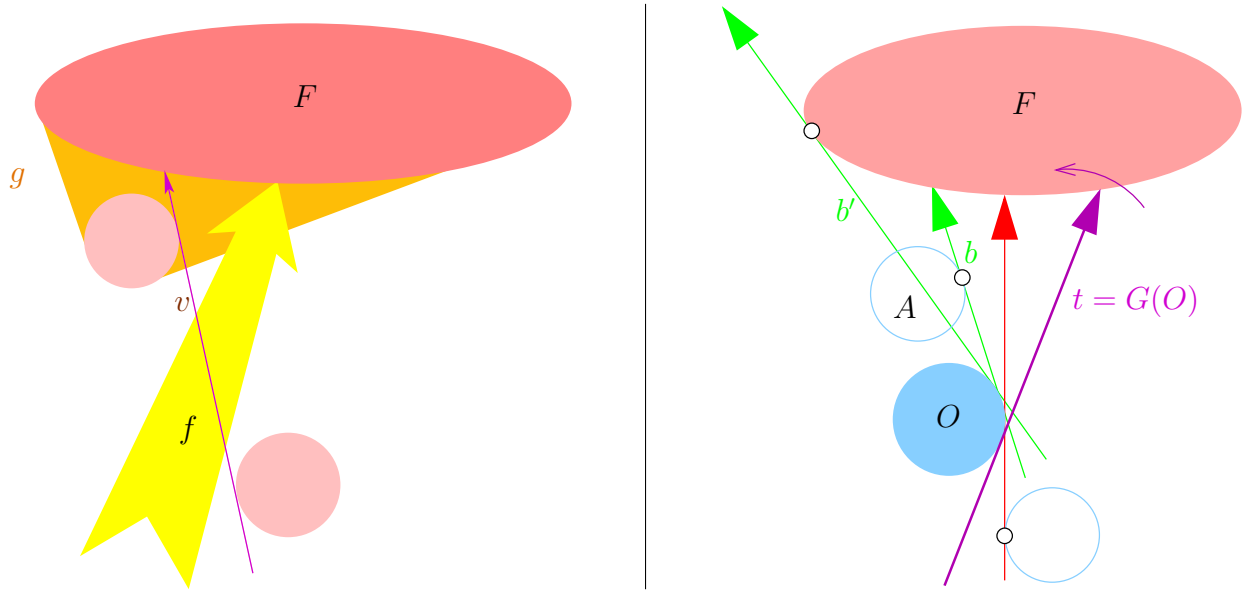


FIG. 4.10. À gauche, illustration du lemme de propagation de la zone. **À droite**, recherche du prochain croisement de bitangente pour t_0. Les bitangentes vertes sont celles que peut croiser t lorsqu'elle tourne dans le sens direct.

Preuve : D'après l'hypothèse que f est une face d'une zone, d'une part, et d'après les propriétés **P2** et **P3**, d'autre part, nous déduisons que la courbe de segments correspondant à cette zone possède une sous-partie qui traverse la tour T_F de sa frontière droite à sa frontière gauche. En particulier, cette courbe joint la frontière gauche de T_F à la chaîne gauche de f. Voir la courbe $\mathcal{C}_{sub}$ sur les deux exemples de la Figure 4.11.

Par définition, toutes les faces traversées par $\mathcal{C}_{sub}$ sont des faces de la zone. Ainsi, nous devons prouver que $\mathcal{C}_{sub}$ traverse g. Les exemples de la Figure 4.11 montrent clairement que $\mathcal{C}_{sub}$ ne peut aller de la chaîne gauche de f vers la frontière gauche de T_F sans passer par g. C'est ce que nous montrons ci-dessous.

En premier lieu, g est définie comme la face en dessous du sommet maximal v de f. f et g se touchent en v. Cela nous montre que l'étendue angulaire de g « va » plus haut que celle de f, et que g forme un barrage à $\mathcal{C}_{sub}$ dans cette région.

Nous devons vérifier que g barre complètement la route de $\mathcal{C}_{sub}$ vers la frontière gauche de T_F, de sorte que $\mathcal{C}_{sub}$ doit traverser g. Deux cas se présentent :
• g atteint la frontière droite de T_F. Ce cas est illustré dans l'exemple 1 du haut de la figure. Alors, g forme un barrage complet entre la chaîne gauche de f et la frontière gauche de T_F. Donc, $\mathcal{C}_{sub}$ doit passer au travers de g.
• g ne touche pas la frontière droite de T_F. Le lemme 4.12 nous dit que g atteint la branche-basse de f. À nouveau, g forme un barrage complet et $\mathcal{C}_{sub}$ doit traverser g pour atteindre la frontière gauche de T_F. □

Nous revenons maintenant sur le *visible zone algorithm*. Pour l'initialisation, nous calculons la décomposition radiale initiale $\mathcal{D}_{P(0)}$, en ne gardant en mémoire que les $2k$ tangentes qui en forment ses murs radials. Nous devons ensuite remplir la queue de priorité $\mathcal{E}$ qui stocke les futurs événements de visibilité $\mathcal{V}_{P(0)}$. $\mathcal{E}$ contient au plus un événement par tangente dans le polygone de visibilité, ce qui contraste avec l'algorithme de maintenance de $\mathcal{D}_P$, qui devait stocker $O(3k)$ événements.

Pour chaque objet O de la scène, nous maintenons deux listes : la liste $Ar(O)$ des tangentes pour lesquelles O est bloqueur arrière et la liste $Av(O)$ des tangentes pour

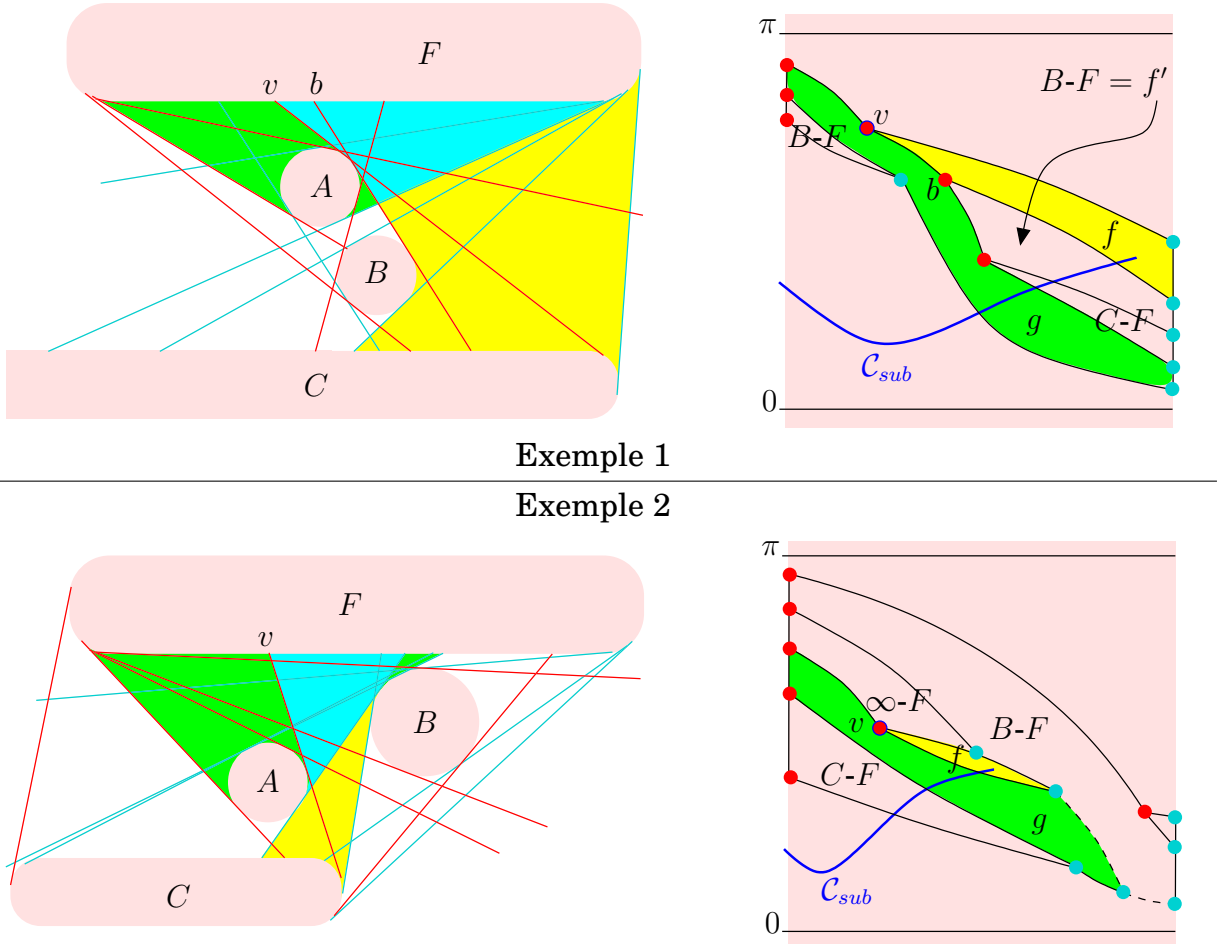


FIG. 4.11. Deux configurations illustrent le lemme de propagation de la zone. La partie **gauche** montre la scène. La partie **droite** montre la paramétrisation de la tour T_F de F. f est en jaune, et est une face d'une zone. Le lemme de propagation de la zone prouve que la face verte, g, est aussi une face de la zone, en montrant que $\mathcal{C}_{sub}$ doit aussi traverser g. Sur l'exemple 2 (en bas), le chemin en pointillés est la branche-basse de f.

lesquelles O est bloqueur avant. O « connaît » aussi ses deux tangentes à gauche $G(O)$ et à droite $D(O)$. La maintenance de ces listes est triviale et ne nécessite un coût de stockage que linéaire.

Pour remplir $\mathcal{E}$, nous examinons chaque tangente t de $\mathcal{V}_{P(0)}$. Soit F le bloqueur avant de t (le bloqueur arrière de t est le point de vue P). Soit O l'objet de tangence de t (Figure 4.10 (à droite)). Quitte à modifier en pensée l'orientation de t et/ou à regarder le plan $\mathbb{R}^2$ « par l'autre côté »,[5] nous pouvons supposer d'une part, que t pivote dans le sens direct, et donc se déplace vers le haut dans la tour t_F de F,[6] et d'autre part, que la prochaine bitangente que t va rencontrer sera soit une bitangente entre O et F, soit une bitangente entre O et A où A est un objet situé entre O et F, voir Figure 4.10 (à droite).

Soit $T(t) = Av(F) \cup \{G(F), D(F)\}$ l'ensemble des tangentes dont le bloqueur avant est F, augmenté des deux tangentes à F. Soit $Next(t)$ l'ensemble des objets qui sont bloqueur arrière ou objet de tangence d'une tangente de $T(t)$. Chaque objet dans

[5] Comme si l'on passait de l'autre côté d'une vitre figurant le plan.

[6] On suppose momentanément que la trajectoire de P est suffisamment simple pour que P se déplace toujours dans le même sens.

$Next(t)$ induit une bitangente susceptible d'être croisée par t quand cette dernière pivote. De fait, le prochain événement de visibilité pour t aura bien lieu sur la bitangente induite par un élément de $Next(t)$ qui a la pente la plus proche de t (tout en restant supérieure). Cette propriété nous est garantie par le lemme de propagation de la zone, ce que nous expliquons maintenant.

Soit b la prochaine bitangente que t va croiser.
• Si b est la bitangente entre O et F (voir la bitangente b' sur la Figure 4.10 à droite), on a gagné puisque O se trouve bien dans $Next(t)$.
• Sinon, la bitangente b est un sommet maximal intérieur de la tour T_F, voir la Figure 4.10 (à droite) et la Figure 4.11. Les choses se compliquent... Soient f' la face de T_F dont b est le sommet maximal (voir 4.11, en haut). f' est une face de la zone de la courbe de segment $\mathcal{C}$. Ainsi, d'après le lemme de propagation de la zone, la face g située juste en dessous de b est également une face de la zone de $\mathcal{C}$. Cette face, est précisément la face de T_F comprenant des segments libres orientés reliant A et F. On en déduit (d'après la définition d'une zone) qu'il existe forcément une tangente dans $T(t)$ dont le bloqueur arrière est A ou bien qui est tangente à A. Dans les deux cas, on a montré que A appartient bien à $Next(t)$. Ainsi, nous avons la garantie de trouver le prochain événement de visibilité pour t en examinant l'ensemble $Next(t)$.

Soit b la prochaine bitangente que t va croiser. Pour que le croisement soit possible, il faut que la seconde tangente t' impliquée dans le croisement soit « dans la bonne position », c'est-à-dire qu'elle puisse être pivotée jusqu'à b sans croiser d'autre bitangente. Pour le vérifier, la procédure de recherche que nous venons de décrire est appliquée récursivement pour t'.

La seule différence est que, puisque t' n'est pas attachée au point de vue, nous devons définir l'ensemble $T(t')$ par $T(t') = Av(F) \cup Ar(H) \cup \{G(F), D(F), G(H), D(H)\}$ où, bien entendu, F et H sont respectivement les bloqueurs avant et arrière de t' (et non de t).

Nous référons le lecteur aux travaux de Hall-Holt [66–68] pour les détails sur la correction et la terminaison de cet algorithme.

4.8 Détails

Dans cette section, nous présentons un détail d'implémentation du VZA absent de l'exposition de Hall-Holt, ainsi qu'une remarque sur la structure du graphe formé par les arêtes d'une tour.

4.8.1 Recherche du prochain événement

Soit t_0 une tangente dont nous connaissons la direction de rotation : t_0 tourne autour de son objet de tangence vers le haut (dans le sens direct, inverse des aiguilles d'une montre) ou vers le bas (dans le sens indirect). Soient A et B les deux bloqueurs de t_0. Nous avons vu que la prochaine bitangente que t_0 croisera lors de son mouvement est définie par t_0 (bien sûr) et une autre tangente t_1 qui touche A ou B. La recherche s'effectue donc linéairement sur l'ensemble des tangentes possédant A ou B comme bloqueur. Nous précisons maintenant les détails de cette recherche. Il nous faut, en effet, comparer les pentes des bitangentes potentielles avec celle de t_0. Or, t_0 n'a pas de pente bien définie puisqu'elle n'est décrite que par son objet de tangence O,

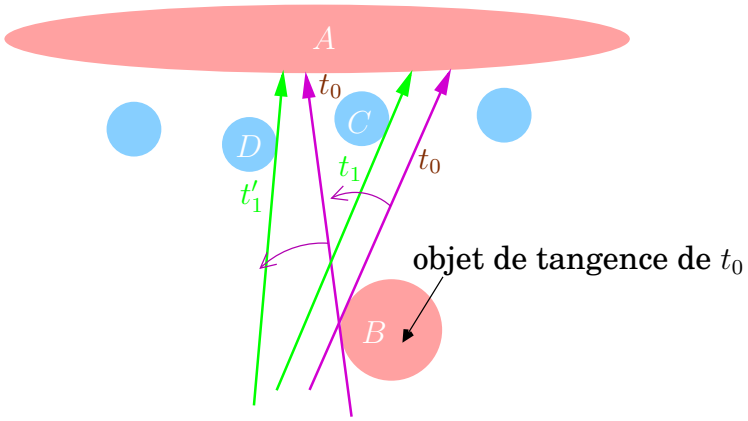


FIG. 4.12. Selon la position exacte de t_0, le prochain événement sera le croisement d'une bitangente définie par t_0 et t_1 (BC) ou t'_1 (BD)... Comment choisir le bon ?

ses bloqueurs et sa position par rapport à O (gauche ou droite). Nous savons sur quelle arête du complexe de visibilité se trouve chaque tangente, mais nous ne connaissons pas, généralement, les sommets de ces arêtes (qui sont des bitangentes).

Soit T l'ensemble des tangentes qui touchent A ou B. Supposons que t_0 tourne dans le sens direct. Nous voulons donc chercher dans T la tangente qui engendrera, avec t_0 la bitangente b d'angle minimal supérieur à la pente de t_0. La Figure 4.12 illustre la confusion possible. Puisque nous ne connaissons pas exactement la position de t_0 (voir les deux positions en violet sur la figure), il apparaît difficile de discriminer t_1 de t'_1 comme candidat au prochain croisement. Heureusement nous pouvons, sans surcoût, obtenir plus d'informations sur la localisation des tangentes, ce qui permettra une comparaison correcte.

Deux attributs de position pour les tangentes Rappelons nous qu'à l'initialisation de l'algorithme, toutes les tangentes construites sont alignées avec le point de vue en position initiale $P(0)$. Tant qu'une tangente n'est pas associée à un événement de croisement, nous pouvons donc supposer qu'elle reste « conceptuellement » alignée avec $P(0)$. Cette information nous donne la position exacte de la tangente, si l'on garde en mémoire la position initiale du point de vue. De même, si une tangente a pour bloqueur arrière le point de vue $P(t)$, alors nous connaissons aussi sa position exacte. Pour chacune de ces tangentes, nous définissons sa position de référence comme sa position d'alignement avec son *point de référence* ($P(0)$ ou $P(t)$) avec lequel elle peut s'aligner sans changer la combinatoire de $\mathcal{C}$.

Pour les autres tangentes, la situation est différente, puisque nous ne pouvons leur attribuer un point de référence avec lequel s'aligner sans changement combinatoire. En revanche, toutes ces tangentes ont participé à au moins un événement (croisement de deux tangentes à travers une bitangente). Comme nous pouvons garder en mémoire la dernière bitangente croisée par une tangente, nous savons que chacune de ces tangentes peut être infiniment rapprochée de cette bitangente sans changer la combinatoire de $\mathcal{C}$. La dernière bitangente croisée par une tangente définit une position de référence de cette tangente.

Puisque toutes les tangentes de notre structure de données possèdent une position de référence atteignable sans changement combinatoire de $\mathcal{C}$, nous pouvons alors effectuer la recherche dans T en comparant ces positions de référence.

4.8.2 Remarques sur la structure en double-arbre d'une tour

Les sommets à l'intérieur et sur le bord d'une tour ont une structure particulière décrite sur la Figure 4.7. Soit F un objet et T_F sa tour. Le bord gauche de T_F contient uniquement des sommets maximaux. Chacun de ces sommets peut être vu comme la racine d'un arbre (de degré au plus 2) de sommets maximaux dont tous les nœuds (sauf la racine) sont à l'intérieur de la tour. De même, le bord droit de T_F contient uniquement des sommets minimaux, et chacun de ces sommets est la racine d'un arbre dont tous les nœuds (sauf la racine) sont des sommets minimaux à l'intérieur de la tour.

Fixons arbitrairement un sommet M du bord gauche de la tour T_F. En descendant le long du bord gauche de T_F, nous relions par des arêtes les autres sommets du bord gauche de la tour, et lions ainsi tous les arbres de sommets maximaux en un seul arbre, contenant tous les sommets maximaux, et de racine M.

Fixons arbitrairement un sommet m du bord droit de la tour T_F. En montant le long du bord droit de T_F, nous relions par des arêtes les autres sommets du bord droit de la tour, et lions ainsi tous les arbres de sommets minimaux en un seul arbre, contenant tous les sommets minimaux, et de racine m. Voir Figure 4.13.

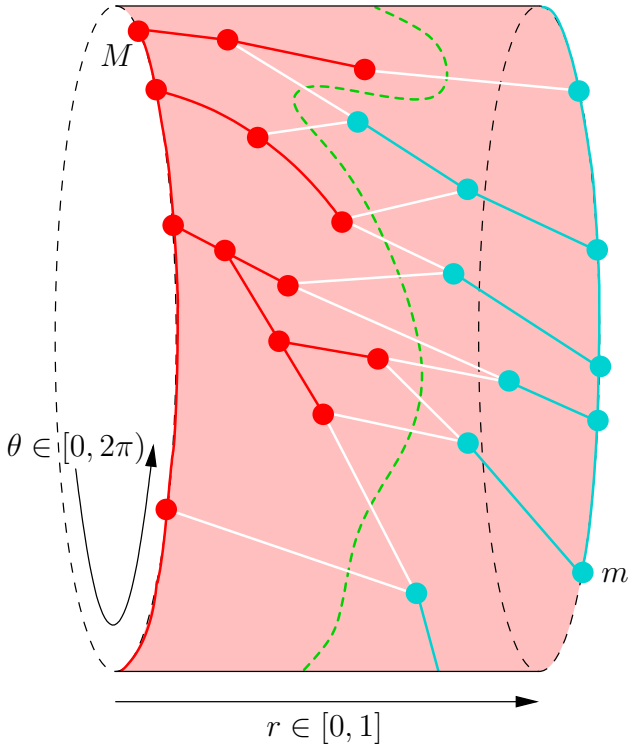


FIG. 4.13. Les sommets d'une tour forment deux arbres. (Ce dessin ne respecte pas la règle de la Figure 4.8.)

Nous pouvons en déduire l'existence d'une courbe fermée de segments libres maximaux dans T_F passant par toutes ses faces. Une telle courbe est illustrée en vert sur la figure.

Chapitre 5

Maintenance de la visibilité 3D d'un point mobile

5.1 Introduction

Soit un ensemble $\mathcal{O}$ de k polytopes disjoints dans $\mathbb{R}^3$, de complexité totale n. Étant donné un point de vue V dans l'espace libre, nous souhaitons maintenir, de façon continue et exacte le polyèdre de visibilité $\mathcal{V}_V$ de V lorsque le point de vue V suit, dans l'espace libre, une trajectoire connue au moins à court terme.

Tous les travaux précédents se sont penchés sur une version plus restrictive du problème, dans laquelle la trajectoire du point de vue est rectiligne. L'algorithme de Bern *et al.* [15] calcule tous les événements EEE transparents (dont le nombre est k_t) en temps $O((n^2+k_t)\log n)$, en balayant successivement les n ensembles de droites passant par la trajectoire rectiligne du point de vue et une arête d'un polygone de la scène. Mulmuley [106] propose de construire une décomposition de l'espace dépendante de la ligne ℓ formant la trajectoire du point de vue V, pour en déduire les instants auxquels le polyèdre de visibilité de V change combinatoirement lorsque V se déplace le long de ℓ.

Nous décrivons dans ce chapitre une extension de la notion de décomposition radiale au cas 3D. Cette décomposition radiale 3D est une partition de l'espace libre (non occupé par un polytope) induite par l'ensemble des polytopes et un point central. Elle contient – en particulier – le polyèdre de visibilité de son centre V. On la note $\mathcal{R}_V$.

En fait, la structure de notre décomposition radiale est une sous-structure de la décomposition proposée par Mulmuley. Ce dernier rend ensuite cette décomposition convexe en l'intersectant avec tous les plans passant par la trajectoire ℓ et un sommet d'un polygone de la scène. Au lieu de suivre cette approche, nous rendons notre subdivision convexe en identifiant chaque cellule à un polygone sphérique que nous triangulons. Nous nous affranchissons ainsi de la contrainte linéaire sur la trajectoire du point de vue.

Nous verrons que chaque cellule de cette décomposition peut être vue à la fois comme un volume de $\mathbb{R}^3$ et comme un ensemble de dimension 2 de segments libres maximaux dont la droite support est alignée avec le point central (qui sera notre point de vue). Ainsi, chaque cellule C pourra être projetée sur la « sphère des directions orientées » $\mathbb{S}^2$ pour former un polygone sphérique en bijection avec les segments libres maximaux de C. La Figure 5.1 montre la forme polygonale des faces de $\mathcal{R}_V$ formées

des segments libres maximaux touchant à la fois le point de vue V et l'infini ∞. Ces images ont été produites par notre implémentation de l'algorithme de construction de $\mathcal{R}_V$ à l'aide de la bibliothèque CGAL [22].

Le mouvement du point de vue V induit une déformation des frontières de toutes les faces de $\mathcal{R}_V$. Aussi, pour pouvoir suivre ces déformations et maintenir correctement la topologie et la combinatoire des faces de $\mathcal{R}_V$, nous les triangulons chacune séparément et suivons la déformation de chaque « triangle sphérique ».

Après triangulation, la décomposition radiale contient assez d'informations pour pouvoir être maintenue continûment quel que soit le mouvement du point de vue et des polytopes.[1] Cette structure permet donc de résoudre le problème général de la maintenance d'une vue 3D d'un ensemble de polytopes par un point de vue mobile. Nous verrons, en revanche, que son efficacité (en tant que structure de données cinétique (KDS)) n'est pas optimale.

Il suit directement des travaux de Efrat *et al.* [50] que la taille de $\mathcal{R}_V$ est $\Theta(kn)$ (c'est-à-dire qu'elle est en $O(kn)$ et il existe des configurations de polytopes tels que la complexité de $\mathcal{R}_V$ soit $\Omega(kn)$). Dans le même papier, Efrat *et al.* montrent que le nombre de changements combinatoires subis par $\mathcal{R}_V$ est $\Theta(kn^2)$ lorsque la trajectoire du point de vue est algébrique de degré constant. Malheureusement, le nombre de changements de $\mathcal{R}_V$ sur une même trajectoire pourra être plus grand, car nous devons également tenir compte des changements combinatoires nécessaires à la maintenance de la triangulation de chaque face de $\mathcal{R}_V$. (Notons que [50] ne donne pas d'algorithme de maintenance de $\mathcal{R}_V$ lorsque la trajectoire est algébrique.)

5.2 La décomposition radiale 3D

Nous avons vu que, dans le plan, la décomposition radiale 2D possède une double nature, à la fois décomposition de l'espace libre, et en même temps courbe (avec embranchement) dans le complexe de visibilité 2D. Il en va de même après ajout d'une dimension, puisque chaque face de la décomposition radiale 3D peut aussi être vue comme l'union de segments libres maximaux possédant tous une orientation distincte. Ainsi, nous commençons par décrire la décomposition radiale 3D en terme de surface dans le complexe de visibilité 3D.

Soit $\mathcal{L}_V$ l'ensemble des droites passant par V. Dans l'espace des droites, $\mathcal{L}_V$ est une surface fermée sans bord. Soit $\mathcal{S}^V$ l'ensemble des segments libres maximaux de l'espace libre privé de $\{V\}$: $\mathcal{F}\backslash\{V\} = \mathbb{R}^3\backslash(\mathcal{O}\cup V)$. Nous définissons l'intersection de $\mathcal{L}_V$ et $\mathcal{S}^V$ comme suit : un segment maximal s de $\mathcal{S}^V$ appartient à $\mathcal{L}_V \cap \mathcal{S}^V$ si et seulement si sa droite support ℓ_s est dans $\mathcal{L}_V$ (ℓ_s contient V). Cette équivalence définit totalement la décomposition radiale de l'espace libre :

$$\mathcal{R}_V = \mathcal{L}_V \cap \mathcal{S}^V.$$

Notons que $\mathcal{L}_V \cap \mathcal{S}^V$ est une intersection « symbolique » puisque $\mathcal{L}_V$ et $\mathcal{S}^V$ ne vivent pas dans le même espace. Notons également que certaines cellules du complexe de visibilité associées à $\mathcal{S}^V$ — celles qui sont attachées à V — sont très dégénérées puisqu'elles sont au maximum de dimension 2 au lieu de 4.

[1] On imaginera par exemple, que les trajectoires sont représentées par des polynômes.

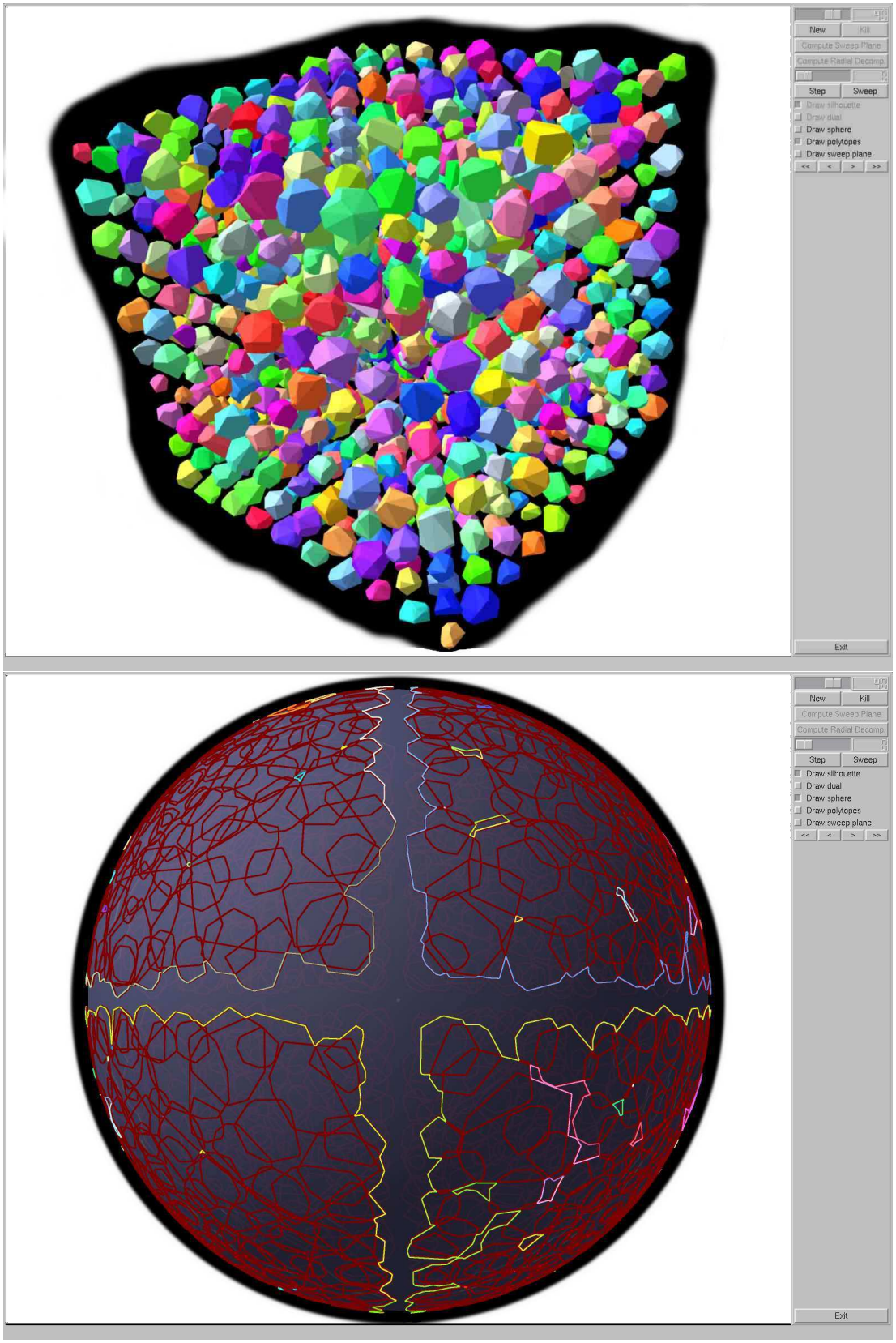


FIG. 5.1. En haut, l'ensemble $\mathcal{O}$ contient 1000 polytopes comprenant en tout à peu près 20000 arêtes. **En bas**, les arêtes silhouettes sont projetées en rouge sombre sur la sphère des directions centrée sur le point de vue V. Les contours des faces de $\mathcal{R}_V$ ayant V et ∞ pour bloqueurs sont dessinés en couleurs claires. Le calcul exact de $\mathcal{R}_V$ a pris quelques minutes (moins de 10). (La couleur de fond des dessins était noire, aussi nous avons blanchi une grande partie du fond pour ne pas trop fatiguer les imprimantes.)

On peut définir $\mathcal{R}_V$ autrement — et peut-être plus « proprement » — à partir de l'ensemble plus « canonique » $\mathcal{S}$ des segments libres maximaux dans $\mathcal{F}$:

$$\mathcal{R}_V = \{s \setminus \{V\}, s \in \mathcal{L}_V \cap \mathcal{S}\}$$

où $s \setminus \{V\}$ est à comprendre au sens de la soustraction de l'ensemble $\{V\}$ à un autre ensemble de points (s est pris ici comme un sous-ensemble de $\mathbb{R}^3$). Ainsi, si $s \in \mathcal{L}_V \cap \mathcal{S}$ contient V, alors $s \setminus \{V\}$ produit deux segments s_1 et s_2 distincts, qui émanent de V avec deux orientations opposées et s'étendent jusqu'au premier objet visible : s est coupé en deux. Si s ne contient pas V, alors il est inchangé.

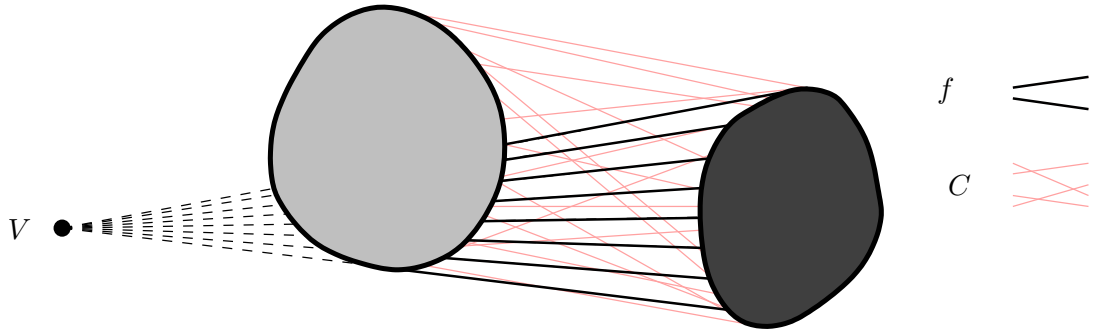


FIG. 5.2. f est un sous-espace de C, de dimension 2. L'oculteur proche de f est en gris clair ; l'oculteur éloigné de f est en gris foncé.

Rappelons que le complexe de visibilité $\mathcal{VC}$ de $\mathcal{S}$ est la partition de $\mathcal{S}$ en cellules maximalement connexes dans chacune desquelles chaque segment libre maximal « voit » la même paire d'objets (ici deux polytopes de $\mathcal{O}$). La structure du complexe de visibilité $\mathcal{VC}^V$ de $\mathcal{S}^V$ nous permet de structurer $\mathcal{R}_V$ de la façon suivante. Si C est une 4-cellule du complexe de visibilité $\mathcal{VC}^V$ contenant un segment libre colinéaire avec V, alors $\mathcal{R}_V \cap C$ comporte un nombre fini (typiquement 1) de composantes maximalement connexes formant les faces f_C de $\mathcal{R}_V$ (Figure 5.2). Similairement à la définition du complexe de visibilité $\mathcal{VC}$, une face de $\mathcal{R}_V$ est une composante maximalement connexe de segments libres de $\mathcal{R}_V$ possédant le même ensemble visible. Nous commettons ici l'abus de langage consistant à identifier $\mathcal{R}_V$, l'ensemble des segments libres de $\mathcal{S}^V$ colinéaires avec V, et $\mathcal{R}_V$, sa partition en cellules où la visibilité est constante.[2]

Soit f une face de $\mathcal{R}_V$ à l'intérieur de la 4-cellule C du complexe $\mathcal{VC}$. Puisque chaque segment libre d'une face f a une direction unique (dans f), f peut être visualisée comme un sous-ensemble connexe (et polygonal) de la « sphère des directions » $\mathbb{S}^2$. Le bord d'une face f est constitué de segments libres maximaux tangents à un (ou deux) polytopes, qui font partie de la frontière de C dans $\mathcal{S}$. Les segments de f tangents à deux polytopes sont en fait en nombre fini et apparaissent en tant que sommets du polygone correspondant à f sur la sphère des directions. Plusieurs faces de $\mathcal{R}_V$ peuvent apparaître dans une même 4-cellule C du complexe de visibilité (Figure 5.3).

Nous appellerons **oculteur éloigné de** f le polytope visible par les segments libres maximaux de f, le plus éloigné de V. L'autre polytope visible par ces segments sera l'**oculteur proche**. $\{V\}$ sera toujours un « oculteur proche » et l'infini (ou ciel bleu, ou simplement ∞) sera toujours un oculteur éloigné.

[2] Nous n'avons pas fait cet abus de langage pour la partition $\mathcal{VC}$ de l'ensemble des segments libres $\mathcal{S}$ vivant dans l'espace libre.

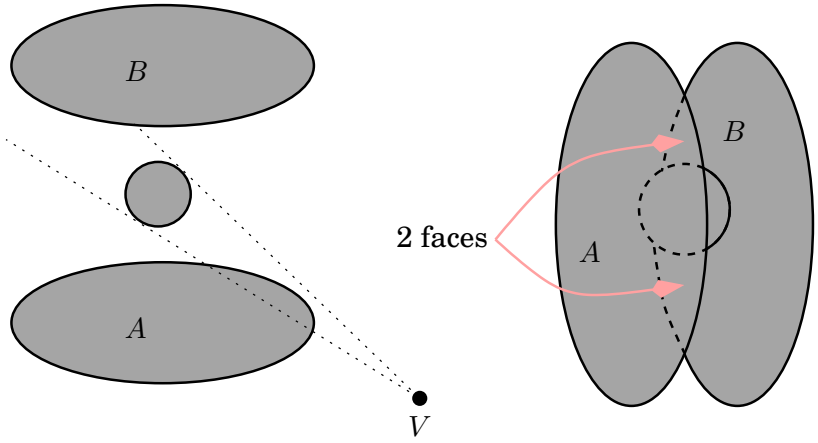


FIG. 5.3. À gauche, la scène figurant 3 objets et le point de vue V. **À droite**, vue de la scène depuis le point de vue V. La présence du petit oculteur entre les deux grands A et B induit, pour la position donnée de V la présence de deux faces dans $\mathcal{R}_V$, contenues dans la même 4-cellule (A,B) du complexe de visibilité.

Double nature de $\mathcal{R}_V$

Remarquons enfin que tous les segments maximaux de $\mathcal{R}_V$ sont deux-à-deux disjoints, en tant qu'ensembles de points de $\mathbb{R}^3$. De plus, leur union (toujours dans $\mathbb{R}^3$) est égale à l'espace libre privé de $\{V\}$: $\mathcal{F} \setminus \{V\}$. Ainsi, $\mathcal{R}_V$ revêt une double nature ; à la fois surface 2D dans le complexe de visibilité de $\mathcal{O} \cup \{V\}$, et décomposition de l'espace libre en cellules « radialement monotones ». Cette double nature se retrouve identiquement dans le plan, et nous nous en servirons pour la mise au point d'un algorithme de construction de la décomposition radiale $\mathcal{R}_V$, à l'aide d'un double balayage.

5.2.1 Description de $\mathcal{R}_V$ en tant que subdivision spatiale

Nous donnons ici une deuxième description de la décomposition radiale 3D $\mathcal{R}_V$, en nous plaçant cette fois-ci dans l'espace primal $\mathbb{R}^3$, dans lequel $\mathcal{R}_V$ est une subdivision de l'espace libre. Cette description est plus proche de l'implémentation C++ que nous avons faite. Nous explicitons la construction des **murs radials** de $\mathcal{R}_V$, dont découlera la description de ses faces. Notons que cette construction est très proche de celle d'une décomposition verticale d'un ensemble de triangles ou d'un arrangement (voir par exemple, [130] et les références incluses). Ainsi, en « poussant » le point de vue V à l'infini, $\mathcal{R}_V$ devient comparable à une décomposition verticale (dont les faces ne sont pas convexes, cependant).

Soit e une arête silhouette, par rapport au point de vue V, d'un polytope de $\mathcal{O}$. Soit q un point de e et $s(q)$ le segment libre maximal dans $\mathcal{F}$ de droite support (V, q), passant par q. La fonction

$$\text{vis}(s) : s \mapsto (A, B) \in (\mathcal{O} \cup \{V\}) \times (\mathcal{O} \cup \{\infty\})$$

qui associe à un segment libre ses deux oculteurs proche et lointain (son « ensemble de visibilité »), partitionne l'ensemble des points de e en sous-ensembles connexes maximaux de points $q \in e$ pour lesquels $\text{vis}(s(q))$ est constante. Notons $\{I_1, I_2, ..., I_{k_e}\}$ ces intervalles sur e.

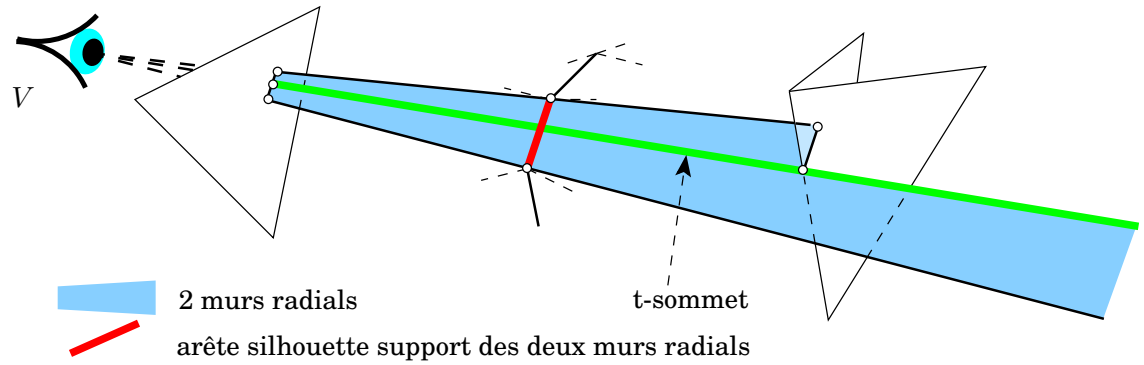


FIG. 5.4. Illustration montrant 2 murs radials supportés par une même arête silhouette. Le segment libre maximal vert est un t-sommet.

Un mur radial W_i de support e est défini en tant qu'union dans $\mathbb{R}^3$ par

$$W_i = \bigcup_{q \in I_i} s(q).$$

La Figure 5.4 montre deux murs radials. Considérons l'arrangement dans $\mathbb{R}^3$ induit par tous les murs radials ainsi définis (supportés par chaque arête silhouette dans $\mathcal{O}$) et les frontières des polytopes de $\mathcal{O}$. Nous pouvons distinguer dans cet arrangement deux types de faces de dimension 3. D'une part, l'intérieur de chaque polytope forme une face. D'autre part, toutes les autres cellules, qui ne sont pas l'intérieur d'un polytope, forment la décomposition radiale $\mathcal{R}_V$ vue comme subdivision spatiale de l'espace libre.

Exemple : disposons deux convexes disjoints dans l'espace et plaçons le point de vue de telle sorte que les deux convexes soient visuellement disjoints (un à gauche, un à droite, par exemple). Alors, la décomposition radiale 3D $\mathcal{R}_V$ comporte 5 faces.

Adjacences dans $\mathcal{R}_V$

La connectivité des faces de $\mathcal{R}_V$ est la même que dans le complexe de visibilité : chaque arête de bordure d'une face de $\mathcal{R}_V$ (un mur radial, donc un ensemble unidimensionnel de segments libres) est adjacente à trois faces de $\mathcal{R}_V$. Un t-sommet est l'intersection visuelle de deux arêtes silhouettes. On le considérera selon le contexte comme un segment libre maximal bitangent, ou comme un point de la carte de visibilité d'un point de vue situé sur la droite support de ce segment libre (voir Figure 5.4, en vert). Chaque t-sommet est adjacent à 4 murs radials et à 6 faces.

Le polyèdre de visibilité de V

Considérons l'ensemble A des segments des faces de $\mathcal{R}_V$ dont l'oculteur proche est $\{V\}$. Alors, l'union (dans $\mathbb{R}^3$) des segments de A vus comme sous-ensembles de $\mathbb{R}^3$ constitue précisément le polyèdre de visibilité de V, que nous noterons $\mathcal{V}_V$, et dont la frontière constitue la partie de $\mathcal{O}$ visible depuis V. Plus précisément, les murs radials faisant partie de la frontière de $\mathcal{V}_V$ ne doivent pas être considérés comme visibles, puisqu'ils sont « immatériels ».

5.2.2 Calculer et maintenir la vue de V à partir de $\mathcal{R}_V$

Nous venons de voir que la partie de $\mathcal{O}$ visible depuis V, c'est-à-dire l'ensemble des points de la scène visibles par V, ou plus simplement l'ensemble des objets visibles au moins partiellement par V, peut être extrait directement de la décomposition radiale $\mathcal{R}_V$: ce sont les oculteurs éloignés des faces dont l'oculteur proche est $\{V\}$. Construire $\mathcal{R}_V$ comme nous allons le voir plus loin est donc un moyen de calculer la vue de V. Construire $\mathcal{R}_V$ a cependant un autre avantage : il est relativement aisé de maintenir $\mathcal{R}_V$ lorsque V bouge dans l'espace libre. On peut alors connaître l'évolution de la vue de V quand ce dernier se « promène » dans l'espace libre $\mathcal{F}$. En fait, $\mathcal{R}_V$ est une structure relativement rigide puisque tous ses segments libres maximaux sont contraints à rester alignés avec le point de vue V. Nous verrons que c'est une force puisque $\mathcal{R}_V$ peut aussi être maintenue de la même manière dans le cas plus général où les polytopes sont autorisés à bouger ou à se déformer. C'est malheureusement aussi une faiblesse puisque cette contrainte augmente la complexité en temps de l'algorithme de maintenance, qui ne sera pas optimal.

Quand V bouge, tous les murs radials sont contraints à suivre ce mouvement pour que leurs segments constitutifs restent alignés avec V. Ainsi, les frontières de chaque face de $\mathcal{R}_V$ se déforment en même temps que V bouge. Il nous faut suivre cette déformation pour pouvoir mettre à jour la combinatoire et la topologie des faces de $\mathcal{R}_V$. Nous commençons par examiner les faces de $\mathcal{R}_V$ plus en détail.

5.3 Les faces de $\mathcal{R}_V$

Chaque face f de la décomposition radiale $\mathcal{R}_V$ est, dans le complexe de visibilité, un ensemble bidimensionnel de segments libres ayant les propriétés suivantes :

- tous ses segments libres ont la même visibilité ; ils voient les deux mêmes oculteurs.
- tous ses segments libres sont alignés avec le point de vue V ; leur droite support contient V.
- f est maximalement connexe vis-à-vis de la première propriété ci-dessus.

Une des caractérisations de ces faces est la donnée de leur bloqueur (ou oculteur) proche et de leur bloqueur lointain. Soit A et B les deux oculteurs d'une face f de $\mathcal{R}_V$. Rappelons nous qu'un oculteur proche peut ne pas appartenir à $\mathcal{O}$ (dans ce cas, cet « oculteur » est le point de vue V). De même, un oculteur lointain peut être le ciel bleu ∞. La distinction dans ce dernier cas est importante :

Si A ou B est un polytope ($A \in \mathcal{O}$ ou $B \in \mathcal{O}$), alors l'ensemble des directions orientées des segments libres de f : $\Omega_f \in \mathbb{S}^2$ est inclu dans une demi-sphère ouverte de $\mathbb{S}$. Cela découle directement de la convexité des oculteurs de $\mathcal{O}$. Il est alors pratique de mettre f en bijection avec un polygone planaire (éventuellement avec des trous), à l'aide d'une projection en perspective de centre V, comme illustré par la figure 5.5. Nous appellerons ces faces des *faces génériques*.

Dans le deuxième cas, on a $A = V$ et $B = \infty$. Appelons les faces ayant ces deux bloqueurs les *faces invisibles*.[3] Il n'y a pas de contrainte sur la forme que peuvent prendre ces faces, et l'intérieur de certaines d'entre elles peut contenir deux segments libres de directions antipodales. Pour ces faces, la projection bijective sur un polygone

[3] Ce sont les directions dans lesquelles le point de vue ne voit rien.

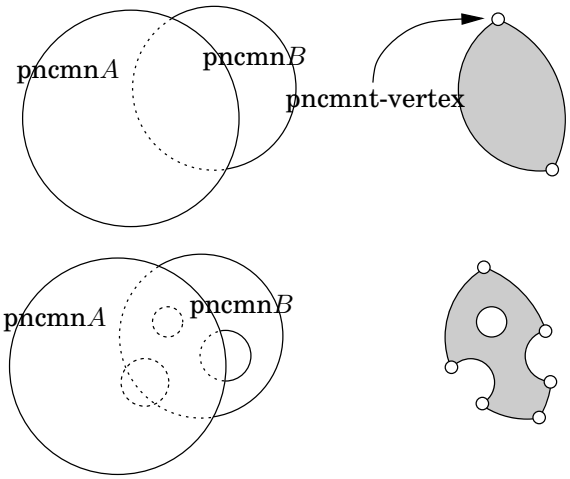


FIG. 5.5. Visualiser une face générique de $\mathcal{R}_V$ par bijection avec un polygone planaire.

planaire est impossible. Nous les qualifierons de *faces non-génériques*. Remarquons qu'une face invisible peut être générique, et une face ayant un bloqueur dans $\mathcal{O}$ est toujours générique.

Notons enfin que si V est à l'extérieur de l'enveloppe convexe de l'ensemble $\mathcal{O}$ des oculteurs, alors il existe une unique face non-générique et celle-ci contient une demi-sphère des directions dans lesquelles aucun oculteur n'est visible. Nous reviendrons plus loin sur le traitement de ces faces.

La figure 5.5 montre un exemple (en haut de la figure) extrêmement simple d'une face f de $\mathcal{R}_V$ dont les bloqueurs sont les objets A et B. Plus généralement, il y a un certain nombre d'objets entre A et B qui vont occulter certaines parties de f, créant ainsi des trous dans le polygone planaire de la projection de f. Le deuxième exemple (en bas de la figure) montre une face un peu plus complexe, dont le bord a été « mangé » par deux objets situés entre A et B, et dont l'intérieur a été « troué » par un troisième objet.

5.3.1 Triangulation des faces de $\mathcal{R}_V$

Rappelons que nous aurons besoin, quand les faces se déformeront du fait du mouvement du point de vue, de suivre leur évolution pour pouvoir maintenir leur combinatoire et leur topologie. Chaque face étant identifiée à sa projection sur la sphère $\mathbb{S}^2$ centrée en V, nous devons nous attacher à maintenir correcte la description géométrique des polygones sphériques correspondant à chaque face de $\mathcal{R}_V$. Cette tâche est infiniment simplifiée si les polygones sphériques sont préalablement triangulés. Alors, nous devrons suivre l'évolution de chaque triangle sphérique et agir en conséquence lorsqu'ils « s'écrasent », c'est-à-dire lorsque leurs trois sommets deviennent colinéaires.

Une face f dont un des bloqueurs est un occulteur de $\mathcal{O}$ peut être simplement triangulée en utilisant un algorithme de triangulation d'un polygone planaire à trous [36, Chapitre 3]. En effet, nous pouvons dans ce cas utiliser une projection azimutale fournissant une bijection entre les directions de la face f et un polygone planaire ayant la propriété de transformer les arêtes de f (des morceaux d'équateurs) en segments de longueur finie sur le plan. Trianguler le polygone planaire ainsi obtenu et le re-

projeter sur la sphère des directions nous fournit une triangulation « sphérique » de f.

Soit f une face dont un des bloqueurs est un oculteur de $\mathcal{O}$. Notons O_f cet oculteur : $O_f \in \mathcal{O}$. f est une face générique. Pour choisir la normale du plan de projection sur lequel on va projeter f, on pourra, par exemple, déterminer le point p de O_f le plus proche de V, puis utiliser la direction de vue de p comme normale.

Le problème se complique pour une face g dont les bloqueurs sont $\{V\}$ et ∞. Nous avons vu, en effet, qu'une telle face ne peut être aisément projetée sur un plan pour obtenir un polygone planaire. Nous proposons une solution simple mais peu satisfaisante.

Pour trianguler une face *invisible* g, nous allons au préalable la découper à l'aide d'arêtes sphériques (des segments d'équateurs) supplémentaires, de telle sorte que chaque partie résultante du découpage soit projetable sur un plan. Soit $\mathcal{T}$ un tétraèdre régulier centré au point de vue V (ses 4 sommets sont à distance 1 de V). En projetant les arêtes de $\mathcal{T}$ sur la sphère, nous partitionnons celle-ci en 4 triangles sphériques. L'intersection de la face g avec ces 4 triangles sphériques est alors calculée. Il en résulte au plus 4 polygones sphériques, que nous pouvons aisément trianguler séparément puisque chacun est projetable sur un plan. Lorsque nous suivrons l'évolution de ces triangles, il faudra faire attention à traiter correctement les arêtes des triangles provenant d'une arête du tétraèdre $\mathcal{T}$ (projetée sur la sphère).

Nous remarquons que pour trianguler correctement une face invisible *g, nous avons contourné le problème en la subdivisant au préalable en parties pouvant être projetées sur un plan, et donc triangulées simplement. Il nous semble que cet artifice (l'utilisation d'un tétraèdre pour découper la face g) peut être évité, et que les faces* invisibles *ont une structure suffisamment « bonne » pour pouvoir être triangulées directement sans ajout de sommet artificiel. Nous laissons cette étude pour de futures investigations.*

5.4 Construction de $\mathcal{R}_V$

Avant de continuer plus avant sur la maintenance de $\mathcal{R}_V$ lors du mouvement de V, nous nous concentrons sur la construction de la décomposition radiale 3D, pour laquelle nous proposons un algorithme efficace. La construction de $\mathcal{R}_V$ procède en deux étapes. Sans perte de généralité, nous supposons que V est situé à l'origine du repère orthonormé (O, x, y, z). D'abord, nous calculons l'intersection de $\mathcal{O}$ avec le plan $\Pi_0 = (O, x, y)$. Dans ce plan apparaît donc un ensemble (éventuellement vide) de polygones convexes. Quitte à effectuer une petite rotation de la scène autour de l'axe Oy, nous pouvons supposer qu'aucun polytope n'engendre une intersection ponctuelle. Nous calculons alors dans ce plan la décomposition radiale 2D $\mathcal{R}_0^2$ centrée à l'origine. Ensuite, nous faisons tourner ce plan autour de l'axe vertical Oy, continûment, sur 180 degrés. Notons $\Pi_t, t \in [0, \pi)$ le plan contenant l'axe Oy et formant un angle dièdre t avec le plan Π_0. Lors de cette rotation nous maintenons l'intersection de Π_t avec $\mathcal{O}$ ainsi que la décomposition radiale $\mathcal{R}_t^2$.

La décomposition radiale 3D $\mathcal{R}_V$ est ainsi balayée par le plan tournant Π_t, et nous la construisons au fur et à mesure que sa « coupe » planaire $\mathcal{R}_t^2$ est mise à jour. Nous

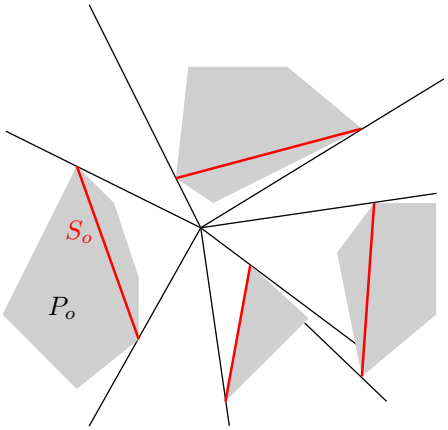


FIG. 5.6. La décomposition radiale 2D d'un ensemble de polygones convexes est combinatoirement équivalente à celle d'un ensemble de segments (rouges).

utiliserons les notations suivantes :

- n est la taille de l'ensemble $\mathcal{O}$ des polytopes (soit le nombre total d'arêtes).
- s est le nombre total d'arêtes silhouettes par rapport à l'origine.
- k est le nombre de polytopes. $k = |\mathcal{O}|$.
- m est le nombre de t-sommets dans $\mathcal{R}_V$. $0 \leq m = \Theta(kn) \leq O(s^2)$.

La suite de cette section peut être résumée par le théorème suivant :

Théorème 5.1. *Soit k polytopes disjoints de complexité totale n et un point V dans l'espace libre. Alors, $\mathcal{R}_V$ peut être construite en temps $O(n + s \log s + (s + m) \log k + k \log^2 k)$, où m est le nombre de t-sommets dans $\mathcal{R}_V$, et s le nombre d'arêtes silhouettes par rapport à l'origine.*

L'expression de la complexité en temps de la création de $\mathcal{R}_V$ ci-dessus est un peu complexe, mais a le mérite d'être précise. On peut supprimer le facteur s en utilisant l'inégalité $s \leq n$.[4] La complexité de la construction de $\mathcal{R}_V$ est alors en $O(n \log n + m \log k + k \log^2 k)$.

On sait que $m = \Theta(kn)$ [50] ; ce qui nous permet d'obtenir une borne en temps $O(n \log n + kn \log k + k \log^2 k) = O(kn \log n)$ pour notre algorithme. On peut également supprimer le facteur k en utilisant l'inégalité $k \leq n$. On obtient alors une complexité plus grossière en $O(n \log^2 n + m \log n)$. Nous remarquons que cette dernière borne est tout à fait similaire à la complexité du calcul d'autres décompositions de l'espace ayant beaucoup de points communs à la notre : citons la décomposition verticale d'un ensemble de triangles dans l'espace [37, 130] ou bien la décomposition de Mulmuley [106] utilisée en précalcul pour calculer efficacement, par la suite, le polyèdre de visibilité d'un point quelconque de l'espace libre.

5.4.1 Initialisation

Calcul des silhouettes. La première étape est le calcul de l'intersection de $\mathcal{O}$ avec Π_0, résultant en au plus k polygones convexes dans le plan Π_0. Cette opération se fait simplement en temps $O(n)$. On peut optimiser l'algorithme ainsi : chaque polytope est examiné un par un. Pour un polytope o, on examine les arêtes intersectant le plan Π_0.

[4] La complexité de la silhouette est parfois plus simple. Par exemple, si un polytope est une approximation fine et régulière d'un objet convexe, on a $s \approx O(\sqrt{n})$; voir, par exemple [6].

Dès qu'une telle arête est trouvée, il est aisé de construire l'intersection de o avec Π_0 en temps proportionnel à sa taille, au lieu d'examiner systématiquement toutes les arêtes de o. Cependant, l'accélération n'est que pratique, puisque la recherche d'une première arête intersectant Π_0 reste en $O(n)$ (complexité sur l'ensemble des polytopes de $\mathcal{O}$).[5]

Calcul de $\mathcal{R}_0^2$. Soit $o \in \mathcal{O}$ un polytope de la scène. Soit P_o son intersection avec le plan de balayage. Si P_o n'est pas vide, nous notons S_o le segment, dans le plan de balayage, reliant les deux sommets silhouettes de P_o par rapport à l'origine. Remarquons que la décomposition radiale 2D $\mathcal{R}_0^2$ est combinatoirement identique à la décomposition radiale 2D de l'ensemble des segments S_o de chaque oculteur en intersection avec Π_0 (voir la Figure 5.6). La construction de $\mathcal{R}_0^2$ peut donc se faire en temps $O(n + k \log k)$. Dans le plan Π_t (au cours du balayage), nous ne maintenons donc pas l'intersection complète de chaque polytope o de $\mathcal{O}$ avec le plan, mais seulement le segment S_o reliant les deux sommets silhouettes de l'intersection. Ce segment est inclu dans P_o car o est convexe, et il remplace avantageusement P_o pour la maintenance de $\mathcal{R}_t^2$ puisque sa complexité est constante. À un instant t donné, notons p_t le nombre de tels segments dans Π_t. Bien entendu, $p_t = O(k)$ puisque chaque oculteur produit au plus un segment dans Π_t. Alors, $\mathcal{R}_t^2$ compte $3p_t = O(k)$ faces.

5.4.2 Balayage de $\mathcal{R}_V$

Localisation dans $\mathcal{R}_t^2$. Lors du balayage de la scène par les plans Π_t, les segments $S_o, o \in \mathcal{O}$ vont se déformer, puisque leurs extrémités sont l'intersection d'une arête silhouette 3D (fixe) avec le plan Π en mouvement. À certains moments, un segment va apparaître ou disparaître, quand le plan Π commence ou cesse de balayer un oculteur. En particulier, lorsqu'un nouveau segment apparaît, il nous faut trouver dans quelle face de $\mathcal{R}_t^2$ l'événement se produit, afin de la subdiviser. Il nous faut donc munir $\mathcal{R}_t^2$ d'une structure de localisation de points. Cependant, les déformations des segments S_o induisent naturellement une déformation continue de la décomposition radiale $\mathcal{R}_t^2$ de Π_t. La localisation de la face contenant un point donné est donc impossible avec un algorithme de localisation planaire statique : la structure de localisation dans le plan Π_t doit être cinétique car les segments bougent continûment, et dynamique car tous les segments apparaissent et disparaissent à des moments différents. Nous montrons, au Chapitre 6 (lemme 6.1, page 89), que nous pouvons maintenir efficacement la décomposition radiale $\mathcal{R}_t^2$, de telle sorte que la localisation du « trapèze radial » contenant un point quelconque du plan s'effectue en temps $O(\log^2 k)$.

Balayage de $\mathcal{R}_V$. Au cours de la construction de $\mathcal{R}_V$, les contours (des chaînes de murs radials) de chaque face sont construits incrémentalement. À la fin du balayage, chaque face contient un ensemble de morceaux de contours, que l'on « recollera » pour former les contours finals.

Le plan Π tourne autour de l'axe y et passe continûment par les positions $\Pi_t, t \in [0, \pi)$. Ce faisant, la décomposition radiale $\mathcal{R}_t^2$ évolue, et c'est précisément ses changements que nous devons détecter pour mettre à jour $\mathcal{R}_t^2$ elle-même, ainsi qu'augmenter

[5] Avec les mêmes hypothèses que la note-de-bas-de-page précédente, on a une bonne chance de tomber sur une arête silhouette après $\sqrt{n}$ tirages aléatoires d'une arête.

la construction de $\mathcal{R}_V$. Bien entendu, pour tout t et t' dans $[0, \pi)$ tels que $t \neq t'$, les décompositions radiales 2D $\mathcal{R}^2_t$ et $\mathcal{R}^2_{t'}$ sont différentes. Mais, les « changements » qui nous intéressent sont combinatoires et non géométriques. Fort heureusement, lors du balayage de $\mathcal{R}_V$, ces changements combinatoires sont en nombre fini. Quels sont-ils ?

Tout d'abord, nous devons maintenir les extrémités de chacun des segments S_o apparaissant dans le plan Π. Ces extrémités sont l'intersection d'une arête silhouette de $\mathcal{O}$ avec Π. Une extrémité change quand le plan Π croise un sommet silhouette de $\mathcal{O}$ (c'est-à-dire un sommet d'une arête silhouette).

Il sera également nécessaire de détecter quand le plan Π *commence à* et *finit d'*entrer en intersection avec un oculteur, auquel cas il nous faut rajouter ou supprimer un segment dans le plan. Pour ce faire, on marquera, pour chaque oculteur, les premier et dernier sommets silhouettes qui seront intersectés par Π lors de sa rotation.

Tous ces événements sont connus à l'avance (puisque nous connaissons les sommets des arêtes silhouettes de $\mathcal{O}$) et sont au nombre de s, qui est le nombre d'arêtes silhouettes de $\mathcal{O}$. Nous les appelons les événements *irréguliers*.[6] Pour savoir dans quel ordre les traiter, nous commençons par les trier radialement autour de l'axe vertical y. Nous marquons également les sommets silhouettes extrêmes de chaque polytope. Cette opération se fait en temps $O(s \log s)$ et permet de stocker tous les événements irréguliers ordonnés dans une pile $\mathcal{E}_{\text{irr}}$ que l'on « dépilera » pour traiter chaque événement irrégulier. Notons qu'aucun nouvel élément ne sera ajouté à $\mathcal{E}_{\text{irr}}$ par la suite : lors du balayage, la taille $\mathcal{E}_{\text{irr}}$ ne fera que diminuer.

Il y a un troisième type d'événement intervenant dans la création des t-sommets. Ces événements seront appelés des événements *réguliers* et seront stockés dans une file d'attente $\mathcal{E}_{\text{rég}}$ permettant les manipulations usuelles en temps constant ou logarithmique, par exemple une queue de priorité [34]. Les événements réguliers sont les disparitions de faces de $\mathcal{R}^2_t$ qui ne sont pas dûes à la fin du balayage d'un polytope. Nous y reviendrons plus loin. Comme $\mathcal{R}^2_t$ contient $O(k)$ faces, la file $\mathcal{E}_{\text{rég}}$ aura une taille $O(k)$. Son initialisation coûte $O(k \log k)$.

Nous aurons donc deux listes triées d'événements à traiter. Pour déterminer le prochain événement à traiter, il suffit de choisir le premier parmi les premiers de chaque liste.

Effets des changements de $\mathcal{R}^2_t$ sur la construction de $\mathcal{R}_V$

Apparition d'un nouveau segment dans Π. Lorsque le plan de balayage croise un sommet silhouette S marqué comme *premier* sommet d'un oculteur o, nous devons insérer le segment S_o qui représente l'intersection de o avec Π. Il y a k événements de ce type.

L'insertion du nouveau segment dans Π nécessite la mise à jour de $\mathcal{R}^2_t$. On localise d'abord la face de $\mathcal{R}^2_t$ contenant le point S (en temps $O(\log^2 k)$, voir Chapitre 6). Cette face est alors subdivisée en 4 nouvelles faces (Figure 5.7).

Pour tout $t \in \mathbb{R}$ nous notons t^+ un réel strictement plus grand que t et infiniment proche de t, et t^- un réel strictement plus petit que t et infiniment proche de t.

Soit Π_t le plan de balayage contenant le sommet S responsable de cet événement d'apparition du segment S_o. Les deux sommets de S_o sont chacun l'intersection de Π_{t^+} avec une arête silhouette de o adjacente à S.

[6] Nous utilisons la terminologie de Mulmuley [106].

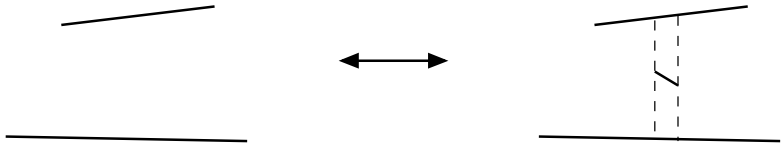


FIG. 5.7. De gauche à droite : l'apparition d'un segment dans le plan provoque la création de trois nouvelles faces. De droite à gauche : disparition d'un segment.

Dans $\mathcal{R}_V$ (la décomposition radiale 3D), deux nouvelles faces sont ajoutées, la première a o pour oculteur proche, et la seconde comprend o comme oculteur lointain. Un nouveau contour de face est créé. Ce dernier comprend deux murs (dont S est un sommet), en cours de balayage, et est copié dans chacune de ses 3 faces adjacentes, dont les deux nouvelles faces font partie.

La structure de localisation dans la décomposition radiale $\mathcal{R}_t^2$ est mise à jour en temps $O(\log k)$ (voir Chapitre 6). Enfin, nous devons mettre à jour la file d'attente $\mathcal{E}_{\text{rég}}$ pour prendre en compte la déformation des nouvelles faces créées dans $\mathcal{R}_{t^+}^2$ (voir le paragraphe **Construire les t-sommets** un peu plus loin), ce qui se fait en temps $O(\log k)$. Leur coût total est donc $O(k(\log^2 k + \log k + \log k)) = O(k \log^2 k)$.

Disparition d'un segment dans Π. Lorsque le plan de balayage croise un sommet silhouette marqué comme *dernier* sommet d'un oculteur o, nous devons supprimer le segment qui représente l'intersection de o avec Π. Il y a k événements de ce type. De même, leur coût total est $O(k \log k)$. Le terme en $\log^2 k$ n'apparaît pas puisqu'il n'y a pas de recherche à effectuer parmi les faces de $\mathcal{R}_t^2$.

Deux faces de $\mathcal{R}_V$ finissent d'être balayées. Nous devons alors fermer le contour extérieur de ces deux faces (rappelons nous qu'une face possède un contour extérieur et éventuellement plusieurs contours intérieurs). Enfin, le contour intérieur d'une troisième face est fermé.

Changement de l'arête support d'un sommet d'un segment de Π. Quand Π traverse un sommet silhouette de o qui n'est ni *premier* ni *dernier*, alors une arête silhouette de o cesse d'être balayée et une autre commence à être balayée. Un des sommets du segment représentant o doit donc mettre à jour son arête support. Il y a $s-2k$ événements de la sorte, dont le traitement peut se faire en temps $O((s-2k)\log k)$ (toujours pour mettre à jour la file $\mathcal{E}_{\text{rég}}$).

Dans $\mathcal{R}_V$, un mur radial vient de finir d'être balayé et donc ses extrémités sont connues ; le mur est construit complètement. Un nouveau mur commence à être balayé ; il est ajouté au contour des 3 faces de $\mathcal{R}_V$ qui lui sont adjacentes.

Construire les t-sommets. Nous n'avons pas encore pris en compte tous les événements possibles qui peuvent arriver à $\mathcal{R}_t^2$ quand t passe de 0 à π. Du point de vue de cette décomposition radiale, chaque événement est en fait la disparition de face(s) (car deux murs fusionnent), l'apparition de faces, ou bien le changement de l'arête silhouette support d'un mur radial. Nous avons pris en compte l'apparition et la disparition d'une face due à l'apparition ou disparition d'un segment S_o dans le plan de balayage, mais ces événements se produisent aussi dans le cas où les deux murs d'une face sont attachés (tangents) à deux segments différents dans Π. Auquel cas, un certain nombre de faces disparaissent et autant réapparaissent. Autrement dit, la

structure de $\mathcal{R}_t^2$ est localement reconfigurée. C'est exactement la situation rencontrée au Chapitre 4, lorsque la décomposition radiale 2D est modifiée alors que deux murs radials se croisent (voir Figure 4.3 page 42). Ce sont ces événements réguliers que nous stockons dans la file $\mathcal{E}_{\text{rég}}$.

Au moment où les deux murs se croisent, leur union forme un segment libre maximal qui est tangent à deux oculteurs et dont la droite support passe par V. C'est donc un t-sommet de la décomposition radiale 3D $\mathcal{R}_V$.

Ainsi, pour pouvoir construire complètement $\mathcal{R}_V$, nous calculons pour chaque face F de $\mathcal{R}_0^2$ l'instant t_F où ses deux murs vont se croiser (il se peut que t_F n'existe pas). Pour ce faire, souvenons nous qu'un mur est supporté par une arête silhouette d'un polytope de $\mathcal{O}$. Notons e_d et e_g ces arêtes pour les deux murs de la face F. Il suffit alors de calculer s'il existe une droite passant par V, e_d et e_g. Si elle existe, nous insérons alors cet événement de croisement dans la file d'attente des événements futurs. Bien entendu, si t_F est inférieur au temps du dernier événement traité, il est inutile de l'insérer.

La procédure ci-dessus n'est pas suffisante pour garantir que les t-sommets de $\mathcal{R}_V$ sont correctement détectés. En effet, il se peut qu'après avoir inséré un futur événement « t-sommet » dans la file $\mathcal{E}_{\text{rég}}$, d'autres événements aient lieu qui modifient la face F, auquel cas l'événement placé dans $\mathcal{E}_{\text{rég}}$ devient caduque et il faut le supprimer. Ainsi, lorsqu'une face est modifiée ou supprimée, nous vérifions s'il existe un événement concernant cette face dans $\mathcal{E}_{\text{rég}}$ et, le cas échéant, nous le supprimons de la liste. Si la face est modifiée (et non supprimée), nous insérons alors éventuellement un nouvel événement dans $\mathcal{E}_{\text{rég}}$.

Notons m le nombre de t-sommets dans $\mathcal{R}_V$. Nous observons que le nombre total de t-sommets *potentiels* insérés dans la liste des événements futurs est $m + O(s)$ puisque chaque événement traité donne lieu à la modification d'un nombre constant de faces de $\mathcal{R}_t^2$ (pour chacune desquelles on insérera éventuellement un événement dans $\mathcal{E}_{\text{rég}}$). Traiter les événements de type « t-sommet » prend donc un temps $O((s+m)\log k)$, dont le facteur $s \log k$ a déjà été compté dans la description des événements irréguliers. Lors du balayage d'un t-sommet par le plan Π, 6 faces de $\mathcal{R}_V$ sont modifiées. Parmi ces 6 faces, une est créée, une est fermée et 4 voient leur contour de murs radials prolongés.

Fin du balayage. À la fin du balayage, les murs radials de $\mathcal{R}_V$ bordant la silhouette d'un polytope en intersection avec Π_0 ne sont pas complets. Il faut donc éventuellement effectuer un « recollement » de $\mathcal{R}_0^2$ avec $\mathcal{R}_{\pi^-}^2$ pour finaliser les contours de chaque face. Chaque polytope en intersection avec Π_0 voit la chaîne de murs radials l'entourant coupée exactement en deux parties. On peut donc effectuer le recollement en temps $O(k)$.

Notons enfin qu'il faut traiter séparément le cas des polytopes en intersection avec l'axe vertical (O, y). Il n'y a cependant pas de difficulté particulière et nous n'entrerons donc pas dans les détails.

En sommant les complexités en temps décrites ci-dessus, nous obtenons la borne en temps du théorème 5.1.

5.5 Maintenance de $\mathcal{R}_V$

Nous avons vu dans l'état de l'art (page 25) que maintenir la vue d'un point mobile V peut se faire en maintenant la combinatoire d'un graphe sur une sphère. Plus précisément, ce graphe est la projection de toutes les arêtes silhouettes de la scène sur la sphère unité centrée sur V, que nous noterons $\mathbb{S}$. De même, nous visualiserons la décomposition radiale $\mathcal{R}_V$ comme un ensemble de sous-ensembles de la sphère unité $\mathbb{S}$. Un point P de $\mathbb{S}$ sera interprété comme un segment libre inclu dans la demi-droite $[V, \infty)$ de direction $V \rightarrow P$.

Considérons la projection de toutes les arêtes silhouettes des polytopes de $\mathcal{O}$ sur $\mathbb{S}$. Leur arrangement (un arrangement de géodésiques sur $\mathbb{S}$) est un graphe. Lorsque les objets ou le point de vue bougent, ce graphe évolue de concert. Un changement combinatoire de ce graphe a lieu quand 3 arêtes se croisent en un même point (de $\mathbb{S}$). Soit un sommet croise l'intérieur d'une arête (on a un événement de type VE) ; soit les trois arêtes se croisent en un point qui est intérieur à chacune (on a alors un événement de type EEE, voir Figure 5.8(b)).

La visibilité de V peut changer uniquement lors de tels événements. Nous qualifions ces événements de *transparents* car l'occlusion n'est pas prise en compte ; ainsi, un événement transparent peut ne pas avoir de conséquence sur la vue de V. Les événements qui modifient effectivement la vue de V sont appelés *opaques*. Enfin, un événement au cours duquel les trois arêtes convernées sont mutuellement visibles sera appelé *semi-opaque*. Les événements semi-opaques sont précisément les événements qui modifient la combinatoire de la décomposition radiale 3D $\mathcal{R}_V$.

Nous expliquons ci-dessous comment on peut maintenir $\mathcal{R}_V$ lorsque le point de vue bouge. Après que chaque face de $\mathcal{R}_V$ (vues comme des polygones sphériques) a été triangulée, la maintenance de $\mathcal{R}_V$ consiste simplement à détecter quand un triangle s'applatit, et à mettre à jour $\mathcal{R}_V$, sa triangulation et la file d'attente des futurs événements. Ce processus est relativement simple, mais comporte de nombreux cas, assez fastidieux à traiter complètement. Aussi, nous contenterons nous de donner une vue d'ensemble de la maintenance de $\mathcal{R}_V$.

Maintenance des arêtes silhouettes

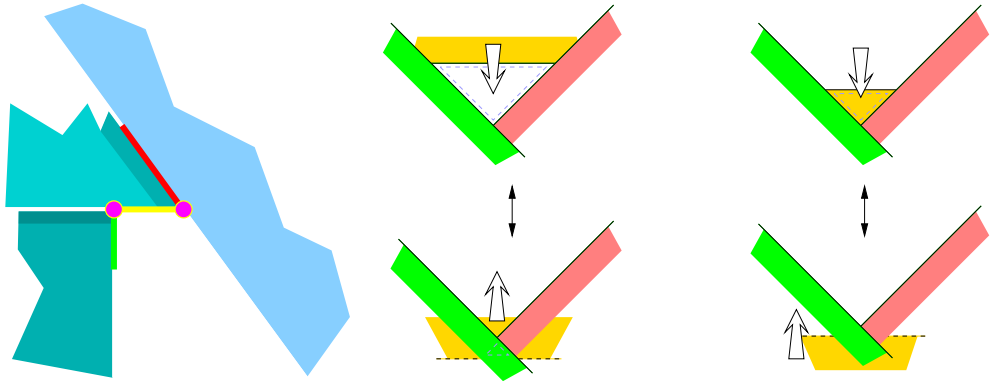
Les arêtes silhouettes de chaque polytope doivent être maintenues. On utilise pour cela un certificat pour chaque triangle adjacent à une arête silhouette. Ce certificat a pour but de garantir que le point de vue reste du même côté du plan support de ce triangle. Quand un tel certificat faillit, la silhouette est mise à jour (une arête en devient deux, ou vice-versa). Il faut également mettre à jour les informations sur les arêtes support des murs radials qui étaient supportés précédemment par la (ou les) arête(s) qui cessent d'être silhouettes. Si l'arête e cesse d'être silhouette, la mise à jour peut se faire en temps $O(m_e \log |\mathcal{R}_V|)$ où m_e est le nombre de murs radials supportés par l'arête e et $\mathcal{R}_V$ est la taille de la décomposition radiale 3D.

5.5.1 Maintenir une face de $\mathcal{R}_V$

Détection des événements EEE

Supposons qu'un événement de type EEE ait lieu à l'instant t, pour les arêtes e_1, e_2, et e_3 (voir Figure 5.8(a)). Alors, à l'instant $t - \epsilon$, il y a un mur radial w_2 (en jaune sur

la figure) supporté par e_2, dont les sommets sont des t-sommets adjacents aux murs radials w_1 et w_3 (respectivement supportés par les arêtes e_1 et e_3). Nous pouvons ainsi détecter l'événement en examinant les murs adjacents à un mur donné, via ses deux sommets.



(a) (b)

FIG. 5.8. Événements de type EEE.

Détection des événements VE

Contrairement aux événements de type EEE, les trois arêtes liées à un événement de type VE ne sont pas forcément reliées d'une manière simple dans la structure de données de $\mathcal{R}_V$. Nous devons donc structurer chaque face afin de mieux pouvoir détecter les événements VE. Nous avons vu précédemment comment on pouvait trianguler les faces de $\mathcal{R}_V$.

Dès lors, une solution simple est de trianguler chaque face de $\mathcal{R}_V$, et de maintenir la propriété que chaque triangle doit avoir une aire non-nulle. Lorsqu'un événement de type « le triangle T va s'applatir au temps t » survient (au temps t), la triangulation et la décomposition radiale $\mathcal{R}_V$ sont mises à jour. L'ensemble des événements de type VE est un sous-ensemble de ceux garantissant que la triangulation d'une face reste bien une triangulation. Ainsi, chaque événement VE sera détecté.

Traitement des événements

Lorsqu'un triangle $T = (v_1, v_2, v_3)$ d'une face générique f s'applatit au temps t, les modifications à apporter à $\mathcal{R}_V$ et à la triangulation de f dépendent du type de T. Nous supposons qu'à l'instant t, le sommet v_1 se trouve entre v_2 et v_3.

- L'arête v_2v_3 n'est pas un mur de f (Figure 5.9). La triangulation de f est mise à jour en *flippant* l'arête v_2v_3. Il n'est pas nécessaire de modifier la décomposition radiale.
- L'arête v_2v_3 de T est un mur de f. Les autres arêtes ne le sont pas (Figure 5.10). Nous avons affaire à un événement VE (opaque ou semi-opaque). Les deux murs adjacents à v_1 forment un angle *reflex*. Par conséquent, v_1 ne peut pas être un t-sommet : v_1 est un sommet régulier (attaché à un sommet d'un oculteur). Si le genre de f est 0, alors f est coupée en deux faces. Sinon, le genre de f est diminué d'une unité : deux contours de murs radials sont fusionnés.
- L'arête v_1v_3 de T n'est pas un mur de f. Les deux autres arêtes sont des murs de f (Figure 5.11). On a un événement de type VE qui n'induit pas d'apparition ou

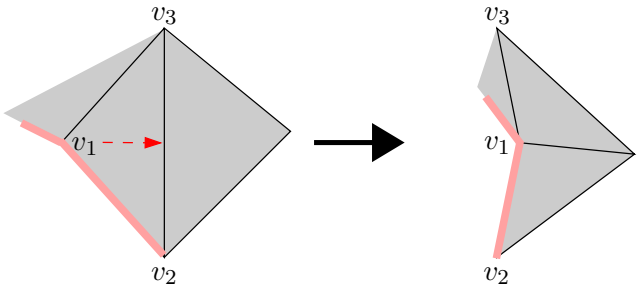


FIG. 5.9. Applatissement d'un triangle de la triangulation d'une face de $\mathcal{R}_V$. Les contours épais représentent le bord éventuel d'une face de $\mathcal{R}_V$.

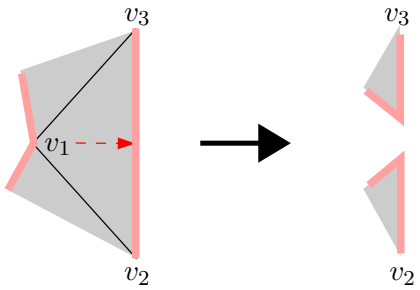


FIG. 5.10. L'applatissement de ce triangle induit un événement de type VE sur la décomposition radiale 3D.

de disparition de faces dans $\mathcal{R}_V$. On a simplement un « glissement » d'un sommet silhouette à travers un mur radial : le t-sommet v_2 voit l'une de ses deux arêtes support changer.

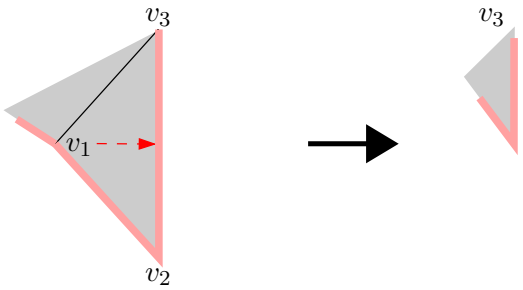


FIG. 5.11. Le t-sommet v_2 change de combinatoire.

– Les 3 arêtes de T sont des murs de f. Si les 3 sommets sont des t-sommets, on a alors affaire à un événement de type EEE.

Remarque. Certains événements de type EEE peuvent être détectés via l'écrasement d'un triangle ; c'est par exemple le cas pour les événements illustrés sur la Figure 5.8(b) s'ils sont lus de haut en bas (sur la figure). En effet, ces événements correspondent à la disparition d'une face de $\mathcal{R}_V$ qui a une forme triangulaire juste avant de disparaître. D'autres événements de type EEE ne peuvent pas être détectés ainsi. C'est le cas pour l'exemple de la Figure 5.8(a) et les exemples de la Figure 5.8(b) lus de bas en haut. On devra donc prendre garde à ne pas détecter plusieurs fois le même événement.

5.5.2 Discussion

Performances

La structure de données cinétique $\mathcal{R}_V$ que nous avons présentée permet de maintenir continûment le polyèdre de visibilité d'un point de vue en mouvement parmi un ensemble de polytopes disjoints. C'est, il nous semble, la première structure présentée pour résoudre ce problème *pour un mouvement algébrique quelconque*. Notons de plus que rien n'empêche les polytopes d'être également mobiles, voire de subir des déformations (à condition que leur convexité reste garantie).[7] La décomposition radiale 3D est cependant loin d'être optimale. Examinons les performances de notre structure selon les critères d'analyse des structures de données cinétiques (voir page 13).

$\mathcal{R}_V$ est *locale*. Ainsi, chaque élément géométrique de $\mathcal{R}_V$ est impliqué dans un nombre constant de certificats (de futurs événements potentiels). La compacité de $\mathcal{R}_V$ est plus difficile à caractériser. Rappelons qu'une KDS est dite *compacte* si sa taille est linéaire (à un facteur logarithmique près) en le nombre d'éléments mobiles dans la scène. Ici, nous avons une scène de taille n mais l'attribut que nous maintenons sur cette scène (le polyèdre de visibilité) peut avoir une taille quadratique, tout comme la taille de $\mathcal{R}_V$. Mais, il se peut également que la taille du polyèdre de visibilité soit en $O(n)$ alors que celle de $\mathcal{R}_V$ soit, au même moment, quadratique (imaginons une grille dont chaque barreau est un polytope, cachée derrière un grand polytope). $\mathcal{R}_V$ n'est donc pas une structure compacte. $\mathcal{R}_V$ n'est pas non plus *répondante* puisque le coût d'un événement de type « changement d'arête silhouette » peut coûter un temps $O(n)$ (dans certains cas de figure particulièrement retors).

Vers une partition de $\mathcal{R}_V$ sensible à sa complexité

Lorsque nous triangulons chaque face de $\mathcal{R}_V$, nous constatons que le nombre de triangles créés est proportionnel au nombre de murs radials formant les frontières de chaque face. Supposons que les oculteurs soient en fait réellement des sphères $S_i, i = 1..k$. Alors, la complexité de $\mathcal{R}_V$ sera en $O(k^2)$. Supposons maintenant que les polytopes de la scène soient des approximations très fines des sphères S_i. Nous avons alors $k \ll n$. Cependant, la complexité de $\mathcal{R}_V$ reste en $\theta(kn)$. Pourrait-on structurer $\mathcal{R}_V$ différemment, de telle manière que sa taille soit sensible à la disposition des oculteurs les uns par rapport aux autres et moins sensible aux détails de leur tessélation ? Nous laissons la question en suspens mais suggérons quelques pistes qui nous paraissent intéressantes. Nous référons également le lecteur aux travaux de Efrat *et al.* [50] qui, dans ces mêmes conditions géométriques, donnent des bornes sur le nombre d'événements ayant lieu lorsque le point de vue décrit une trajectoire algébrique ou linéaire.

Les contours de chaque face gagneraient peut-être à être stockés implicitement. En effet, nous pouvons facilement voir que la donnée des t-sommets et de la position du point de vue suffit à reconstituer tous les murs radials de $\mathcal{R}_V$ en temps proportionnel à leur taille, en marchant simplement le long des silhouettes des polytopes.

De même, nous souhaiterions réduire le coût mémoire associé à la triangulation de chaque face de $\mathcal{R}_V$. À l'instar des travaux de Kirkpatrick *et al.* [83–85], nous proposons d'utiliser autant que possible des pseudo-triangulations à la place des triangulations,

[7] Cela augmentera par contre le degré des équations dont les racines sont les moments auxquels un événement peut avoir lieu.

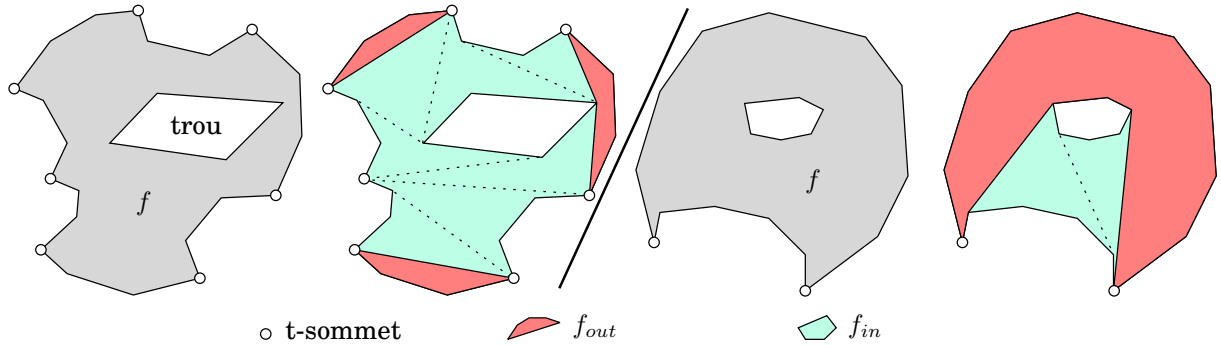


FIG. 5.12. Une face générique est partitionnée en une partie intérieure f_{in} et un ensemble de « croissants protecteurs », f_{out}. Les pointillés indiquent une pseudo-triangulation de f_{in}.

pour subdiviser les faces de $\mathcal{R}_V$.[8] Nous avons entrepris quelques travaux préliminaires qui nous permettent de penser que l'on pourrait trouver une alternative à la triangulation, mêlant pseudo-triangles et triangles, de telle sorte que la subdivision de chaque face soit sensible au nombre de t-sommets et sensible à la « séparation » des bords d'une face provenant de polytopes différents.

Soit f une face générique de $\mathcal{R}_V$ dont les bloqueurs sont A et B. Soit $\mathrm{Ext}(f)$ l'ensemble des sommets de f correspondant à des segments libres tangents à un oculteur n'appartenant pas à $\{A, B\}$. $\mathrm{Ext}(f)$ est l'union des sommets concaves de f et des t-sommets de f qui ne correspondent pas à un segment libre maximal bitangent à A et B. Nous partitionnons f en deux parties, f_{in} et f_{out} (Figure 5.12) que nous traiterons séparément :

$$f_{in} = f \cap ConvexHull(\mathrm{Ext}(f))$$
$$f_{out} = f \setminus f_{in}.$$

Les croissants protecteurs de f_{out}

f_{out} consiste en un ensemble de polygones en forme de croissant, que nous appelons les « croissants protecteurs », constitués de deux chaînes convexes. Soit ☾ un croissant protecteur. Notons c_e et c_i ses deux chaînes convexes, de telle sorte que c_e soit sur le bord de f et c_i soit adjacent à f_{in} ou à un trou de f. Pour maintenir ☾ , il faut maintenir ces deux chaînes et également détecter à quel moment un sommet de c_i viendrait à passer au travers de c_e. La chaîne extérieure c_e est constituée de murs radials supportés par un seul polytope et ne nécessite donc aucune maintenance (autre que celle des arêtes silhouettes). Il faudra en revanche s'assurer que la chaîne intérieure c_i garde sa convexité.

Enfin, nous devons être capables de détecter quand un sommet de c_i passera au travers de c_e. Pour ce faire, nous pseudo-triangulons une partie de ☾ pour former un « barrage » entre c_i et c_e (Figure 5.13). En partant du sommet v_1 (voir la figure), nous ajoutons le pseudo-triangle capable d'englober le plus de sommets de la chaîne intérieure, et nous répétons le processus à partir du sommet du pseudo-triangle que l'on vient d'ajouter sur la chaîne c_e.

Le nombre de pseudo-triangles ainsi créés est dépendant de la proximité (ou de la séparation) des deux chaînes de ☾ . On rompt ainsi la dépendance entre la taille de la structure de détection des événements et le nombre $O(s + m)$ de murs radials de $\mathcal{R}_V$.

[8] Un pseudo-triangle dans le plan est un ensemble dont la frontière est formée de trois courbes concaves [118].

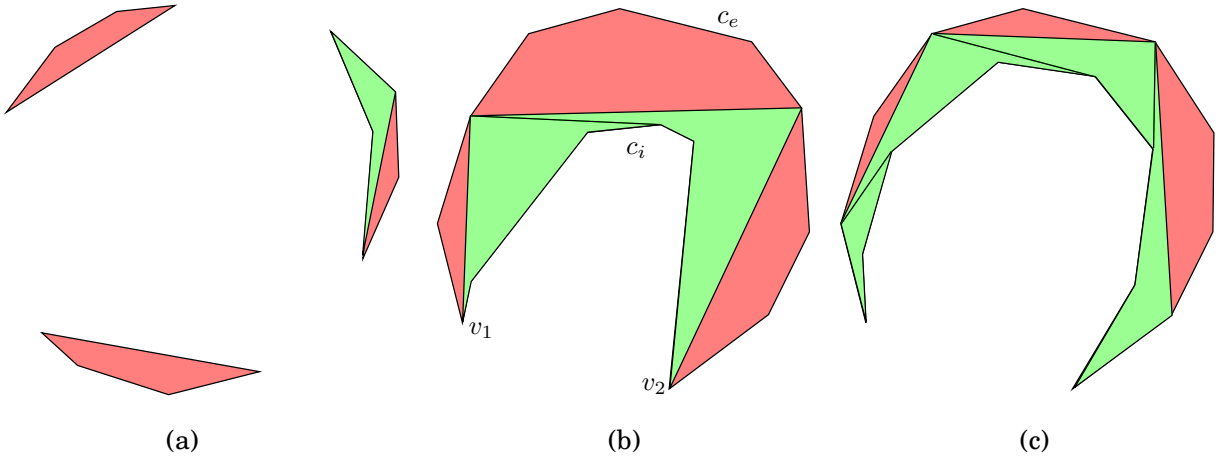


FIG. 5.13. Création d'un barrage de pseudo-triangles pour séparer les deux chaînes convexes d'un croissant protecteur. **(a)** et **(b)** montrent les barrages créés pour les croissants protecteurs de la Figure 5.12. **(c)** montre l'accroissement du nombre de pseudo-triangles lorsque les chaînes c_i et c_e se rapprochent.

Pour chaque pseudo-triangle τ, deux certificats sont créés qui garantissent que l'angle interne formé aux deux sommets de τ sur c_i reste non nul. On peut observer facilement que la maintenance de ce barrage entre c_i et c_e permet de détecter le passage d'un sommet de c_i à travers c_e, et donc de détecter les événements semi-opaques de type VE dans cette région de $\mathcal{R}_V$.

Nous arrêtons ici notre examen des croissants protecteurs, mais il est évident que plus de travail est nécessaire pour fournir un algorithme précis.

Maintenir f_{in}

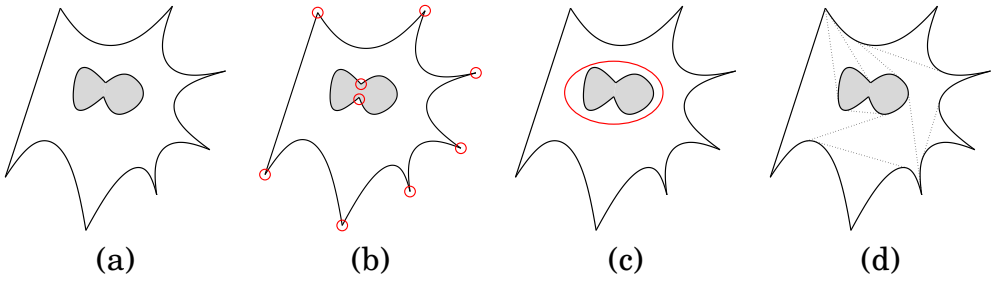


FIG. 5.14. **(a)** f_{in} (en blanc). **(b)** 9 t-sommets. **(c)** un trou. **(d)** une pseudo-triangulation de f_{in} comportant $9 - 2 + 2 \times 1 = 9$ pseudo-triangles.

Pour détecter les événements apparaissant sur les frontières de f_{in}, nous utilisons une pseudo-triangulation de ce polygone sphérique (Figure 5.12). Nous avons vu que le nombre de pseudo-triangles dans f_{out} est dépendant de la séparation des deux chaînes de chaque croissant protecteur. Nous allons voir que le nombre de pseudo-triangles dans f_{in} ne dépend que du nombre de ses t-sommets et du nombre de ses trous.

Le bord extérieur de f_{in} est une série de chaînes concaves (Figure 5.14(a)). Comptons le nombre PT de pseudo-triangles dans une pseudo-triangulation de f_{in} (Figure 5.14(d)). Soit T le nombre de t-sommets le long du bord de f_{in} (Figure 5.14(b)) et H le nombre de trous dans f (Figure 5.14(c)). Nous avons alors :

$$\boxed{PT = T - 2 + 2H}$$

Preuve : Considérons le graphe planaire $G = (W, E)$ dont les sommets W sont les t-sommets T de f ainsi que les sommets W_t de la frontière de f_{in} auxquels sont tangents les bitangentes de la pseudo-triangulation ($W = T \cup W_t$, W_t contient seulement les sommets qui sont pas également des t-sommets). Les faces de G sont les pseudo-triangles de f_{in} et les trous H de f. La relation d'Euler pour un graphe planaire nous dit que

$$|H| + PT - |E| + |W| = 1 \quad (5.1)$$

$$\text{On a également} \quad |W| = T + |W_t| \quad (5.2)$$

Soit B l'ensemble de toutes les bitangentes dans la pseudo-triangulation. Les sommets de $W = W_t \cup T$ sont situés sur les contours de f_{in}, ils génèrent donc $|W_t| + |T|$ arêtes du graphe G. Les autres arêtes sont les bitangentes ajoutées dans f_{in} pour la pseudo-trianguler. On a donc :

$$|E| = T + |W_t| + |B| \quad (5.3)$$

$$\text{Les égalités 5.2 et 5.3 impliquent} \quad -|E| + |W| = -|B| \quad (5.4)$$

$$\text{Les égalités 5.1 et 5.4 impliquent} \quad |H| + PT - |B| = 1 \quad (5.5)$$

Nous montrons maintenant que le nombre d'adjacences Adj entre les pseudo-triangles et leurs 3 coins est

$$Adj = 3PT = T + 2|B| \quad (5.6)$$

Pour $Adj = 3PT$, c'est facile. Pour $Adj = T+2|B|$, on observe que chaque bitangente donne deux adjacences. Soit b le nombre de bitangentes adjacentes à un t-sommet t. Au point t, nous avons $b + 1$ adjacences, or b adjacences ont déjà été comptées avec les bitangentes ; donc il nous reste une adjacence à compter par t-sommet. D'où l'égalité ci-dessus. En combinant les équations 5.5 et 5.6, nous obtenons le résultat. □

Maintenant, faisons deux remarques.

1. un t-sommet est adjacent à exactement 6 faces de $\mathcal{R}_V$.
2. le nombre total de trous dans les faces ne peut excéder k, puisqu'un polytope ne peut pas contribuer à plus d'un trou. De plus, un trou apparaît en tant que trou dans exactement une face.

Soit $\mathcal{PT}_{in}$ le nombre total de pseudo-triangles dans toutes les parties intérieures des faces génériques de $\mathcal{D}$. Nous avons

$$\mathcal{PT}_{in} = \sum_{f \in \mathcal{D}} (T_{f_{in}} - 2 + 2H_{f_{in}})$$

$$\mathcal{PT}_{in} \leq 6m + 2k$$

$$\boxed{\mathcal{PT}_{in} = O(m + k)}$$

Autrement dit, les événements semi-opaques apparaissant dans les parties f_{in} de toutes les faces f génériques de $\mathcal{R}_V$ peuvent être détectés à l'aide d'une structure maintenant $O(m + k)$ certificats. Ici encore, il apparaît que nous pouvons rompre la dépendance entre la taille de la structure de détection des événements et le nombre $O(s + m)$ de murs radials de $\mathcal{R}_V$.

5.6 Implémentation

Nous avons implémenté une partie seulement de l'algorithme de maintenance de visibilité que nous venons de décrire : la construction de la décomposition radiale initiale $\mathcal{R}_V$. Les éléments manquants sont d'une part la triangulation de chaque face de $\mathcal{R}_V$ et d'autre part la maintenance de $\mathcal{R}_V$ lors d'un mouvement du point de vue. Ces deux éléments ne semblent pas poser de problème majeur à une implémentation.

Les calculs géométriques d'intersections et de prédicats ont été programmés avec l'aide de la bibliothèque CGAL et un type numérique exact, garantissant l'absence d'erreur numériques. L'implémentation produit donc la décomposition radial exacte. On a cependant fait l'hypothèse que la scène est en position générique ; on suppose ainsi qu'aucun quadruplet de sommets ne sont coplanaires, qu'aucune paire de sommets n'apparaît au même moment lors du balayage rotationnel pour la construction de la décomposition radiale, ou encore qu'aucune face de $\mathcal{R}_V$ n'est dégénérée (aire nulle, sommet au milieu d'une arête, *etc.*).

Du côté théorique, le plus grand manque concerne la pseudo-triangulation des faces de $\mathcal{R}_V$. Nous avons montré que nous pouvions les pseudo-trianguler de manière à rendre le nombre d'événements de simulation dans la file d'attente aussi indépendant que possible de la complexité combinatoire des polytopes. Cependant, nous n'avons pas proposé d'algorithme ni pour la construction des pseudo-triangulations, ni pour leur maintenance. C'est un sujet intéressant qu'il conviendrai d'approfondir.

Chapitre 6

Localisation d'un point dans une décomposition verticale cinétique de segments disjoints

Dans l'algorithme de construction de la décomposition radiale 3D présenté au Chapitre 5, il est nécessaire de maintenir une décomposition radiale 2D *dessinée* sur un plan Π tournant autour d'un axe fixe. Les polytopes disposés dans l'espace 3D induisent la présence, sur Π, d'un ensemble de segments disjoints et mobiles, dont on souhaite calculer et maintenir la décomposition radiale 2D. Cette décomposition 2D est l'intersection de la décomposition radiale 3D avec Π. Une décomposition radiale 2D est complètement analogue à une décomposition verticale (ou trapézoïdale) du plan. Nous présentons dans ce chapitre une structure de données cinétique capable de maintenir une telle décomposition verticale (ou radiale) et de répondre rapidement à des requêtes de localisation du trapèze contenant un point donné.

Nous nous intéressons donc au problème de localisation suivant : étant donnée une décomposition verticale [1] $\mathcal{V}$ d'un ensemble $\mathcal{S}$ de segments disjoints, et un point P, trouver dans quelle région de $\mathcal{V}$ se trouve P. Les segments sont, de plus, autorisés à se mouvoir continûment (tout en restant disjoints) et à apparaître ou disparaître quand leur longueur devient nulle. Notre structure de données doit donc être « cinétique ».

La structure de données que nous utiliserons est celle de Goodrich et Tamassia [62]. Elle a été créée pour maintenir une subdivision monotone du plan de manière dynamique. Nous décrivons dans ce chapitre les modifications que nous avons apportées pour en faire une structure de données *cinétique* de maintenance de la décomposition radiale d'un ensemble de segments disjoints et mobiles.

Nous donnons d'abord quelques résultats relatifs à la construction et la mise à jour d'une décomposition verticale dans un cadre dynamique en-ligne, et à une structure de localisation dynamique plus générale.

6.1 Décomposition verticale et localisation

Soit $\mathcal{S}$ un ensemble de n segments finis du plan. Sans perte de généralité, on supposera qu'aucun segment de $\mathcal{S}$ n'est vertical. La décomposition verticale $\mathcal{V}$ de $\mathcal{S}$ est

[1] En 2D, la décomposition verticale d'un ensemble de segments est généralement appelée *décomposition trapézoïdale*.

une partition du plan en régions convexes de complexité constante. Elle est obtenue en traçant, depuis chaque sommet de segment et chaque intersection de deux segments, deux « murs » verticaux l'un dirigé vers le haut, l'autre vers le bas. Ces murs s'étendent jusqu'au premier segment rencontré (Figure 6.1). Soit k le nombre d'intersections entre deux segments. Il apparaît clairement que la taille de $\mathcal{V}$ est en $O(n+k)$ ($0 \leq k \leq n(n-1)$).

On peut construire $\mathcal{V}$ de manière incrémentale randomisée en temps espéré $O(n \log n + k)$ [107] et augmentée (dans le même temps) d'une structure de données permettant la localisation d'un point en temps $O(\log n)$. Dans la même référence, on peut trouver une version *en-ligne* et complètement dynamique s'exécutant en temps $O(n \log n + k \log n)$ pour presque toute séquence de mise à jour de la décomposition.

Ces algorithmes randomisés ne semblent pas évidents à « cinétiser », c'est-à-dire à adapter au cas de segments bougeant continûment. Il existe (entre autres !) une autre structure plus générale et un peu moins efficace qui se prête cependant plus facilement à une « cinétisation ».

6.1.1 La structure de Goodrich et Tamassia

Il s'agit d'une structure de données proposée par Tamassia et Goodrich [62], qui permet de maintenir une structure de localisation d'un point dans une partition monotone $\mathcal{P}$ du plan. En particulier, l'insertion ou la suppression d'un sommet ou d'une arête dans cette structure s'effectue en temps $O(\log n)$, les requêtes de localisation s'exécutent en temps $O(\log^2 n)$ et la structure occupe une taille linéaire en celle de la subdivision $\mathcal{P}$. Nous donnons quelques définitions avant d'attaquer à la maintenance cinétique de $\mathcal{V}$.

Monotonie. Une subdivision polygonale du plan est monotone si chacune de ses faces est monotone vis-à-vis de l'axe y. Une face f (un polygone simple) est monotone vis-à-vis de l'axe y si toute droite horizontale (parallèle à l'axe x) intersecte f en au plus un segment (connexe). Par exemple, une triangulation d'un nombre fini de points est une subdivision monotone (en excluant la face infinie), puisqu'un triangle est convexe.

Arbre couvrant. En supposant les arêtes de $\mathcal{P}$ orientées vers le bas, un arbre couvrant T est construit sur $\mathcal{P}$ en choisissant pour chaque sommet v de $\mathcal{P}$ une arête sortante (pointant vers le bas, par rapport au sommet v). Un arbre dual D est défini dans le graphe dual de $\mathcal{P}$ (dans lequel chaque nœud correspond à une face de $\mathcal{P}$). Il est composé des arêtes e^* duales des arêtes $e \in \mathcal{P}$ ne faisant pas partie de l'arbre couvrant T.

Subdivision annexe $\mathcal{P}'$. La seule hypothèse sur $\mathcal{P}$ est sa monotonicité. Le nombre de faces voisines d'une face donnée n'est pas borné. Il s'ensuit que l'arbre dual D n'est pas de degré borné (la valence de ses nœuds n'est pas bornée). Pour y remédier, Goodrich et Tamassia raffinent $\mathcal{P}$ pour obtenir une nouvelle subdivision de taille $O(n)$ dans laquelle un arbre couvrant T spécifique est défini de telle manière que l'arbre dual D soit au plus de degré 3, donc borné. C'est une propriété qui sera exploitée pour obtenir une localisation rapide dans $\mathcal{P}$.

Nous avons assez de détails pour décrire la « cinétisation » de la décomposition verticale $\mathcal{V}$. Nous décrirons donc plus tard comment s'effectue la localisation d'un point dans $\mathcal{V}$ ou $\mathcal{P}$.

6.2 Maintenance cinétique de $\mathcal{V}$

Chaque face de la décomposition verticale est convexe, donc $\mathcal{V}$ est une subdivision monotone du plan. Nous pouvons donc penser appliquer la structure de recherche dynamique de Goodrich et Tammasia [62] pour mettre à jour la structure de recherche lorsqu'un segment apparaît ou disparaît et quand $\mathcal{V}$ subit un changement combinatoire.

Au lieu de raffiner la décomposition comme dans la structure originelle, nous tirons profit de la « verticalité » de notre décomposition $\mathcal{V}$ pour garantir que l'arbre dual D soit de degré borné (au plus 3).

6.2.1 Construction de la structure de localisation

Étant donnée la décomposition verticale $\mathcal{V}$, nous définissons une *forêt* T couvrant les sommets de $\mathcal{V}$ comme suit : chaque arête de $\mathcal{V}$ qui est incluse dans un segment de $\mathcal{S}$ est une arête de T (orientée de droite à gauche). Ensuite, nous ajoutons à T les n arêtes descendant verticalement du sommet droit de chaque segment de $\mathcal{S}$, en les orientant de bas en haut, voir Figure 6.1a-b. Nous construisons un *arbre dual* D comme suit. Les nœuds v_i^* de D sont en bijection avec les trapèzes t_i de $\mathcal{V}$. Les arêtes de D sont duales des arêtes de $\mathcal{V}$. L'arête duale e^* entre les nœuds v_i^* et v_j^* apparaît dans D si l'arête séparant les trapèzes t_i et t_j existe et n'est pas dans la forêt couvrante T, voir Figure 6.1c.

La forêt T et l'arbre D possèdent plusieurs propriétés qui nous permettront de localiser le trapèze contenant un point donné en temps $O(\log^2 n)$:

- Chaque arête e^* de D définit un unique cycle C_{e^*} dans la forêt T à laquelle on ajoute l'arête e. Ce cycle est la frontière d'un polygone monotone vis-à-vis de la direction horizontale : C_{e^*} est donc un polygone formé de deux chaînes d'arêtes, une supérieure et une inférieure (en ignorant les arêtes extrêmes verticales, à gauche et à droite). La Figure 6.1d montre un tel polygone. C_{e^*} est défini ainsi : soit v_{up} et v_{down} les deux sommets de l'arête e duale de e^*. v_{up} et v_{down} appartiennent respectivement à deux arbres T_1 et T_2 de la forêt T (éventuellement le même arbre). Notons $expose(v)$ le chemin dans un arbre entre le nœud v et la racine de l'arbre. $expose(v_{up})$ et $expose(v_{down})$ forment deux chaînes monotones géométriquement, et pouvant posséder un suffixe commun combinatoirement. Les deux préfixes distincts de $expose(v_{up})$ et $expose(v_{down})$, respectivement $up(C_{e^*})$ et $down(C_{e^*})$, sont deux lignes polygonales monotones qui bordent le polygone C_{e^*} par le haut et par le bas respectivement. Notons qu'une partie de ces chaînes peut être définie seulement implicitement lorsqu'elle se trouve en $y = \pm\infty$. C'est notamment le cas pour la chaîne $down(C_{e^*})$ de la Figure 6.1d, dont le dernier segment (en $y = -\infty$) n'est défini qu'implicitement.
- Bien que la décomposition verticale $\mathcal{V}$ ne soit pas de degré borné (un trapèze peut être adjacent à un nombre quelconque d'autres trapèzes), nous avons pris soin de définir la forêt T de telle sorte que l'arbre dual D soit de degré borné

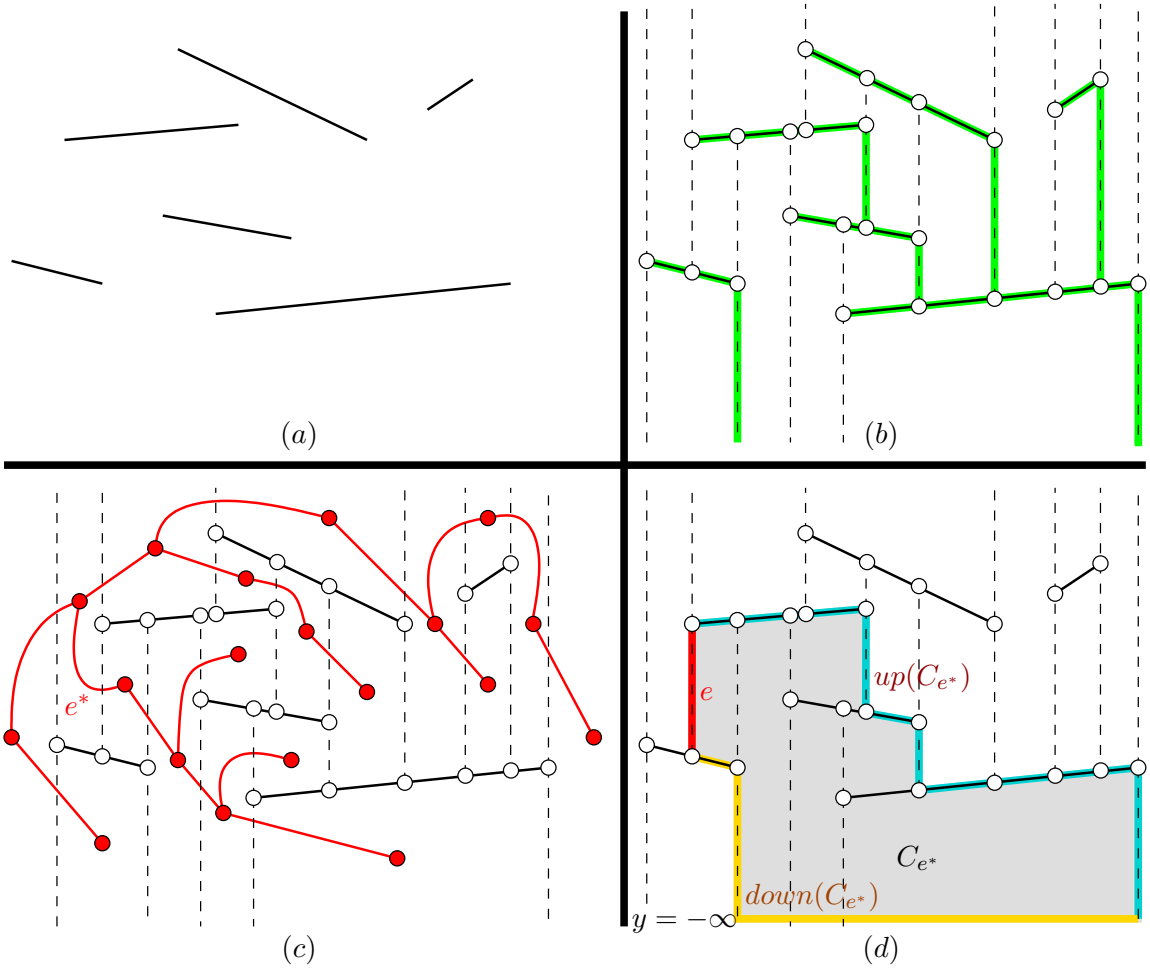


FIG. 6.1. (a), l'ensemble $\mathcal{S}$ de segments dans le plan. **(b)**, la décomposition verticale du plan, en noir, et la forêt couvrante T, en vert. **(c)**, l'arbre dual D, en rouge. **(d)**, le polygone monotone C_{e^*} associé à l'arête e^* de D.

(et au plus 3). Cette propriété est vérifiée puisque chaque arête de $\mathcal{V}$ contenue dans un segment de $\mathcal{S}$ appartient à la forêt T, aussi, un nœud de D ne peut être connecté (dans D) à un autre nœud que via l'arête duale d'un mur vertical. Or, un trapèze possède au plus 4 murs verticaux sur sa frontière [36, Chapitre 6]. Notre définition de T limite donc à 3 le degré de chaque nœud de D.

Le degré borné de D permet de former une *décomposition centroïdale* de D. Considérons un arbre D de taille n sans racine, et de degré borné. Une *arête centroïdale* de D coupe, si on la supprime, l'arbre D en deux sous-arbres de taille au plus $\frac{2n}{3} + 1$. Dès que la taille de D est plus grande que 1, il existe toujours une arête centroïdale. On peut donc décomposer en temps linéaire un arbre de degré borné (sans racine) en un *arbre de décomposition centroïdale*, de hauteur $O(\log n)$ dont les nœuds internes sont des arêtes de D et les feuilles sont les nœuds de D. La Figure 6.2 montre un arbre de degré 3 et une de ses décompositions centroïdales possibles. Une décomposition centroïdale peut être construite en temps linéaire [24]. Cependant, dans l'algorithme de Goodrich et Tamassia, cette décomposition est stockée de manière implicite dans la structure d'arbre utilisée, qui permet de trouver une arête centroïdale en temps $O(\log n)$.

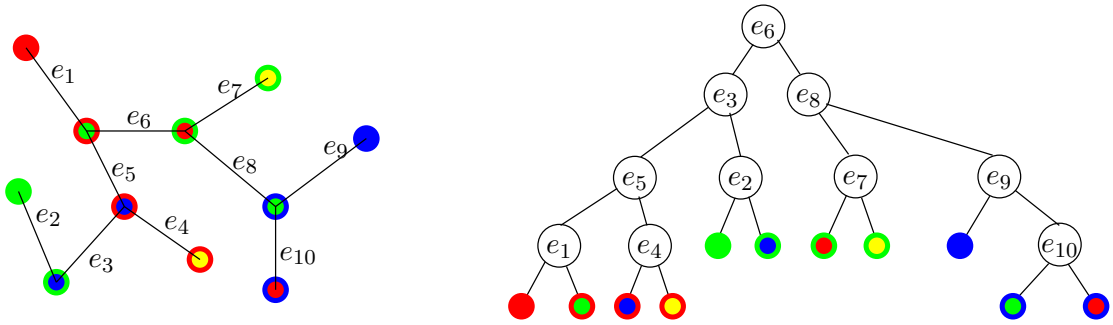


FIG. 6.2. (a), un arbre de degré borné, D. **(b)**, une décomposition centroïdale possible de D.

Représentation implicite avec des arbres dynamiques. La grande astuce que proposent Goodrich et Tamassia [62] est de représenter la forêt T [2] et l'arbre dual D à l'aide de la structure de données *link-cut trees* de Sleator et Tarjan [51, 132]. Cette dernière permet, entre autre

- de calculer une arête centroïdale d'un arbre de taille > 1 en temps $O(\log n)$ [62].
- de récupérer un arbre binaire représentant la chaîne $expose(v)$ pour un nœud v d'un arbre, en temps $O(\log n)$ [132].

Nous référons le lecteur aux deux articles sus-mentionnés pour de plus amples détails.

6.2.2 Requête de localisation d'un point

Nous avons maintenant tous les outils pour localiser le trapèze dans lequel se trouve un point « requête » P. La recherche est récursive.

1. Si D est une feuille (et correspond donc à un trapèze de $\mathcal{V}$), on vérifie si P est dans D en temps constant.
2. Sinon, trouver une arête centroïdale e^* de l'arbre D (en temps $O(\log n)$). e^* sépare D en deux sous-arbres D_{in} et D_{out}. D_{in} est le sous-arbre localement à droite de l'arête e^*. D_{out} est localement à gauche.
3. Calculer les deux arbres binaires représentant les chaînes supérieure et inférieure du polygone monotone C_{e^*} ($O(\log n)$ à l'aide de la fonction $expose(e^*)$ de [132]). Notons que C_{e^*} est le contour de l'union des trapèzes qui sont des feuilles de D_{in}.
4. Vérifier si P est dans C_{e^*} ($O(\log n)$).
   - si oui, itérer la recherche dans D_{in}.
   - si non, itérer la recherche dans D_{out}.

La recherche d'une arête centroïdale garantit que l'on explorera au plus $O(\log n)$ polygones monotones du type C_{e^*}. La recherche s'effectue donc en temps $O(\log^2 n)$.

6.2.3 Maintenance cinétique

La maintenance cinétique de la décomposition verticale est développée ici en vue de son intégration dans la construction d'une « décomposition radiale 3D » (voir cha-

[2] Dans leur article, T est un arbre car toutes les branches descendent vers une racine commune qui est le point le plus bas de la subdivision monotone (vis-à-vis de la direction verticale). Pour notre décomposition verticale, une forêt est plus adaptée, mais on pourrait en faire un arbre à l'aide d'une branche horizontale allant de $(+\infty, -\infty)$ à $(-\infty, -\infty)$.

pitre 5). Nous faisons donc l'hypothèse que les segments de $\mathcal{S}$ ne s'intersectent et ne se touchent jamais. De plus, dans l'application qui nous intéresse, les segments ne passent jamais par la verticale,[3] mais nous traiterons quand même ce cas puisqu'il n'est pas compliqué à gérer.

Les événements qui modifient $\mathcal{V}$, T et D sont donc

- l'apparition d'un segment de longueur nulle et croissante ;
- la disparition d'un segment après qu'il ait rétréci jusqu'à une longueur nulle ;
- le croisement de deux murs verticaux bordant un même trapèze (en excluant le cas ci-dessus).

Soit n_t le nombre de segments présents dans le plan à un instant t. $\mathcal{V}$ comporte $O(n_t)$ trapèzes et chacun donne lieu à 0 ou 1 certificat indiquant la date d'« écrasement » du trapèze. La file d'attente des événements à traiter contient donc $O(n_t + N_a)$ événements, où N_a est le nombre d'événements du type *apparition d'un nouveau segment*. Les apparitions et disparitions de segments sont des événements en nombre a priori inconnu.

En supposant connu le mouvement de chaque extrémité de segment, et exprimé sous forme d'expression algébrique de degré borné (par une constante), le nombre total d'événements pouvant advenir est en $O(n^2)$ où n est le nombre total de segments mis en jeu au cours de la simulation. Cette borne provient du fait qu'un événement met en jeu au plus deux extrémités de segments.

Apparition et disparition

Les événements de type *apparition* ou *disparition* d'un segment sont simples à gérer. La décomposition verticale est mise à jour en temps constant (3 trapèzes sont supprimés ou créés et un trapèze voit sa combinatoire modifiée). Un nombre constant de certificats doivent être supprimés ou ajoutés à la file des événements futurs, ce qui se fait en temps $O(\log n)$.

Notons enfin que les opérations de maintenance de D et T (indiquées schématiquement sur la Figure 6.3(a)) sont en nombre constant et s'effectuent en temps $O(\log n)$. Nous référons le lecteur aux articles [51, 62, 132] pour de plus amples détails.

Écrasement d'un trapèze

Lors de l'écrasement d'un trapèze, il n'y a pas changement du nombre total de trapèzes, mais seulement leur réorganisation (voir Figure 6.3(b)). Notons simplement qu'il peut y avoir une apparition d'un nouvel arbre dans la forêt T ou bien la « fusion » de deux arbres en un. Ces opérations s'effectuent toujours en temps $O(\log n)$.

Passage d'un segment de $\mathcal{S}$ par la verticale

Cet événement est très simple à gérer puisqu'il suffit d'échanger les coordonnées des deux sommets du segment.

[3] Cela correspondrait à la présence, dans la scène, d'un polytope convexe (3D) de volume nul.

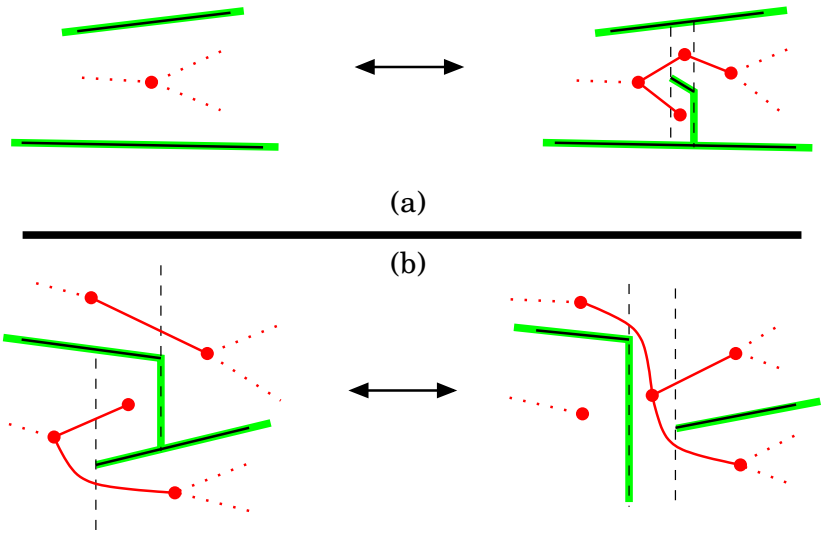


FIG. 6.3. **(a)**, mise à jour de D et T lors de l'apparition ou la disparition d'un segment. **(b)**, mise à jour de D et T lors de l'écrasement d'un trapèze.

6.3 Décomposition radiale

La construction de la décomposition radiale 3D met en jeu un plan de balayage de la scène, tournant autour d'un axe fixe. Nous maintenons dans ce plan une décomposition *radiale* 2D, qui diffère d'une décomposition verticale du plan en deux points : premièrement les murs verticaux sont remplacés par des murs « radials », alignés avec l'origine d'un repère du plan. On peut voir la décomposition radiale à peu près comme une décomposition verticale dans le système de coordonnées (θ, r) au lieu de (x, y). Deuxièmement, la décomposition radiale est *cyclique* ; topologiquement équivalente à une décomposition verticale dessinée à la surface d'un cylindre vertical.

En reprenant la définition de T et D décrite plus haut, la cyclicité de la décomposition radiale 2D rend le *graphe* dual D également cyclique. D n'est donc plus un arbre et peut même comporter plusieurs composantes connexes, voir Figure 6.5.

Pour rompre cette cyclicité, nous considérons le rayon r vertical ayant la même origine que la décomposition radiale. La forêt T est coupée par r et le graphe dual redéfini comme indiqué sur la Figure 6.4. La maintenance des cellules découpées par le rayon r est effectué de même que décrit précédemment. De plus le nombre de nœuds dans le graphe dual D est au plus triplé, et D reste de degré égal à 3. La Figure 6.5 montre comment la coupre est effectuée sur un exemple de décomposition radiale.

Nous résumons la discussion ci-dessus en un lemme pour pouvoir s'y référer plus aisément.

Lemme 6.1. *Il existe une structure de données cinétique permettant la maintenance de la décomposition radiale d'un ensemble de segments disjoints et mobiles. Cette structure est de taille linéaire en le nombre de segments. Elle supporte la localisation d'un point du plan en temps $O(\log^2 n)$, ainsi que l'insertion et l'extraction d'un segment de longueur nulle en temps $O(\log n)$. Cette structure de données cinétique est répondante, efficace, compacte et locale.*

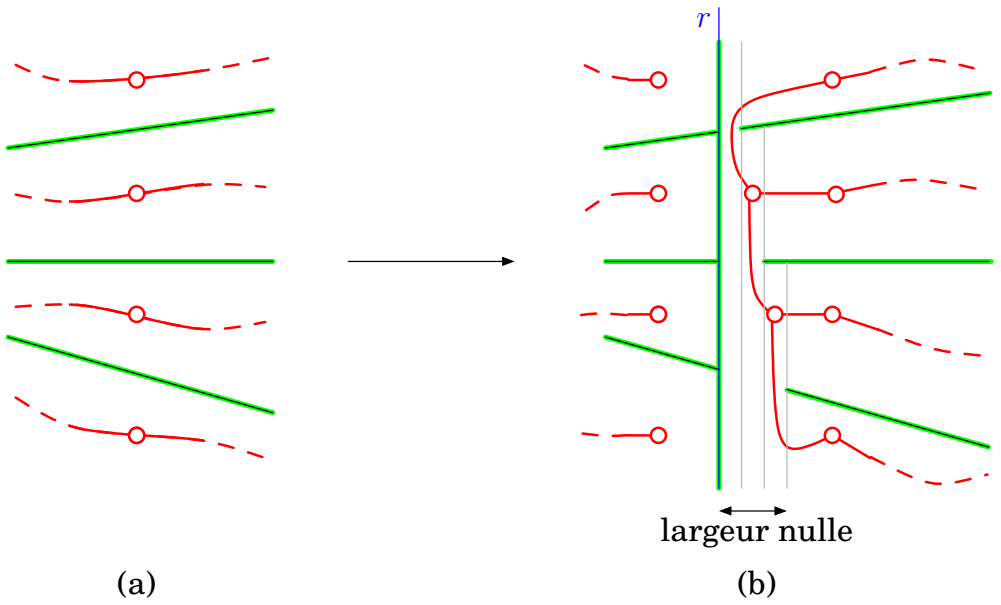


FIG. 6.4. (a), l'ensemble des segments du plan qui sont en intersection avec le rayon r. Dans ce scéma non radial (mais vertical) le rayon r est placé à une coordonnée x arbitraire. **(b)**, la nouvelle forêt T et le nouveau graphe dual D. Ce dernier voit son nombre de nœuds au plus triplé tandis que son acyclicité est rompue. D ne contient plus qu'une composante connexe.

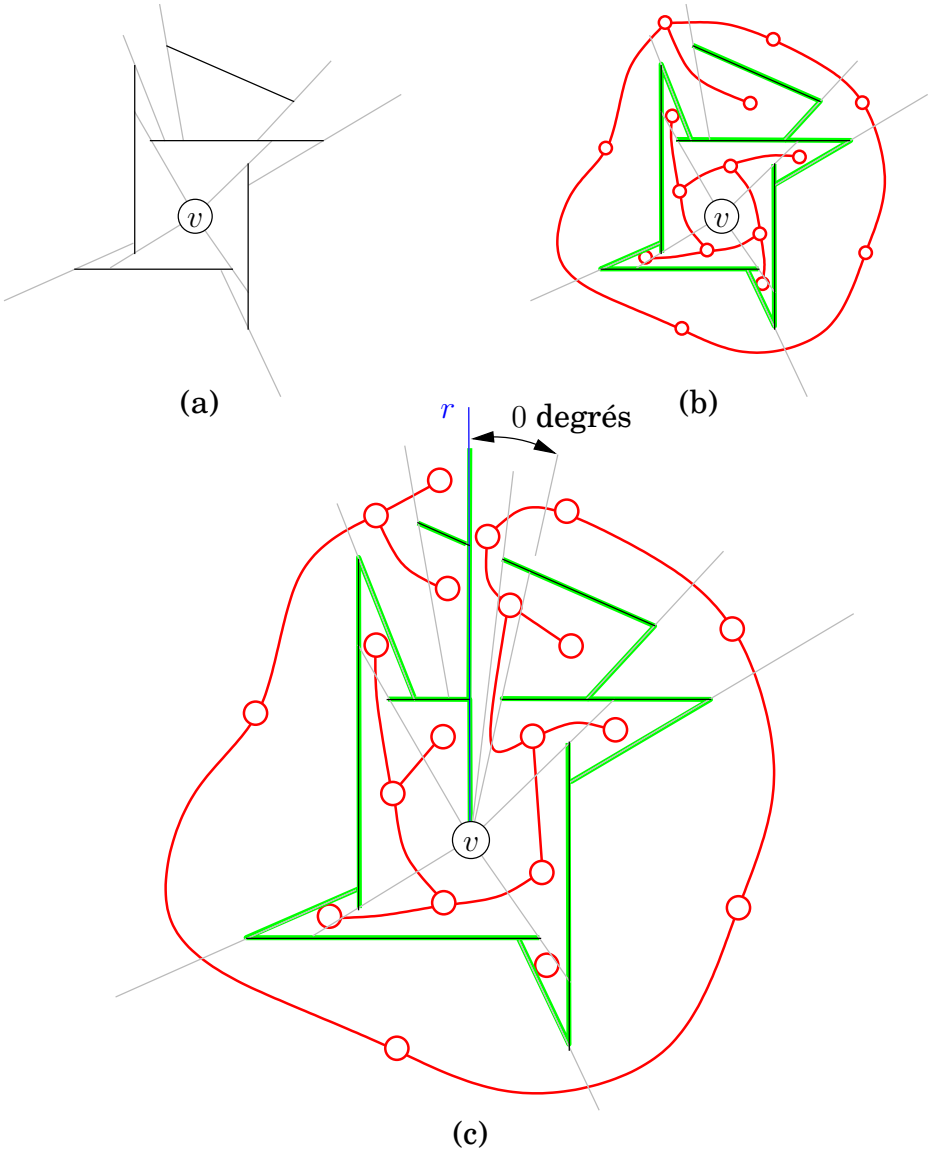


FIG. 6.5. (a), un ensemble de segments dans le plan, et la décomposition radiale vis-à-vis du « point de vue » v. **(b)**, *en vert*, la forêt T ; *en rouge*, le dual D n'est plus un arbre. **(c)**, La coupure de la décomposition radiale induite par le rayon r permet de retrouver un graph dual D de degré 3.

Chapitre 7

Construction du complexe de visibilité de polytopes disjoints

Dans ce chapitre, nous nous intéressons au problème de la construction du complexe de visibilité d'un ensemble de polytopes disjoints (un polytope est un polyèdre intersection finie de demi-espaces). En 2D, le *Greedy Flip Algorithm* [118] effectuant la construction du complexe de visibilité d'un ensemble de convexes dans le plan (cercles, ellipses, polygones) a été implémenté par Pierre Angelier [7] dans le cadre de la librairie C++ CGAL [22]. Cette implémentation a permi de mesurer expérimentalement sa complexité pour des scènes d'objets uniformément répartis et de densité variable [53]. Cette étude montre que la taille du complexe de visibilité dans le plan est raisonnable en pratique (linéaire avec une faible constante) et encourage donc l'étude pratique de la taille du complexe en dimension 3. Une telle expérience devrait montrer si l'utilisation du complexe 3D serait envisageable comme préalable à la résolution d'autres problèmes (calcul de facteurs de forme, *ray-tracing*, *etc.*). Cependant, aucune implémentation de la construction du complexe 3D existe à ce jour (et nous ne faisons pas exception). Nous proposons dans ce chapitre un algorithme de construction du complexe de visibilité d'un ensemble de polytopes disjoints, rendu possible grâce à une propriété de connexité de certaines parties du complexe, que nous démontrons également dans ce chapitre.

Ces travaux ont commencé lors d'une visite au MIT, en collaboration avec Frédo Durand et Florent Duguet. Ils ont été poursuivis en collaboration avec Maxime Wolff.

Taille théorique du complexe de visibilité 3D

La borne la plus simple que l'on puisse donner pour une scène de taille n (polyédrique avec n arêtes par exemple, ou n sphères) est $O(n^4)$. C'est évidemment une borne un peu intimidante pour qui voudrait se lancer dans une implémentation de la construction du complexe. Aussi, on s'est attaché à réduire la taille théorique du complexe de visibilité pour des scènes un peu plus « réalistes ». Durand [46] montre que la taille du complexe d'un ensemble de n objets de taille bornée répartis uniformément dans une sphère est $O(n^{8/3})$. Il étaye ce résultat par quelques expérimentations montrant une variation presque quadratique du nombre de quadritangentes dans une scène polygonale.

Plus récemment, Devillers *et al.* ont montré que la complexité em moyenne du complexe est linéraire pour des objets répartis uniformément (sphères unités, polytopes épais) [38]. Brönnimann *et al.* [18] ont également montré que la taille dans le cas le

pire du complexe de visibilité d'un ensemble de k polytopes quelconques de complexité totale n est $O(n^2k^2)$. Ces résultats importants incitent également à implémenter effectivement la construction du complexe pour en étudier la taille expérimentalement.

L'ensemble des droites de $\mathbb{R}^3$ qui ne traversent aucun objet est l'union de toutes les 4-cellules du complexe de visibilité dont les bloqueurs sont (∞, ∞). Agarwal *et al.* [1] ont montré que la complexité combinatoire de cet ensemble (l'ensemble des *droites libres*) est en $O(n^{3+\varepsilon})$ pour tout $\varepsilon > 0$, si les objets sont des boules unités.

Algorithmes de construction du complexe 3D

Durand [46] propose un algorithme par double balayage au cours duquel le complexe de visibilité 2D est maintenu dans une coupe planaire de la scène. L'algorithme n'est présenté que très succintement, et permet de construire le complexe en temps $O((q + n^3) \log n)$, où q est le nombre de 0-cellules du complexe (le nombre de quadritangentes).

Goaoc [60] propose un algorithme de construction du complexe pour des objets représentés sous forme d'ensembles semi-algébriques. Un ensemble semi-algébrique a la propriété intéressante que l'ensemble de ses droites perçantes forme également un ensemble semi-algébrique exprimé à l'aide des coordonnées de Plücker. Cet algorithme fait appel à une « boîte noire » effectuant les calculs algébriques sur les ensembles de droites perçant un objet. Il est intéressant du fait de sa simplicité conceptuelle. En revanche, certaines opérations algébriques utilisées ne sont pas inplémentées et empêchent, provisoirement, sa réalisation pratique.

Ici, nous abordons le problème différemment. De manière analogue à la représentation d'un objet 3D par sa frontière (B-rep, utilisé à peu près dans tous les domaines liés au virtuel), nous souhaitons représenter une 4-cellule du complexe par le treillis d'adjacence des faces de sa frontière (des 3-, 2-, 1-, et 0-cellules). Nous rappelons qu'une 4-cellule du complexe est connexe par définition. Nous commençons, dans la section suivante, par montrer que la frontière (un ensemble de 3-cellules) d'une 4-cellule est elle-même connexe. Ainsi, une 4-cellule ne peut pas contenir de « cavité » interne (Figure 7.1). La description d'une 4-cellule peut donc être donnée simplement par le treillis d'adjacence des 3-cellules de sa frontière, sans se préoccuper de distinguer plusieurs composantes disjointes : construire l'ensemble des cellules de dimension inférieure ou égal à 3 (le 3-squelette) sera donc suffisant pour extraire la description de chaque 4-cellule du complexe de visibilité. Nous montrons ensuite comment construire le 3-squelette du complexe de visibilité d'un ensemble fini de polytopes disjoints, en promenant un « point de vue » le long de toutes les arêtes de ces polytopes.

7.1 Le complexe de visibilité 3D

Nous définissons un *oculteur* comme un sous-ensemble de $\mathbb{R}^3$ ouvert et convexe. Soit $\mathcal{O}$ un ensemble fini et non vide d'oculteurs disjoints dans l'espace. L'*espace libre* $\mathcal{F}$ est le complémentaire de l'union des oculteurs.

$$\mathcal{F} = \mathbb{R}^3 \setminus \bigcup_{o \in \mathcal{O}} o$$

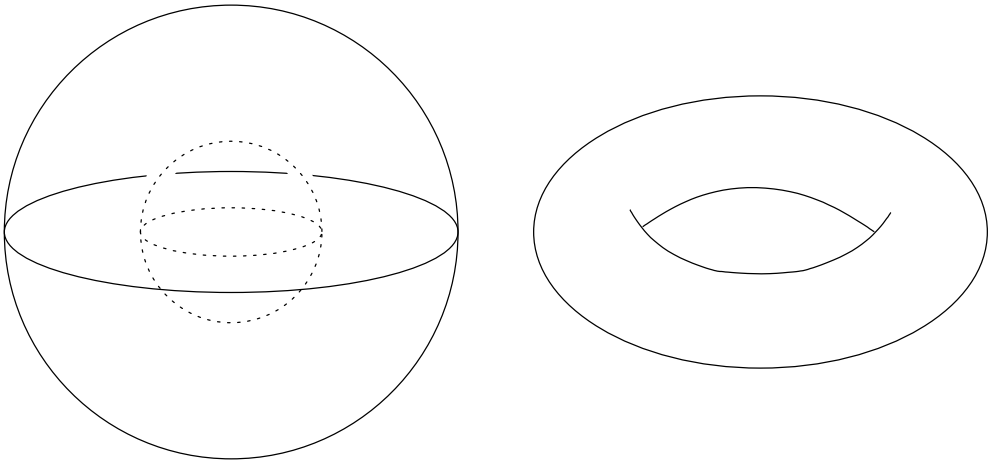


FIG. 7.1. À gauche, une boule B (pleine) comprenant une cavité interne : B est connexe, mais sa frontière ne l'est pas. **À droite**, un tore plein est connexe et de frontière connexe.

Un *segment libre maximal* s est un segment maximal contenu dans l'espace libre : s ne peut pas être étendu à une de ses extrémités sans intersecter un oculteur. Pour pouvoir définir proprement les segments libres maximaux, nous exigeons que la distance entre deux oculteurs soit strictement positive. Ainsi, tout segment libre maximal a une longueur strictement positive, et pour chaque segment libre maximal s, il existe une unique paire $\mathrm{vis}(s) = (A, B) \in (\mathcal{O} \cup \{\infty\})^2$ telle que les extrémités de s soient respectivement sur ∂A et ∂B. Nous appelons cette paire l'*ensemble visible de* s. En général, A peut être égal à B, mais ce n'est pas vrai si les oculteurs sont convexes (sauf pour ∞).

Soit $\mathcal{S}$ l'ensemble des segments libres maximaux. La fonction

$$vis : \mathcal{S} \rightarrow (\mathcal{O} \cup \{\infty\})^2$$

définit une partition de $\mathcal{S}$ en classes d'équivalence dans chacune desquelles deux segments sont équivalents s'ils « voient » les mêmes oculteurs à leurs extrémités. Nous raffinons un peu plus ces classes d'équivalence en en séparant les composantes maximalement connexes. Chaque composante maximalement connexe constitue alors une *4-cellule du complexe de visibilité* $\mathcal{VC}$. $\mathcal{VC}$ est la partition de $\mathcal{S}$ en ces composantes maximalement connexes.

7.1.1 Définition d'une topologie sur $\mathcal{S}$

En s'inspirant de la définition du complexe de visibilité 2D [119] nous redéfinissons l'ensemble $\mathcal{S}$ des segments libres maximaux comme le quotient par une relation d'équivalence du produit de deux espaces topologiques simples. Il en découle une topologie naturelle sur le complexe de visibilité.

Un segment libre maximal s peut être décrit à l'aide d'une paire (x_s, ω_s) où $x_s \in \mathcal{F}$ est un point de s, et $\omega_s \in \mathbb{RP}^2$ est la direction de s. Nous définissons

$$\overline{\mathcal{S}} \equiv \mathcal{F} \times \mathbb{RP}^2.$$

Clairement, tout élément de $\overline{\mathcal{S}}$ définit un unique segment libre maximal. Cependant, cette application n'est pas bijective. Nous introduisons donc la relation d'équivalence

$\approx$:

$$\forall (x_1,\omega_1),(x_2,\omega_2) \in \overline{\mathcal{S}}, x \approx y \Leftrightarrow \left\{ \begin{array}{ll} & \omega_1 = \omega_2 = \omega_{x_1x_2}, \\ \text{et} & [x_1,x_2] \in \mathcal{F} \end{array} \right.$$

où $\omega_{x_1x_2}$ est la direction (dans $\mathbb{RP}^2$) du vecteur $\overrightarrow{x_1x_2}$. Nous définissons alors $\mathcal{S}$ comme le quotient de $\overline{\mathcal{S}}$ par $\approx$:

$$\mathcal{S} \equiv \overline{\mathcal{S}}/\approx$$

La topologie de $\mathcal{F}$ est induite par celle de $\mathbb{R}^3$: les ouverts de $\mathcal{F}$ sont les intersections d'un ouvert de $\mathbb{R}^3$ avec $\mathcal{F}$. $\overline{\mathcal{S}}$ est muni de la topologie produit, induite par les topologies de $\mathcal{F}$ et $\mathbb{RP}^2$: un ouvert de $\overline{\mathcal{S}}$ est l'union de produits cartésiens d'un ouvert de $\mathcal{F}$ et d'un ouvert de $\mathbb{RP}^2$. Enfin, la topologie de $\mathcal{S}$ est la topologie quotient induite par la relation d'équivalence $\approx$: un sous-ensemble A de $\mathcal{S}$ est ouvert si la préimage de A par la surjection canonique $\Pi : \overline{\mathcal{S}} \to \mathcal{S} = \overline{\mathcal{S}}/\approx$ est un ouvert de $\overline{\mathcal{S}}$.

Lemme 7.1. *$\mathcal{S}$ muni de cette topologie quotient, est un espace séparé.*

Preuve : Par définition, un espace topologique est séparé si deux éléments distincts admettent deux voisinages respectifs dont l'intersection est nulle.

Nous commençons par définir des familles de voisinages ouverts $S(s,x,r,\alpha)$ d'un segment libre maximal s, où x est un point du segment s, r est un réel positif et α est un réel positif inférieur à π. Soit $B(x,r) \subset \mathcal{F}$ l'intersection de la boule ouverte de centre x et de rayon r avec l'espace libre. $B(x,r)$ est non vide car $x \in \mathcal{F}$ et est un ouvert de $\mathcal{F}$. Soit $\mathcal{B}(\omega_s,\alpha)$ l'ensemble des directions dont l'angle formé avec ω_s est inférieur à α. (Rappellons que $\omega_s \in \mathbb{RP}^2$ est la direction du segment s.) $\mathcal{B}(\omega_s,\alpha)$ est un ouvert de $\mathbb{RP}^2$. Nous définissons

$$S(s,x,r,\alpha) = \Pi(B(x,r) \times \mathcal{B}(\omega_s,\alpha)).$$

$S(s,x,r,\alpha)$ est simplement l'ensemble des segments libres maximaux passant par $B(x,r)$ et ayant une direction dans $\mathcal{B}(\omega_s,\alpha)$. Vu comme un ensemble de points dans $\mathcal{F}$, $S(s,x,r,\alpha)$ à la forme d'un sablier.

Remarque. Si r est assez grand, $B(x,r)$ peut avoir la forme d'une boule « mangée » par certains oculteurs de la scène. $S(s,x,r,\alpha)$ peut même comporter plusieurs composantes connexes.

Montrons maintenant que $S(s,x,r,\alpha)$ est un ouvert. Pour ce faire, il faut montrer que $X \equiv \Pi^{-1}(S(s,x,r,\alpha))$ est un ouvert de $\overline{\mathcal{S}} = \mathcal{F} \times \mathbb{RP}^2$. Nous allons montrer que chaque élément $(y \in \mathcal{F}, u \in \mathbb{RP}^2)$ contenu dans X, admet un voisinage $W(y)$ ouvert et contenu dans X. X sera alors l'union de ces voisinages pour tous les éléments de X, preuve que X est un ouvert de $\overline{\mathcal{S}}$.

Soit $(y,u) \in X$.

- Si $y \in B(x,r)$. Soit $B(y,\rho)$ une boule contenue dans $B(x,r)$. $W(y) \equiv B(y,\rho) \times \mathcal{B}(\omega_s,\alpha)$ est un ouvert de $\overline{\mathcal{S}}$ contenu dans X.
- Si $y \notin B(x,r)$. Il existe un point $c \in B(x,r)$ appartenant au segment libre $\Pi((y,u))$, et une boule ouverte $B(c,\rho) \subset B(x,r)$. À partir de y, faisons glisser un point $p \in \mathbb{R}^3$ jusqu'à ce que l'ensemble des directions prises par les droites reliant p à $B(c,\rho)$ forment une boule ouverte $\mathcal{B}(u,\beta)$ contenue dans $\mathcal{B}(\omega_s,\alpha)$ (une telle position de p existe car $\mathcal{B}(\omega_s,\alpha)$ est un ouvert et $u \in \mathcal{B}(\omega_s,\alpha)$. $\Pi(\{p\} \times \mathcal{B}(u,\beta))$ forme un cône de sommet p et de section circulaire. Il existe donc une boule ouverte $B(y,\rho')$ épousant complétement le bord de ce cône. Il est facile de vérifier que $W(y) \equiv B(y,\rho') \times \mathcal{B}(u,\beta)$ est un voisinage ouvert de (y,u) contenu dans X.

Nous avons donc démontré que $S(s,x,r,\alpha)$ est bien un ouvert de $\mathcal{S}$.

Venons en à la preuve du lemme. Posons $\Pi_{\mathcal{F}} : (y,u) \mapsto y$ la projection canonique de $\overline{\mathcal{S}}$ sur $\mathcal{F}$. Soit s et s' deux segments libres maximaux distincts. Il existe deux point $x \in s$ et $x' \in s'$ distincts, et $\epsilon > 0$ tel que $B(x,\epsilon) \cap B(x',\epsilon) = \emptyset$ (on prendra par exemple, $\epsilon = \frac{||x-x'||}{10}$).Il existe $\alpha > 0$ suffisament petit pour que $\Pi_{\mathcal{F}}(S(s,x,\epsilon,\alpha))$ n'intersecte pas $B(x',\epsilon)$, et $\alpha' > 0$ suffisament petit pour que $\Pi_{\mathcal{F}}(S(s',x',\epsilon,\alpha'))$ n'intersecte pas $B(x,\epsilon)$. Il s'ensuit que l'intersection de $S(s,x,\epsilon,\alpha)$ et $S(s',x',\epsilon,\alpha')$ est vide. Conclusion, $\mathcal{S}$ est un espace séparé.

Remarque. Notons que si s et s' ont la même droite support, on prendra ϵ suffisament petit pour que tous les segments reliant $B(x,\epsilon)$ à $B(x',\epsilon)$ intersectent l'intérieur du bloqueur commun de s et s'. □

7.1.2 $\mathcal{S}$ est un complexe cellulaire (un CW-complexe ou *cell space*)

Pour nous préparer à la section suivante, nous montrons ici que $\mathcal{S}$ est un complexe cellulaire. Un complexe cellulaire X — aussi appelé un CW-complexe — est un espace topologique union disjointe d'un ensemble de cellules B^k_α (où $k \in \mathbb{N}$ est la dimension de la cellule B^k_α et α fait partie d'un ensemble quelconque). Voici sa définition [70] : Nous notons $D^k \subset \mathbb{R}^k$ le disque (la boule) de dimension k, et $S^{k-1} \subset \mathbb{R}^k$ la sphère de dimension $k-1$, frontière de D^k : $S^{k-1} = \partial D^k$. X est construit de façon inductive :

1. On part d'un ensemble discret X^0 de points. Ces points forment les cellules de dimension 0 (les 0-cellules) de X.
2. On construit le k-**squelette** X^k de X à partir de X^{k-1} en attachant des k-cellules B^k_α à X^{k-1} à l'aide de fonctions $\varphi_\alpha : S^{k-1} \mapsto X^{k-1}$. X^k est l'espace $X^{k-1} \coprod_\alpha D^k_\alpha$ union disjointe de X^{k-1} et d'une collection de k-disques fermés D^k_α, et quotienté par la relation $x \equiv \varphi_\alpha(x)$ pour tout $x \in \partial D^k_\alpha$.
3. Le processus inductif ci-dessus peut être arrêté pour un k fini ($k \in \mathbb{N}$) — auquel cas on obtient $X = X^k$, ou continué indéfiniment pour obtenir $X = \cup_k X^k$.

Pour notre objet d'étude, le complexe de visibilité, ou plus précisément l'ensemble des segments libres maximaux, on s'arrêtera à $k = 4$. On va voir que l'on peut donner à $\mathcal{S}$ une structure de complexe cellulaire de dimension finie (4) et comportant un nombre fini de cellules.

Les CW-complexes ont de bonnes propriétés et forment un cadre dans lequel le théorème de Mayer-Vietoris, que nous utiliserons plus tard, fonctionne bien.

Pour prouver qu'un espace topologique est un CW-complexe, on peut simplement exhiber sa structure sous forme d'union de cellules satisfaisant les conditions données ci-dessus. Nous connaissons déjà une structuration de l'espace $\mathcal{S}$ des segments libres maximaux : c'est le complexe de visibilité $\mathcal{VC}$. Nous avons également vu que ce complexe $\mathcal{VC}$ n'est pas un CW-complexe puisque certaines de ses cellules peuvent comporter des « tunnels » (elles ne sont pas homéomorphes à un disque D^4).

Nous montrons maintenant que l'on peut raffiner le complexe $\mathcal{VC}$ en un complexe $\mathcal{VC}^n$, pour $n \in \mathbb{N}$ assez grand, de telle sorte que $\mathcal{VC}^n$ soit une structuration de $\mathcal{S}$ en un CW-complexe et que chaque cellule de $\mathcal{VC}$ soit une union finie de cellules de $\mathcal{VC}^n$. Nous prouvons ainsi que $\mathcal{S}$ est bien un CW-complexe.

Nous commençons par définir certaines partitions $\overline{\mathcal{VC}}^n$ de $\mathcal{S}$, puis expliquons comment obtenir $\mathcal{VC}^n$ à partir de $\overline{\mathcal{VC}}^n$.

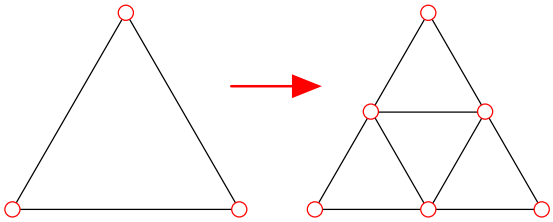


FIG. 7.2. Le schéma de subdivision.

Les partitions $\overline{\mathcal{VC}}^n$ de $\mathcal{S}$ sont obtenues itérativement en subdivisant les oculteurs de $\mathcal{O}$. Nous définissons d'abord la subdivision itérative $S^2 = S_0^2, S_1^2, S_2^2, S_3^2, \cdots$ de la sphère des directions orientées S^2. Cette subdivision utilise la subdivision d'un triangle équilatéral en 4 triangles équilatéral de même taille. Elle subdivision est appliquée ici sur un simple tétraèdre (en fait à peu près n'importe quelle schéma de subdivision ferait notre affaire). Soit T_1 un tétraèdre régulier dont le centre de masse est posé à l'origine. S_1^2 est obtenue en normalisant tous les points de la surface de T_1. Ainsi, chaque triangle du tétraèdre correspond à une cellule de la partition S_1^2 de la sphère des directions orientées. T_{i+1} est obtenu à partir de T_i en subdivisant chaque triangle de T_i en 4 triangles équilatères et égaux (Figure 7.2). Puis, S_{i+1}^2 est obtenue à partir de T_{i+1} en normalisant chaque point de la surface de T_{i+1} (on gonfle le tétraèdre subdivisé pour former une sphère). Passons à la subdivision des oculteurs. Pour chaque oculteur $o \in \mathcal{O}$, on fixe un point o_{ctr} à l'*intérieur* de o, que l'on appelle le *centre* de o. Alors, on peut associer à chaque point p de $o \setminus \{o_{ctr}\}$ la direction orientée $p - o_{ctr}$. On subdivise donc o[1] en une collection o_i de sous-ensembles de points dont la direction associée appartient à une même cellule de S_i^2. On définit ainsi pour tout $i \in \mathbb{N}$ un ensemble $\overline{\mathcal{O}}_i$ d'oculteurs, avec $\overline{\mathcal{O}}_0 = \mathcal{O}$ et tel que $\overline{\mathcal{O}}_{i+1}$ soit une subdivision des oculteurs de $\overline{\mathcal{O}}_i$.

Pour tout $n \in \mathbb{N}$, considérons le complexe de visibilité $\overline{\mathcal{VC}}^n$ associé à $\overline{\mathcal{O}}_n$. Remarquons que chaque cellule de $\mathcal{VC}$ est l'union disjointe d'un nombre fini de cellules de $\overline{\mathcal{VC}}^n$ (de dimensions variées).

Il nous reste à observer que pour $n \in \mathbb{N}$ assez grand, plus aucune cellule de $\mathcal{VC}^n$ ne peut contenir de « trous » : leur genre devient nul. Considérons deux éléments de surfaces A et B issus de la subdivison $\overline{\mathcal{O}}_n$. Soit $T_{A,B}$ le *shaft* entre A et B. $T_{A,B}$ est un volume de $\mathbb{R}^3$ égal à l'union, dans $\mathbb{R}^3$ des segments libres maximaux reliant A et B. Alors, un tunnel apparaîtra dans la 4-cellule A-B seulement si un autre oculteur de $\mathcal{O}$ existe à l'intérieur de $T_{A,B}$. La présence d'un tel oculteur de $\mathcal{O}$ dans un *shaft* devient impossible dès lors que la subdivision des oculteurs devient suffisamment fine, puisque A et B deviennent infiniment petits.

Pour les cellules de $\overline{\mathcal{VC}}^n$ dont un bloqueur est ∞, notre argument ne tient pas. Pour y remédier, nous ajoutons simplement un cube englobant la totalité de la scène. Les 6 faces de ce cube sont alors considérées comme des objets distincts et subdivisés, pour $n \in \mathbb{N}$ en 4^n carrés identiques. Ainsi, toutes les cellules de $\mathcal{VC}$ se voient raffinées dans $\overline{\mathcal{VC}}^n$.

Les arguments ci-dessus permettent de montrer que l'on se débarrasse aussi des tunnels dans les cellules de $\overline{\mathcal{VC}}^n$ de dimension inférieure à 4, pour un n assez grand.

[1] Plus précisément, on subdivise $o \setminus \{o_{ctr}\}$ mais cela n'a que peu d'importance puisque l'intérieur de o n'est jamais « visible » et n'intervient pas dans le complexe de visibilité. Seule la subdivision de la surface de o nous intéresse.

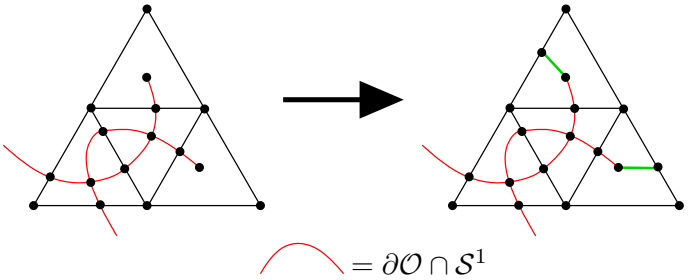


FIG. 7.3. Raffinement de la surface des occulteurs pour obtenir $\mathcal{VC}^n$.

Nous allons maintenant raffiner $\overline{\mathcal{VC}}^n$ pour exhiber une structuration de $\mathcal{S}$ en un CW-complexe. Soit C une 4-cellule de $\overline{\mathcal{VC}}^n$, et A et B ses deux bloqueurs. A et B sont clairement homéomorphes au disque D^2. Donc $A \times B$ est homéomorphe à D^4. Cependant, cela ne nous donne pas d'homéomorphisme entre D^4 et C puisque certains segments de $A \times B$ peuvent être ocultés par d'autres objets de la scène. Alors, pour expliciter un tel homéomorphisme, nous allons encore découper un peu plus la surface des occulteurs initiaux ($\mathcal{O}$). Considérons l'ensemble $\mathcal{S}^1$ des segments libres maximaux de type $T+T+T$ et T ++ T. Les segments libres de type $T+T+T$ sont les segments tangents à 3 occulteurs de $\mathcal{O}$. Les segments libres de type T ++ T sont les segments tangents à deux occulteurs et contenus dans un plan bitangent à ces deux occulteurs (voir [45]). Ces deux familles de segments libres sont localement de dimension 1, et forment dans l'espace $\mathbb{R}^3$ les *surfaces d'événements visuels* : lorsqu'un point de vue P se balade dans l'espace libre, la topologie de sa carte de visibilité change précisément au moment où P passe au travers d'un telle surface.

Dessinons sur les occulteurs. La subdivision des occulteurs que nous avons utilisée pour obtenir $\overline{\mathcal{VC}}^n$ peut être vu comme suit : on a d'abord dessiné des régions « triangulaires » à la surface de chaque occulteur (en dessinant incrémentalement un nouveau triangle à l'intérieur de chaque triangle). Ensuite, on a considéré l'arrangement de ces courbes dessinées sur la surface des occulteurs, et chaque 2-face de cet arrangement a été considérée comme un objet de la scène à part entière, distinct des autres 2-faces.

Pour obtenir $\mathcal{VC}^n$ nous continuons selon ce principe. Nous *ajoutons* donc de nouvelles courbes à la surface des occulteurs. Ces courbes sont définies dans $\mathbb{R}^3$ par l'intersection $\partial\mathcal{O} \cap \mathcal{S}^1$ de $\partial\mathcal{O}$ (la surface des occulteurs) avec l'ensemble $\mathcal{S}^1$ vu comme un ensemble de points de $\mathbb{R}^3$. Nous obtenons un nouvel arrangement sur $\partial\mathcal{O}$ que nous raffinons encore de manière à ce que tout point de $\partial\mathcal{O} \cap \mathcal{S}^1$ soit sur le bord de deux 2-faces distinctes. Voir Figure 7.3.

Lemme 7.2. *La surface des occulteurs de $\mathcal{O}$ est alors partitionnée en un ensemble $\mathcal{O}_n$ de 2-faces homéomorphes à un disque de telle sorte qu'aucun point de vue ne peut se balader sur la surface d'un objet et croiser une surface d'événements visuels sans passer d'une 2-face f_1 à une autre, f_2, distincte de f_1.*

Nous définissons $\mathcal{VC}^n$ comme le complexe de visibilité associé à $\mathcal{O}_n$. Il nous reste à démontrer que chaque 4-cellule de $\mathcal{VC}^n$ est homéomorphe à D^4 donc à une boule de dimension 4. Soit C une 4-cellule de $\mathcal{VC}^n$, dont les bloqueurs sont les 2-faces A et B à la surface de deux occulteurs de $\mathcal{O}$. Soit P un point de vue posé à l'intérieur de A. Soit B_P la partie de B visible par P. Remarquons que pour un n assez grand, B_P est simplement connexe, grâce à la définition de $\overline{\mathcal{O}}_n$. Montrons que B_P forme en fait une seule

composante connexe et est donc homéomorphe au disque D^2. Supposons le contraire, c'est-à-dire que B_P soit formée d'au moins deux composantes connexes. Considérons alors b_1 et b_2 à l'intérieur de deux composantes connexes distinctes de B_P. Ces deux points forment, avec P, deux segments libres maximaux contenus à l'intérieur de C : $s_1 = [Pb_1]$ et $s_2 = [Pb_2]$. Comme C est connexe, il existe un chemin χ continu les reliant dans C : $\chi : [0,1] \mapsto C, \chi(0) = s_1, \chi(1) = s_2$. Considérons alors $B_{\chi(t)\cap A}$, la famille, indexée par $t \in [0,1]$, des sous-ensembles $B_{\chi(t)\cap A}$ de B formés des points de B visibles par l'extrémité de $\chi(t)$ contenue dans A. $B_{\chi(t)\cap A}$ doit être vu comme la déformation continue de B_P vers B_P. Clairement, les deux composantes connexes contenant b_1 et b_2 doivent, au cours de leur déformation, fusionner à un moment donné, autrement le chemin χ ne pourrait pas être continu. Or, au moment t d'un telle fusion, le point $\chi(t) \cap A$ passe nécessairement au travers d' une surface d'événements visuels, et sort donc de A. On a une contradiction. Conclusion : B_P ne contient qu'une composante connexe homéomorphe au disque D^2.

Avec le même raisonnement, on montre que la partie A_C de A visible par les segments libres contenus dans C, forme également une seule composante connexe et simplement connexe. Enfin, puisque B_P se déforme continûment lorsque P bouge sur l'intérieur de A_C, C est bien homéomorphe à D^4.

7.2 Un résultat de connexité

Nous montrons dans cette section le théorème suivant

Théorème 7.3. *Soit C une 4-cellule du complexe de visibilité d'un ensemble fini de polytopes disjoints et bornés. Soit ∂C la frontière de C. Alors, ∂C est connexe.*

En fait, la preuve de ce théorème ne fait pas intervenir la qualité de *polytope*. Ainsi, le théorème reste vrai pour n'importe quel ensemble fini de convexes disjoints et bornés. Nous l'appliquerons ensuite au cas d'un ensemble de polytopes dont on veut construire le complexe de visibilité. Nous supposons donc que notre scène est composée d'objets convexes disjoints. Nous commençons par étudier quelques propriétés de l'espace libre $\mathcal{F}$ et de l'ensemble $\mathcal{S}$ des segments libres maximaux.

Lemme 7.4. *L'espace libre $\mathcal{F}$ est connexe par arc et simplement connexe.*

Preuve : Cette propriété est assez évidente puisque les oculteurs sont convexes et disjoints. Nous donnons cependant une preuve par induction le nombre n d'oculteurs.

Nous noterons $\mathcal{I} = [0,1] \subset \mathbb{R}$.

Pour $n = 0$, on a $\mathcal{F} = \mathbb{R}^3$. $\mathcal{F}$ est trivialement connexe par arc et simplement connexe. Supposons maintenant que notre scène comporte n oculteurs $o_1, o_2, \cdots, o_n$, avec $n \geq 0$. Définissons $\mathcal{F}' = \mathbb{R}^3 \setminus \bigcup_{i=1}^{n-1} o_i$. Par l'hypothèse d'induction, $\mathcal{F}'$ est connexe par arc et simplement connexe. $\mathcal{F}$ est obtenu à partir de $\mathcal{F}'$ en lui ôtant l'oculteur o_n : $\mathcal{F} = \mathcal{F}' \setminus o_n$. Nous allons seulement montrer que $\mathcal{F}$ est simplement connexe, puisque la preuve de connexité par arc est tout à fait similaire.

Soit $x_0 \in \mathcal{F}$ et f une boucle dans $\mathcal{F}$ dont le point de base est x_0. f est une application continue $f : \mathcal{I} \mapsto \mathcal{F}$ telle que $f(0) = f(1) = x_0$. Puisque $\mathcal{F} \subset \mathcal{F}'$, f est aussi une boucle dans $\mathcal{F}'$. Comme $\mathcal{F}'$ est simplement connexe, il existe une homotopie F, dans $\mathcal{F}'$, de f vers la boucle constante x_0 :
$F : \mathcal{I} \times \mathcal{I} \mapsto \mathcal{F}'$ est continue et vérifie $F(0,t) = f(t)$ et $F(1,t) = x_0$ pour tout $t \in \mathcal{I}$.

Nous allons construire une homotopie $G : \mathcal{I} \times \mathcal{I} \mapsto \mathcal{F}$ qui vérifiera $G(0,t) = f(t)$ et $G(1,t) = x_0$ pour tout $t \in \mathcal{I}$. Soit $\mathrm{Im}\, F$ l'image de $\mathcal{I} \times \mathcal{I}$ par F. Comme $\mathrm{Im}\, F$ est bidimensionnel, il existe un point $o_n^* \in o_n$ qui n'est pas dans $\mathrm{Im}\, F$.

Soit $p \in \mathbb{R}^3 \setminus \{o_n^*\}$. On définit p^* comme l'unique intersection de la frontière ∂o_n de o_n avec la demi-droite ayant pour origine o_n^* et pour direction $p - o_n^*$. Il est aisé de remarquer que $p^* \in \mathcal{F}$.

Nous définissons l'application

$$\mathrm{push} : \mathcal{F}' \setminus \{o_n^*\} \mapsto \mathcal{F}, \mathrm{push}(p) = \begin{cases} p, & \text{si } p \notin o_n \\ p^*, & \text{si } p \in o_n \end{cases}$$

Puisque $\mathrm{Im}\, F$ ne contient pas o_n^*, la composition $\mathrm{push} \circ F$ est continue, et est donc une homotopie de f vers la boucle constante x_0 dans $\mathcal{F}$. On a démontré que $\mathcal{F}$ est simplement connexe. □

Lemme 7.5. *L'ensemble $\mathcal{S}$ des segments libres maximaux est connexe par arc et simplement connexe.*

Preuve : Nous omettons la preuve de connexité par arc car elle est similaire à la preuve de simple-connexité.

Soit $c : S^1 \mapsto \mathcal{S}$ une boucle fermée dans le complexe de visibilité. De façon équivalente, c est une fonction continue de l'intervalle $[0,1]$ dans $\mathcal{S}$, $c(0) = c(1)$. Afin de prouver le lemme 7.5, nous devons montrer que c est continûment contractible en un point de $\mathcal{S}$ (un segment libre maximal).

Nous commençons par extraire une courbe fermée $p : S^1 \mapsto \mathcal{F}$ telle que $p(t)$ soit un point du segment $c(t)$. Puisque $\mathcal{F}$ est simplement connexe, (lemme 7.4), nous pourrons alors contracter p en un point. Cette contraction nous servira à construire une contraction de c en un segment libre de $\mathcal{S}$, prouvant ainsi le lemme 7.5.

Soit $\mathcal{S}^3$ le squelette de dimension 3 de $\mathcal{VC}^n$ (le complexe cellulaire défini plus haut). Par souci de simplicité, nous faisons l'hypothèse que c passe d'une 4-cellule de $\mathcal{VC}^n$ à une autre de manière transversale : si $c(t) \in \mathcal{S}^3$, alors il existe $\varepsilon > 0$ tel que $0 < |t' - t| < \varepsilon$ implique $c(t') \notin \mathcal{S}^3$.

La longueur des segments $c(t)$ varie continûment sauf quand c croise la frontière d'une 4-cellule du complexe. Soit L l'ensemble des points de $[0,1]$ où les segments de c changent de longueur discontinûment. Supposons que le cardinal $|L|$ de l soit infini. Puisque $L \subset [0,1]$, L admet un point d'accumulation a. Cela contredit notre hypothèse de transversalité ; on en déduit que L est un ensemble fini :

$L = \{t_i \in [0,1], i = 0..N_c\}$ est l'ensemble des points t de $[0,1]$ où la longueur des segments de c change discontinûment.

Soit $t \in [0,1]$. Si $t \notin L$, alors nous définissons $p(t)$ comme le point milieu du segment $c(t)$. p n'est pour le moment pas défini aux points $t_i, i = 0..N_c$. Cependant, $c(t_i)$ est tangent à au moins un oculteur et $c(t_i)$ contient (en tant qu'ensemble de points de $\mathbb{R}^3$) deux points limites :

$$T_i = \lim_{t \to t_i^-} p(t); T_i \in c(t_i)$$

$$T_i' = \lim_{t \to t_i^+} p(t); T_i' \in c(t_i)$$

Au lieu de définir p au point t_i, nous insérons un segment de droite $[T_i, T_i'] \subset c(t_i)$ pour « recoller » les parties de p déjà définies. Ce recollement s'effectue en élargissant,

dans $[0,1]$, tous les singletons $\{t_i\}$ en un petit intervalle $[a_i, b_i]$ de longueur non nulle (on effectue donc une reparamétrisation de c pour « faire de la place »). Ensuite, nous « mappons » cet intervalle sur le segment $[T_i, T_i']$. Maintenant, p est une boucle continue dans $\mathcal{F}$. De même, nous construisons l'application continue $\omega : [0,1] \mapsto \mathbb{RP}^2$ telle que

$$(p(t), \omega(t)) = c \circ r(t), t \in [0,1]$$

où r est une reparamétrisation continue de $[0,1]$ (r « écrase » sur t_i chaque intervalle du type $[a_i, b_i]$ dont l'image est $[T_i, T_i']$).

Comme $\mathcal{F}$ (lemme 7.4) et $\mathbb{RP}^2$ sont simplement connexes, on peut contracter $c \approx (p, \omega)$ en un point de $\mathcal{S}$. □

Lemme 7.6. *Soit C une 4-cellule de $\mathcal{VC}$, $C \subset \mathcal{S}$. Soit $C^c = \mathcal{S} \setminus C$ le complémentaire de C. Alors C^c est connexe.*

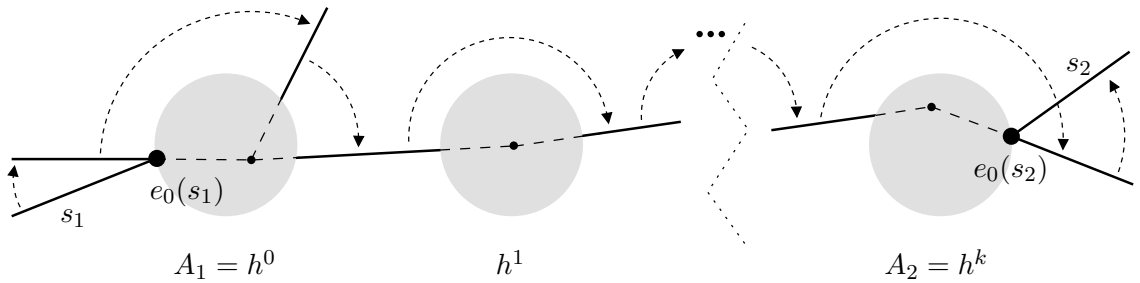
Preuve : Soit s_1 et $s_2 \in C^c$. Notons $vis(s_1) = (A_1, B_1)$ et $vis(s_2) = (A_2, B_2)$ les ensembles visibles de s_1 et s_2 respectivement. Nous noterons $e_0(s)$ et $e_1(s)$ les points extrémités (éventuellement infinies) d'un segment libre maximal s. Notons (A_C, B_C) l'ensemble visible de la 4-cellule C, que l'on cherchera à éviter en bougeant s_1 continûment vers s_2. Pour $i = 1, 2$, si $vis(s_i) = (A_C, B_C)$ (c'est-à-dire que s_i voit les mêmes objets que C, mais n'est pas dans la même composante connexe), alors nous déplaçons s_i dans une direction aléatoire jusqu'à ce que son ensemble visible change. Maintenant, sans perte de généralité, nous avons $A_i \notin \{A_C, B_C\}$ pour $i = 1, 2$.

Cas 1 : $A_C = \infty$ et $B_C = \infty$. Soit

$$H = \{h^0 = A_1, h^1, ..., h^{k-1}, h^k = A_2\}$$

la séquence ordonnée des oculteurs traversés par le segment de droite $[e_0(s_1), e_0(s_2)]$. Notons que $e_0(s_1)$ est sur la frontière de A_1 et que $e_0(s_2)$ est sur la frontière de A_2 (voir le schéma ci-dessous). Nous définissons un déplacement en *saut* qui déplace continûment $e_0(s_1)$ de h^i vers h^{i+1}.

Fixons un point h^i_{ctr} appellé « centre » à l'intérieur de chaque occulteur h^i. La première étape du saut consiste à aligner la droite support de s_1 avec h^i_{ctr}, en faisant effectuer à s_1 une rotation autour de son extrémité $e_0(s_1)$. La seconde étape du saut déplace s_1 jusqu'à ce qu'il touche h^{i+1} et que sa droite support passe par h^{i+1}_{ctr}. Ce déplacement est effectué en faisant tourner s_1 autour du point h^i_{ctr}. À la fin du saut, s_1 se trouve sur la droite $(h^i_{ctr} h^{i+1}_{ctr})$; nous échangeons alors les rôles des extrémités $e_0(s_1)$ et $e_1(s_1)$ de telle sorte que $e_0(s_1)$ apparaisse sur la frontière de h^{i+1}.



Après k sauts, $e_0(s_1)$ se trouve sur la frontière de $h^k = A_2$ sur laquelle se trouve $e_0(s_2)$ également. Il est maintenant très simple de déplacer s_1 vers s_2 en contraignant $e_0(s_1)$ à rester sur ∂A_2.

Cas 2 : $A_C \neq \infty$. Nous devons trouver une séquence de sauts depuis A_1 vers A_2 tout en évitant l'oculteur A_C. C'est très facile puisque les oculteurs sont convexes. Notons cependant, qu'il pourra être nécessaire d'ajouter un *oculteur virtuel* si A_C est un oculteur très grand situé entre A_1 et A_2 et qu'aucun autre oculteur n'est atteignable par des sauts depuis A_1 ou A_2. □

Preuve du théorème 7.3. Nous donnons enfin la preuve de notre théorème principal. Nous allons utiliser quelques propriétés des complexes cellulaires, les lemmes donnés ci-dessus ainsi qu'un outil de la théorie de l'homologie singulière : la séquence exacte de Mayer-Vietoris [70]. Soit $\{G_n, n \in \mathbb{N}\}$ une famille de groupes.

Une **séquence exacte** est une séquence d'homomorphismes de groupes $\phi_n, n \in \mathbb{N}$, de la forme

$$\cdots \xrightarrow{\phi_{n+1}} G_{n+1} \xrightarrow{\phi_n} G_n \xrightarrow{\phi_{n-1}} G_{n-1} \xrightarrow{\phi_{n-2}} \cdots \xrightarrow{\phi_0} G_0 \to 0$$

telle que pour tout n, $\operatorname{Im}\phi_{n+1} = \operatorname{Ker}\phi_n$ (et donc $\phi_n \circ \phi_{n+1} = 0$). En particulier, soit X un CW-complexe, U et V deux ouverts de X tels que $X = U \cup V$, alors la séquence de Mayer-Vietoris, définie ainsi :

$$\cdots H_n(U \cap V) \to H_n(U) \oplus H_n(V) \to H_n(X) \to H_{n-1}(U \cap V) \to \cdots$$

est une séquence exacte, où $H_n(A)$ est le n-groupe d'homologie de l'espace topologique A [70, page 523]. (Les homomorphismes de groupes de la séquence sont construits explicitement, voir [70].)

Preuve (théorème 7.3) : Soit C une 4-cellule du complexe de visibilité $\mathcal{VC}$ et ∂C sa frontière. Il existe un voisinage B de ∂C homotope à ∂C [70, proposition A.5]. Soit C^C le complémentaire de C dans $\mathcal{S}$. Nous définissons $U = C \cup B$ et $V = C^C \cup B$. D'après le lemme 7.6, U et V sont connexes.

La séquence de Mayer-Vietoris est une séquence exacte :

$$\cdots \to H_1(U \cup V = \mathcal{S}) \to H_0(U \cap V) \to H_0(U) \oplus H_0(V) \to H_0(U \cup V) \to 0.$$

$\mathcal{S}$ est simplement connexe par le lemme 7.5 ; donc

$$H_1(U \cup V) = H_1(\mathcal{S}) = 0$$

Ainsi,

$$\cdots \to 0 \to H_0(U \cap V) \to H_0(U) \oplus H_0(V) \to H_0(U \cup V) \to 0$$

Puisque $\mathcal{S}$ (lemme 7.5), U et V (lemme 7.6) sont connexes, on a :

$$\cdots \to 0 \xrightarrow{\phi_3} H_0(B) \xrightarrow{\phi_2} \mathbb{Z}^2 \xrightarrow{\phi_1} \mathbb{Z} \xrightarrow{\phi_0} 0$$

où $\mathbb{Z}^k$ est le groupe libre engendré par k éléments.

Par définition d'une séquence exacte, on a $\operatorname{Im}\phi_1 = \operatorname{Ker}\phi_0 = \mathbb{Z}$, donc ϕ_1 est une surjection. De même, $\operatorname{Im}\phi_3 = \operatorname{Ker}\phi_2 = 0$, donc ϕ_2 est une injection :

$$\cdots \to 0 \xrightarrow{\phi_3} H_0(B) \xhookrightarrow{\phi_2} \mathbb{Z}^2 \xrightarrow{\phi_1} \mathbb{Z} \xrightarrow{\phi_0} 0$$

Une telle séquence, de la forme $0 \to G_3 \to G_2 \to G_1 \to 0$, est dite *courte* et a la propriété d'être *scindable* : c'est à dire que G_2 est isomorphe à $G_3 \oplus G_1$ [70, page 147].

Nous avons donc montré que $\mathbb{Z}^2 \approx H_0(B) \oplus \mathbb{Z} \approx \mathbb{Z}^k \oplus \mathbb{Z} \approx \mathbb{Z}^{k+1}$ puisque $H_0(B) \approx \mathbb{Z}^k$ pour un certain entier k.

Nous avons $\mathbb{Z}^2 \approx \mathbb{Z}^{k+1}$. Le théorème fondamental sur les groupes finiment générés nous permet de conclure que $k+1 = 2$ et $k = 1$.

Ainsi, le groupe d'homologie $H_0(B)$ est isomorphe à $\mathbb{Z}$. Ce qui montre que B est connexe, et donc connexe par arc puisque ces deux propriétés sont équivalentes dans un CW-complexe [70, page 523]. Or, comme ∂C est homotope à B, on en déduit que ∂C est connexe. □

Nous venons de démontrer que la frontière de chaque 4-cellule du complexe de visibilité est connexe.

Remarque 7.1. *Tous les lemmes présentés ci-dessus, sauf le lemme 7.6, restent valides avec la même preuve (ou presque) si on définit un oculteur comme un sous-ensemble de $\mathbb{R}^3$ ouvert et homéomorphique à la boule unité $\{x \in \mathbb{R}^3, |x| < 1\}$. (Les oculteurs sont des « patatoïdes ».) La preuve du lemme 7.6 utilise cependant l'hypothèse de convexité des oculteurs. Ainsi, le théorème 7.3 n'est prouvé que pour le cas convexe. Nous conjecturons que ce théorème reste vrai quand les oculteurs sont homéomorphes à la boule unité.*

7.3 Construire le complexe de visibilité d'un ensemble de polytopes

Un objet tridimensionnel est le plus souvent représenté en explicitant sa surface extérieure ainsi que la surface de ses cavités éventuelles. De la même manière, nous souhaitons construire la frontière de chaque 4-cellule du complexe, pour représenter le complexe lui-même. Nous présentons ici un algorithme simple pour construire le complexe de visibilité d'un ensemble de polytopes disjoints. Plus précisément, nous construisons explicitement le 3-squelette du complexe $\mathcal{VC}$ puis utilisons la propriété de connexité démontrée ci-dessus pour extraire du 3-squelette chacune des 4-cellules du $\mathcal{VC}$. Puisque la frontière d'une 4-cellule est connexe, la reconstruction de cette cellule pourra se faire en calculant une certaine composante connexe dans le graphe d'adjacence donné par le 3-squelette. Plus précisément, soit $\mathcal{G} = (\mathrm{Sk}_3, E_3)$ le graphe où Sk_3 est l'ensemble des 3-cellules de $\mathcal{VC}$ et $E_3 \subset \mathrm{Sk}_3 \times \mathrm{Sk}_3$ indique les relations d'adjacence entre les 3-cellules. Chaque 3-cellule b est marquée avec un ensemble $\mathrm{adj}(b)$ contenant les trois paires $(A, B) \in (\mathcal{O} \cup \{\infty\})^2$ correspondant aux ensembles visibles des trois 4-cellules adjacentes à b. Étant donnée une 4-cellule C dont l'ensemble visible est (A, B) et une 3-cellule $b \in \partial C$, nous pouvons retrouver toute la frontière de C comme la composante $K_b \subset \mathrm{Sk}_3$ maximalement connexe (dans $\mathcal{G}$) telle que pour tout $b' \in K_b$, on a $(A, B) \in \mathrm{adj}(b')$. Concentrons nous maintenant sur la construction du 3-squelette de $\mathcal{VC}$: (Sk_3, E_3).

$\mathcal{O}$ est un ensemble de polytopes. Ainsi, une 3-cellule b consiste en un ensemble maximalement connexe de segments libres maximaux tangents à une même arête e d'un polytope, et ayant le même ensemble visible. Si l'ensemble visible de la 3-cellule b est (A,B) on dira que b est de la forme (e, A, B). Pour chaque arête e d'un polytope, nous allons construire les 3-cellules de la forme (e, o_1, o_2), où $o_1, o_2 \in (\mathcal{O} \cup \{\infty\})$.

Nous supposons que le 1-squelette du complexe de visibilité est déjà construit. Ce peut être effectué en temps $O(n^2k^2 \log n)$ avec l'algorithme de Goaoc [60]. k est le

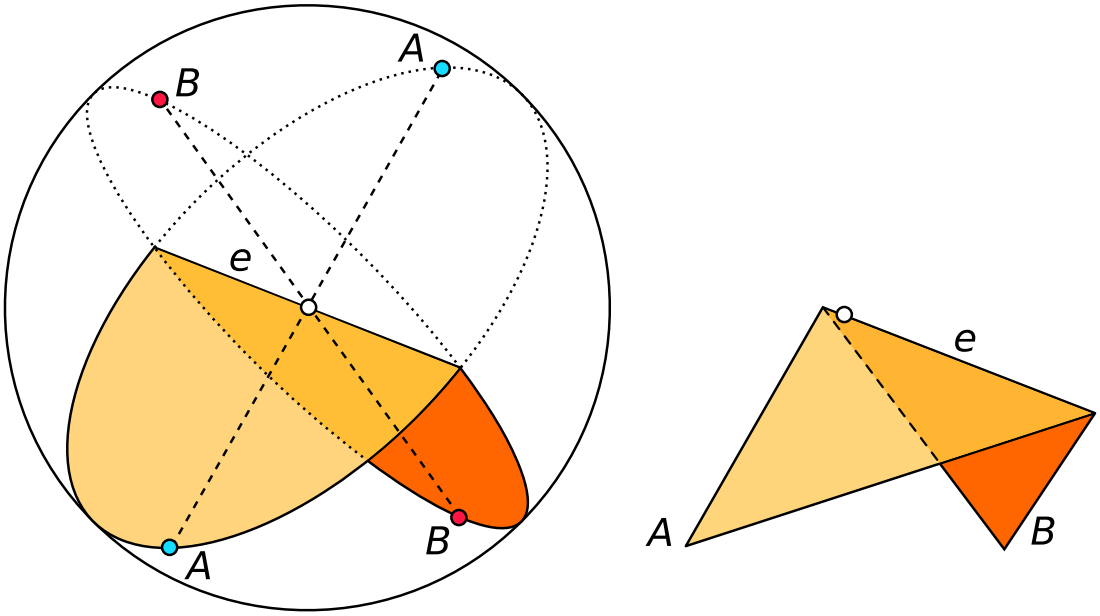


FIG. 7.4. Balayage d'une 3-cellule de la forme $(e, *, *)$. Le point blanc indique la position du point P à un instant du balayage. **À droite**, on observe l'arête e en train d'être balayée ainsi que ses deux polygones adjacents. **À gauche**, on observe comment ces deux polygones partitionnent la sphère des directions (non orientées) autour du point P.

nombre de polytopes et n est le nombre total d'arêtes. Cet algorithme balaye l'ensemble des segments libres tangents à une arête e avec un plan contenant e et tournant autour de cette dernière. Bien qu'il semble possible de construire au cours du même balayage l'ensemble des cellules de dimension 3 de la forme $(e, *, *)$, nous préférons utiliser un balayage différent plus facile à visualiser.

Coupe radiale de dimension 2 dans $\mathcal{S}$

Soit P un point à l'intérieur de l'espace libre $\mathcal{F}$. Le graphe de visibilité $V(P)$ de P est la projection des arêtes silhouettes de $\mathcal{O}$ visibles depuis P, sur la sphère unité centrée en P. $V(P)$ est un graphe « sphérique ». Un sommet de $V(P)$ est soit un sommet sur la silhouette d'un polytope, soit un *t-sommet*, c'est-à-dire l'intersection visuelle (sur la sphère unité) de deux arêtes silhouettes.

Considérons maintenant l'ensemble $\mathcal{S}_P$ des segments libres maximaux passant par P. $\mathcal{S}_P$ peut être mis en bijection avec $\mathbb{RP}^2$. Comme pour $V(P)$, nous partitionnons $\mathcal{S}_P$ en composantes maximalement connexes de segments ayant le même ensemble visible. Cette opération équivaut à considérer l'intersection de la partition $\mathcal{VC}$ de $\mathcal{S}$ avec $\mathcal{S}_P$. Un telle décomposition de $\mathcal{S}_P$ peut être visualisée comme l'arrangement de $V(P)$ avec la version symétrique de $V(P)$ par rapport à P ; $\mathcal{S}_P$ est un graphe planaire partitionnant $\mathbb{RP}^2$. Quand P est dans l'espace libre, les sommets de $\mathcal{S}_P$ sont des points dans quelques 2-cellules de $\mathcal{VC}$, c'est-à-dire des bitangentes ou des tangentes passant par un sommet de $\mathcal{O}$. Les arêtes de $\mathcal{S}_P$ sont des courbes (de dimension 1) dans quelques 3-cellules de $\mathcal{VC}$, c'est-à-dire des ensembles de segments tangents à une même arête. Finalement, les faces de $\mathcal{S}_P$ sont des surfaces de dimension 2 dans quelques 4-cellules du complexe de visibilité, c'est-à-dire l'intersection de $\mathcal{S}_P$ avec l'intérieur d'une 4-cellule.

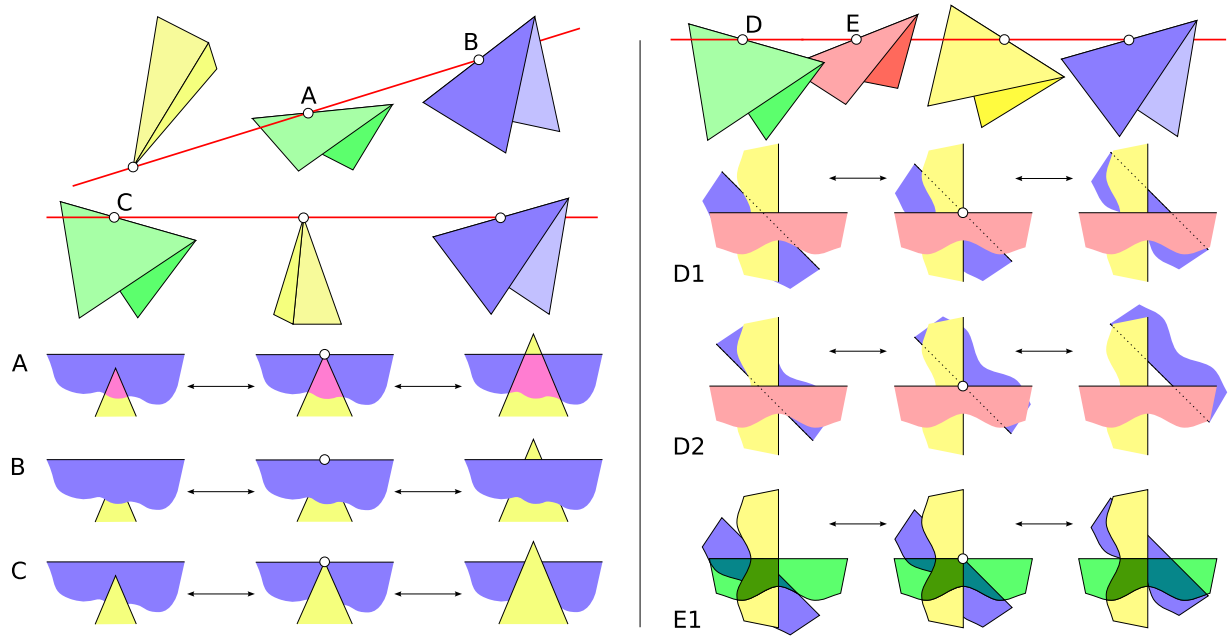


FIG. 7.5. Mise à jour de $\mathcal{S}_P$ lors du croisement d'une droite poignardante extrême de type VEE (à gauche) et EEEE (à droite). Dans les diagrammes du bas, chaque région colorée est une coupe 2D d'une 3-cellule. À droite, les cas D1 et D2 sont symétriques. Le cas E2, symétrique du cas E1, n'est pas illustré ici.

Nous plaçons maintenant P sur une arête d'un polytope de $\mathcal{O}$. En effet, nous sommes intéressés par les tangentes à ces arêtes. En plaçant ainsi P sur l'arête e, $\mathcal{S}_P$ devient l'ensemble des segments libres qui sont tangents à e en P (Figure 7.4).[2] Lorsque P se déplace le long de e, $\mathcal{S}_P$ balaye toutes les 3-cellules de la forme $(e, *, *)$. Si nous pouvons maintenir la décomposition de $\mathcal{S}_P$ alors que P se déplace d'un bout à l'autre de e, nous pourrons construire toutes les 3-cellules dont l'arête de tangence est e. L'intersection de $\mathcal{S}_P$ avec $\mathcal{VC}$ est un graphe planaire sur $\mathbb{RP}^2$ qui varie continûment lors du déplacement de P, sauf en certaines positions de P (en nombre fini) quand 3 arêtes du graphe se croisent (il y a un changement combinatoire du graphe). De fait, les segments libres maximaux où ont lieu ces événements sont extrémaux. Ce sont des 0-cellules de $\mathcal{VC}$, des segments quadritangents. Si le squelette de visibilité (le 1-squelette de $\mathcal{VC}$) a été construit auparavant, nous pouvons donc connaître directement les changements combinatoires du graphe qui auront lieu lors du mouvement de p sur e.

Nous avons vu, informellement, comment $\mathcal{VC}$ peut être construit à partir de son 1-squelette.

7.3.1 Construction des 3-cellules attachées au segment e

Soit $e = \overline{v_0 v_1}$ une arête d'un oculteur $\mathcal{O}$. e est adjacent à 2 facettes $t_1(e)$ et $t_2(e)$. Nous dénotons par $\mathcal{D}(e)$ le sous-ensemble de $\mathbb{RP}^2$ égal à l'ensemble des directions des tangentes à e en son intérieur. $\mathcal{D}(e)$ est une double-lune sur la sphère des directions comprise entre les plans supports de $t_1(e)$ et $t_2(e)$ (Figure 7.4). Soit $\mathcal{VC}(e)$ le sous-ensemble de $\mathcal{VC}$ contenant les segments libres tangents à e. Nous appelons $\mathcal{VC}(e)$

[2] On ignore les segments libres non tangents à e en P.

la *colonne* de e. Les 3-cellules dans $\mathcal{VC}(e)$ sont construites, ensembles, comme une partition de $[0,1] \times \mathcal{D}(e)$. D'abord, le point P est placé en v_0 et les 2-cellules attachées au sommet v_0 sont construites à partir du 1-squelette. Ces 2-cellules forment les bords de certaines 3-cellules de $\mathcal{VC}(e)$. Ensuite, nous déplaçons P le long de e jusqu'à v_1.

Construction de $\mathcal{S}_{v_0}$ et sa maintenance le long de e

Hoberock [74] propose l'utilisation d'une structure en demi-arêtes (*halfedge data structure*) pour représenter les 2-cellules du complexe de type V-cellule, dont est composé $\mathcal{S}_{v_0}$. Comme le 1-squelette de $\mathcal{VC}$ est déjà calculé, on connaît déjà toutes les arêtes et sommets de $\mathcal{S}_{v_0}$. On peut donc construire les 2-cellules de $\mathcal{S}_{v_0}$ à l'aide d'un balayage de la sphère des directions centrée en v_0. Ce balayage est tout à fait analogue au balayage d'un ensemble de polygones dans le plan. En particulier, aucune intersection ne sera trouvée et le balayage permet seulement d'associer à chaque 2-cellule l'ensemble de ses frontières. Ainsi, la construction de tous les $\mathcal{S}_v$ pour tous les sommets v des polytopes de la scène peut se faire en temps $O(n^2k^2 \log(n^2k^2)) = O(n^2k^2 \log n)$.

Soit $e_1, e_2, \cdots, e_j$ les arêtes de $\mathcal{O}$ adjacentes à v_0. Le support de $\mathcal{S}_{v_0}$ dans $\mathbb{RP}^2$ est l'union des $\mathcal{D}(e_i)$ pour $i = 1..j$, puisque chaque segment libre tangent à l'intérieur d'un e_i peut être translaté jusqu'à v_0 (puisque $\mathcal{O}$ est convexe). Aussi, chaque segment libre s de $\mathcal{S}_{v_0}$ sera adjacent à deux colonnes : les colonnes $\mathcal{VC}(e)$ et $\mathcal{VC}(e')$ des deux arêtes e et e' silhouettes de $\mathcal{O}$ dans sa vue orthographique de direction s.

Pour commencer le balayage de $\mathcal{VC}(e)$, on commence par extraire de $\mathcal{S}_{v_0}$ les parties incluses dans $\mathcal{D}(e)$. Nous obtenons ainsi $\mathcal{S}_P$ pour un point P sur e infiniment proche de v_0. Chaque face de $\mathcal{S}_P$ donne naissance à une 3-cellule bordée en partie par les 2-cellules de $\mathcal{S}_{v_0}$.

Traiter les événements du balayage

Soit $P_0 \in e$ un point où un changement combinatoire a lieu dans $\mathcal{S}_P$ lorsque P passe au travers de P_0. Alors, dans $\mathcal{S}_{P_0}$, 3 arêtes se croisent en un même point u_0 de $\mathbb{RP}^2$ (on a un événement de type VE ou EEE). Cela indique la présence d'un segment libre maximal $s = (P_0, u_0)$ tangent à e et à 3 autres arêtes de polytopes. s est donc tangent à 4 arêtes : $\{s\}$ est une 0-cellule, et a donc déjà été construite comme élément du 1-squelette de $\mathcal{VC}$.

Toutes les 0-cellules du 1-squelette de $\mathcal{VC}$ correspondantes à un segment libre tangent à e sont donc extraites puis triées en fonction de la position de leur point d'intersection avec l'arête e, et le balayage de la colonne de e peut commencer. La gestion de chaque événement est très similaire à celle des événements apparaissant dans la maintenance de la décomposition radiale présentée au chapitre 5 (Figure 7.5). Le tri des événements ayant lieu le long de toutes les arêtes se fait en temps $O(n^2k^2 \log n)$.

Après avoir calculé $\mathcal{S}_v$ pour tous les sommets de la scène, en temps $O(n^2k^2 \log n)$, on peut donc calculer toutes les 3-cellules du complexe de visibilité en temps proportionnel à leur taille, et borné par la taille du squelette, $O(n^2k^2)$. La recherche des composantes connexes correspondant à chaque 4-cellule dans le graphe (Sk_3, E_3) est également linéaire.

Théorème 7.7. *Le complexe de visibilité $\mathcal{VC}$ d'un ensemble de k polytopes de complexité totale n peut être construit en temps $O(n^2k^2 \log n)$.*

Remarque 7.2. *La description de l'algorithme de construction du complexe, que nous venons d'exposer, ne donne pas de preuve de sa correction. Un élément essentiel de cette correction est le fait que, dans la description combinatoire du complexe de visibilité de polytope, on garantit que chaque k-cellule possède une frontière formée de $(k-1)$-cellules, ce qui n'est pas le cas, par exemple, dans le cas d'une scène composée d'un seul objet convexe et lisse (auquel cas, le complexe de visibilité ne comprend aucune cellule de dimension 0, 1 et 2). Dans le cas de polytopes, par exemple, la droite support d'une arête d'un polytope est considérée comme une 0-cellule, puisqu'elle est tangente au polytope et passe par deux sommets.*

Dans la section suivante, nous revenons sur la discussion entamée à la page 12 à propos de la représentation des faces du complexe de visibilité par des structures de données.

7.4 Structure de données pour le complexe de visibilité

Soit $\mathcal{O} = \{P_1, P_2, \cdots, P_k\}$ un ensemble de polytopes disjoints dont on souhaite calculer le complexe de visibilité $\mathcal{VC}$. Soit T_{P_i} l'ensemble des segments libres tangents au polytope P_i tel que décrit page 12, c'est-à-dire indépendamment de la présence des autres polytopes. Le complexe $\mathcal{VC}$ est l'arrangement des ensembles $T_{P_i}, i = 1..n$, dans $\mathcal{S}$, l'ensemble des segments libres maximaux.

La représentation des relations d'adjacence entre les nouvelles faces apparaissant dans l'arrangement $\mathcal{VC}$ ne pose pas de difficulté particulière. Aussi, nous examinons le seul point délicat dans la description des adjacences entre les faces de T_P pour un polytope P considéré. Il s'agit de décrire les relations d'adjacences entre les segments tangents à P en un sommet v de P, et les segments tangents à P en une ou deux arêtes adjacentes à v. Examinons par exemple la Figure 7.6(a). La 1-cellule c_{v,e_2} (l'ensemble des segments libres passant par le sommet v et l'intérieur de l'arête e_2) est incluse dans la 2-cellule c_v et y est un point d'entrée dans la cellule c_{e_1,e_2}. Nous représentons l'ensemble des segments tangents en v par une sphère S_v dont on identifiera les points diamétralement opposés ($\mathbb{RP}^2$). Ainsi, sur cette sphère devra apparaître une arête (un morceau d'équateur) correspondant à la 1-cellule c_{v,e_2} et égale à l'ensemble des directions prises par les segments libres de c_{v,e_2}. Examinons maintenant la Figure 7.6(e). L'ensemble des segments libres passant par v et appartenant au plan support π_f du polygone f formera également une arête dans la sphère S_v. Notons que cet ensemble unidimensionnel de segments libres n'est pas représentable sur la frontière d'une paramétrisation de c_{e_1,e_2} sur le carré $[0,1]^2$ (Figure 7.6(g)). Aussi, nous devons paramétrer c_{e_1,e_2} comme un triangle dans l'espace dual du plan π_f (Figure 7.6(h)).

Suite à ces considérations, nous proposons de représenter c_v et ses adjacences dans T_P comme suit.

Soit v un sommet d'un polytope P. Nous appelons $f_0, f_1, \cdots, f_{N-1}$ les polygones du bord de P adjacents à v. La Figure 7.7 (à gauche) montre comment sont nommés les polygones, arêtes et sommets voisins du sommet v.

Soit S_v la sphère unité centrée en v. Pour $i = 0..N-1$, nous notons π_i le plan support du polygone f_i. Chaque plan support π_i intersecte S_v selon un équateur $\mathcal{E}_i$. Soit

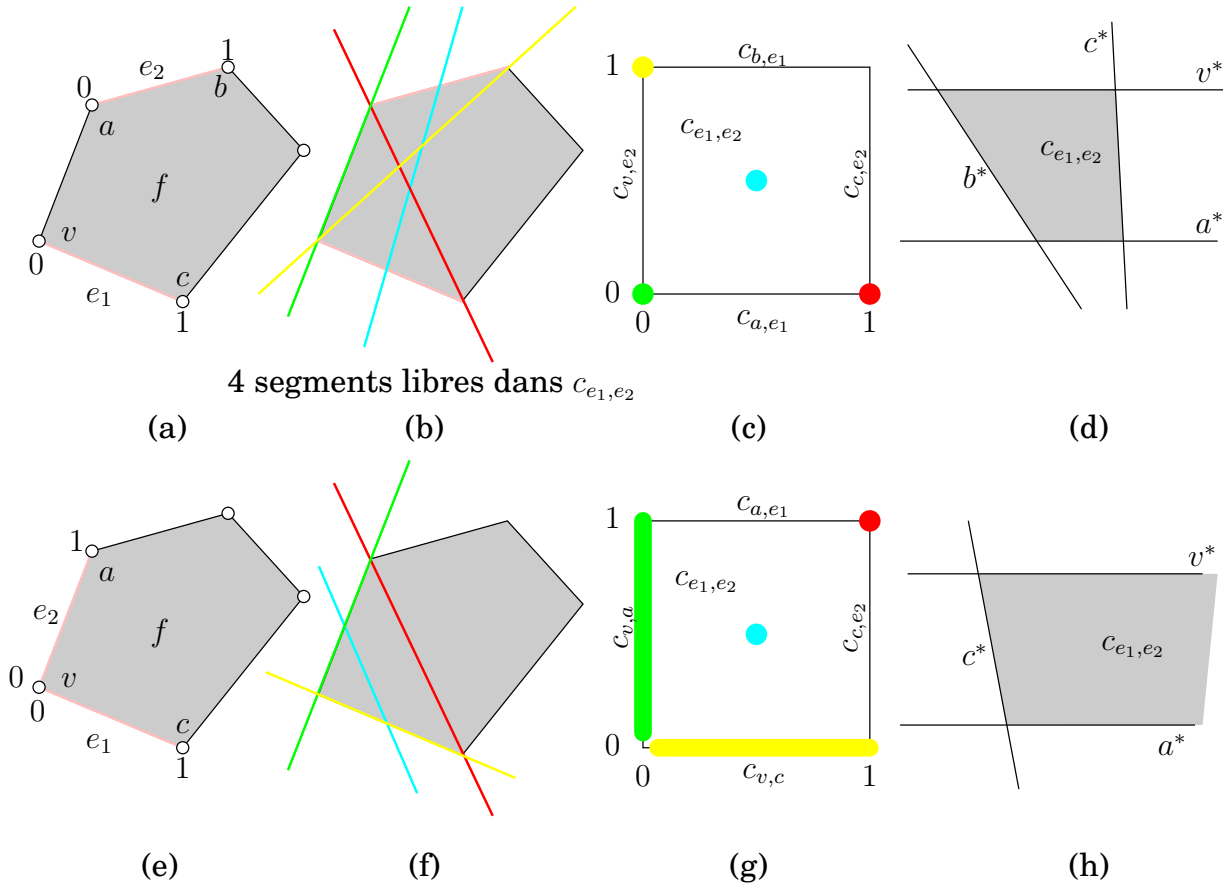


FIG. 7.6. (a), après paramétrisation des arêtes e_1 et e_2, la cellule c_{e_1,e_2} **(b)** peut-être paramétrée sur le carré $[0,1]^2$ **(c)**, ou comme un polygone convexe dans l'espace dual du plan support de f **(d)**. Si e_1 et e_2 partagent le sommet v **(e)**, la paramétrisation de c_{e_1,e_2} n'est pas continue en $(0,0)$ **(g)**. Elle reste en revanche continue sur un triangle de l'espace dual **(h)**.

A_v l'arrangement des N équateurs ainsi définis. A_v comprend $N(N-1)+2$ faces (des polygones sphériques) et $2\binom{N}{2}$ sommets. Remarquons que deux faces de cet arrangement (ou une seule, si on les identifie dans $\mathbb{RP}^2$) correspondent à des segments libres maximaux non-tangents au polytope P ; nous ne considérerons donc pas ces faces. Toutes les autres faces (au nombre de $\frac{N(N-1)}{2}$ dans $\mathbb{RP}^2$) forment une décomposition de l'ensemble des segments libres maximaux tangents à P en son sommet v. Soit p un point de l'arrangement A_v, nous noterons $s(p)$ le segment libre maximal passant par v (le centre de la sphère) et p. Nous décrivons maintenant les éléments de A_v en tant de cellules de T_P.

• Soit ϵ une arête sphérique de A_v portée par l'équateur $\mathcal{E}_i$. L'ensemble $\{s(p)|p \in \epsilon\}$ est une 1-cellule de T_P
  - de type V ou de type VE, selon qu'elle est générée ou non par une arête du polytope P.
  - adjacente à une ou deux 2-cellules dans A_v.
  - adjacente à la 2-cellule $c_{e_i,e_{i+1}}$ (indices modulo N).

• Soit p un sommet de A_v, intersection de deux équateurs $\mathcal{E}_i$ et $\mathcal{E}_j$. $s(p)$ est une 0-cellule de T_P
  - de type VV si le segment $s(p)$ est supporté par un autre sommet que v, et de type indéterminé (ou V), sinon.
  - adjacente à au plus quatre 2-cellules dans A_v.

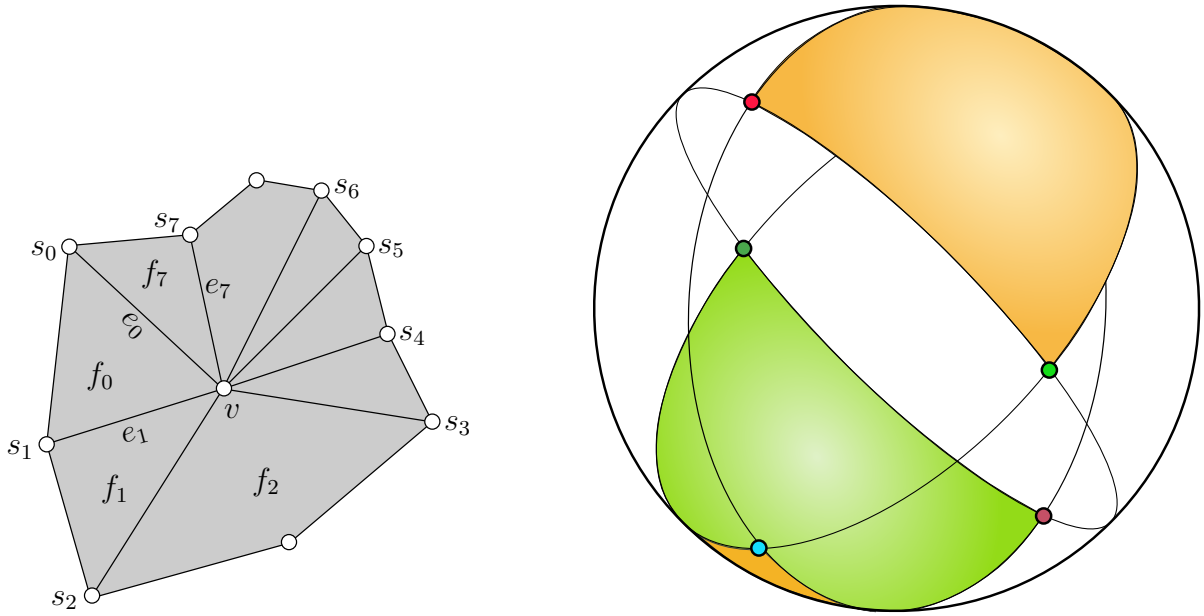


FIG. 7.7. À gauche, la configuration géométrique du polytope P autour de son sommet v. **À droite**, l'arrangement A_v lorsque que v est adjacent à 3 arêtes sur le bord du polytope P.

  - adjacentes aux deux 2-cellules $c_{e_i,e_{i+1}}$ et $c_{e_j,e_{j+1}}$ (indices modulo N).

• Soit g une face bidimensionnelle de l'arrangement A_v. L'ensemble $\{s(p)|p \in g\}$ est une 2-cellule de T_P
  - de type V.
  - adjacentes à deux 3-cellules de T_P : c_{e_i} et c_{e_j} où e_i t e_j sont les deux arêtes de P adjacentes à v qui sont des arêtes silhouettes par rapport à la direction de vue donnée par le segment (v, p) où $p \in g$. Notons que i et j sont bien indépendants du choix de p dans g.

Remarque 7.3. *Nous n'avons pas décrit ci-dessus toutes les relations d'adjacences entre les cellules de dimensions variées mentionnées. Notre lecteur les déduira facilement.*

Remarque 7.4. *Si $N > 3$ alors l'arrangement A_v fait bien apparaître des 0-cellules générées par... un seul sommet ! Mais leur présence est nécessaire pour représenter correctement toutes les adjacences entre les cellules de T_P.*

Troisième partie

Rendu interactif

Chapitre 8

ZP+ : ombres pochées

Dans ce chapitre, nous étudions un problème plus pratique, qui concerne le dessin rapide d'ombres dures générées par une source lumineuse ponctuelle, et pour lequel nous proposons un nouvel algorithme.

Dans de nombreuses applications interactives de type jeu-vidéo en 3D ou visite virtuelle d'un lieu, le dessin des ombres portées améliore le réalisme de la simulation et donnent des indices importants sur les relations spatiales des objets les uns par rapport aux autres. De nombreuses méthodes ont été développées pour permettre le calcul rapide des parties ombrées d'une séquence d'images. Une de ces méthodes est connue sous le nom de *stencil shadow volumes*. Elle utilise la géométrie de la scène 3D pour en extraire les volumes de l'espace non éclairé par une source lumineuse ponctuelle. Ces volumes sont ensuite dessinés dans un image appellée « *stencil buffer* » pour y obtenir un masque indiquant quels pixels sont dans l'ombre de la source lumineuse ponctuelle. Il existe deux techniques de base pour « dessiner » ces volumes d'ombres dans le *stencil buffer*. Elles sont connues sous les noms de *Z-pass* et *Z-fail*. La première technique présente quelques défauts dont la correction fait l'objet de ce chapitre.

Le marquage des parties ombrées d'une scène dans le *stencil buffer* (tampon pochoir) du matériel graphique 3D permet par la suite de masquer les pixels ombrés lors du calcul de l'éclairage. La première méthode découverte pour ce faire est connue sous le nom de *Z-pass*. Bien qu'elle soit la plus populaire, *Z-pass* n'est pas toujours correcte. La méthode *Z-fail*, symétrique à *Z-pass*, est correcte (nous dirons *robuste*) mais possède quelques inconvénients. Nous présentons un nouvel algorithme, nommé *ZP+*, qui ajoute une passe de correction à la méthode *Z-pass*. *ZP+* est bien adapté aux capacités des cartes graphiques récentes à détecter rapidement les triangles invisibles ; *ZP+* rend également possible l'utilisation de formatage des données optimisées pour l'affichage rapide, tel que les *triangle-strips* (bandes de triangles), qui sont inutilisables par la méthode robuste *Z-fail*. Alors que *Z-fail* peut être jusqu'à 80% plus lent que *Z-pass*, nous observons expérimentalement que *ZP+* est typiquement moins de 10% plus lent que *Z-pass*. En fin de chapitre, nous comparerons les trois méthodes : quand une scène contient beaucoup d'informations géométriques, *ZP+* est toujours plus rapide que *Z-fail*. Nous expliquons également pourquoi, dans certaines situations, *Z-pass* (et donc *ZP+*) est plus lent que *Z-fail* sur des cartes graphiques récentes ; une observation surprenante.

Ces travaux ont été effectués en collaboration avec Jared Hoberock (University of Illinois, Urbana-Champaign), John C. Hart (UIUC) et Sylvain Lefebvre (Microsoft

Research), dans le cadre d'une collaboration CNRS-UIUC. Ils ont été présentés au *Symposium on Interactive 3D Graphics and Games* [76].

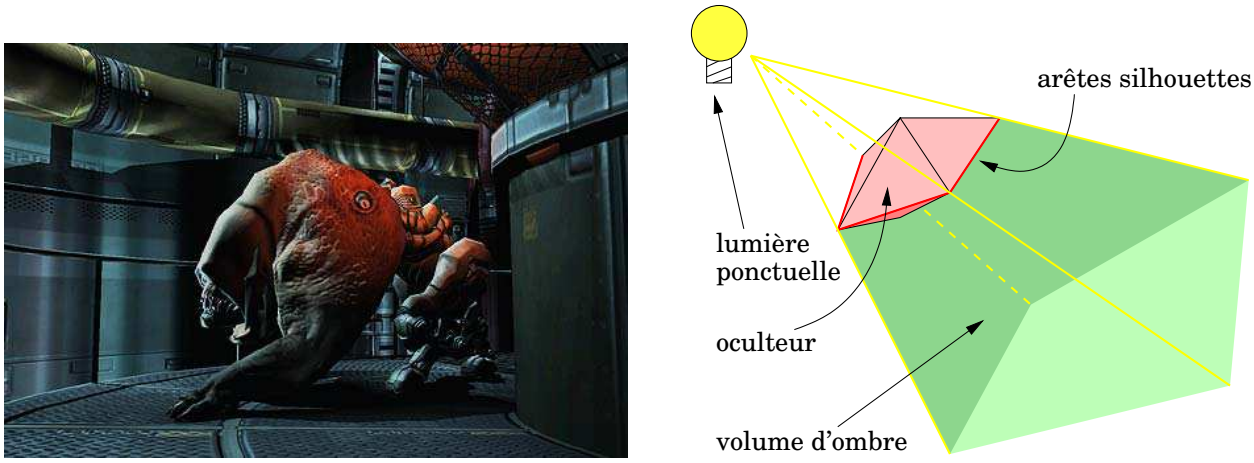


FIG. 8.1. À gauche, une capture d'écran du jeu vidéo Doom 3. Le dessin d'ombres en temps-réel permet de donner cette ambiance dramatique. **À droite**, illustration d'un volume d'ombre.

8.1 Introduction

Les ombres sont une indication visuelle très importante pour la perception des formes et distances dans une représentation 2-D d'une scène 3D. Pour certaines scènes, c'est même la seule indication de la relation spatiale entre deux objets disjoints.

La recherche de méthodes nouvelles pour rendre les jeux vidéo, environnements virtuels et applications graphiques interactives plus réalistes, a vu un essor récent de travaux portant sur la génération d'ombres en temps-réel [2, 5, 23, 54, 69, 82, 94]. L'état de l'art actuel, en matière d'ombrage temps-réel, se fonde sur une utilisation efficace des volumes d'ombres. On peut en apprécier les résultats dans les derniers « moteurs graphiques » des jeux vidéo, tel que celui utilisé pour Doom 3, de iD Software (Figure 8.1, à gauche). Dans ce chapitre, nous décrivons un nouvel algorithme appelé *ZP+*, qui accélère la technique de calcul des ombres dures utilisée dans Doom 3 et d'autres applications et jeux interactifs.

Étant donnée une source lumineuse ponctuelle l, on définit le volume d'ombre d'un objet polyédrique comme le maillage polyédrique de la frontière de la région de l'espace invisible depuis l [35] (voir Figure 8.1, à droite). La technique du *stencil shadow volume* utilise ce maillage pour déterminer si un point doit être éclairé ou pas (en testant si le point est inclu dans le volume d'ombre). Cette technique est populaire et très utilisée car elle permet le dessin d'ombres bien nettes par opposition au *shadow mapping*, autre technique populaire, qui souffre de la résolution limitée de la texture d'ombre utilisée. La technique de *shadow mapping* transforme, en effet, la position 3D d'un fragment dans un repère attaché à la lumière, et compare sa profondeur à celle stockée dans une carte de profondeur précalculée dans ce système de coordonnées.

Un fragment est l'ensemble des informations nécessaires (dans le système OpenGL) au traitement et à l'affichage d'un pixel. Il contient entre autres les informations de couleur, position écran, position 3D, normale, etc...

Description du volume d'ombre. Par abus de langage, nous identifierons le volume d'ombre avec sa frontière polyédrique. Un polygone (ou *une face*) orienté définit un demi-espace positif (du côté où pointe son vecteur normal). Une face d'un objet de la scène est dite *face-avant,* par rapport à la lumière, si cette dernière est dans le

demi-espace positif de la face. Sinon, c'est une *face-arrière*. Une *arête silhouette* est une arête d'un polyèdre dont les deux faces adjacentes sont respectivement une face-avant et une face-arrière. Le volume d'ombre peut alors être décomposé en 3 parties : le *bouchon-éclairé* est l'ensemble des faces-avant. Le *bouchon-ombré* est l'ensemble des faces-arrière « éloignées » vers l'infini par une homothétie dont le centre est la position de la lumière. Enfin, les *côtés* sont un ensemble de quadrilatères résultant de l'extrusion des arêtes silhouettes vers l'infini, par la même homothétie. Cette définition ne correspond pas vraiment à la frontière proprement dite du volume d'ombre, mais cette structure est plus facile à calculer et on peut facilement vérifier qu'elle ne change pas la correction des algorithmes décrits dans ce chapitre. C'est donc cette structure en 3 parties que l'on utilise en pratique.

Il existe plusieurs variantes de la technique de *stencil shadow volume*. La méthode *Z-pass* est la plus ancienne [73]. Elle est illustrée ici sur la figure 8.2 (gauche). La méthode *Z-pass* commence par initialiser le *stencil buffer* à zéro et le *z-buffer* (tampon de profondeur) avec les valeurs de profondeur des parties visibles de la scène. Ensuite, les côtés du volume d'ombre sont *rasterisés*. Pour chaque pixel ainsi dessiné, *Z-pass* incrémente la valeur correspondante du *stencil buffer* si le vecteur normal de ce fragment fait face à la caméra, et la décrémente sinon. Notons que le *stencil* est modifié uniquement pour les fragments du volume d'ombre qui passent le test en Z, c'est-à-dire pour les fragments qui se trouvent géométriquement *devant* les objets de la scène (d'où le nom de la méthode : *Z-pass*). Nous pouvons alors vérifier qu'en général, les pixels dont la valeur dans le *stencil buffer* est nulle correspondent à des points 3D éclairés par la source lumineuse, alors que ceux dont la valeur est non nulle sont bien dans l'ombre. Pour s'en convaincre, il est utile de « visualiser » l'algorithme autrement. En réarrangeant les opérations effectuées sur le *stencil buffer*, nous voyons que la méthode est équivalente à « lancer », pour chaque pixel, un rayon depuis la caméra, dans la direction donnée par la position de ce pixel et des paramètres de la caméra, puis à compter le nombre d'entrées et sorties du volume d'ombre jusqu'à ce que le rayon touche un objet de la scène. Si le nombre de sorties est égal au nombre d'entrées, alors le point de l'objet touché est éclairé ; sinon il est dans l'ombre de la lumière. Nous pouvons donc détecter, avec une résolution égale à celle de l'image à dessiner, quelles parties sont éclairées ou dans l'ombre.

Z-pass ne marche plus quand le volume d'ombre traverse le plan-avant et plus précisément, le rectangle-avant (la figure 8.3 illustre la terminologie que nous utilisons). Dans l'exemple de la figure 8.2 (gauche), le fragment le plus en haut est classifié incorrectement comme étant dans l'ombre car le volume d'ombre créé par le cercle est coupé par le plan-avant. Ainsi, une partie du volume d'ombre n'est pas détectée par le « rayon » lancé depuis la caméra et le compte des entrées/sorties devient faux. De nombreuses méthodes ont été proposées pour boucher le volume d'ombre coupé à l'aide de polygones additionnels [14, 40, 82]. Mais, elles s'avèrent coûteuses en temps de calcul et souffrent toutes de problèmes de robustesse qui font apparaître des artefacts très visibles.

Ce problème de coupe par le plan-avant a mené certains, dont Carmack [20], à examiner la méthode dite « *Z-fail* » qui modifie le *stencil buffer* pour les fragments du volume d'ombre qui échouent au test de profondeur ; voir figure 8.2 (milieu). Nous verrons que *Z-fail* déplace le problème du plan-avant au plan-arrière. Mais, cette spécificité permet de le traiter de manière robuste en « déplaçant » le plan-arrière à l'infini [54] (pour *Z-pass*, il n'est pas possible de déplacer le plan-avant à distance nulle de

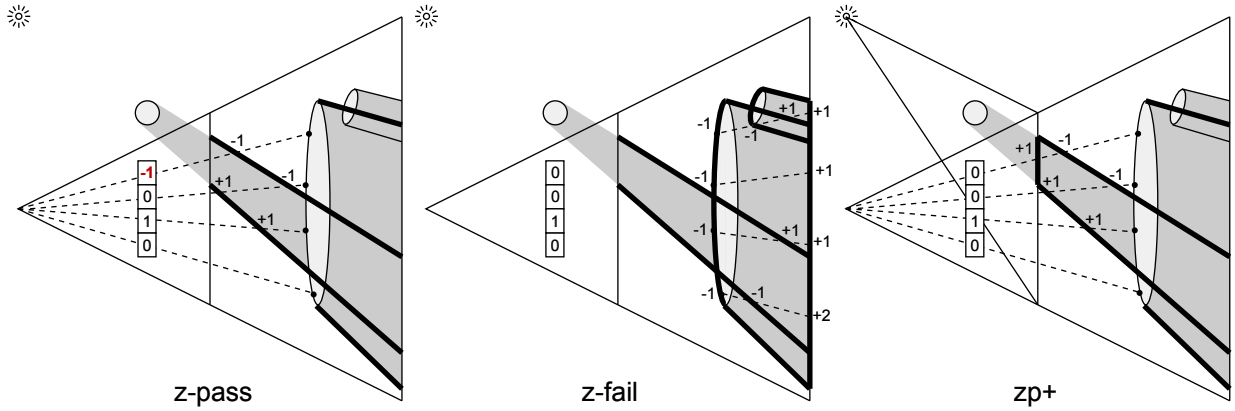


FIG. 8.2. À gauche, la méthode *Z-pass* dessine d'abord la scène dans le *z-buffer*, puis dessine les volumes d'ombre (en gras), en incrémentant ou décrémentant le *stencil buffer* quand un fragment passe le test en Z. *Z-pass* ne donne pas une ombre correcte quand le volume d'ombre est coupé par le plan-avant du système de visualisation. À l'inverse, la méthode *Z-fail* (**au milieu**) compte les fragments du volume d'ombre qui échouent au test en Z (les fragments invisibles depuis la caméra). *Z-fail* utilise également un « bouchon » sur le plan-arrière pour garantir la correction des sommations dans le *stencil buffer*. **À droite**, notre méthode, *ZP+*, initialise le *stencil buffer* en dessinant la scène vue depuis la lumière pour positionner le *stencil* à la valeur donnée par *Z-pass en l'absence de plan-avant*. Ainsi, *ZP+* rend *Z-pass* aussi robuste que *Z-fail*.

l'observateur). Cependant, cette robustesse est obtenue aux dépens des performances. En effet, pour que le calcul du masque d'ombre dans le *stencil buffer* soit correct, il faut, en plus de dessiner les côtés du volume d'ombre, en dessiner le *bouchon-éclairé* et le *bouchon-ombré* faisant face à la caméra, et à ceux tournant le dos à la caméra et projetés (voir Figure 8.3). De plus, *Z-fail* ne peut tirer avantage des optimisations *hardware* du test de profondeur que *Z-pass* met à profit pour éviter le traitement des fragments du volume d'ombre ocultés de la caméra par la scène 3D.

Le nouvel algorithme que nous présentons, *ZP+* (pour *Z-pass*-plus), allie la robustesse de *Z-fail* à une vitesse plutôt comparable à la méthode *Z-pass*. La méthode *ZP+*, illustrée sur la Figure 8.2 (droite), construit, lors d'une passe de dessin préliminaire, un frustum de vue cisaillé (*sheared frustum*) qui s'étend depuis la lumière jusqu'au rectangle-avant original. Puisque le rectangle-arrière de ce nouveau frustum est aligné avec le rectangle-avant de la caméra, il contient exactement la portion de la scène qui génère un volume d'ombre coupé sur le rectangle-avant initial. Une fois ce frustum de vue mis en place, le fait de dessiner la scène projette ses fragments sur le plan-avant de la caméra exactement là où les volumes d'ombres étaient coupés. Ce dessin permet donc de « réparer » les erreurs induites par la méthode *Z-pass*, et rend cette dernière robuste.

Mentionnons que suite à ces travaux, Samuli Laine [87] propose une méthode pour combiner *ZP+* et *Z-fail* en utilisant une technique dans les zones de l'image où elle est plus efficace que l'autre. La technique de Laine subdivise l'image en carreaux de petite taille (8×8) et permet de décider dans chaque carreau, selon la configuration géométrique de la scène, s'il vaut mieux utiliser *Z-fail* ou *ZP+* pour minimiser le nombre de pixels touchés au cours du dessin des ombres.

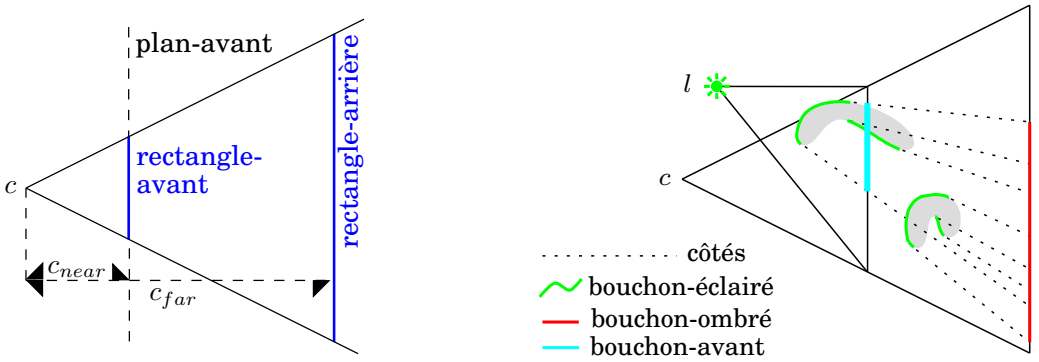


FIG. 8.3. Terminologie utilisée. c est la position de la caméra. l est la source lumineuse ponctuelle. L'image de droite montre les différents éléments qui composent un volume d'ombre : le *bouchon-éclairé* est l'ensemble des polygones faisant face à la lumière. Le *bouchon-ombré* est l'ensemble des polygones qui « tournent le dos » à la caméra et projetés sur le plan-arrière. Le *bouchon-éclairé* et le *bouchon-ombré* sont utilisés par la méthode *Z-fail*. Le *bouchon-avant* est l'intersection du volume d'ombre avec le plan-avant de la caméra ; il est utilisé dans la nouvelle méthode *ZP+*. Les côtés du volume d'ombre (*sides*) sont utilisés dans les trois méthodes *Z-pass*, *Z-fail* et *ZP+*.

Plan du chapitre. Dans la section 8.2, nous comparons *ZP+* aux méthodes déjà existantes, et montrons en quoi *ZP+* est nouveau et utile. La section 8.3 décrit l'algorithme de *ZP+* dans un cadre plus général que la situation décrite sur la Figure 8.2 (à droite), et propose une nouvelle extension OpenGL qui en faciliterait l'implémentation. Dans la section 8.4, nous expliquons pourquoi *ZP+* est plus rapide que *Z-fail*, et, dans la section 8.5, nous en analysons la robustesse en examinant comment la différence entre *rasterisation* et *clipping* peut faire apparaître des petits artefacts, et expliquons comment nous pouvons supprimer ces artefacts. Nous observerons qu'il existe un cas, rare, où les erreurs numériques sont trop grandes pour être corrigées, et pour lequel nous sommes obligés d'utiliser la méthode *Z-fail*. La section 8.6 démontre la qualité et la vitesse de rendu des ombres dessinées par notre nouvelle méthode, que nous avons testée sur deux modèles récents [1] de cartes graphiques 3D.

8.2 Travaux antérieurs

8.2.1 Correction des défauts de *Z-pass*

Nous avons vu dans l'introduction que l'algorithme original *Z-pass* [73] n'est pas correct lorsque le volume d'ombre intersecte le rectangle-avant (la partie du plan-avant visible par la caméra). En effet, *Z-pass* suppose implicitement que ce rectangle-avant est *en dehors* de toute ombre.

Le problème à résoudre est donc de pouvoir initialiser le *stencil buffer* en marquant les parties du *buffer* qui sont dans l'ombre (et qui correspondent à des parties ombrées du plan-avant). Ce problème est plus difficile que de savoir si la position de la caméra (le point c) est dans l'ombre ou pas ; nous pouvons en effet remarquer que c peut être éclairé bien que le rectangle-avant soit partiellement (voire complètement) ombré. La Figure 8.4(a) illustre une telle configuration.

Pour corriger ces défauts, certains proposent de générer des polygones additionnels sur le plan-avant pour *boucher* le volume d'ombre. Batagelo *et al.*[14] suggèrent

[1] Récents il y a deux ans...

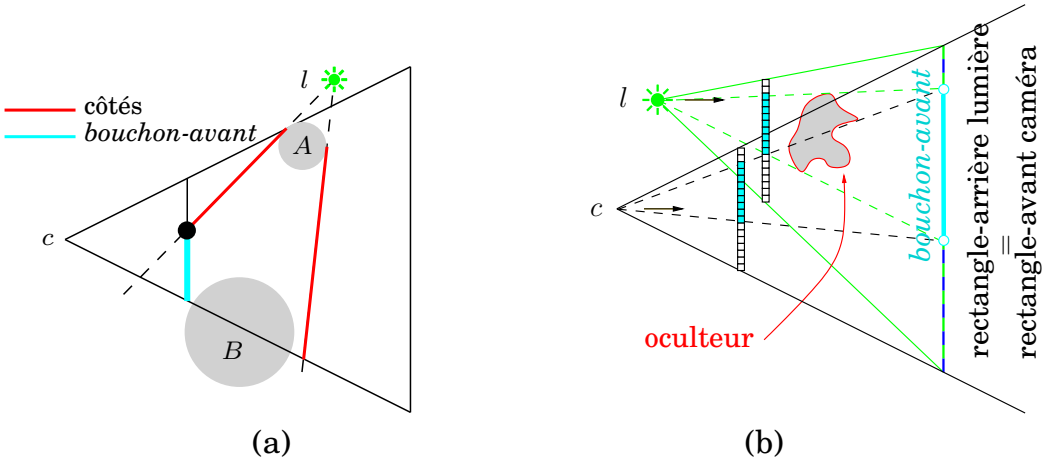


FIG. 8.4. (a) Dans cette situation, *Z-pass* ne marche pas car une partie des côtés du volume d'ombre est supprimée par le plan-avant, ce qui fausse l'éclairage de l'objet B. Notre méthode dessine en plus le *bouchon-avant* (bleu clair), corrigeant ainsi le défaut introduit par *Z-pass*. **(b)** Configuration géométrique du *bouchon-avant* sur le rectangle-avant de la caméra.

de calculer directement puis de mailler l'intersection du volume d'ombre avec le plan-avant pour en « boucher » les parties ombrées. Ce calcul d'intersection est cependant sujet à de nombreuses erreurs numériques qui apparaissent sous la forme de « fendillements » dans l'image finale. Diefenbach [40] propose de dessiner l'extrusion des arêtes silhouettes de l'oculteur vers la source de lumière. Cependant, Everitt et Kilgard [54] montrent que l'approche de Diefenbach n'est pas robuste dans toutes les configurations en montrant des configurations géométriques qui ne réalisent pas d'ombre correcte. Kilgard [82] propose une autre solution au problème de robustesse de *Z-pass*. Celle-ci est assez directe. En effet, il propose de boucher géométriquement le volume d'ombre en projetant explicitement les faces-arrière sur le plan-avant. La procédure est assez complexe car elle nécessite de prendre en compte plusieurs cas, selon qu'un triangle se projette finiment sur le plan-avant ou pas (il se projette alors en partie à l'infini). De plus, des erreurs numériques peuvent « retourner » un triangle (changer son orientation par rapport à la caméra), ce qui contraint à des tests supplémentaires pour détecter ces cas. Aucune de ces méthodes ne se montre satisfaisante à cause de la complexité du traitement géométrique effectué sur le CPU, et dont les résultats numériques ne coïncident généralement pas avec les calculs effectués par le GPU. En fait, pour ce genre d'application graphique, il est communément admis que toute méthode nécessitant des calculs géométriques est intrinsèquement fragile en raison de l'imprécision des calculs en virgule flottante. Ces imprécisions se manifestent ici sous la forme de zones de l'écran « craquelées », où la propriété d'ombre/lumière est inversée, ce qui est, bien-sûr, inacceptable. Ces considérations suggèrent qu'il serait judicieux de trouver une autre solution à ce problème, n'utilisant que les opérations de *rasterisation* disponibles sur les cartes graphiques.

GPU signifie Graphical Processing Unit. *Il s'agit du processeur principal d'une carte accélératrice 3D.*

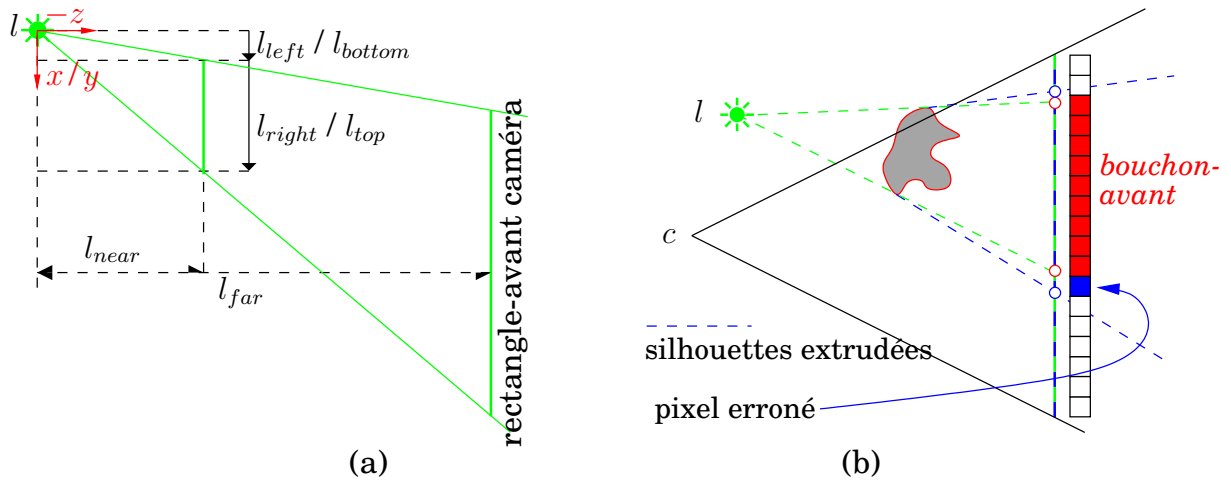


FIG. 8.5. (a) Paramètres de la fonction `glFrustum` pour la mise en place de la matrice de projection OpenGL. **(b)** Mise en évidence de l'erreur introduite par la *rasterisation* de la même arête géométrique par deux transformations différentes.

8.2.2 *Z-fail*

En constatant les difficultés d'implémentation robuste de la méthode *Z-pass*, Carmack [20] découvre qu'il est possible « d'inverser » la manière dont les calculs sont effectués, ce qui aboutit à la méthode *Z-fail*. Cet algorithme, en quelque sorte symétrique à *Z-pass*, supprime le problème lié au plan-avant mais le replace au niveau du plan-arrière. Conceptuellement, *Z-pass* compte les entrées et sorties du volume d'ombre pour un rayon partant de la caméra vers l'infini (comme pour le ray-tracing). Le comptage, pour *Z-pass* s'arrête dès lors que le rayon atteint un objet de la scène. Symétriquement, *Z-fail* compte ces entrées-sorties pour un rayon allant de l'infini vers la caméra, et s'arrête dès lors que le « rayon » n'est plus invisible pour la caméra. Cette symétrie fait clairement apparaître que le problème de coupe du volume d'ombre s'est maintenant déplacé vers le plan-arrière.

Un peu plus tôt, Bilodeau et Songy [16] avaient développé une idée similaire appelée maintenant *reversed Z-pass*. Au lieu de compter les fragments qui ne passent pas le test de profondeur (comme *Z-fail*), leur méthode inverse le test de profondeur et compte les fragments qui passent ce test inversé. Bien que très similaire, *Z-fail* est plus efficace que la méthode du *reversed Z-pass* sur les GPU actuels qui stoppent le traitement d'un fragment aussitôt que le test en profondeur échoue.

Le problème de coupe du volume d'ombre par le plan-arrière nous empêche toujours d'obtenir le bon compte dans le *stencil buffer*. Comme illustré sur la Figure 8.3(à droite) et la Figure 8.2(au milieu), nous devons, pour *Z-fail*, fermer le volume d'ombre à l'aide du *bouchon-éclairé* et du *bouchon-ombré*. Si le *bouchon-ombré* est coupé par le plan-arrière, l'intérieur du volume d'ombre est « exposé » et des valeurs incorrectes sont produites dans le *stencil buffer* car l'entrée du rayon (conceptuel) dans le volume d'ombre ne sera pas détectée pour certains pixels. Everitt et Kilgard [54] résolvent ce problème en repoussant le plan-arrière à l'infini grâce à une matrice de projection modifiée, et en poussant de même le *bouchon-ombré* à l'infini (opération simplement réalisée à l'aide de coordonnées homogènes). Cette méthode *Z-fail*, bien que robuste, ajoute à *Z-pass* le dessin, coûteux en temps, du *bouchon-éclairé* et du *bouchon-ombré*, qui doivent être traités séparément puisque le *bouchon-ombré* est éloigné à l'infini de sa position originale. Cette séparation des deux « bouchons » en fonction de l'orien-

tation des polygones d'un objet, empêche de prendre avantage de la structure du maillage à l'aide, par exemple, de *triangle-strips* ou de *Vertex Buffer Object*.

*Un VBO (*Vertex Buffer Objet*) est un tableau de coordonnées 3D résidant en mémoire vidéo. Il bénéficie donc d'un accès plus rapide par le GPU.*

8.2.3 Carte d'ombre (shadow mapping)

Les cartes d'ombre ont été proposées par Williams [146] comme une technique simple, générale et efficace pour générer des ombres lors du rendu d'une scène 3D. Cette technique est maintenant implémentée « en hard » dans les cartes graphiques 3D. Puisqu'il s'agit d'une technique à base d'images (la profondeur d'une scène est échantillonnée en certains points), la technique des cartes d'ombre fait montre de nombreux défauts d'aliassage dus à la discrétisation de la profondeur, et à la projection en perspective. Plusieurs travaux récents se penchent sur ces problèmes [98, 133, 148]. En particulier, Chong et Gortler [28] décrivent une technique pour les cartes d'ombre qui permet un échantillonnage parfait sur un « plan d'intérêt » choisi au préalable. Notre méthode, *ZP+*, utilise une approche similaire mais dans un contexte différent. *ZP+* construit effectivement une carte d'ombre pour initialiser le *stencil buffer* qui correspond à l'intersection (avec multiplicité) des volumes d'ombres avec le rectangle-avant, « bouchant » ainsi les trous laissés par la méthode *Z-pass* classique. Notons cependant, que nous n'écrivons pas de valeur de profondeur, mais plutôt, pour chaque pixel du *stencil buffer*, le nombre de volumes d'ombre contenant le point correspondant sur le plan-avant. Dessiner le *bouchon-avant* n'est cependant pas si simple que l'on pourrait le penser à première vue. Une implémentation directe fera apparaître quelques pixels isolés où l'ombre n'est pas correcte. Nous verrons à la section 8.5 comment modifier les règles de coupe des polygones sur le plan-avant pour corriger ces défauts.

8.2.4 Optimisation de la méthode des ombres par pochoir

Récemment, de nombreux travaux ont tenté avec succès d'améliorer la méthode des volumes d'ombre. Lloyd *et al.* [94] proposent une technique pour réduire le taux de remplissage (*fill rate*) nécessaire au dessin des volumes d'ombre en rejetant ceux qui n'ont pas d'influence sur le résultat final, et en les coupant pour ne garder que la partie des volumes nécessaire au calcul correct des pixels ombrés. Aila et Akenine-Möller [2] utilisent une méthode hiérarchique qui ne dessine les volumes d'ombre à la précision du pixel, que sur les bords de l'ombre. De manière similaire, Chan et Durand [23] réduisent le taux de remplissage en utilisant d'abord une carte d'ombre pour déterminer l'ensemble des pixels se trouvant proches du bord de l'ombre, où une plus grande précision est requise. Cet ensemble de pixels est alors utilisé comme un masque, qui indique les pixels devant être modifiés lors du dessin des volumes d'ombre. Enfin, Aldridge et Woods [5] proposent une méthode robuste pour produire des ombres correctes pour des objets qui ne sont pas des variétés (*non-manifold*). *ZP+* corrige la méthode de base des ombres par pochoir et est donc complémentaire à toutes celles décrites ci-dessus.

8.3 *ZP+*

Comme décrit précédemment, la méthode *Z-pass* traite les volumes d'ombre efficacement mais incorrectement s'ils intersectent le plan-avant. De plus, les solutions proposées jusqu'à présent se sont montrées moins robustes que *Z-fail* puisque présentant systématiquement des artefacts visibles. La méthode *Z-fail* est robuste (au sens où les ombres produites sont toujours correctes) à condition de dessiner, en plus des côtés, les *bouchon-éclairé* et *bouchon-ombré* du volume d'ombre. *Z-fail* profite également moins bien des optimisations de rejet rapide des fragments invisibles, puisque tous les volumes d'ombre invisibles doivent être dessinés complètement (c'est là l'essence même de *Z-fail*). Cette section décrit notre algorithme, *ZP+*, qui corrige les erreurs de *Z-pass* sur le plan-avant, à l'aide d'une seule passe de rendu des oculteurs dans le *stencil buffer*. Lors de la description de notre méthode, nous nous restreignons à l'interaction d'une source lumineuse et d'un objet oculteur. Les détails concernant une implémentation complète d'un système d'ombres par pochoir peut être trouvée dans [100].

Le coût élevé de *Z-fail* comparé à *Z-pass* nous pousse à chercher une technique plus simple fournissant un résultat identique. Nous pouvons remarquer, sur la Figure 8.2 (droite) par exemple, que les problèmes de *Z-pass* se « situent » à l'intérieur du frustum défini par la position de la lumière et le plan-avant de la caméra. Tout morceau d'oculteur se trouvant dans ce frustum, projette une ombre sur le plan-avant, et introduit une erreur lorsque la méthode *Z-pass* est utilisée.

À l'aide d'une projection centrale dont le centre est la source lumineuse, on peut projeter l'oculteur sur le plan-avant pour fermer le « trou » apparu à cause de la coupe de son volume d'ombre. Les tentatives précédentes pour combler ce trou grâce à l'ajout de géométrie supplémentaire [14, 40, 82] souffrent de calculs longs et peu précis puisque la géométrie est *explicitement* projetée sur le plan-avant. *ZP+* se contente de dessiner la géométrie *existante* à l'aide d'une projection spéciale. La Figure 8.4(b) illustre la configuration géométrique pour l'algorithme décrit ci-dessous :

1. Créer un frustum ayant pour sommet la position de la lumière.
2. Si la caméra et la lumière sont du même côté du plan-avant,
   (a) alors, orienter le nouveau frustum de vue parallèlement au frustum de vue original (celui de la caméra).
   (b) sinon, l'orienter dans le sens opposé.
3. Aligner (par cisaillement) le plan-arrière du nouveau frustum de vue avec le plan-avant du frustum de la caméra. Voir Figure 8.5a.
4. Dessiner la scène en rejetant les *back-faces* (par rapport au nouveau frustum), en ajoutant 1 au *stencil buffer* pour chaque fragment dessiné.
5. Continuer la méthode standard *Z-pass* pour finir le calcul des pixels ombrés dans l'image.

Nous positionnons donc un frustum de vue perspective cisaillée ayant pour sommet la position de la lumière, que nous appelons le *frustum de la lumière*. Le cisaillement permet d'aligner son rectangle-arrière avec le rectangle-avant de la caméra. Dès lors, dessiner l'oculteur génère correctement les fragments qui vont fermer la portion ouverte du volume d'ombre sur le plan-avant. Nous utilisons ensuite ces fragments (à l'aide du *stencil test*) pour initialiser correctement la valeur du *stencil buffer*, redonnant ainsi une propriété de robustesse à la méthode *Z-pass*.

Cet alignement fournit une identification entre tous les pixels du *stencil buffer* avec les segments connectant la lumière et le point correspondant sur le plan-avant. Ainsi, la nouvelle projection perspective cisaillée permet d'initialiser correctement le *stencil buffer* en dessinant les polygones faisant face à la lumière.

Techniquement, la mise en place du frustum de la lumière s'effectue d'abord à partir de la matrice de positionnement des objets (MODELVIEW_MATRIX). Cette dernière est multipliée par la gauche par une matrice de translation, et, éventuellement, l'orientation opposée est obtenue par une rotation de $180°$ autour de l'axe des Y. La projection perspective (PROJECTION_MATRIX) est ensuite mise en place comme expliquée sur la Figure 8.5(a).

Lors du dessin de l'oculteur pour le frustum de la lumière, nous configurons la librairie OpenGL pour que le *stencil buffer* soit incrémenté pour chaque fragment dessiné et que les polygones tournant le dos à la lumière soient rejetés. Une fois ce dessin fini, chaque pixel du *stencil buffer* contient exactement le nombre de volumes d'ombre contenant le point correspondant sur le plan-avant. Nous pouvons alors utiliser *Z-pass* pour calculer correctement l'ensemble des pixels ombrés.

8.3.1 Mise en place du frustum de la lumière

La seule addition à *Z-pass* est la rasterisation du *bouchon-avant* dans le *stencil buffer*. Nous décrivons maintenant précisément comment configurer les matrices *modelview* et *projection* d'OpenGL pour le dessin du *bouchon-avant*.

La matrice *modelview* $\mathcal{M}_l$ est vue comme un repère dont l'origine est à la position de la lumière. Ce « repère-lumière » a le même axe vertical Y que le repère-caméra, et un vecteur de direction de vue $-Z$ parallèle à celui du repère-caméra, et orienté vers le plan-avant. Ainsi, le passage de la *modelview*-lumière à la *modelview*-caméra nécessite une translation et éventuellement une rotation de $180°$ autour de l'axe Y si la caméra et la lumière sont de part et d'autre du plan-avant.

La matrice *projection* $\mathcal{P}_l$ décrit une projection perspective décentrée. Nous pouvons la mettre en place à l'aide de la fonction `glFrustum`. Cependant, pour minimiser les erreurs numériques, nous lui affectons directement ses coefficients comme suit :

$$\mathcal{P}_l = \begin{pmatrix} \frac{2\alpha l_{far}}{c_{width}} & 0 & \frac{-2\Delta_x}{c_{width}} & 0 \\ 0 & \frac{2l_{far}}{c_{height}} & \frac{-2\Delta_y}{c_{height}} & 0 \\ 0 & 0 & \frac{l_{near}+l_{far}}{l_{near}-l_{far}} & \frac{2l_{near}l_{far}}{l_{near}-l_{far}} \\ 0 & 0 & -1 & 0 \end{pmatrix}$$

où :

- $\alpha = 1$ si la lumière et la caméra sont du même côté du plan-avant de la caméra, $\alpha = -1$ sinon.
- c_{width} (*resp.* c_{height}) est la dimension horizontale (*resp.* verticale) du plan-avant de la caméra.
- $\Delta = l - c$, exprimé dans le repère-caméra, i.e., la position de la lumière l dans ce repère.

La matrice ci-dessus est obtenue directement à partir de la matrice fournie dans le livre de référence d'OpenGL et à l'aide de la Figure 8.5(a).

Nous envoyons maintenant la géométrie de l'oculteur dans le pipeline graphique pour rasteriser le *bouchon-avant*. Puisque nous voulons compter pour chaque pixel du

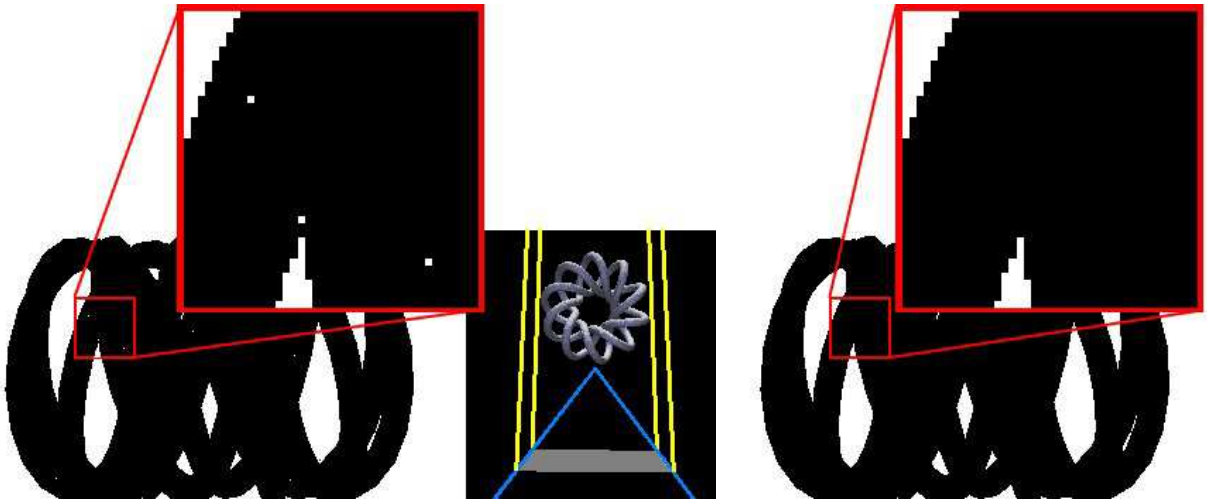


FIG. 8.6. À gauche, artefacts : le zoom révèle 3 pixels éclairés qui devraient être dans l'ombre. Sur l'image complète, on peut voir en tout 8 pixels dont l'éclairage est ainsi inversé. **Au milieu**, vue de dessus. Le rectangle gris (en bas) est le *near-rectangle* de la caméra. Le frustum de la lumière est en jaune, et le frustum de la caméra en bleu. Le récepteur de l'ombre est un plan placé devant la caméra (non visible ici). **À droite**, une ombre sans artefact.

stencil buffer le nombre de volumes d'ombre intersectant le point 3D correspondant, nous configurons OpenGL afin de :

(1) rejeter les faces à l'envers (*resp.* à l'endroit) si $\alpha = 1$ (*resp.* $\alpha = -1$).

(2) incrémenter le *stencil buffer* pour chaque fragment dessiné.

(3) désactiver l'écriture dans les tampons de couleur et de profondeur.

(4) désactiver le test de profondeur.

Ensuite, nous suivons strictement la méthode *Z-pass*.

8.3.2 Mise en place du plan-avant de la lumière

Lors de la rasterisation du *bouchon-avant*, seule la partie de l'oculteur contenue dans le frustum de la lumière produit des fragments visibles sur *stencil buffer*. Appelons « pyramide de la lumière » la pyramide ayant pour sommet la position de la lumière et pour base le rectangle-avant du frustum de la lumière. Si une partie de l'oculteur se trouve dans cette pyramide, elle ne sera pas dessinée dans le *stencil buffer* car elle sera coupée par le plan-avant de la lumière. Il est donc crucial que la pyramide de la lumière soit vide. Si la lumière est à l'extérieur de tout oculteur (ce que nous supposons vrai), il est toujours possible de trouver une distance l_{near} de la lumière à son plan-avant de telle sorte que la pyramide de la lumière soit vide. Cependant, la valeur de l_{near} doit être aussi grande que possible pour éviter de trop grandes pertes de précision numérique. Voici une procédure simple pour calculer l_{near}.

Supposons connue la distance d de la lumière à l'oculteur. Par exemple, d peut être calculée en utilisant les fonctionnalités de la plupart des systèmes de détection de collision, présentes dans de nombreuses d'applications interactives.

Soit d_{max} la plus grande distance entre la lumière et son rectangle-arrière (ou bien le rectangle-avant de la caméra). Alors, $l_{near} = \frac{l_{far} d}{d_{max}}$.

8.3.3 Proposition d'une nouvelle extension OpenGL

Les cartes graphiques récentes de NVIDIA implémentent une extension du système OpenGL appelée NV_depth_clamp qui, au lieu de supprimer les fragments coupés par les plans avant et arrière du frustum, tronque leur valeur de profondeur à celle du plan de coupe le plus proche et dessine effectivement le fragment avec sa nouvelle profondeur. Une fois le fragment transformé, sa profondeur est donc projetée sur le segment $[0, 1]$. Cette extension serait très utile pour garantir que l'oculteur n'est pas coupé par le plan-avant de la lumière. Cependant, en activant cette extension OpenGL, les fragments se trouvant *derrière* le plan-arrière de la lumière resteraient eux aussi visibles, ce que nous ne désirons évidemment pas. Une simple modification portée à cette extension pour « tronquer » sur un des deux plans du frustum (par exemple, le plan-avant) et « couper » sur l'autre (le plan-arrière), permettrait de s'affranchir du calcul de l_{near} et faciliterait l'implémentation de *ZP+*.

8.4 Discussion

Dans cette section, nous soulignons les particularités de *ZP+* qui le rendent plus rapide, en général, que *Z-fail*.

Tout d'abord, en comparaison avec *Z-fail*, *ZP+* élimine le besoin d'un *bouchon-ombré* à l'infini. *Z-fail* nécessite de dessiner l'oculteur deux fois, pour incrémenter et décrémenter le *stencil buffer* pour le *bouchon-éclairé* et le *bouchon-ombré*. *ZP+* n'a pas besoin de dessiner le *bouchon-ombré* car les fragments du volume d'ombre qui influent sur le *stencil buffer* sont ceux qui *passent* le test de profondeur. La partie du volume d'ombre coupée par le plan-arrière n'a donc aucune influence sur le résultat.

Deuxièmement, *Z-fail* ne peut pas tirer profit des divers moyens d'accélérer le dessin, comme les *triangle-strips* ou les *VBO*. Par exemple, les *triangle-strips* empêchent la classification d'un sommet comme appartenant au *bouchon-éclairé et* au *bouchon-ombré* puisqu'un sommet peut n'apparaître qu'une seule fois dans le triangle strip, et ne peut donc être traité qu'une fois par la carte accélératrice 3D (voir Figure 8.7). *ZP+* n'a pas besoin de traiter différemment les sommets d'un maillage et peut donc tirer avantage de ces accélérations.

Lors du dessin du *bouchon-avant* dans la méthode *ZP+*, une très grande partie des triangles sont rapidement rejetés, puisque le volume du frustum de la lumière est bien plus petite que celui du frustum de la caméra utilisé pour dessiner les bouchons avant et arrière dans *Z-fail*. Aussi, puisque *Z-fail* dessine les bouchons dans le repère de la caméra, la classification face-avant (par rapport à la lumière, pour le *bouchon-éclairé*) / face arrière (pour le *bouchon-ombré*) doit être faite en *software*. À l'inverse, *ZP+* dessine le *bouchon-avant* dans le repère de la lumière, et peut donc profiter du rejet automatique des faces arrières (par rapport à la lumière) offert par la carte 3D.

Enfin, du point de vue du traitement des fragments (pixels), nous pensons que *Z-fail* met peu à profit les optimisations du test de profondeur qu'offrent les cartes graphiques récentes, qui peuvent régler ces optimisations pour *rejeter* rapidement les fragments qui échouent à ce test, plutôt que pour rapidement laisser passer les fragments qui passent le test avec succès. Nous validons ces prédictions dans la section 8.6.

ZP+ est compatible avec tous les travaux antérieurs effectués pour optimiser le dessin d'ombres à l'aide de volumes d'ombre et du *stencil buffer* : *hybrid shadows*

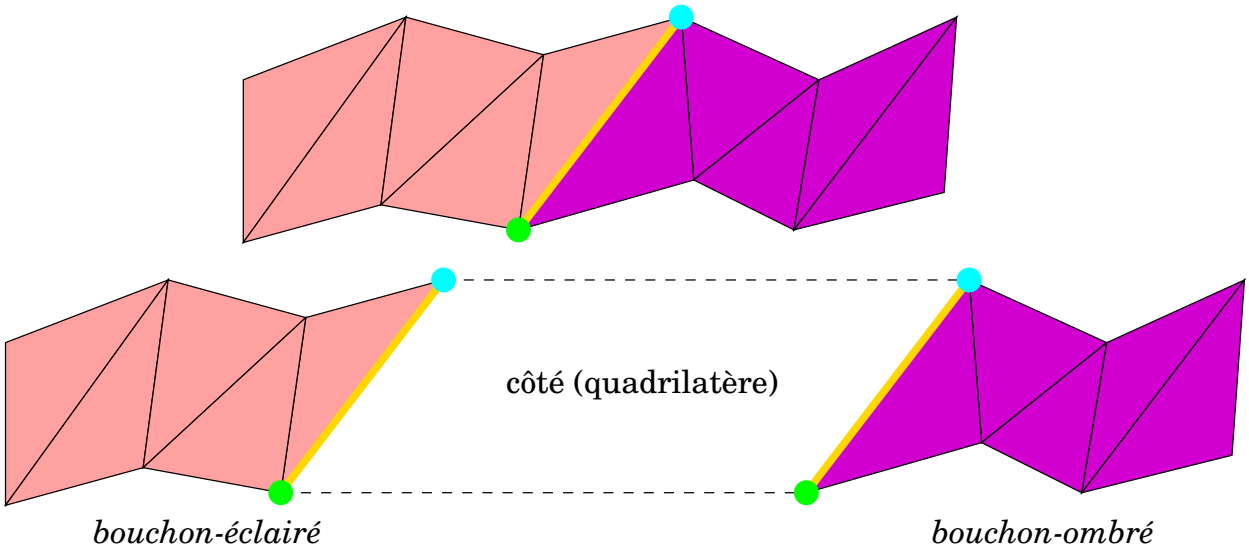


FIG. 8.7. Les deux sommets de l'arête silhouette (jaune) n'apparaissent qu'une seule fois dans la bande de triangles. Ils ne peuvent donc pas être partagés par les *bouchon-éclairé* et *bouchon-ombré*.

[23], *hierarchical shadow volumes* [2], *CC-shadow volumes* (volumes d'ombre coupés et tronqués) [94], et d'autres optimisations diverses [100].

8.5 Supprimer les artefacts

L'implémentation décrite ci-dessus produit quelques artefacts, comme illustrés sur la Figure 8.6 : lorsque le rectangle-avant de la caméra est partiellement ombré, nous pouvons observer quelques (pas plus de 10 en général) pixels épars qui sont éclairés au lieu d'être ombrés ou vice-versa. Ces artefacts ne sont pas nécessairement une gêne et selon l'application, on pourra garder cette implémentation puisqu'ils ne sont pas très visibles ; pour des calculs de facteurs de forme ou d'ombres douces, par exemple. Nous avons en effet testé *ZP+* pendant plusieurs heures avant de nous rendre compte du problème. En observant ces artefacts plus en détail, nous remarquons que ces pixels « inversés » se trouvent sur la frontière du *bouchon-avant*, le long des arêtes où les côtés du volume d'ombre rencontrent le plan-avant (voir le rond noir sur la Figure 8.4(a)).

Dans le repère attaché à l'écran, les coordonnées de ces arêtes sont calculées de deux manières différentes : d'une part, lors du dessin du *bouchon-avant*, ce sont alors des arêtes silhouettes projetées par les matrices spéciales de *modelview* $\mathcal{M}_l$ et de projection $\mathcal{P}_l$; d'autre part, lors du dessin des côtés du volume d'ombre, ce sont les coordonnées de quelques quadrilatères extrudés, et coupés par le plan-avant. À cause de l'arithmétique non exacte, les résultats de ces deux transformations diffèrent légèrement et la *rasterisation* de ces arêtes ne sont plus identiques au pixel près. La Figure 8.5(b) illustre cette disparité.

Dans certaines applications cependant, il peut être désirable de supprimer ces artefacts. Nous présentons une solution que nous avons implémentée avec succès.

La suppression des artefacts se fait en deux étapes. Premièrement, nous choisissons de ne modifier en rien le dessin du *bouchon-avant*. Deuxièmement, nous modifions la manière dont les côtés du volume d'ombre sont coupés sur le plan-avant. Si

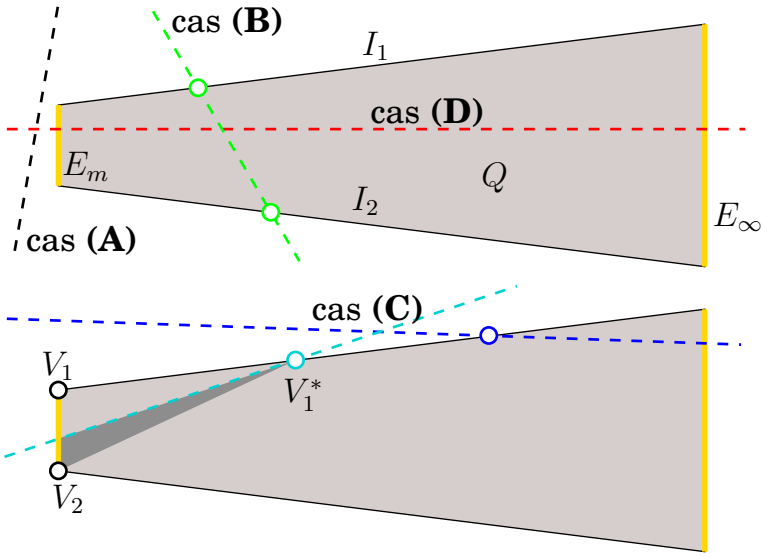


FIG. 8.8. Le quadrilatère Q a deux arêtes infinies, I_1 et I_2, et deux arêtes finies. Une des deux arêtes finies, E_m, est aussi une arête du maillage, alors que l'autre, E_∞, est finie dans l'espace des orientations $\mathcal{S}^2$, et se trouve à l'infini. Les lignes en pointillés indiquent les positions possibles du plan-avant, modulo les symétries.

un sommet A d'un quadrilatère doit être coupé (car il se trouve devant le plan-avant), nous remplaçons sa position sur l'écran par celle obtenue via les matrices $\mathcal{M}_l$ et $\mathcal{P}_l$, puis nous forçons sa profondeur à celle du plan-avant (0, une fois la transformation effectuée). Nous appelons le point ayant ces nouvelles coordonnées « le vis-à-vis avant » de A. Cette transformation est effectuée dans un *vertex program*.

Un vertex program *est un petit programme modifiable, qui est exécuté par la carte graphique 3D pour chaque sommet qu'on lui fournit.*

Jetons un œil sur un des quadrilatères Q qui définissent les côtés du volume d'ombre (Figure 8.8). Quatre cas peuvent être distingués :

- **(A) :** Le plan-avant n'intersecte pas Q. Il n'y a rien à faire, puisque ce cas-là ne peut produire d'artefact à l'écran.
- **(B) :** Le plan-avant intersecte à la fois I_1 et I_2 (ligne verte sur la Figure 8.8). Les deux sommets coupés sont remplacés par leur vis-à-vis avant (les deux petits ronds verts). Ainsi, la « rasterisation » du quadrilatère se joindra parfaitement à la partie correspondante de la frontière du *bouchon-avant*.
- **(C) :** Le plan-avant intersecte une arête infinie I_1 ou I_2 et une arête finie E_m ou E_∞ (lignes cyan et bleu sur la Figure 8.8).
- **(D) :** Le plan-avant intersecte E_m et E_∞ (ligne rouge sur la Figure 8.8).

Dans le cas **(C)**, un ou trois sommets seront coupés par le plan-avant, selon l'orientation de ce dernier. Comme pour le cas **(B)**, nous les remplaçons par leur vis-à-vis avant. Ce faisant, malheureusement, un « trou » triangulaire se forme dans le quadrilatère. Examinons le cas du plan-avant de couleur cyan : supposons que le plan-avant soit orienté de telle sorte que le sommet V_1 soit coupé et remplacé par son vis-à-vis avant V_1^*, tandis que les trois autres sommets restent en place. Alors, le trou prend la forme du triangle gris foncé de la Figure 8.8. Nous bouchons ce trou en dessinant un nouveau triangle $V_1V_2V_1^*$. Les détails d'implémentation ci-dessous expliquent comment nous créons efficacement ces triangles correctifs à l'aide de quadrilatères dégénérés.

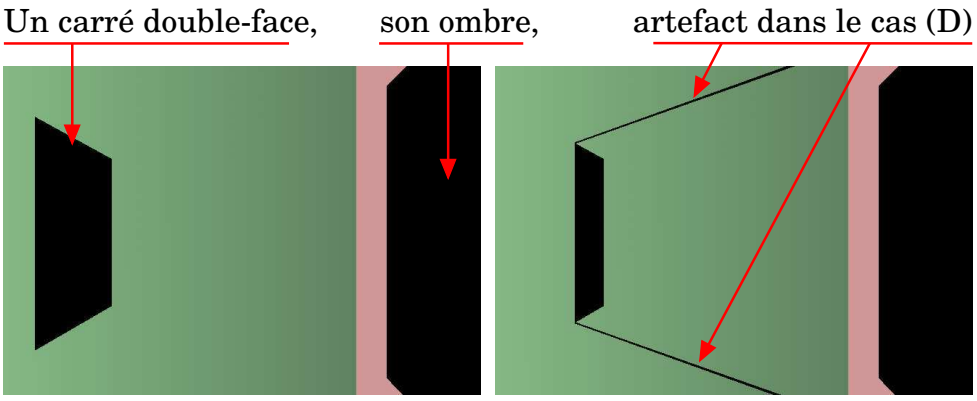


FIG. 8.9. À gauche, l'oculteur est un quadrilatère « double face ». La lumière est sur la gauche de l'image. **À droite**, nous translatons le plan-avant en avant pour que la lumière en soit aussi proche que possible. Des artefacts apparaissent car les erreurs numériques trop grandes translatent le *bouchon-avant* trop à droite. Remarquons qu'en général, le cas **(D)** ne produit pas de si grands artefacts ; l'image de droite montre le pire artefact que nous ayons obtenu.

Le cas **(D)** est plus complexe. Cette situation géométrique apparaît lorsque le frustum de la lumière devient très cisaillé et plat, quand la lumière est à la fois loin de la caméra et proche du plan-avant. La Figure 8.9 illustre les artefacts obtenus dans le cas **(D)**. Dans ce cas, nous ne pouvons pas utiliser le vis-à-vis avant de quelque sommet (qui n'aurait pas la position voulue). Cependant, nous observons que la détection du cas **(D)** est aussi une indication (conservative en pratique) que les matrices $\mathcal{M}_l$ et $\mathcal{P}_l$ ont un très mauvais comportement numérique. Pour ces raisons, quand le cas **(D)** est détecté pour un oculteur, nous changeons temporairement de méthode et utilisons *Z-fail*, seulement pour cette paire lumière / oculteur.

Chaque cas est détecté lors de la recherche des arêtes silhouettes. Nos expériences montrent que, heureusement, le cas **(D)** se produit très rarement. Lors d'une navigation libre (d'un utilisateur) dans nos scènes de test, le cas **(D)** ne se produit presque jamais. Et, ayant créé un chemin de caméra spécial pour révéler le cas **(D)**, nous remarquons que la transition vers *Z-fail* se passe sans accroche, ni autre artefact. Nous discutons plus en détail de la transition temporaire vers *Z-fail* à la fin de cette section.

Le cas **(C)** peut être source d'artefacts, car nous ne pouvons pas corriger parfaitement le quadrilatère à l'intersection du plan-avant et de $E_{m|\infty}$, comme nous l'avons fait pour le cas **(B)**. Cependant, dans nos expériences, nous n'avons pu déceler d'artefact dû à cette déficience, lors d'une utilisation « normale » de la caméra. Nous avons parfois observé quelques artefacts lorsque l'oculteur est coupé par le rectangle-avant, c'est-à-dire que nous pouvons voir la coupure de l'oculteur à l'écran, situation qui est en général à éviter dans toute application car elle défait l'illusion de scène 3D. Le cas **(B)** est le plus courant (comme le cas **(A)**) et nous avons vu que sa correction à l'aide des vis-à-vis avant est exacte. Il en résulte qu'en pratique, nous n'avons plus d'artefact.

8.5.1 Détails sur le *vertex program*

La Figure 8.10 présente le *vertex program* utilisé pour dessiner les côtés du volume d'ombre (un ensemble de quadrilatères). Les coordonnées d'entrée x, y, z décrivent la position d'un sommet d'une arête silhouette. La coordonnée w prend par convention les valeurs $0, 1, 2$ ou 4. Les valeurs 0 et 1 portent la signification standard des coordon-

```
 0 uniform mat4 lightMVMatrix;
 1 uniform mat4 lightPMatrix;
 2 uniform vec4 lightPosition;

 3 void main(void)
   {
 4     vec4 eyePos = gl_Vertex;
       // SHOULD WE EXTRUDE TO INFINITY ?
 5     if( gl_Vertex.w == 0.0 || gl_Vertex.w == 4.0)
       {
 6         eyePos = eyePos - lightPosition;
 7         eyePos.w = 0.0;
       }
       // SHOULD WE STICK THE VERTEX TO ITS ORIGINAL POS ?
 8     if( gl_Vertex.w == 2.0 )
       {
 9         eyePos.w = 1.0;
       }

       // TRANSFORM IN CAMERA FRUSTUM
10     eyePos = gl_ModelViewProjectionMatrix * eyePos;

       // IS IT MOVABLE ...
11     if( ( gl_Vertex.w < 1.5 ) &&
        // ... AND CLIPPED BY THE NEAR PLANE ?
12         ( (eyePos.w <= 0.0)||(eyePos.z < - eyePos.w) ) )
       {   // YES: RE-TRANSFORM THE VERTEX IN LIGHT FRUSTUM
13         eyePos = gl_Vertex;
14         eyePos.w = 1.0;
15         eyePos = lightMVMatrix * eyePos;
16         eyePos = lightPMatrix * eyePos;
17         eyePos.z = -eyePos.w; // PUSH VERTEX ON NEAR-PLANE
       }
       // OK, WE ARE DONE
18     gl_Position = eyePos;
   };
```

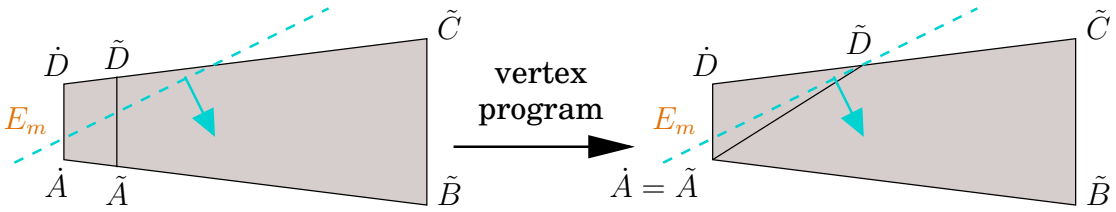


FIG. 8.10. Le *vertex program* utilisé pour dessiner les côtés du volume d'ombre.

nées homogènes : 1 ne change pas la position du sommet, tandis que 0 indique que le sommet doit être extrudé à l'infini en s'éloignant de la lumière (lignes 6, 7). Si le sommet est coupé par le plan-avant (ligne 12), alors le vis-à-vis avant est produit en transformant un nouveau sommet $(x, y, z, 1)$ (ligne 13, 14) de la même manière que lors de la « rasterisation » du *bouchon-avant* (lignes 15, 16), puis en le projetant sur le plan-avant (ligne 17). Un sommet V avec $w = 0$ ou 1 est appelé un *sommet flottant* et noté $\tilde{V}$.

Si $w = 2$ ou 4, le sommet ne bouge pas ($w = 2$) ou bien est extrudé ($w = 4$), et il ne sera *pas* modifié s'il est coupé par le plan-avant (ligne 11). Un sommet V avec $w = 2$ ou 4 est appelé un *sommet cloué* et est noté $\dot{V}$. Les sommets cloués sont utilisés pour créer les quadrilatères correctifs, comme voici :

Dans le cas **(C)**, il nous faut ajouter un triangle correctif pour boucher le « trou » laissé par la déformation du quadrilatère. Plutôt qu'un triangle, nous créons un quadrilatère correctif. Pour deux raisons : premièrement, cela permet de gérer les cas symétriques de façon automatique (ainsi, lorsque le plan-avant intersecte [I_1 et E_m] ou [I_2 et E_m], le même quadrilatère correctif est utilisé). Deuxièmement, et peut-être est-ce le plus important, nous pouvons concaténer la liste des quadrilatères correctifs à celle des quadrilatères formant les côtés du volume d'ombre. Nous n'avons donc pas à traiter cet ensemble géométrique séparément, lors du transfert de données vers la carte graphique.

Soit AD une arête silhouette formant, par extrusion, un quadrilatère Q dans le cas **(C)**. AD est orientée de telle sorte que le triangle ADl pointe en dehors du volume d'ombre (nous rappelons que l est la position de la lumière).

Le cas **(C1)** correspond à la ligne de couleur cyan (le plan-avant intersecte E_m). Le quadrilatère correctif pour **(C1)** est $\{(A, 2); (A, 1); (D, 1); (D, 2)\}$.

Le cas **(C2)** correspond à la ligne de couleur bleue (le plan-avant intersecte E_∞). Le quadrilatère correctif pour **(C2)** est $\{(A, 0); (A, 4); (D, 4); (D, 0)\}$.

Dans tous les cas, deux sommets du quadrilatère correctif fusionnent et le quadrilatère devient un triangle. L'illustration au bas de la Figure 8.10 explique le cas **(C1)** : l'arête E_m de Q est dupliquée pour créer un second quadrilatère (correctif) dont deux sommets sont cloués.

8.5.2 Transition vers *Z-fail*

Nous rappellons que lorsque le cas **(D)** est détecté pour un paire lumière / occulteur, nous devons nous rabattre sur la méthode *Z-fail*. Cependant, dans le contexte plus général d'un système complet d'ombrage par pochoir, c'est-à-dire avec plusieurs lumières et occulteurs, la transition vers *Z-fail* pour une seule paire lumière / occulteur n'est pas possible. En effet, *ZP+* commence par dessiner la scène dans le tampon de profondeur, avec des plans avant et arrière à distance finie de la caméra. Partant, le dessin du *bouchon-avant* de *Z-fail* produira des fragments dont la profondeur (après normalisation dans $[0, 1]$) sera incompatible avec les valeurs stockées dans le tampon de profondeur. Nous proposons trois moyens de résoudre ce problème. Tout d'abord, nous pouvons tout à fait utiliser un plan-arrière poussé à l'infini avec la méthode *ZP+*. Ainsi, les profondeurs resteront compatibles. Sinon, nous pouvons utiliser une alternative à la méthode *Z-fail* : si l'extension OpenGL NV_depth_clamp est disponible, la méthode *Z-fail* peut l'activer et il n'est dès lors plus nécessaire de pousser le plan-arrière à l'infini. L'autre possibilité est d'utiliser l'approche *ZF+* que nous décri-

rons brièvement dans la conclusion. Si nous choisissons *ZF+*, nous avons également la garantie d'un résultat correct, car la transition de *ZP+* vers une méthode alternative se fait lorsque nous rencontrons le cas **(D)**, i.e., quand la lumière est proche du plan-avant. À moins qu'une arête silhouette soit assez longue pour intersecter à la fois les plans avant et arrière, nous sommes sûrs que le cas **(D)** ne sera pas détecté pour cette même paire lumière / oculteur avec l'approche *ZF+*.

Notons que la méthode alternative ne doit être utilisée que pour les paires lumière / oculteur qui génèrent les cas **(D)**. Par souci d'efficacité, il sera également utile de traiter ces paires ensemble.

8.6 Mesures de performance

Nous avons implémenté les trois méthodes : *Z-pass*, *Z-fail* et *ZP+*. *Z-pass* ne produit pas d'ombre correcte en général, mais nous l'utilisons comme référence pour comparer les performances de *Z-fail* et *ZP+*. Pour que cette comparaison soit aussi juste que possible, nous avons optimisé les trois implémentations de ces méthodes :

- **Optimisation commune aux trois méthodes :** les coordonnées des côtés du volume d'ombre sont stockées dans un tableau de sommets dynamique (nous avons utilisé l'extension OpenGL ARB_vertex_buffer_object).
  La deuxième rasterisation des côtés (pour décrémenter les valeurs du *stencil buffer*) s'effectue alors à l'aide d'un seul appel à une fonction OpenGL.
- ***Z-fail* :** une version du maillage de l'oculteur est stockée comme un ensemble de triangles indépendants (sans information de connectivité) dans un tableau de sommets dans lequel chaque entrée contient la position du sommet ainsi que la normale du triangle auquel il appartient. Nous utilisons alors un *vertex program* pour décider si le sommet doit être ou non projeté à l'infini, en fonction de son orientation par rapport à la lumière. Dessiner ce maillage avec ce *vertex program* rasterise ainsi le *bouchon-ombré* et le *bouchon-éclairé*. Ce maillage est dessiné deux fois, pour incrémenter puis décrémenter le *stencil buffer*.
- ***ZP+* :** une version du maillage de l'oculteur est stockée sous la forme de bandes de triangles dans un tableau de sommets, dans lequel chaque entrée ne contient que les coordonnées du sommet. Ce maillage n'est dessiné qu'une seule fois pour rasteriser le *bouchon-avant*. Les côtés sont dessinés avec les quadrilatères correctifs et en utilisant le *vertex program* décrit à la Section 8.5.

Notre scène de test consiste en un oculteur central (voir Figure 8.11) au milieu d'une boîte, et d'une source ponctuelle. Un chemin 3D est créé, puis la caméra le suit trois fois, tandis que la scène est dessinée avec les trois différentes méthodes tour à tour. Le dessin d'une image implique entre autres, la recherche (linéaire) des arêtes silhouettes, la passe initiale de rendu pour initialiser le tampon de profondeur, le dessin du volume d'ombre dans le *stencil buffer*, et enfin, la passe d'éclairage pour illuminer les pixels marqués dans le *stencil buffer*.

Il est difficile de comparer les trois méthodes. En effet, les temps de rendu d'une image sont très dépendants de la scène, tant du point de vue de sa complexité (nombre de triangles) que de sa configuration spatiale. Il dépend aussi en grande partie du nombre de pixels couverts par les côtés du volume d'ombre. De plus, l'ensemble des pixels du *stencil buffer* modifiés pendant le dessin du volume d'ombre diffère entre *Z-pass* et *ZP+* (on ajoute le *bouchon-avant*) et entre *ZP+* et *Z-fail* : l'ensemble des pixels

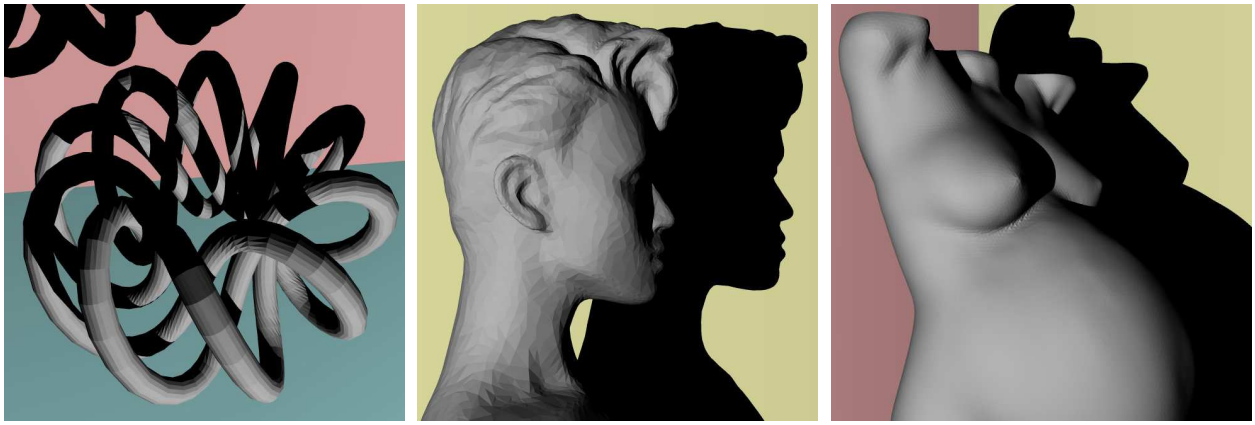


FIG. 8.11. Les oculteurs utilisés pour nos tests. Le *tore vissé* (12000 triangles) est intéressant pour la complexité de son volume d'ombre. La *tête* (20222 triangles) et la *femme enceinte* (83666 triangles) sont utilisés pour tester l'augmentation de la quantité de données géométriques.

du *stencil buffer* modifiés par *ZP+* et l'ensemble de ceux modifiés par *Z-fail* forment une partition de l'ensemble des pixels générés lors de la « rasterisation » du volume d'ombre. La taille respective de ces deux ensembles n'est en général pas équilibrée.

Les mesures de performance ci-dessous incluent la suppression des artefacts décrits à la Section 8.5. Nous avons observé qu'elle n'induit pas de surcoût significatif sur le temps de rendu.

8.6.1 Tests avec une GeForce4

Nous avons d'abord testé *ZP+* sur un PC de bureau (Xeon 2,4 GHz) équipé d'une carte graphique 3D GeForce4 Ti 4800. Les résultats sont présentés sous forme d'un tableau de graphes sur les Figures 8.12 et 8.13.

- Comme attendu, *ZP+* est plus lent que *Z-pass*.
- *ZP+* est plus rapide que *Z-fail*.
- Les performances de *ZP+* sont plus proches de celles de *Z-pass* que de *Z-fail*. Ceci est de plus en plus remarquable quand la complexité géométrique de l'oculteur augmente.

Pour s'adapter à la grande variabilité des temps de rendu, nous avons formé plusieurs chemins de caméra spécifiques, mettant chacun en valeur un aspect particulier de la méthode d'ombrage. La Figure 8.12 montre le comportement de *ZP+* et *Z-fail* relativement à *Z-pass*. Les chemins utilisés dans les deux premières colonnes de la figure placent la caméra en permanence à l'intérieur du volume d'ombre. Ainsi, le résultat produit par *Z-pass* est toujours incorrect et il est obligatoire d'utiliser une méthode alternative, *Z-fail* ou *ZP+*. Le chemin de caméra « *Eye looks away* » met la caméra face à un mur du cube, dos à l'oculteur ; dans ce cas, la plupart des tests de profondeur échouent. Ce test favorise donc *ZP+*. Le chemin de caméra « *Eye looks at occluder* » place la caméra face à l'oculteur, auquel cas la plupart des tests de profondeur réussissent, ce qui favorise *Z-fail*, comme nous pouvons le voir dans le fait que les deux courbes rouge et verte sont plus proches dans la deuxième colonne. Selon ces deux tests, cependant, il est clairement préférable d'utiliser *ZP+*.

La Figure 8.12 nous permet également d'observer que le temps de rendu relatif de *Z-fail* augmente avec la complexité de l'oculteur, alors que celui de *ZP+* se stabilise autour des 10%. L'« applatissement » des courbes de la *femme enceinte* vient du fait que nous atteignons les limites de la capacité de traitement géométrique de la carte

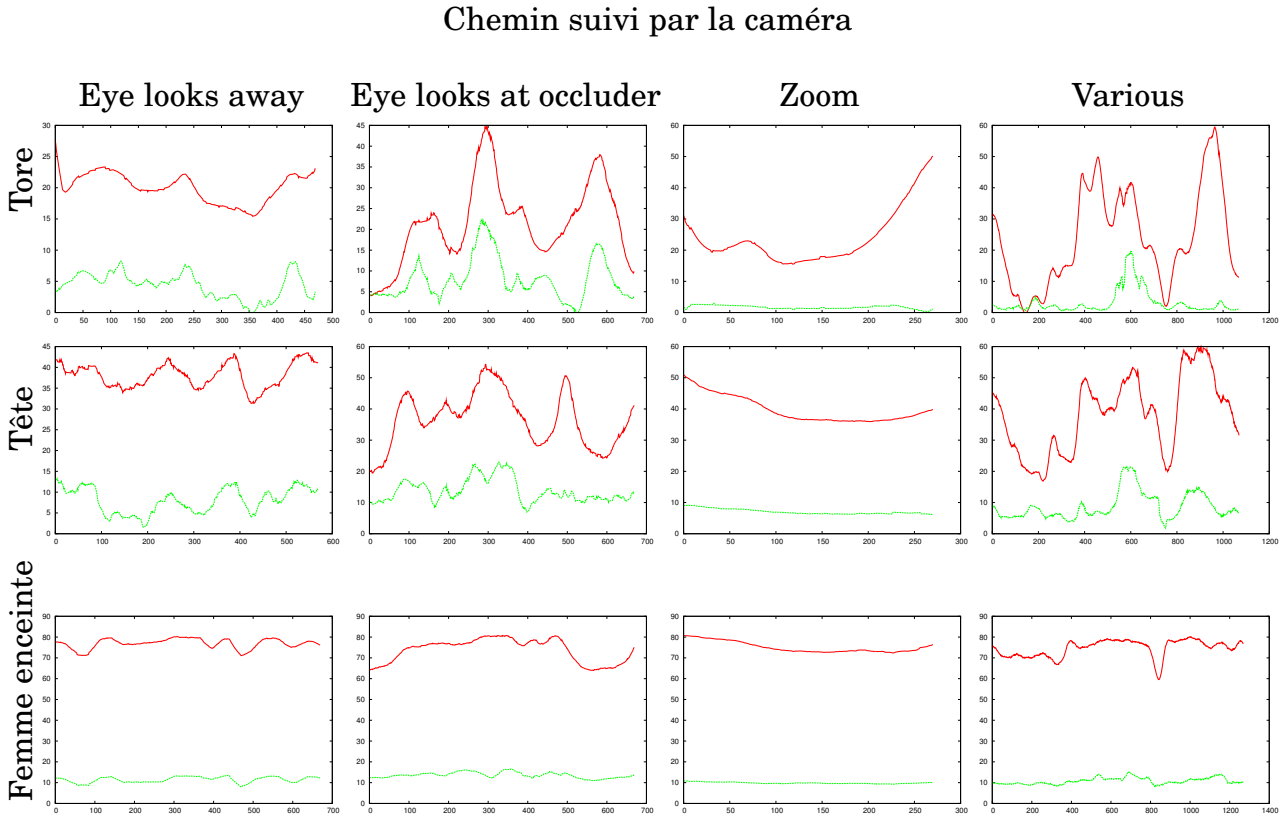


FIG. 8.12. Temps de rendu relatifs – une courbe basse indique des performances meilleures. Chaque graphe représente le temps de rendu d'une image (en %) au cours d'une même animation pour les méthodes *ZP+* (en vert) et *Z-fail* (en rouge) relativement à *Z-pass*. L'axe des ordonnées indique le numéro de l'image dessinée. Par exemple, une valeur de 30 à l'image 100 signifie qu'autour de l'image numéro 100, le temps de rendu moyen pour chaque image est 30% plus lent que *Z-pass*. Chaque ligne du tableau montre les résultats obtenus pour un oculteur. Chaque colonne montre les résultats obtenus pour un chemin de la caméra particulier : les chemins utilisés pour les deux premières colonnes laissent la caméra entièrement dans l'ombre de l'oculteur de telle sorte que *Z-pass* produit un résultat incorrect. Le chemin intitulé *Zoom* est un zoom en avant pour lequel l'oculteur couvre à peu près 100 pixels dans la première image, et couvre tout l'écran dans la dernière ; l'ombre étant portée vers la gauche de l'écran. Le chemin intitulé *Various* fait bouger la caméra aléatoirement autour de l'oculteur.

graphique.

ZP+ se montre bien plus adapté que *Z-fail* pour des oculteurs très détaillés géométriquement, comme illustré sur le graphe de droite de la Figure 8.13.

La Section 8.5 expliquait pourquoi, dans de rares cas, nous ne pouvons utiliser *ZP+* et devons temporairement utiliser une méthode alternative comme *Z-fail*. Dans tous les tests de la Figure 8.12, la transition vers *Z-fail* n'a jamais été nécessaire (le cas **(D)** n'ayant jamais été détecté). Nous avons créé un chemin de caméra spécial pour révéler cette transition. Le chemin fait bouger la caméra lentement quand le plan-avant traverse à la fois la lumière et l'oculteur. Les temps de rendu relatifs sont reportés Figure 8.13 (gauche). Comme attendu, le cas **(D)** est détecté sur quelques images seulement.

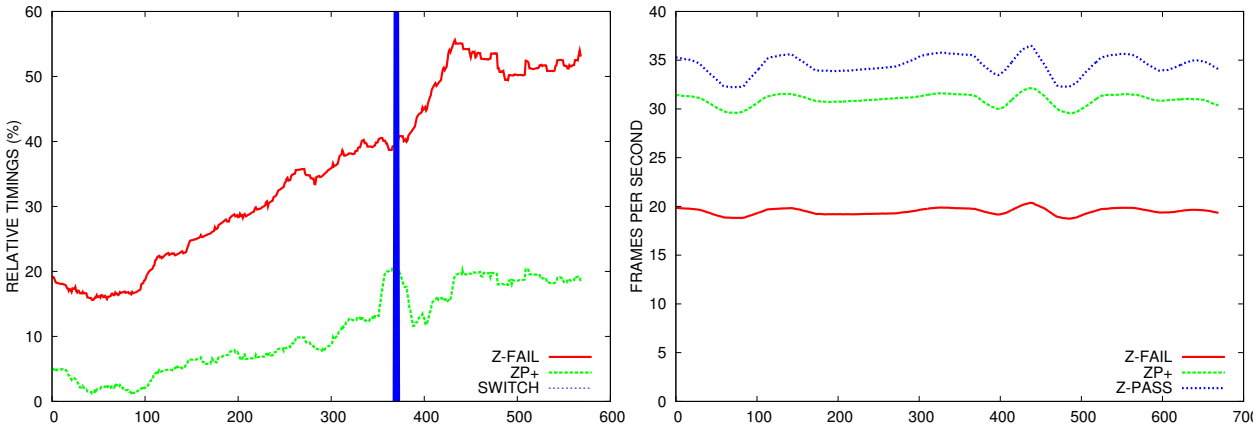


FIG. 8.13. À gauche, la barre bleue indique les images pour lesquelles le cas **(D)** a été détecté et la méthode *Z-fail* utilisée. On peut clairement observer une augmentation du temps de rendu à cet endroit. **À droite**, nombre d'images par seconde pour le chemin de caméra « *Eye looks away* » et l'oculteur *femme enceinte*. *Z-pass* culmine à 35 images par seconde mais produit une ombre incorrecte tout au long du test. *ZP+* produit une ombre correcte et peut encore dessiner 50% plus d'images par seconde que *Z-fail*.

8.6.2 Tests sur une GeForce FX

Nous avons également testé *ZP+* sur un PC de bureau équipé d'un processeur Xeon à 3 Ghz et d'une carte graphique GeForce FX 5900 Ultra. Les résultats sont présentés sur la Figure 8.14. En particulier, nous avons activé l'extension OpenGL EXT_stencil_two_side (non disponible sur la GeForce4) de telle sorte que le rendu des côtés et des bouchons puisse se faire en une seule passe pour *Z-fail*, et le rendu des côtés se fasse en une seule fois pour *ZP+*.

Le traitement des sommets et des fragments sur la GeForce FX est bien plus rapide que sur la GeForce4. En conséquence, les graphes de la Figure 8.14 distinguent *Z-fail* et *ZP+* d'une façon particulière et intéressante, qui n'était pas apparente lors de nos précédents tests. Quand l'oculteur est peu complexe géométriquement, les performances relatives de *Z-pass* sur *Z-fail* dépendent fortement de la configuration spatiale de la scène (voir les graphes à gauche de la figure). Bien que le nombre de sommets traités soit bien plus grand pour *Z-fail*, *Z-pass* (et donc *ZP+*) est effectivement plus lent dans certaines configurations géométriques de la scène. Nous pouvons en déduire que lorsque l'oculteur est simple, le temps de traitement des sommets est négligeable. De plus, les performances de *Z-pass* / *ZP+* varient avec le nombre de fragments qui réussissent le test de profondeur, en opposition à la manière dont varient les performances de *Z-fail*.

Quand la complexité de l'oculteur grandit, *ZP+* est alors systématiquement plus rapide que *Z-fail* car le principal goulot d'étranglement devient le traitement des sommets et non plus le traitement des fragments. La possibilité pour *ZP+* d'utiliser des bandes de triangles prend alors toute sa mesure (voir les mesures reportées sur la droite des Figures 8.13 et 8.14).

Les graphes de la Figure 8.14 montrent clairement que *ZP+* n'inflige qu'un temps de rendu supplémentaire presque négligeable. Ainsi, nous sommes maintenant équipés de deux méthodes pour « rasteriser » un volume d'ombre de manière robuste. Nos observations montrent que pour un oculteur de faible complexité, le coût de cette rasterisation dépend fortement de la proportion de fragments qui réussissent (*Z-pass* et

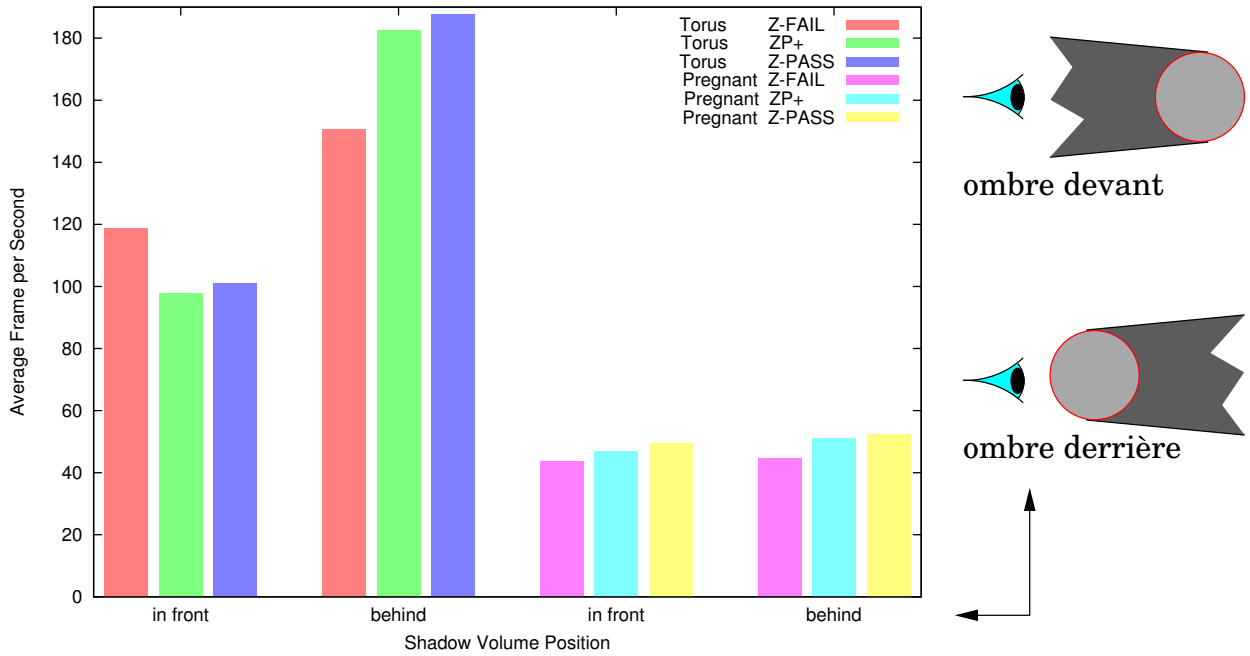


FIG. 8.14. Nombre moyen d'images par seconde sur une GeForce FX 5900 Ultra, en utilisant l'extension OpenGL EXT_stencil_two_side. La caméra fait un zoom en avant vers l'oculteur. Le volume d'ombre admet deux positions relativement à la caméra.

ZP+) ou échouent (*Z-fail*) au test de profondeur : *ZP+* peut être plus lent que *Z-fail* mais cela est simplement dû au fait que, dans une telle situation, *Z-pass* est plus lent que *Z-fail*. C'est une observation surprenante qui ouvre de nouvelles pistes de recherche : il ne semble pas simple, a priori, de décider à l'avance quelle sera la proportion de fragments qui passeront ou échoueront au test de profondeur, étant donné un volume d'ombre et une scène 3D.

Nous avons également effectué des tests avec une carte graphique ATI Radeon 9700. Les résultats sont très semblables à ceux obtenus avec les cartes NVIDIA.

8.7 Conclusion

Nous avons présenté une méthode simple, appelée *ZP+*, pour corriger les défauts de la méthode *Z-pass* pour le rendu d'ombres dures. *ZP+* n'utilise, en plus de *Z-pass*, qu'une seule « rasterisation » de l'oculteur, avec des matrices de transformation spéciales. Bien que *ZP+* soit mathématiquement exacte, la nature discrète du calcul par ordinateur génère quelques petits artefacts. Nous avons décrit d'une part les raisons de leurs apparitions, et d'autre part, un moyen efficace de les supprimer. Nous avons vu que *ZP+* est en général plus rapide que *Z-fail* et se comporte bien mieux lorsque les oculteurs sont complexes. Nous avons également décrit une modification simple de l'extension OpenGL NV_depth_clamp qui simplifierait en partie l'implémentation de *ZP+*.

Les observations faites lors des tests sur la GeForce FX induisent la question suivante : peut-on décider efficacement quelle méthode utiliser (*ZP+* ou *Z-fail*) lorsque l'oculteur est suffisamment simple pour que le traitement des sommets soit négligeable comparé au traitement des fragments ? Samuli Laine [87] apporte une réponse positive à cette question.

Nous pensons qu'il est possible d'exprimer $\mathcal{P}_l$ différemment afin d'éviter les problèmes de précision numérique qui pour le moment nous forcent à utiliser *Z-fail* dans quelques cas rares.

Finalement, nous pouvons remarquer que *ZP+* peut être adapté à une méthode plus proche de *Z-fail* (que nous appellerons *ZF+*), en bouchant le plan-arrière plutôt que le plan-avant, i.e., en dessinant un *bouchon-arrière* plutôt qu'un *bouchon-avant*. Nous souhaiterions examiner *ZF+* plus en détail, et chercher une heuristique qui pourrait décider dans quelle situation utiliser l'une ou l'autre.

Chapitre 9

Décomposition en cellules et portails

La décomposition de données géométriques est couramment utilisée dans la synthèse d'images. Elle aide à l'organisation des informations relatives à une scène, et permet souvent de réduire le coût de certaines méthodes en les appliquant seulement sur les sous-parties nécessaires. L'affichage interactif d'une grande scène 3D en est une application typique. Par exemple, la décomposition d'une scène en composants plus petits, ainsi que la donnée de leurs relations de visibilité mutuelle, est utilisée pour éviter de dessiner la scène entière, en ignorant les composants dont l'invisibilité est certaine [140]. Pour effectuer un calcul d'une solution de radiosité, il est nécessaire de calculer l'ensemble des objets visibles depuis certaines régions de l'espace, comme une source lumineuse, ou un morceau de surface réfléchissant [45, 103]. Ce genre de subdivision est aussi utile pour précalculer des relations de visibilité pour le calcul d'éclairage global d'une scène [102, 141], en calculant la puissance des échanges lumineux entre les surfaces seulement là où c'est le plus nécessaire.

Nous présentons une méthode de décomposition automatique d'une scène polygonale en un graphe de cellules et portails (GCP). Le graphe résultant satisfait deux contraintes définies par l'utilisateur : une borne sur le coût de rendu de chaque cellule, et des bornes inférieure et supérieure sur la taille de chaque cellule. Cette décomposition permet le rendu en temps-réel de grandes scènes d'intérieur, et est plus particulièrement adaptée aux ballades virtuelles dans des scènes architecturales, telles que dans les jeux vidéo. Notre méthode est basée sur une subdivision binaire de l'espace, puis sur un algorithme sélectionnant et regroupant certains portails générés par la subdivision. Enfin, le GCP est construit et traité pour satisfaire les contraintes données par l'utilisateur. Nous proposons une métrique pour mesurer la qualité d'un portail, que nous utilisons pour le traitement final du graphe. Notre algorithme de simplification peut aussi être utilisé sur n'importe quel GCP pour réduire sa complexité. Nous présentons la méthode générale et quelques détails pratiques d'implémentation. Les résultats montrent que les portails générés ont de bonnes propriétés géométriques (ils sont la plupart du temps placés sur les portes et les fenêtres).

Pour répondre rapidement à un grand nombre de requêtes de visibilité, on utilise souvent des structures de subdivision de l'espace comme les arbres BSP (*binary space subdivision*), des boîtes englobantes hiérarchiques, ou des graphes de cellules et portails. Un arbre BSP représente une subdivision hiérarchique de l'espace dans laquelle chaque nœud est coupé par un plan (un *plan de coupe*) pour créer deux sous-nœuds. Il

est en général souhaitable d'obtenir un arbre de petite taille et bien balancé, mais il est difficile de calculer de telles subdivisions [114]. Ce sont ainsi des heuristiques qui sont utilisées pour trouver, à chaque niveau de la subdivision, un bon plan de coupe souvent choisi parmi les plans supports des polygones qui se trouvent à l'intérieur du nœud. Les hiérarchies de boîtes englobantes sont utilisées pour éliminer les objets extérieurs à la pyramide de vue (*view frustum culling*) quand il est possible de décomposer un objet complexe (par exemple, un bureau) en sous-objets séparés (par exemple, la chaise, le pot à crayons, les stylos, la gomme, *etc.*). Pour des applications de type « visite interactive » (*walk-through*) d'un bâtiment ou jeu vidéo se déroulant en intérieur, les graphes de cellules et portails sont fort utiles pour déterminer rapidement les parties visibles depuis un point de vue donné. Une *cellule* est un polyèdre dont les faces sont soit des polygones « solides » faisant partie du modèle 3D, soit des *portails*. Un portail est un polygone « transparent [1] » qui connecte ou *recolle* deux cellules adjacentes, encodant ainsi explicitement la relation spatiale et la visibilité existante entre les deux cellules (les polygones d'une cellule qui ne sont pas des portails doivent être opaques). La technique de dessin de la scène à l'aide d'un graphe de cellules et portails et détaillée à la Section 9.1.

Revenons aux arbres BSP. Ils sont très utilisés dans le domaine des jeux vidéo,[2] d'une part pour le calcul rapide de collision entre un objet mobile et un décor fixe (le décor est décomposé en un BSP), et d'autre part pour précalculer les relations de visibilité entre les cellules d'une subdivision de la scène (on utilise les feuilles d'un arbre BSP décomposant la scène comme « cellules de base »). Des outils comme `qbsp3`, `qvis3` [3] ou `zhlt` [4] sont disponibles gratuitement et utilisés dans de nombreux jeux vidéo. Cependant, la taille des scènes 3D s'est considérablement accrue (et continue d'augmenter [5] !) et le coût du calcul des PVS (*potentially visible sets*) pour chaque cellule de l'arbre BSP devient prohibitif. Cela explique l'attrait de plus en plus important des GCP dans les jeux récents (par exemple, Doom 3 / id Software). Dans un graphe de cellules et portails, les cellules sont plus grandes (comparées aux cellules d'un arbre BSP) et les portails peu nombreux, ce qui permet d'utiliser des techniques de rendu plus efficaces (algorithmique simple avec le « rendu par portails » — voir plus bas), et permet une bonne utilisation des capacités des cartes graphiques. Alors que les arbres BSP sont relativement simples à implémenter, les graphes de cellules et portails sont difficiles à obtenir automatiquement, et sont donc le plus souvent spécifiés à la main par les infographistes, à l'aide d'outils spécialement conçus. Cette tâche n'est pas forcément intéressante pour un artiste 3D et gagnerait à être automatisée. Comme les GCP sont de simples graphes plongés dans un espace 3D, ils pêchent par l'absence d'une forte structuration, comme celle qu'offrent les arbres BSP. De fait, la définition d'un *bon* graphe de cellules et portails est difficile à énoncer (nous y reviendrons plus loin). Des algorithmes très spécialisés de construction de GCP existent et utilisent des règles de « bon sens général » portant sur la taille et l'orientation des pièces dans un bâtiment pour construire un GCP à partir d'un *plan au sol* [102].

[1] Il permet de partitionner la scène, mais ne doit pas être dessiné.

[2] Quoique de plus en plus remplacés par des GCP, ou des structures plus complexes fournies par des développeurs spécialisés [41].

[3] `www.planetquake.com/lfire/Level_Editing/Tutorial/tutorial1.html`

[4] `http ://collective.valve-erc.com/`

[5] Le « moteur » 3D de Unreal 3 est capable d'afficher un décor d'un demi-million de triangles, accompagné d'une pléthore d'effets graphiques (*HDR, soft-shadows, pixel-shader complexes,* etc.). Voir `www.unrealtechnology.com/`

Ils ne sont pas généraux, et en particulier, ne peuvent générer de portails avec une orientation arbitraire.

Il existe une méthode pour construire automatiquement un graphe de cellules et portails sur une scène 3D, qui a été développée par Denis Haumont [71] parallèlement à ce travail [88]. Nous y reviendrons à la Section 9.2.

9.1 Motivations techniques

Un modèle 3D *étanche (watertight)* est un modèle polygonal dont l'intérieur est bien défini. La surface d'un modèle polygonal n'a donc pas de trou puisque la présence d'un trou empêcherait la séparation claire de l'intérieur et l'extérieur du modèle. Soit B un arbre BSP décomposant une scène 3D étanche $\mathcal{S}$. Chaque feuille de B est un polytope (polyèdre convexe) dont les faces appartiennent à deux catégories :

– la face est un sous-polygone d'un polygone de $\mathcal{S}$.
– la face est « transparente ».

Nous pouvons donc voir B comme un graphe de cellules et portails décomposant la même scène $\mathcal{S}$. Cependant, B en tant que GCP, est très peu efficace, comme nous allons le voir.

Étant donné un GCP G de $\mathcal{S}$, il existe deux méthodes pour dessiner $\mathcal{S}$ en tirant parti de G ; l'une est *en-ligne* (*rendu par portails*) et l'autre use de précalculs (PVS). Si le point de vue se trouve dans la cellule C, le *rendu par portails* de $\mathcal{S}$ procède ainsi :

1. Dessiner les polygones de la cellule C.
2. Pour chaque portail p de C,
   Si p est visible, alors
   Retourner récursivement au point 1 appliqué à la cellule se trouvant de l'autre côté de p.

Il existe plusieurs méthodes pour déterminer si un portail p est visible ou pas [3, 97, 141]. Utiliser un arbre BSP comme graphe de cellules et portails n'est pas adapté pour un « rendu par portails » comme décrit ci-dessus. En effet, la plupart des cellules ont très peu de polygones, et le nombre de portails est trop élevé : trop de temps sera passé à déterminer la visibilité des portails. À l'inverse, si les cellules sont assez grandes et que le nombre de portails est restreint (ce qui n'est pas le cas dans un arbre BSP), le rendu par portails peut alors bénéficier du débit et du *fill-rate* très élevés des cartes graphiques récentes. En effet, nous pouvons optimiser l'usage du débit (*bandwidth*) à l'aide de *vertex arrays* à accès rapide ou simplement à l'aide de *display lists*. Ainsi, la géométrie de chaque cellule est envoyée rapidement à la carte graphique en même temps que nous traversons le GCP. De plus, les cartes graphiques permettent maintenant de faire des requêtes d'occlusion (*occlusion query*) qui facilitent grandement l'implémentation du rendu par portails : le *hardware* prend alors la charge de déterminer la visibilité de chaque portail rencontré. (Voir les travaux récents de Wimmer et Bittner [147].)

Pour le rendu d'une scène à l'aide de PVS (précalculés), chaque feuille C du BSP est augmentée d'un PVS : une liste des feuilles potentiellement visibles depuis C. Une feuille C' est potentiellement visible par C si elle est visible depuis au moins un point de C. Si le point de vue se trouve dans la feuille C, il suffit alors d'envoyer à la carte la géométrie de chacune des feuilles contenues dans le PVS de C. Cette méthode fut introduite par Airey *et al.* [3, 4] et Teller et Séquin [142].

Si n est le nombre de polygones dans une scène, son arbre BSP peut contenir $O(n^2)$ feuilles [114] et $O(n)$ polygones visibles par feuille (beaucoup moins en pratique). Comme les cartes graphiques bénéficient maintenant d'une grande capacité mémoire, nous pourrions en tirer parti en stockant la géométrie sur la carte graphique ; cette idée est difficile à concilier avec la méthode de rendu par PVS, car nous aurions un trop grand nombre de toutes petites *display lists*. Concilier le stockage de la géométrie sur la carte et le rendu par portails est plus réaliste cependant, puisque chaque polygone sera alors stocké une seule fois sur la carte et, pourvu qu'un bon GCP soit fourni, la géométrie pourra être dessinée par la carte par paquets de taille quasi optimale.

Certains travaux se sont penchés sur le problème de la compression de PVS : Van de Panne et Stewart [143] obtiennent de très bons taux de compression (avec ou sans perte). Cependant, nous aimerions aussi réduire le temps de précalcul, et donc chercher des alternatives à la construction initiale des PVS.

Notre but est donc de construire, à partir d'une scène statique, un graphe de cellules et portails dont les cellules seront de taille presque optimale vis-à-vis des cartes graphiques récentes à grande capacité de traitement. Nous proposons une méthode de décomposition d'une scène en GCP, capable de gérer des scènes architecturales 3D sans restriction, et offrant un certain contrôle sur la complexité des cellules, produisant ainsi des cellules adaptées au rendu par portails. Nous présentons également une heuristique simple pour évaluer et comparer la qualité des portails.

Dans la section suivante, nous présentons les travaux existant en lien avec les graphes de cellules et portails. La Section 9.2.1 propose une vue générale de notre méthode qui sera détaillée dans les Sections 9.3 et 9.4. La Section 9.5 discute de l'implémentation pratique de notre méthode et nous concluons à la Section 9.6.

9.2 Travaux antérieurs

Jones introduit en 1971 la notion de décomposition (en cellules et portails) de l'espace environnant un objet [81], et l'utilise pour donner une solution au fameux *hidden line removal problem*. Il décompose manuellement l'espace en cellules convexes telles que les faces de l'objet coïncident avec certaines faces des cellules. Le dessin de l'objet se fait à peu près tel que décrit dans l'introduction.

Fuchs *et al.* [55] initient l'application des arbres BSP à des problèmes de visibilité et proposent des heuristiques pour les construire. Un arbre BSP permet de parcourir rapidement l'ensemble des polygones de la scène du « fond » vers le « devant » (et réciproquement), et donne ainsi une certaine efficacité à « l'algorithme du peintre » pour dessiner une scène 3D. Cet algorithme consiste simplement à dessiner un polygone situé derrière un autre avant ce dernier.

Airey *et al.* et Teller *et al.* [4, 142] proposent d'utiliser une décomposition en GCP pour dessiner rapidement des scènes architecturales. La méthode est ensuite mieux analysée et améliorée pour des scènes 2D, axiales 3D et 3D générales [140, 142]. Dans ces travaux, Teller présente des méthodes et heuristiques pour construire un arbre BSP d'une scène, et, pour chaque feuille (ou cellule) de l'arbre, calculer l'ensemble des cellules potentiellement visibles (les informations de *visibilité cellule-à-cellule*).

Luebke et Georges [97] présentent le « rendu par portails » : une méthode efficace de calcul d'occlusion *en-ligne* pour des requêtes de visibilité de type œil-cellule (point-

cellule), utilisant un GCP. La boîte englobante de la projection 2D (sur l'écran) d'un portail P est utilisée pour rejeter la géométrie de la cellule C située derrière P si P est invisible, et également pour déterminer la visibilité des portails menant de C vers d'autres cellules, si P est visible. Le GCP est créé manuellement.

En 1997, Meneveaux *et al.* [102] proposent un algorithme de décomposition d'une scène 3D en un GCP. Ils utilisent la projection des murs verticaux sur un plan horizontal puis regroupent les segments résultant dans l'espace dual. Ils usent de règles spéciales fondées sur certaines connaissances architecturales pour construire les cellules les unes après les autres, après quoi les portails sont calculés. Leur méthode est rapide, mais ne peut être appliquée à des scènes plus générales puisque leur décomposition est, de fait, de dimension 2.

James *et al.* [80] présentent un algorithme de simplification d'un arbre BSP pour accélérer le dessin de scènes architecturales. Leur méthode ne crée pas explicitement un GCP. L'idée intéressante de ce papier est de fusionner les feuilles du BSP pour obtenir une meilleure décomposition.

Plus récemment, deux travaux se sont attachés au problème que nous abordons ici, la génération automatique d'un graphe de cellules et portails. Haumont, Debeir et Sillion [71] présentent un algorithme de décomposition en cellules et portails d'une scène d'intérieur 3D sans contrainte géométrique. Cette méthode a été développée en même temps que la notre, et se montre en général plus efficace. Ils commencent par construire un champ de distance à la scène qui, pour tout point de l'espace libre, donne une bonne approximation de la distance du point de la scène le plus proche. Ce champ est ensuite utilisé pour faire « grandir » les cellules avec un algorithme de type *watershed* volumique. Quand deux cellules grandissantes se rencontrent, un portail est construit pour les séparer.

Lerner, Chrysanthou et Cohen-Or [93] présentent une méthode dont l'esprit est assez similaire à la notre. Elle s'applique à des scènes 2,5D (un champ de hauteur, par exemple, une ville). La différence fondamentale avec notre méthode est la génération de la subdivision initiale, qui se fait en suivant les contours de la scène qui est représentée par une partition du plan stockée sous la forme d'une *halfedge data structure*. Cette spécificité exploite pleinement l'hypothèse d'une scène d'entrée 2D, et paraît difficilement généralisable au cas 3D.

9.2.1 Vue générale de notre méthode

La méthode que nous présentons génère un GCP automatiquement à partir d'un arbre BSP de la scène, comme illustré sur la Figure 9.1. Il est intéressant de remarquer que, contrastant avec les approches précédentes, notre méthode *commence* par rechercher de *bons* portails et, *ensuite*, construit les cellules en explorant les composantes connexes d'un graphe de cellules plus petites (celles du BSP).

Étant donnée une scène architecturale, nous commençons par y construire des *séparateurs* (des polygones transparents) ayant de bonnes propriétés géométriques. Intuitivement, un séparateur doit fermer hermétiquement un chemin dans la scène. Ces séparateurs (qui sont des portails) définissent un GCP : les cellules sont les volumes bordés par les polygones de la scène et les séparateurs. Ces séparateurs sont créés à partir d'une subdivision initiale de la scène. Il existe de nombreuses manières de subdiviser une scène, mais pour notre problème, les polygones de la scène doivent coïncider avec les bords des cellules de la décomposition initiale. Nous utiliserons un

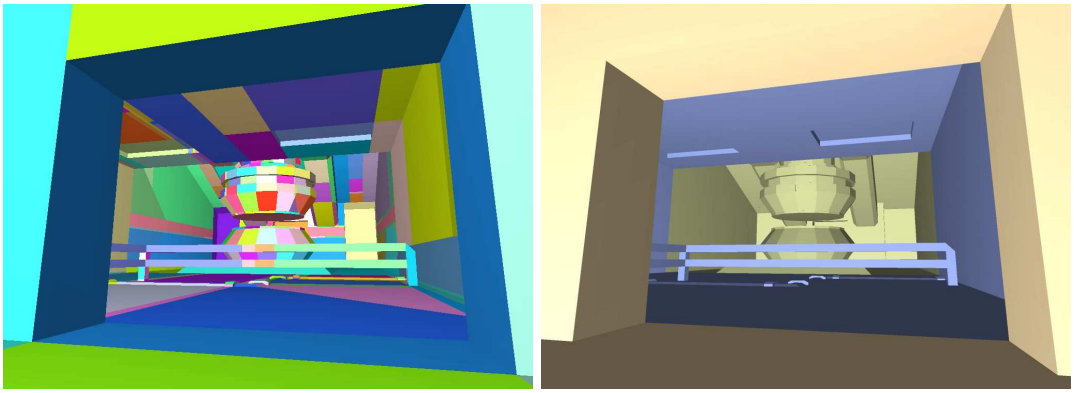


FIG. 9.1. À gauche, l'arbre BSP de la scène. **À droite**, notre décomposition en GCP, utilisable pour des visites interactives. Trois cellules sont visibles.

arbre BSP.

Les cellules que nous créons ne sont pas convexes, en général. Cela n'interdit en rien l'utilisation du rendu par portails, ou du calcul de PVS, ni l'utilisation de certaines optimisations comme l'*occlusion culling* [33]. Mais, si l'on souhaite calculer les PVS de ces cellules, uniquement à l'aide des portails [138], les résultats seront moins précis (plus conservatifs) alors que des cellules convexes donnerait des PVS exacts. Cependant, notre méthode permettra, par exemple, de regrouper certains objets géométriquement complexes (comme des escaliers) en une seule cellule, ce qui est impossible si la convexité des cellules est requise.

Notre méthode consiste en deux étapes : le *regroupement* commence par créer un GCP à partir d'un arbre BSP. Tous les portails de ce GCP sont des séparateurs ayant les bonnes propriétés géométriques définies ci-dessus. Ensuite, une *simplification* est appliquée sur ce GCP en ne gardant que les portails pertinents. Le regroupement procède en 4 étapes :

- *Étape 1.* Nous construisons une décomposition de la scène en un arbre BSP. Cette subdivision fournit un GCP qui est trop complexe pour être directement utilisé. Nous appelons les cellules de cet arbre des *bsp-cells*, qui sont des polytopes (convexes), et les facettes transparentes de ces bsp-cells, des *bsp-portals*. Nous dirons que deux bsp-cells sont connectées si elles partagent un bsp-portal commun.
- *Étape 2.* Nous construisons un ensemble de *séparateurs valides* à partir des bsp-portals de l'arbre BSP. Les séparateurs valides sont des portails, unions de plusieurs bsp-portals coplanaires et adjacents (voir plus bas).
- *Étape 3.* Les cellules sont construites en parcourant les composantes connexes de bsp-cells : deux bsp-cells sont connectées si elles sont connectées dans l'arbre BSP et le bsp-portal les connectant ne fait *pas* partie d'un séparateur valide. La Figure 9.2 présente un exemple simple. Après cette étape, nous obtenons un GCP ne contenant que des séparateurs valides.
- *Étape 4.* Nous post-traitons les cellules afin qu'elles satisfassent les contraintes de coût de dessin maximale (voir plus bas). Cette étape permet d'ajouter éventuellement de nouveaux séparateurs.

Le résultat de ces 4 étapes (et donc du *regroupement*) est un GCP dont les cellules ont un coût de rendu inférieur à une borne maximum définie par l'utilisateur. Nous appliquons alors la *simplification* de ce GCP. Cette dernière étape utilise une métrique sur les portails afin de ne garder que les plus intéressants. La simplifica-

tion peut être, en fait, appliquée à n'importe quel GCP et peut donc être utilisée en post-traitement sur tout autre algorithme de décomposition d'une scène en GCP.

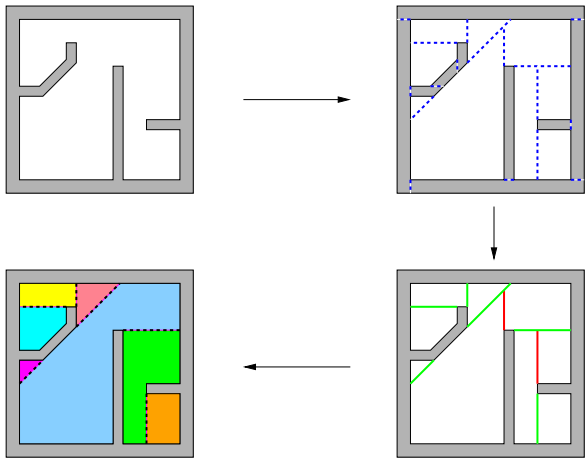


FIG. 9.2. En suivant les flèches, **1** : scène de départ. **2** : arbre BSP. **3** : séparateurs valides (en vert) et non valides (en rouge). **4** : le GCP résultant à la fin du *regroupement*. La *simplification* fusionnera certaines cellules adjacentes, en suivant les contraintes définies par l'utilisateur.

Nous souhaitons générer un ensemble de portails divisant la scène en cellules respectant certaines contraintes : en particulier, des bornes inférieure et supérieure sur le coût de rendu de chaque cellule. Un tel coût peut se définir de plusieurs façons. On pourra prendre en compte, par exemple, le nombre de triangles dans une cellule, et leur aire totale, pondérée par le coût des *shaders* décrivant l'aspect graphique de ces polygones. Le coût d'un tel *shader* doit prendre en compte le nombre de « passes » nécessaires, la complexité des *vertex-* et *pixel-shaders* [112], *etc.*. Nous appelons ces bornes les *contraintes sur les cellules*.

Notre méthode génère un GCP simplifié, dont les cellules satisfont les contraintes données par l'utilisateur. La Figure 9.3 montre une cellule typique générée par notre méthode.

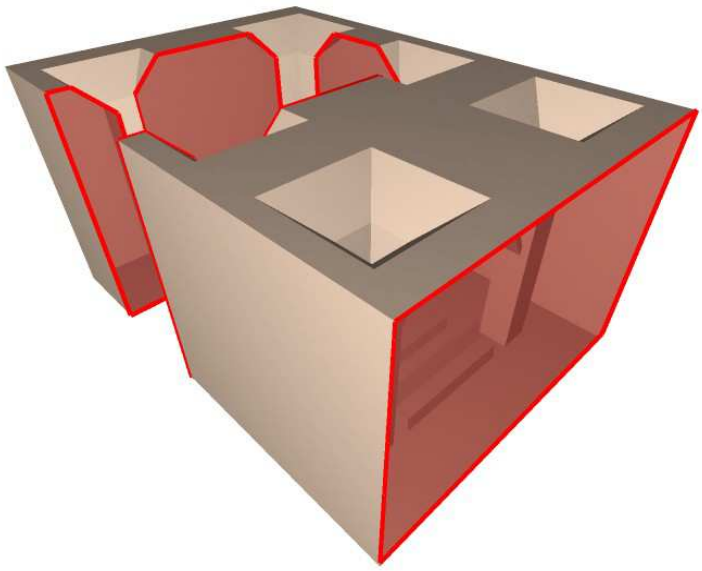


FIG. 9.3. Une cellule créée par notre méthode. Le contour de chaque portail apparaît en rouge.

9.3 Regroupement – construction des séparateurs

Étape 1 : arbre BSP initial

Soit $\mathcal{C}_{init}$ (resp. $\mathcal{P}_{init}$) l'ensemble des bsp-cells (resp. bsp-portals) du BSP initial. Soit $\mathcal{G}_{init}$ le graphe $(\mathcal{C}_{init}, \mathcal{E}_{init})$ où

$$\mathcal{E}_{init} = \{(x, y) \in \mathcal{C}_{init}^2 \ : \exists p \in \mathcal{P}_{init}, p \text{ est un bsp-portal connectant les bsp-cells } x \text{ et } y\}.$$

$\mathcal{G}_{init}$ est le graphe des bsp-cells connectées par les bsp-portals, et il y a bijection entre $\mathcal{E}_{init}$ et $\mathcal{P}_{init}$.

Étape 2 : sélection des séparateurs valides

Nous appelons les polygones de la scène d'entrée des polygones *solides*. Puisque nous souhaitons que les futurs séparateurs ferment certains chemins « hermétiquement », il est nécessaire que leur bord s'appuie entièrement sur des polygones solides. Pour construire l'ensemble des séparateurs, nous commençons par regrouper les bsp-portals en ensembles maximaux de bsp-portals coplanaires. Dans chaque plan support, chaque ensemble connexe de bsp-portals est extrait et forme un séparateur, dont nous vérifions alors la *validité*, voir Figure 9.4, à gauche et au centre.

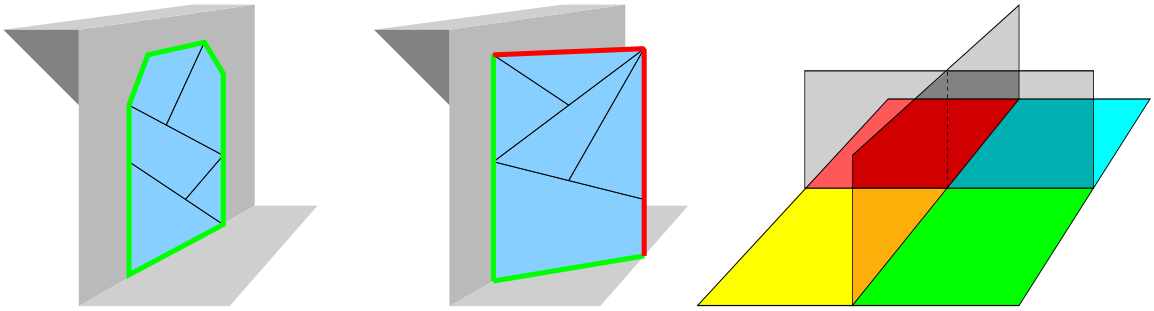


FIG. 9.4. À gauche, 5 bsp-portals (bleus) forment un séparateur valide. **Au milieu**, le séparateur n'est pas valide à cause de la présence de 3 arêtes non valides. **À droite**, deux séparateurs croisés générant 4 cellules.

Plus formellement, un séparateur s est valide si les bsp-portals le constituant sont également valides. Un bsp-portal est valide si chacune de ses arêtes est en contact avec un autre bsp-portal de s ou repose sur un (ou plusieurs) polygone solide (l'arête est alors dite *valide*). Nous garantissons ainsi « l'étanchéité » d'un portail valide. Si une arête d'un bsp-portal n'est pas valide, nous ôtons ce bsp-portal du séparateur et testons à nouveau la validité du séparateur. Il se peut que la suppression d'un bsp-portal coupe le séparateur en plusieurs composantes connexes, créant ainsi de nouveaux séparateurs.

Après n'avoir gardé que les séparateurs valides, nous vérifions qu'aucun séparateur valide ne croise un autre séparateur (Figure 9.4). Il faut alors en sélectionner un pour le supprimer. Nous sélectionnons le séparateur adjacent au plus petit nombre de cellules, ou bien utilisons la métrique sur les portails pour éliminer le moins efficace (voir la Section 9.4).

Étape 3 : création des cellules

Pour construire une cellule, nous sélectionnons une bsp-cell, puis sélectionnons itérativement les bsp-cells adjacentes jusqu'à rencontrer un polygone solide ou un séparateur valide. L'itération a lieu tant que l'expansion est possible. Un séparateur correspond à un ensemble de bsp-portals, et donc à un sous-ensemble de $\mathcal{E}_{init}$

dans le graphe $\mathcal{G}_{init}$. Soit $\mathcal{E}_{valid}$ l'union des bsp-portals formant les séparateurs *valides*. Les cellules nouvellement créées correspondent aux composantes connexes du graphe $(\mathcal{C}_{init}, \mathcal{E}_{init} \setminus \mathcal{E}_{valid})$.

Étape 4 : satisfaction des contraintes sur les cellules

Avant de simplifier le GCP obtenu à l'étape précédente, nous devons garantir que toutes les cellules satisfassent la borne supérieure imposée au coût de chaque cellule car la simplification ne peut qu'accroître le coût de la cellule la moins coûteuse. Si une cellule C est trop coûteuse, nous choisissons un nouveau plan de coupe P pour subdiviser C en cellules plus petites. L'intersection de P et C est construite (en utilisant le BSP), créant ainsi de nouveaux séparateurs que nous gardons, qu'ils soient valides ou pas : le rendu par portails fonctionnera correctement car un nouveau séparateur non valide s'appuie nécessairement sur des séparateurs valides (*sans* les croiser) ou sur des polygones solides. La Figure 9.5 montre comment une cellule trop grande (grise, à gauche) est découpée.

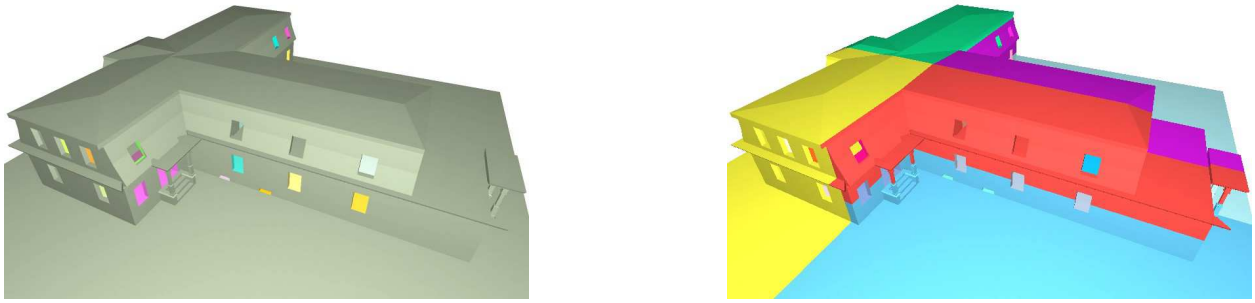


FIG. 9.5. À gauche, dans cette scène, la cellule extérieure (en gris) est trop coûteuse. **À droite**, la cellule est subdivisée.

Pour découper une cellule trop grande, plusieurs plans de coupe sont générés à l'aide d'une heuristique, et plusieurs heuristiques sont testées. Nous retenons le plan créant des sous-cellules les plus balancées possible. La Figure 9.9 montre le résultat des différentes heuristiques décrites ici :

- Un point à l'intérieur de la cellule est généré aléatoirement, de pair avec un vecteur normal (Figure 9.9, à gauche). Cette heuristique donne de bons résultats pour la taille des sous-cellules, mais la forme des cellules devient plus complexe.
- La normale est alignée avec un axe (Figure 9.9, au centre). La forme des cellules est meilleure, mais il est plus rare de tomber sur un bon plan de coupe.
- Nous choisissons le plan support d'un bsp-portal non valide contenu dans la cellule initiale (Figure 9.9, à droite). Les résultats de cette heuristique sont en général les meilleurs.

9.4 Simplification du GCP

Nous souhaitons simplifier le GCP obtenu à la section précédente, de manière à minimiser le nombre de cellules trop petites (ou trop peu coûteuses). Notre procédure est assez semblable à l'algorithme de simplification d'un maillage de Garland et Heckbert [57]. Nous commençons par trier les cellules en fonction de leur coût de rendu. Nous considérons alors la cellule la moins coûteuse (si son coût est en dessous de la borne inférieure prescrite) et examinons ses séparateurs. Nous excluons les séparateurs qui, si on les supprimait, généreraient une cellule (union de deux cellules) trop

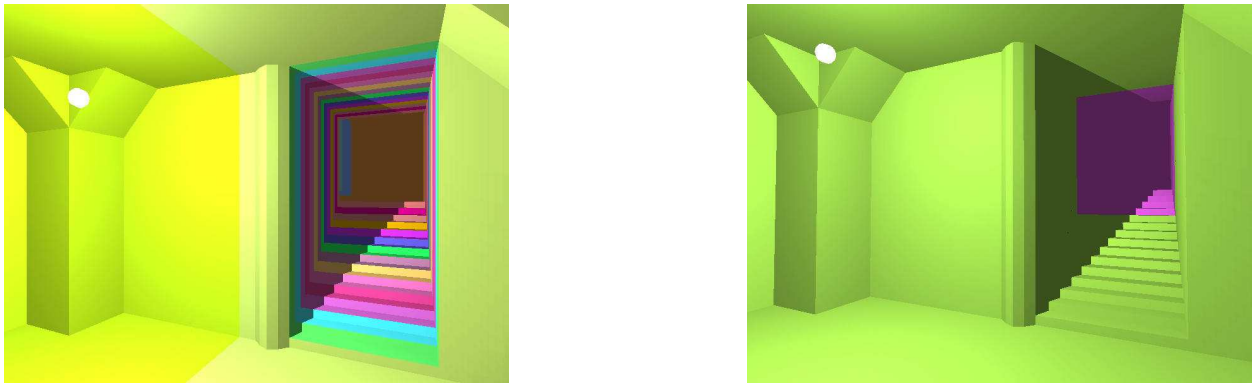


FIG. 9.6. À gauche, avant l'étape de simplification, chaque marche d'escalier donne lieu à une cellule, dans la scène de la clinique. **À droite**, après simplification, seules deux cellules contiennent l'ensemble de l'escalier.

coûteuse. Les séparateurs restants sont alors comparés à l'aide d'une métrique définie plus bas, et le séparateur obtenant la plus grande mesure est supprimé en fusionnant ses deux cellules adjacentes. Le tri des cellules est alors mis à jour et le processus est itéré jusqu'à ce que toutes les cellules satisfassent les contraintes imposées par l'utilisateur. La Figure 9.6 montre comment les marches d'un escalier ont été fusionnées en une cellule plus grande. Il se peut qu'une cellule trop petite ne puisse pas être fusionnée avec une cellule voisine. C'est le cas si toutes les cellules voisines ont déjà un coût assez élevé. Nous pourrions peut-être résoudre ce problème en subdivisant la cellule voisine ayant le plus petit coût, mais cette solution n'a pas été implémentée ni, donc, testée.

Nous décrivons maintenant la métrique utilisée pour comparer les portails.

Qualités d'un portail. Il n'est pas aisé de quantifier la qualité d'un portail, ni même de savoir ce qu'est un « bon » portail. Nous suggérons qu'un « bon » portail devrait avoir les deux bonnes propriétés suivantes :

- un portail doit être aussi caché que possible : nous voulons donc minimiser le volume, dans chaque cellule, depuis lequel le portail est visible. On parlera du *volume visible* du portail.
- un portail doit réaliser un équilibre correct du coût de rendu : nous souhaitons que le portail rejette beaucoup de géométrie quand il n'est pas visible, et ce, des deux côtés du portail. On considèrera donc qu'un portail connectant deux cellules de coût proche est meilleur qu'un portail connectant une petite et une grande cellule.

Remarquons que les « bonnes » positions intuitives d'un portail (portes, fenêtres, *etc.*) satisfont en général les qualités décrites ci-dessus. La Figure 9.7 montre un ensemble de cellules à des itérations successives de la simplification du GCP. La métrique que nous proposons est utilisée pour déterminer quel portail doit être supprimé à chaque itération. Remarquer comme le dernier portail restant correspond à une notion assez intuitive d'un « bon » portail : les coûts de ses 2 cellules voisines sont relativement bien balancés et le portail est relativement peu visible ; il est par exemple moins visible depuis l'étage du bas que le portail en haut de l'escalier sur la $3^{\text{ème}}$ vignette.

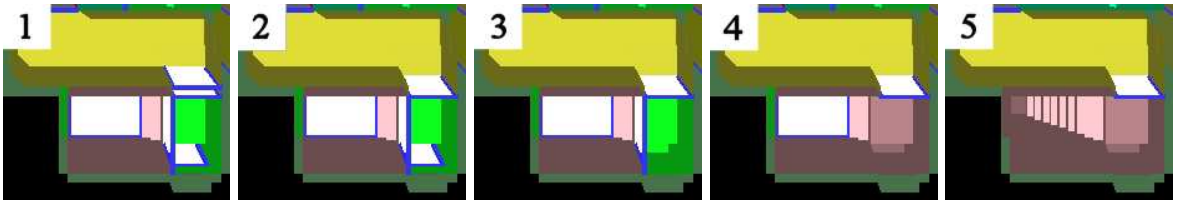


FIG. 9.7. Une vue rapprochée d'un GCP durant les itérations de simplification. Les portails sont dessinés en blanc et leur contour en bleu.

Calcul de la métrique à l'aide du matériel graphique. Nous évaluons rapidement une approximation du volume visible d'un portail à l'aide du matériel graphique. La faible précision de l'approximation n'est pas réellement un problème puisqu'il s'agit seulement de *distinguer* un « mauvais » d'un « bon » portail. Nous évaluons donc le volume visible d'un point P du portail en dessinant la scène vue par P dans une *cube-map* de basse résolution (32x32 pixels). La profondeur de chaque pixel dessiné nous donne alors une approximation du volume visible *à travers* ce pixel (en corrigeant le biais introduit par l'angle de vue). Nous sommons les volumes de chaque pixel pour estimer le volume visible de P. Pour évaluer le volume visible du portail p, nous moyennons le volume visible de plusieurs points générés aléatoirement sur p[6] pour obtenir la valeur $visibleVolume(p)$.

Nous définissons aussi, pour un portail p, la valeur

$$balance(p) = cost(cell_A(p)) - cost(cell_B(p))$$

où $cell_A(p)$ et $cell_B(p)$ sont les deux cellules séparées par le portail p. Finalement, nous définissons la métrique d'un portail p comme

$$score(p) = balance(p) \cdot visibleVolume(p)$$

9.5 Implémentation

Comme évoqué précédemment, le choix d'un arbre BSP comme subdivision initiale semble obligatoire. Par exemple, imaginons que nous utilisions une triangulation de Delaunay (3D) comme subdivision initiale. Dans une telle tétraédralisation, les polygones de la scène de départ correspondent aux faces des tétraèdres.

Malheureusement, il est fort peu probable que nous trouvions deux *Delaunay-portals* coplanaires. Ainsi, nous nous retrouverions sans le moindre séparateur valide. L'utilisation d'un arbre BSP nous donne de meilleures garanties sur ce point.

En suivant ce raisonnement, il apparaît clairement que l'arrangement des plans supports des triangles de la scène ferait une excellente subdivision initiale. Plus de séparateurs valides pourraient être trouvés avant la simplification, ce qui résulterait en un ensemble de cellules plus homogènes. Mais en pratique, la taille de cet arrangement — $O(n^3)$ — ne nous permet pas d'envisager son utilisation.

Le choix d'un BSP permet cependant certaines variations. Pour adapter notre algorithme à divers types de scènes, nous pouvons guider le début de la construction

[6] Le résultat de cette moyenne est plutôt à rapprocher du volume de l'ensemble des rayons lumineux « voyant » le portail. Pour une estimation du volume de l'espace visible par le portail, une sélection du maximum plutôt qu'une moyenne serait préférable.

de l'arbre BSP. Nous pouvons spécifier explicitement les premiers plans de coupe utilisés avant de laisser faire une heuristique pour terminer l'arbre BSP. Par exemple, nous pouvons commencer par un découpage de la scène avec une grille régulière, ou une tétraédralisation régulière de l'espace. Ce dernier type de découpage nous semble particulièrement adapté si la scène modélise, par exemple, un ensemble de galeries sous-terraines. L'utilisateur peut aussi aider cette construction en fournissant explicitement quelques plans de coupe ou des boîtes englobantes isolant certaines parties de la scène.

Heuristiques pour la construction de l'arbre BSP. Pendant la construction d'un arbre BSP, une heuristique est chargée de sélectionner les plans de coupe utilisés. Pour une scène générale contenant n polygones, la construction de l'arbre BSP se fait en temps $O(n^3)$ et en espace $O(n^2)$, dans le pire des cas. Certaines heuristiques garantissent un espace espéré $O(n)$; d'autres optimisent le temps de construction du BSP. Nous référons le lecteur aux travaux de Airey et Teller [3, 140] pour de plus amples discussions sur le choix et la combinaison des heuristiques.

La recherche d'une subdivision initiale, optimale pour la création des séparateurs, est un problème « mal posé ». Puisque nous ne savons pas comparer deux solutions acceptables, nous ne savons pas non plus comment définir une solution optimale. Nous observons cependant que plus nous avons de bsp-portals dans la subdivision, plus nous avons de choix pour leur regroupement. Aussi, nous n'avons pas besoin que l'arbre BSP soit correctement balancé. Nous avons alors expérimenté les heuristiques suivantes :

• **min-cut** minimise le nombre de polygones coupés par le plan de coupe. Si le sous-espace considéré comprend k polygones, **min-cut** opère en temps $O(k^2)$ et l'arbre BSP est construit en temps $O(n^3)$.

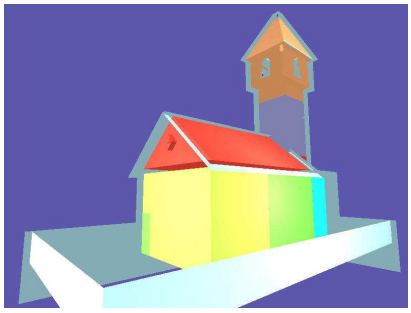
• **max-area** choisit le plan qui contient la plus grande aire couverte par des polygones solides. Si les plans supports des polygones solides sont initialement triés selon cette mesure (ce qui coûte un temps $O(n^2)$), le choix d'un plan de coupe avec cette heuristique coûte un temps $O(k)$.

• **max-ortho** choisit un plan maximisant l'orthogonalité entre lui et les polygones du sous-espace (pondéré par leur aire).

Quelle que soit l'heuristique utilisée, notre algorithme donne de bons résultats. Le GCP final a toujours été utilisable. Cependant, nous avons observé que l'heuristique **min-cut** donne les résultats les plus satisfaisants, puisqu'elle tend à placer les petits objets (tables, chaises, cheminées, ordinateurs, douches, éviers) entièrement à l'intérieur d'une cellule, ce qui est visuellement agréable.

Pour certaines scènes spécifiques, la construction de l'arbre BSP aurait sûrement besoin d'être guidée au début. Par exemple, des scènes de type galerie sous-terraine ne contiennent pas de polygones localement orthogonaux aux paroies. Guider la subdivision initiale à l'aide d'une grille ou une triangulation régulière sera particulièrement utile dans ces cas-là.

Notre implémentation de la construction de l'arbre BSP suppose que les données en entrée sont des polytopes. Cette contrainte facilite grandement les tests *intérieur*/*extérieur* effectués dans la construction de l'arbre. De plus, elle est généralement imposée dans le monde des jeux vidéo, pour les mêmes raisons. Les bsp-portals et bsp-cells intérieurs (dans le *béton* virtuel) peuvent ainsi être facilement détectés et



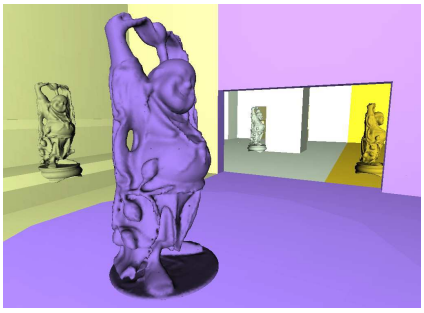


FIG. 9.8. À gauche, un modèle d'église est subdivisé en un GCP. Chaque couleur représente une cellule (pour voir les cellules intérieures, nous dessinons les polygones tournant le dos au point de vue). **À droite**, nous avons placé un modèle du bouddha dans chaque cellule pour construire une scène de 820 000 triangles, contenant à peu près 16 000 triangles par cellule.

éliminés. Nous avons implémenté une version différente qui prend en entrée un ensemble de polygones sans contrainte (une *soupe* de polygones), mais si la scène n'est pas *étanche*, il sera alors impossible de distinguer l'intérieur de l'extérieur, et des cellules seront créées à l'intérieur du modèle (dans les murs par exemple). De plus, le temps de construction de l'arbre BSP est augmenté.

La construction de l'arbre BSP est effectuée jusqu'au bout, c'est-à-dire jusqu'à ce que chaque bsp-cell soit vide de tout autre polygone. Il y a deux raisons à cela. Imaginons que l'on s'arrête dès lors qu'une bsp-cell contient peu de polygones (10 par exemple, pour accélérer la construction de l'arbre). Tout d'abord, nous perdons beaucoup de bsp-portals qui pourraient être cruciaux pour la construction de séparateurs valides. Ensuite, nous ne pouvons plus effectuer de tests *intérieur*/*extérieur* et sommes contraints à garder tous les bsp-portals et bsp-cells indésirables, créés à l'intérieur des murs.

La construction de l'arbre effectue beaucoup de tests géométriques et des problèmes de précisions numériques apparaissent inévitablement. Pour y remédier un tant soit peu, nous assignons un ID unique à chaque plan de coupe utilisé et assignons l'ID d'un plan de coupe à tous les bsp-portals générés lors de la découpe d'un sous-espace par ce plan. Nous pouvons alors regrouper les bsp-portals coplanaires simplement en les regroupant par ID. Remarquons qu'un même plan de coupe peut être utilisé dans plusieurs sous-espaces. Aussi, avant d'utiliser un plan de coupe, nous vérifions qu'il n'a pas déjà été utilisé précédemment. Cette vérification doit se faire numériquement. Le cas échéant, nous modifions l'ID du nouveau plan de coupe et lui assignons l'ID du plan précédemment utilisé. Les calculs de coplanarité sont tous effectués « à ε près ». Le même ε est utilisé pour vérifier si une arête d'un bsp-portal repose contre un polygone solide, lors des tests de validité des séparateurs.

La Figure 9.8, à gauche, montre les cellules créées par notre méthode sur une scène simple.

Le Tableau 9.1 liste les temps de calcul de l'arbre BSP utilisant l'heuristique **min-cut**. Le nombre de bsp-cells créées est à peu près le même que la colonne ◇ **out**. Remarquer le très grand nombre de bsp-portals créés, ce qui rend l'arbre BSP inutilisable pour le rendu par portails.

Les temps de calcul ont été mesurés sur un Athlon@1466 Mhz, équipé de 1Go de RAM.

Le Tableau 9.2 donne les statistiques du regroupement des séparateurs. Remarquer la grande réduction du nombre de cellules (**#cells**, à gauche), et l'effet de la

Nom de la scène	**# in**	**BSP**	**# out**	⋄ **out**
Church	1056	3 s.	3523	2460
Underground	2105	3 s.	4972	3248
Sanatorium	4916	8 s.	10501	8314
Clinic	11597	135 s.	27241	16816
Blockwar	21369	123 s.	47337	31222

TAB. 9.1. Statistiques de la construction de l'arbre BSP. La colonne **# in** indique le nombre de polygones solides en entrée. **BSP** indique le temps de création de l'arbre BSP. **# out** indique le nombre de polygones dans l'arbre BSP, et ⋄ **out** indique le nombre de bsp-portals générés.

borne inférieure sur le coût des cellules (réduisant ainsi le nombre de cellules, **#cells**, à droite).

Nom de la scène	validation	#cells	#portals
Church	2 s.	68/15	18
Underground	5 s.	97/20	13
Sanatorium	19 s.	186/18	33
Clinic	66 s.	630/141	103
Blockwar	176 s.	869/177	94

TAB. 9.2. Statistiques pour le regroupement des séparateurs valides, avec un seuil de 50 polygones minimum par cellule et pas de borne supérieure. La colonne **validation** indique le temps nécessaire pour construire les séparateurs valides et construire le GCP final (bien que la construction de ce dernier prenne un temps négligeable). **#cells** indique le nombre de cellules après le regroupement des séparateurs (à gauche) et après la fusion des petites cellules (à droite). **#portals** indique le nombre final de portails (séparateurs valides).

Pour démontrer l'efficacité de notre décomposition en cellules et portails, nous présentons le nombre d'images dessinées par seconde lors d'une simulation d'une visite d'une scène (voir le Tableau 9.3). Les tests sont effectués avec une carte graphique GeForce4Ti. Le dessin est effectué avec *shader* de type Phong, par pixel.

La colonne **BSP** montre combien un arbre BSP n'est pas efficace si on l'utilise en tant que graphe de cellules et portails, parce qu'il y a beaucoup trop de petits portails et qu'un portail « voit » un grand nombre d'autres portails. De plus, chaque cellule contient trop peu de polygones pour obtenir un bon débit de travail de la carte graphique.

La colonne **GCP** démontre l'efficacité des GCP créés par notre méthode pour le rendu par portails. Il y a bien moins de portails à traiter et chaque cellule est assez large pour pouvoir occuper convenablement la carte graphique, sans temps mort.

La dernière ligne du Tableau 9.3 montre les temps de rendu d'une scène contenant 820 000 triangles, bien qu'un peu artificielle : nous avons placé un modèle de bouddha d'environ 16 000 triangles à l'intérieur de chaque cellule (après construction du GCP [140]). Les résultats montrent l'intérêt d'avoir de grandes cellules pour la détermination de la visibilité *en-ligne* : chaque cellule peut contenir des objets détaillés qui ne seront pas dessinés si la cellule n'est pas visible.

Nom de la scène	BSP	all	GCP
Clinic	7-20-39	20-30-33	75-103-149
Blockwar	3.7-6.3-6.5	18-22-23	87-126-171
Clinic+buddhas	–	– -1.2-1.3	5-18-34

TAB. 9.3. Nombre d'images par seconde : minimum, moyen et maximum lors d'une « promenade virtuelle » dans une scène 3D. Pour la colonne **BSP**, nous utilisons le rendu par portails directement sur l'arbre BSP initial. Pour la colonne **all**, nous envoyons simplement tous les polygones de la scène à la carte graphique. Pour la dernière colonne, nous utilisons le rendu par portails sur le GCP généré par notre méthode.

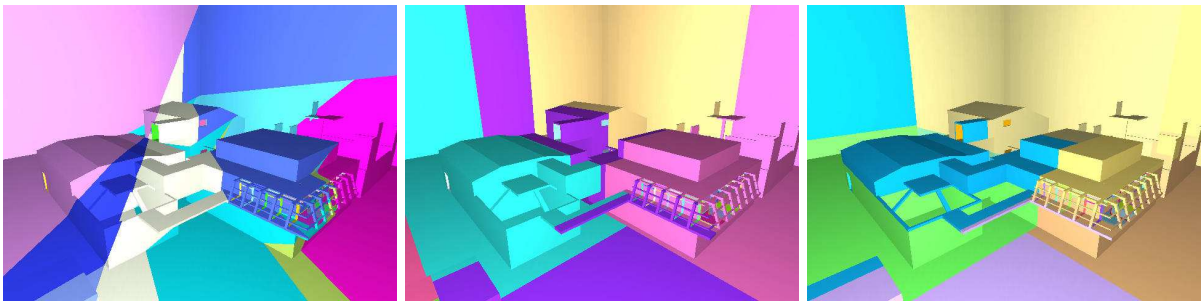


FIG. 9.9. Utilisation de critères différents pour la découpe d'une cellule trop grande. De gauche à droite, choix aléatoire, normale alignée sur les axes, choix aléatoire d'un bsp-portal inutilisé dans la cellule.

9.6 Conclusion et travaux futurs

Nous avons décrit un outil assez général pour la construction automatique d'un graphe de cellules et portails utilisable pour le rendu de scènes de type architectural, en temps-réel, utilisant le rendu par portails.

La méthode est flexible en plusieurs points. Elle prend en entrée un arbre BSP dont on peut varier la méthode de construction pour s'adapter à différents types de scènes. Elle permet la spécification de bornes sur le coût de dessin de chaque cellule.

Le GCP résultant peut cependant ne pas satisfaire l'utilisateur pour diverses raisons : un portail placé en un endroit indésirable, des portails manquants à d'autres endroits à cause de la subdivision initiale, ou bien la présence de cellules trop grandes. Une interface utilisateur devrait venir compléter la méthode automatique pour :

- choisir la méthode de création de la subdivision initiale.
- sélectionner et supprimer un portail. Il faut alors fusionner ses deux cellules adjacentes C_1 et C_2, et supprimer les autres portails connectant C_1 et C_2.
- Ajouter un portail manuellement. Étant donné un polygone donné par l'utilisateur, nous l'insérons dans l'arbre BSP pour le raffiner et reconstruisons localement les cellules du GCP.

Nous souhaiterions utiliser des méthodes plus rapides pour construire la subdivision initiale. Par exemple, la métrique sur les portails que nous proposons pourrait être utilisée plus tôt, lors de la construction de l'arbre BSP, pour guider la sélection des plans de coupe.

La définition d'un « bon » portail (ou d'un « bon » GCP) paraît encore floue, et il serait intéressant d'y donner un sens plus quantitatif (voir Lerner *et al.* [93]). Enfin, les performances de rendu que nous obtenons devraient être comparées plus finement avec d'autres méthodes de dessin de grandes scènes utilisant des calculs d'occlusion.

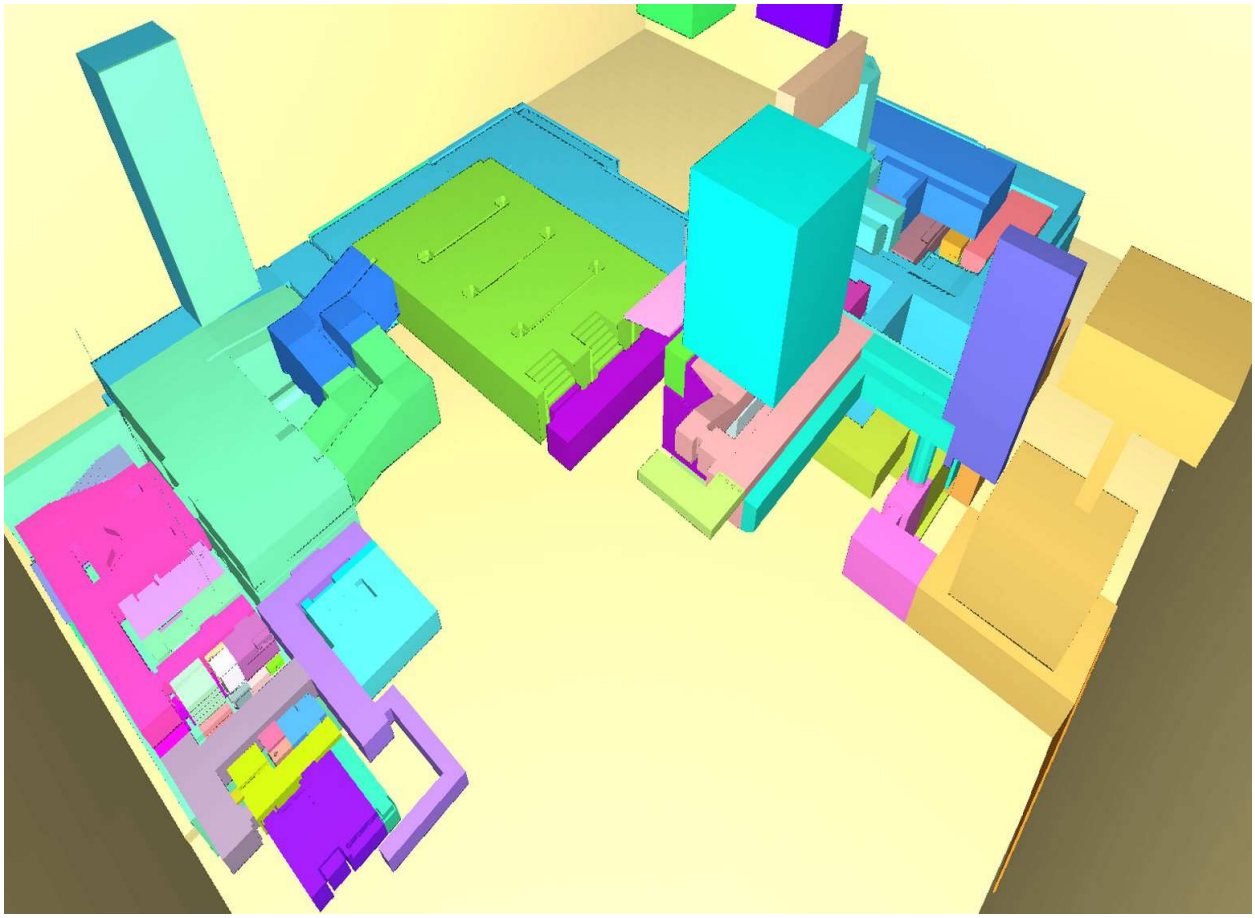


FIG. 9.10. Vue d'une scène utilisée dans un jeu vidéo. Chaque cellule a une couleur différente. Pour ce graphe de cellules et portails, les contraintes demandaient un grand nombre de polygones par cellule. Remarquer comme les marches d'escalier sont regroupées dans une seule cellule.

Conclusion

Cette thèse apporte quelques contributions originales pour la résolution du problème de la maintenance de la visibilité d'un point de vue mobile. Bien entendu, dans une acceptation si générale, il est certain que le problème est insoluble. Mais dès que quelques hypothèses sont ajoutées, des solutions algorithmiques deviennent possibles. Nous avons examiné ce problème sous deux angles *a priori* différents : celui de la géométrie algorithmique d'une part, selon lequel on s'attache à garantir la correction de l'algorithme, l'exactitude des calculs et une complexité algorithmique optimale. Celui de la synthèse d'images d'autre part, selon lequel on souhaite avant tout obtenir de bonnes performances réelles, sans toutefois sacrifier la qualité de l'image.

Les Chapitres 3, 4, 5 et 6 forment l'ensemble de nos contributions au problème de la maintenance *exacte* de la visibilité d'un point mobile. Le Chapitre 9 examine ce même problème, mais dans le but d'obtenir une implémentation efficace pour le dessin d'une scène 3D sur un écran, avec un biais certain vers les jeux vidéo. Les Chapitres 7 et 8 s'éloignent un peu du problème que nous venons d'évoquer. Au Chapitre 7, nous proposons un nouvel algorithme de construction du complexe de visibilité d'un ensemble de polytopes disjoints dans l'espace. Cet algorithme utilise une propriété de connexité que nous démontrons également. Enfin, le Chapitre 8 montre comment rendre robuste un algorithme servant au dessin d'ombres dures en temps-réel. Nous revenons maintenant plus en détail sur chacune de nos contributions.

Au Chapitre 3, nous émettons des hypothèses assez drastiques : la scène est constituée d'un ensemble de points dans le plan. Aussi, la « visibilité » du point de vue se résume à la liste des points triée circulairement autour du point de vue. Nous donnons un algorithme optimal pour maintenir une telle liste circulaire. Les événements sont de deux types : changement dans l'ordre des points ou changement de la trajectoire du point de vue. Dans les deux cas, notre algorithme traite un événement en temps $O(\log n)$.

Le Chapitre 4 est une contribution plutôt pédagogique puisque l'algorithme qui y est présenté est dû à Hall-Holt [67]. Ici, la scène est plane et peuplée d'objets convexes disjoints. La correction de l'algorithme de Hall-Holt repose sur les propriétés de certaines courbes dans le complexe de visibilité de l'ensemble des objets de la scène. Malheureusement, l'exposition de ces propriétés — dans l'Appendice A de la thèse de Hall-Holt — est complexe et difficile à suivre. Dans ce chapitre, nous réexpliquons le *visible zone algorithm* tout en en simplifiant l'exposition à l'aide de nombreuses figures et surtout d'une preuve plus simple d'un lemme clé, qui permet de se passer d'une certaine quantité de notions et propriétés introduites et démontrées dans la thèse de Hall-Holt.

Au Chapitre 5, nous transposons le problème du Chapitre 4 dans l'espace (en dimension 3). Nous y proposons un algorithme de maintenance de la visibilité dans une scène composée de polytopes disjoints. Cet algorithme a la particularité de permettre

au point de vue de suivre une trajectoire algébrique quelconque. Cependant, il est loin d'être optimal au sens décrit dans l'introduction, puisqu'un changement de trajectoire du point de vue peut avoir un coût quadratique en la taille de la scène. Cet algorithme a aussi la propriété de permettre aux polytopes de la scène de bouger ou de se déformer continûment (à condition qu'ils restent convexes !).

Nous avions l'espoir de pouvoir généraliser au cas 3D l'algorithme de Hall-Holt décrit au Chapitre 4 pour la maintenance de la visibilité d'un point de vue mobile dans le plan. L'idée est de voir la décomposition radiale 3D, $\mathcal{R}_V$, comme une surface (2D) dans le complexe de visibilité 3D (qui est de dimension 4). L'algorithme de maintenance de cette décomposition, présenté au Chapitre 5 peut être vu comme la maintenance de l'intersection de cette surface avec le complexe de visibilité lorsque la surface se déforme globalement et très rigidement : chaque segment libre de $\mathcal{R}_V$ est contraint à être aligné avec le point de vue, et le mouvement du point de vue induit donc le mouvement de tous les segments libres de $\mathcal{R}_V$. Comme dans l'algorithme de Hall-Holt, nous voudrions relâcher cette forte contrainte d'alignement et permettre aux segments de $\mathcal{R}_V$ qui ne sont pas directement adjacents au point de vue, de ne plus être alignés avec celui-ci, tout en garantissant certains invariants structurels sur $\mathcal{R}_V$ pour garantir, entre autre, que l'on puisse trouver en temps raisonnable le prochain événement de visibilité. Nous n'avons malheureusement pas obtenu de résultat encourageant dans cette direction, mais restons cependant persuadés de la faisabilité d'une telle approche. En particulier, nous savons que le *visible zone algorithm* peut être interprété comme un balayage topologique paresseux du complexe de visibilité 2D. Cela correspondrait, en 3D, à un « balayage » topologique paresseux du complexe de visibilité 3D (dont la structure est proche d'un arrangement d'hyperplans). Cependant, la « structure de balayage » n'est plus de codimension 1 : la surface de dimension deux que nous souhaitons déformer dans le complexe de visibilité, est de codimension 2. Cela suggère-t'il que cette structure n'est pas suffisament riche pour pouvoir la maintenir au cours du temps ?

Au Chapitre 7, nous avons montré une propriété de connexité des cellules du complexe de visibilité 3D $\mathcal{VC}$ d'un ensemble d'objets convexes disjoints : si C est une 4-cellule de $\mathcal{VC}$ alors sa frontière est connexe. Nous avons utilisé cette propriété pour décrire un algorithme de construction du complexe de visibilité, qui nous semble être le premier algorithme pratiquement implémentable pour ce problème. L'implémenter devient donc, bien entendu, un objectif futur que l'on se donne.

Dans la dernière partie de ce manuscrit, nous nous sommes intéressés à des questions beaucoup plus pratiques dans le domaine de la synthèse d'images en temps-réel.

Au Chapitre 8, nous avons montré qu'une certaine technique de dessin d'ombres dures, connue depuis 1991 et reputée comme non-robuste, pouvait être corrigée de manière simple avec l'aide de certaines spécificités des cartes graphiques récentes (la disponibilité des *vertex-programs*). La technique corrigée se révèle être compétitive avec l'autre solution robuste connue précédemment, et semble être avantageuse lorsqu'un modèle 3D complexe (en terme du nombre de triangles le composant) est utilisé pour projeter une ombre.

Enfin, au Chapitre 9, nous exposons une méthode pour organiser une scène 3D architecturale en un graphe de cellules et portails. Une telle cellule est analogue à une pièce dans une maison, et les portails sont des polygones « transparents » reliant les cellules. Une telle décomposition se révèle fort utile lorsque l'on souhaite simuler des ballades virtuelles à l'intérieur d'un bâtiment. À notre connaissance, aucune méthode

de décomposition de ce genre n'existait avant ces travaux (en dehors du traditionnel BSP). Cependant, la méthode de Haumont *et al.* [71] développée en même temps que la notre, se révèle plus efficace en pratique.

Bibliographie

[1] Pankaj K. AGARWAL, Boris ARONOV, Vladlen KOLTUN et Micha SHARIR : On lines avoiding unit balls in three dimensions. *Discrete and Computational Geometry*, 34(2):231–250, August 2005. 94

[2] Timo AILA et Tomas AKENINE-MÖLLER : A hierarchical shadow volume algorithm. *In Proceedings of Graphics Hardware 2004*, pages 15–23. Eurographics, Eurographics Association, 2004. 115, 121, 126

[3] John M. AIREY : *Increasing Update Rates in the Building Walkthrough System with Automatic Model-Space Subdivision and Potentially Visible Set Calculations*. Thèse de doctorat, Dept. of CS, U. of North Carolina, July 1990. 139, 148

[4] John M. AIREY, John H. ROHLF et Frederick P. BROOKS, JR. : Towards image realism with interactive update rates in complex virtual building environments. *Computer Graphics (1990 Symposium on Interactive 3D Graphics)*, 24(2):41–50, mars 1990. 27, 139, 140

[5] Graham ALDRIDGE et Eric WOODS : Robust, geometry-independent shadow volumes. *In Proc. 2nd International Conference on Computer graphics and Interactive Techniques in Australasia and Southeast Asia (Graphite)*, volume 2, pages 250–253. ACM, ACM Press, June 2004. 115, 121

[6] Helmut ALT, Marc GLISSE et Xavier GOAOC : On the worst-case complexity of the silhouette of a polytope. *In 15th Canadian Conference on Computational Geometry - CCCG 2003, Halifax, Canada*, August 2003. 70

[7] Pierre ANGELIER : *Algorithmique des graphes de visibilité*. Thèse de doctorat, École Normale Supérieure (Paris), February 2002. 10, 93

[8] Pierre ANGELIER et Michel POCCHIOLA : Cgal-based implementation of visibility complexes. Technical Report ECG-TR-241207-01, Effective Computational Geometry for Curves and Surfaces (ECG), 2003. 6, 10

[9] Pierre ANGELIER et Michel POCCHIOLA : A sum of squares theorem for visibility complexes and applications. *In* B. ARONOV, S. BASU, J. PACH et M. SHARIR, éditeurs : *Discrete and Computational Geometry – The Goodman-Pollack Festschrift*, volume 25 de *Algorithms and Combinatorics*, pages 79–139. Springer-Verlag, June 2003. A preliminary version appeared in the proceedings of the 17th Annual ACM Symposium on Computational Geometry (SoCG'01). v, 10

[10] Arthur APPEL : The notion of quantitative invisibility and the machine rendering of solids. *In Proc. ACM 22nd National Meeting*. Thompson Book Co, 1967. 18, 19

[11] Boris ARONOV, Leonidas J. GUIBAS, M. TEICHMANN et Li ZHANG : Visibility queries and maintenance in simple polygons. *Discrete and Computational Geometry*, 27(4):461–483, 2002. 21, 24

[12] Julien BASCH : *Kinetic Data Structures*. Thèse de doctorat, Stanford University, June 1999. 14

[13] Julien BASCH, Leo GUIBAS et J. HERSHBERGER : Data structures for mobile data. *In Proc. 8th Symposium on Discrete Algorithms (SODA'97)*, pages 747–756, 1997. 14, 15, 16

[14] Harlen Costa BATAGELO et Ilaim Costa JUNIOR : Real-time shadow generation using bsp trees and stencil buffers. *In Proc. SIBGRAPI 99*, pages 93–102, October 1999. 116, 118, 122

[15] Marshall BERN, David DOBKIN, David EPPSTEIN et Robert GROSSMAN : Visibility with a moving point of view. *In SODA '90 : Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 107–117, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics. 26, 61

[16] Bill BILODEAU et Mike SONGY : Real time shadows. *In Creative Labs Sponsored Game Developer Conference*. Creative Labs Inc., May 1999. 120

[17] Gerth BRODAL et Riko JACOB : Dynamic planar convex hull. *In Proc. 43rd Annual Symposium on Foundations of Computer Science*, pages 617–626, 2002. 34, 36

[18] Hervé BRÖNNIMANN, Olivier DEVILLERS, Vida DUJMOVIĆ, Hazel EVERETT, Marc GLISSE, Xavier GOAOC, Sylvain LAZARD, Hyeon-Suk NA et Sue WHITESIDES : The number of lines tangent to arbitrary convex polyhedra in 3d. *In SCG '04 : Proceedings of the twentieth annual symposium on Computational geometry*, pages 46–55, New York, NY, USA, 2004. ACM Press. 93

[19] Marie-Paule CANI et Samuel HORNUS : Subdivision curve primitives : a new solution for interactive implicit modeling. *In Shape Modelling International*, Italy, May 2001.

[20] John CARMACK : E-mail to private list. Published on the nVIDIA website, 2000. 116, 120

[21] Ed CATMULL : *A Subdivision Algorithm for Computer Display of Curved Surfaces*. Thèse de doctorat, Computer Science Department, University of Utah, Salt Lake City, UTah, 1974. Report UTEC-CSc-74-133. 27

[22] CGAL : *Computational Geometry Algorithm Library*. `http://www.cgal.org/`. 6, 62, 93

[23] Eric CHAN et Frédo DURAND : An efficient hybrid shadow rendering algorithm. *In Proc. Eurographics Symposium on Rendering*, pages 185–195. Eurographics, Eurographics Association, 2004. 115, 121, 126

[24] Bernard CHAZELLE : A theorem on polygon cutting with applications. *In Proc. 23rd Annual FOCS Symposium*, pages 339 – 349, 1982. 4, 20, 86

[25] Bernard CHAZELLE : Triangulating a simple polygon in linear time. *In Proc. 31st Annual IEEE Symposium on Foundation of Computer Science*, pages 220–230, 1990. 20

[26] Bernard CHAZELLE et Leonidas J. GUIBAS : Visibility and intersection problems in plane geometry. *In Proc. 1st Symposium on Computational Geometry (SoCG)*, 1985. 20

[27] Danny Z. CHEN et Ovidiu DAESCU : Maintaining visibility of a polygon with a moving point of view. *In Proc. 8th Canadian Conference on Computational Geometry (CCCG'96)*, pages 240–245, 1996. 24

[28] Hamilton Yu-Ik CHONG et Steven J. GORTLER : A lixel for every pixel. *In Proc. Eurographics Symposium on Rendering*. Eurographics, Eurographics Association, 2004. 121

[29] James H. CLARK : Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, octobre 1976. 21

[30] Daniel COHEN-OR, Yiorgos CHRYSANTHOU, Silva C. et Frédo DURAND : A survey of visibility for walk-through applications. *IEEE TVCG Journal*, 9(3): 412–431, July-Sept 2003. 24, 29

[31] Satyan COORG et Seth J. TELLER : A spatially and temporally coherent object space visibility algorithm. Rapport technique, MIT LCS Technical Report 546, 1996. 27

[32] Satyan COORG et Seth J. TELLER : Temporally coherent conservative visibility. *In Proc. 12th Annual ACM Symposium on Computational Geometry (SOCG'96)*, pages 78–87, Philadelphia, PA, May 1996. ACM. 27

[33] Satyan COORG et Seth J. TELLER : Real-time occlusion culling for models with large occluders. *In* ACM, éditeur : *Proc. Symposium on Interactive 3D Graphics (SI3D'97)*, pages 83–90, 1997. 27, 142

[34] Thomas CORMEN, Charles LEISERSON, Ron RIVEST et Cliff STEIN : *Introduction to algorithms*. MIT Press, 2002. 6, 72

[35] Franklin C. CROW : Shadow algorithms for computer graphics. *In Proc. SIGGRAPH*, pages 242–248. ACM, ACM Press, 1977. 115

[36] Marc de BERG, Marc van KREVELD, Mark OVERMARS et Otfried SCHWARTZKOPF : *Computational Geometry : Algorithms and Applications*. Springer-Verlag, 2000. 2nd rev. ed., 367 pages. 5, 6, 40, 68, 86

[37] Mark de BERG, Leonidas J. GUIBAS et Dan HALPERIN : Vertical decompositions for triangles in 3-space. *Discrete & Computational Geometry*, 15:35–62, 1996. 70

[38] Olivier DEVILLERS, Vida DUJMOVIĆ, Hazel EVERETT, Xavier GOAOC, Sylvain LAZARD, Hyeon-Suk NA et Sylvain PETITJEAN : The expected number of 3d visibility events is linear. *SIAM Journal on Computing*, 32(6):1586–1620, 2003. 93

[39] Olivier DEVILLERS, Vida DUJMOVIĆ, Hazel EVERETT, Samuel HORNUS, Sue WHITESIDES et Stephen WISMATH : Maintaining visibility information of planar point sets with a moving viewpoint. *In Proc. 17th Canadian Conference on Computational Geometry*, 2005. 33

[40] Paul J. DIEFENBACH : *Multi-pass Pipeline Rendering : Interaction and Realism through Hardware Provisions*. Thèse de doctorat, University of Pennsylvania, 1996. 116, 119, 122

[41] dPVS, Hybrid Graphics. `http://www.hybrid.fi/main/products/devtools.php`. 138

[42] Florent DUGUET et George DRETTAKIS : Robust epsilon visibility. *In* John HUGHES, éditeur : *Proceedings of ACM SIGGRAPH*, Annual Conference Series. ACM, ACM Press, juillet 2002. 13

[43] Frédo DURAND : A multidisciplinary survey of visibility. *In ACM SIGGRAPH course notes : Visibility, Problems, Techniques, and Applications*. ACM, 2000. iii, 24

[44] Frédo DURAND, George DRETTAKIS et Claude PUECH : The 3d visibility complex, a new approach to the problems of accurate visibility. *In 7th Eurographics Workshop on Rendering, Porto*, june 1996. v, 7, 10

[45] Frédo DURAND, George DRETTAKIS et Claude PUECH : Fast and accurate hierarchical radiosity using global visibility. *ACM Transactions on Graphics*, avril 1999. 13, 29, 99, 137

[46] Frédo DURAND, George DRETTAKIS et Claude PUECH : The 3D visibility complex. *ACM Transactions on Graphics*, 2002. 10, 13, 93, 94

[47] Frédo DURAND, George DRETTAKIS, Joëlle THOLLOT et Claude PUECH : Conservative visibility preprocessing using extended projections. *Proceedings of SIGGRAPH 2000*, July 2000. Held in New Orleans, Louisiana. 28

[48] Herbert EDELSBRUNNER et Leonidas J. GUIBAS : Topologically sweeping an arrangement. *In STOC '86 : Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 389–403, New York, NY, USA, 1986. ACM Press. 10, 22, 39

[49] Herbert EDELSBRUNNER, Leonidas J. GUIBAS et Jorge STOLFI : Optimal point location in monotone subdivisions. Rapport technique #2, DEC/SRC, 1984. 21

[50] Alon EFRAT, Leonidas J. GUIBAS, Olaf HALL-HOLT et Li ZHANG : On incremental rendering of silhouette maps of a polyhedral scene. *In Proc. 11th Symp. Discrete Algorithms*, pages 910–917. ACM, ACM and SIAM, Jan 2000. 26, 62, 70, 78

[51] David EPPSTEIN, Giuseppe F. ITALIANO, Roberto TAMASSIA, Robert E. TARJAN, Jeffrey WESTBROOK et Moti YUNG : Maintenance of a minimum spanning forest in a dynamic planar graph. *Journal of Algorithms*, 13(1):33–54, 1992. 87, 88

[52] Jeff ERICKSON : Finite-resolution hidden surface removal. *In Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 901–909, 2000. 23

[53] Hazel EVERETT, Sylvain LAZARD, Sylvain PETITJEAN et Linqiao ZHANG : An experimental assessment of the 2d visibility complex. *In Proceedings of 17th Canadian Conference on Computational Geometry (CCCG)*, 2005. 10, 93

[54] Cass EVERITT et Mark J. KILGARD : Practical and robust stenciled shadow volumes for hardware-accelerated rendering. Rapport technique, nVIDIA, 2002. 115, 116, 119, 120

[55] Henry FUCHS, Zvi M. KEDEM et Bruce F. NAYLOR : On visible surface generation by *a priori* tree structure. *Proceedings of SIGGRAPH'80*, 14(3):124–133, 1980. 21, 140

[56] R. GALIMBERTI et U. MONTANARI : An algorithm for hidden line elimination. *Communication of the ACM*, 12(4), April 1969. 19

[57] Michael GARLAND et Paul S. HECKBERT : Surface simplification using quadric error metrics. *In SIGGRAPH '97 : Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co. 145

[58] Sherif GHALI et James STEWART : Incremental update of the visibility map as seen by a moving viewpoint in two dimensions. *In Seventh International Eurographics Workshop on Computer Animation and Simulation*, pages 1–11, August 1996. 24

[59] Sherif GHALI et James STEWART : Maintenance of the set of segments visible from a moving viewpoint in two dimensions. *In Video Proceedings of Symposium on Computational Geometry (SoCG)*, May 1996. 24

[60] Xavier GOAOC : *Structures de visibilité globales : tailles, calculs et dégénérescences*. Thèse de doctorat, Université de Nancy, 2004. v, 13, 94, 104

[61] Michael T. GOODRICH : A polygonal approach to hidden-line and hidden-surface elimination. *Computer Vision, Graphics, and Image Processing : Graphical Models and Image Processing*, 54(1):1–12, 1992. 22

[62] Michael T. GOODRICH et Roberto TAMASSIA : Dynamic trees and dynamic point location. *In STOC '91 : Proceedings of the twenty-third annual ACM Symposium*

on Theory of computing, pages 523–533, New York, NY, USA, 1991. ACM Press. 83, 84, 85, 87, 88

[63] Stéphane GRABLI, Emmanuel TURQUIN, Frédo DURAND et François SILLION : Programmable style for NPR line drawing. *In Rendering Techniques 2004 (Eurographics Symposium on Rendering)*. ACM Press, june 2004. 19

[64] Leonidas J. GUIBAS, John HERSHBERGER, D. LEVEN, Micha SHARIR et Robert E. TARJAN : Linear time algorithms for visibility and shortest path problems inside simple polygons. *In SCG '86 : Proceedings of the second annual symposium on Computational geometry*, pages 1–13, New York, NY, USA, 1986. ACM Press. 20, 24

[65] Leonidas J. GUIBAS, Lyle RAMSHAW et Jorge STOLFI : A kinetic framework for computational geometry. *In Proc. 24th Annual Symposium on Foundation of Computer Science (FOCS)*, pages 100–111, 1983. 20

[66] Olaf HALL-HOLT : Kinetic visible set maintenance in the plane. submitted for publication, 2001. 44, 57

[67] Olaf HALL-HOLT : *Kinetic visibility*. Thèse de doctorat, Department of Computer Science, Stanford University, August 2002. iv, v, 24, 39, 44, 48, 57, 153

[68] Olaf HALL-HOLT et Szymon RUSINKIEWICZ : Visible zone maintenance for real-time occlusion culling. submitted for publication, 2001. 47, 57

[69] Jean-Marc HASENFRATZ, Marc LAPIERRE, Nicolas HOLZSCHUCH et François SILLION : A survey of real-time soft shadows algorithms. *Computer Graphics Forum*, 22(4):753–774, December 2003. State-of-the-Art Reviews. 115

[70] Allen HATCHER : *Algebraic Topology*. Cambridge University Press, 2001. 97, 103, 104

[71] Denis HAUMONT, Olivier DEBEIR et François SILLION : Volumetric cell-and-portal generation. *Computer Graphics Forum*, 3-22, 2003. 139, 141, 155

[72] Denis HAUMONT, Otso MÄKINEN et Shaun NIRENSTEIN : A low dimensional framework for exact polygon-to-polygon occlusion queries. *In Proc. 16th Eurographics Symposium on Rendering*, pages 211–222, June 2005. 28

[73] Tim HEIDMANN : Real shadows, real time. *In IRIS Universe*, volume 18, pages 23–31. Silicon Graphics, Inc, 1991. 116, 118

[74] Jared HOBEROCK : *The visibility 2-skeleton*. Mémoire de D.E.A., University of Illinois, 2004. 107

[75] Samuel HORNUS, Alexis ANGELIDIS et Marie-Paule CANI : Implicit modelling using subdivision-curves. *The Visual Computer*, 2002.

[76] Samuel HORNUS, Jared HOBEROCK, Sylvain LEFEBVRE et John C. HART : ZP+ : correct z-pass stencil shadows. *In ACM International Symp. on Interactive 3D Graphics and Games*. ACM Press, April 2005. 114

[77] Samuel HORNUS et Claude PUECH : 3d radial decomposition and their kinetic maintenance. DIMACS workshop on Algorithmic Issues in Modeling Motion, Rutgers University, November 2002.

[78] Samuel HORNUS et Claude PUECH : A simple kinetic visibility polygon. *In Proc. 18th European Workshop on Computational Geometry'02*, pages 27–30, 2002. Warsaw University.

[79] Samuel HORNUS et Philippe SCHNOEBELEN : On solving temporal logic queries. *In* Hélène KIRCHNER et Christophe RINGEISSEN, éditeurs : *Proc. 9th Int. Conf. Algebraic Methodology and Software Technology (AMAST'2002)*, volume 2422 de *Lecture Notes in Computer Science*, pages 163–177. Springer, September 2002.

[80] Adam JAMES et A. M. DAY : The hidden face determination tree. *Computers and Graphics*, 23(3):377–387, july 1999. 141

[81] Cliff B. JONES : A new approach to the 'hidden line' problem. *Computer Journal*, 14(3):232–237, août 1971. 140

[82] Mark J. KILGARD : Robust stencil shadow volumes. *In CEDEC Presentation, Tokyo*, 2001. 115, 116, 119, 122

[83] David KIRKPATRICK, Jack SNOEYINK et Bettina SPECKMANN : Kinetic collision detection for simple polygons. *Int. J. of Computational Geometry and Applications*, 12(1-2):3–27, 2002. 78

[84] David KIRKPATRICK et Bettina SPECKMANN : Separation sensitive kinetic separation structures for convex polygons. *In Proc. Japan Conference on Discrete and Computational Geometry*, volume 2098 de *Lecture Notes in Computer Science*, pages 222–236. Springer Verlag, 2001. 78

[85] David KIRKPATRICK et Bettina SPECKMANN : Kinetic maintenance of context-sensitive hierarchical representations for disjoint simple polygons. *In Proc. 18th ACM Symposium on Computational Geometry*, pages 179–188. ACM, 2002. v, 78

[86] Samuli LAINE : A general algorithm for output-sensitive visibility preprocessing. *In ACM International Symp. on Interactive 3D Graphics and Games*. ACM Press, April 2005. 28

[87] Samuli LAINE : Split-plane shadow volumes. *In Proceedings of Graphics Hardware 2005*, pages 23–32. Eurographics Association, 2005. 117, 135

[88] Sylvain LEFEBVRE et Samuel HORNUS : Automatic cell-and-portal decomposition. Rapport technique 4898, INRIA, July 2003. 139

[89] Sylvain LEFEBVRE, Samuel HORNUS et Fabrice NEYRET : *GPU Gems 2 : Techniques for Graphics and Compute-Intensive Programming*, chapitre Hardware implementation of octree textures, pages 593–613. Addison Wesley Professional, March 2005. ISBN : 0-321-33559-7.

[90] Sylvain LEFEBVRE, Samuel HORNUS et Fabrice NEYRET : Texture sprites : Texture elements splatted on surfaces. *In ACM International Symp. on Interactive 3D Graphics and Games*. ACM Press, April 2005.

[91] Sylvain LEFEBVRE, Fabrice NEYRET, Samuel HORNUS et Joëlle THOLLOT : Mobinet : a pedagogic platform for computer science, maths and physics (how to make students love maths by programming video games). *In Eurographics - Education*. Eurographics, august 2004. Grenoble.

[92] Hans-Peter LENHOF et Michiel SMID : Maintaining the visibility map of spheres while moving the viewpoint on a circle at infinity. *Algorithmica*, 13(3):301–312, 1995. 25

[93] Alan LERNER, Yiorgos CHRYSANTHOU et Daniel COHEN-OR : Breaking the walls : Scene partitioning and portal creation. *In Proceedings of Pacific Graphics (PG'03)*, page 303. IEEE Computer Society, 2003. 141, 151

[94] Brandon LLOYD, Jeremy WENDT, Naga GOVINDARAJU et Dinesh MANOCHA : CC shadow volumes. *In Proc. Eurographics Symposium on Rendering*. Eurographics, Eurographics Association, 2004. 115, 121, 126

[95] Philippe Paul LOUTREL : Determination of hidden edges in polyhedral figures : convex case. Rapport technique 400-145, New York U., September 1966. 19

[96] Philippe Paul LOUTREL : A solution to the "hidden line" problem for computer drawn polyhedra. Rapport technique 400-167, New York U., September 1967. *Also in IEEE Transactions on Computers, vol. C-19(3), march 1970*. 19

[97] David LUEBKE et Chris GEORGES : Portals and mirrors : Simple, fast evaluation of potentially visible sets. *In* ACM PRESS, éditeur : *Proc. ACM Symposium on Interactive 3D Graphics, special issue of Computer Graphics*, pages 105–106, Monterey, CA, April 1995. 29, 139, 140

[98] Tobias MARTIN et Tiow-Seng TAN : Anti-aliasing and continuity with trapezoidal shadow maps. *In Eurographics Symposium on Rendering*. Eurographics, Eurographics Association, 2004. 121

[99] Yutaka MATSUSHITA : Hidden lines elimination for a rotating object. *Communications of the ACM*, 15(4), April 1972. 23

[100] Morgan MCGUIRE, John F. HUGHES, Kevin T. EGAN, Mark J. KILGARD et Cass EVERITT : Fast, practical and robust shadows. Rapport technique, nVIDIA, 2003. 122, 126

[101] Michael MCKENNA : Worst-case optimal hidden-surface removal. *ACM Transaction on Graphics*, 26(1):19–28, 1987. 22

[102] Daniel MENEVEAUX, Eric MAISEL et Kadi BOUATOUCH : A new partitioning method for architectural environments. Rapport technique, IRISA, April 1997. 137, 138, 141

[103] Daniel MENEVEAUX, Eric MAISEL, Coudret F. et Kadi BOUATOUCH : Partitioning complex architectural environments for lighting simulation. Rapport technique 2981, INRIA/IRISA, 1996. 137

[104] Joseph S. B. MITCHELL, David M. MOUNT et Christos H. PAPADIMITRIOU : The discrete geodesic problem. *SIAM Journal on Computing*, 16(4):647–668, 1987. 20

[105] Ketan MULMULEY : An efficient algorithm for hidden surface removal. *In SIGGRAPH '89 : Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques*, pages 379–388, New York, NY, USA, 1989. ACM Press. 22

[106] Ketan MULMULEY : Hidden surface removal with respect to a moving viewpoint. *In Proc. Annual ACM Symposium on the Theory of Computing*, 1991. 22, 26, 61, 70, 72

[107] Ketan MULMULEY : *Computational Geometry, An Introduction Through Randomized Algorithms*. Prentice Hall, 1994. 14, 84

[108] Karel NECHVÍLE et Petr TOBOLA : Dynamic visibility in the plane. *In Spring Conference on Computer Graphics*, April 1999. 24

[109] Karel NECHVÍLE et Petr TOBOLA : Local approach to dynamic visibility in the plane. *In Proc. Winter School of Computer Graphics*, February 1999. 24

[110] Shaun NIRENSTEIN et Edwin BLAKE : Hardware accelerated aggressive visibility preprocessing using adaptive sampling. *In Proc. 15th Eurographics Symposium on Rendering (EGSR)*, 2004. 28

[111] Shaun NIRENSTEIN, Edwin BLAKE et J. GAIN : Exact from-region visibility culling. *In Proc. 13th Eurographics Workshop on Rendering (EGWR)*, 2002. 28

[112] *n*VIDIA : `http://developer.nvidia.com/`. web pages. 143

[113] Marc OVERMARS et Jan van LEEUWEN : Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, 23:166–204, 1981. 34, 36

[114] Michael S. PATERSON et Frances Foong YAO : Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete & Computational Geometry*, 1989. 21, 138, 140

[115] Michael S. PATERSON et Frances Foong YAO : Optimal binary space partitions for orthogonal objects. *Journal of Algorithms*, 13(1), 1992. 21

[116] Matt PHARR et Greg HUMPHREYS : *Physically Based Rendering*. Morgan Kaufman, 2004. 5

[117] Michel POCCHIOLA : Graphics in flatland revisited. *In SWAT '90 : Proceedings of the second Scandinavian workshop on Algorithm theory*, pages 85–96, New York, NY, USA, 1990. Springer-Verlag New York, Inc. 21

[118] Michel POCCHIOLA et Gert VEGTER : Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete Computational Geometry*, 16(4):419–453, décembre 1996. Special issue devoted to the proceedings of the 11th Annual ACM Symposium on Computational Geometry (SoCG'95). 6, 39, 79, 93

[119] Michel POCCHIOLA et Gert VEGTER : The visibility complex. *International Journal on Computational Geometry and Applications*, 6(3):279–308, 1996. Special issue devoted to the proceedings of the 9th Annual ACM Symposium on Computational Geometry (SoCG'93). iv, v, 7, 21, 95

[120] Mihai POP, Gill BAREQUET, Christian A. DUNCAN, Michael T. GOODRICH, Wenjing HUANG et Subodh KUMAR : Efficient perspective-accurate silhouette computation. *In Proc. 17th Annual ACM Symposium on Computational Geometry*, pages 60–68. ACM, June 2001. 26

[121] Franco P. PREPARATA, Jeffrey Scott VITTER et Mariette YVINEC : Computation of the axial view of a set of isothetic parallelepipeds. *Transactions on Graphics*, 9(3):278–300, 1990. 22

[122] John H. REIF et Sandeep SEN : An efficient output-sensitive hidden-surface removal algorithm and its parallelization. *In Proc. 4th Annual ACM Symposium on Computational Geometry*, pages 193–200, 1988. 22

[123] Alexander RESHETOV, Alexei SOUPIKOV et Jim HURLEY : Multi-level ray tracing algorithm. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 24(3): 1176–1185, August 2005. 23

[124] Stéphane RIVIÈRE : Topologically sweeping the visibility complex of polygonal scenes. *In Proceedings of the eleventh annual symposium on Computational geometry (SoCG)*, pages 436–437, New York, NY, USA, 1995. ACM Press. 10

[125] Stéphane RIVIÈRE : *Calculs de visibilité dans un environnement polygonal 2D*. Thèse de doctorat, Université Joseph Fourier (Grenoble), 1997. v, 10, 24

[126] Lawrence G. ROBERTS : Machine perception of three-dimensional solids. Rapport technique 315, MIT, 1963.
`http://www.packet.cc/files/mach-per-3D-solids.html`. 17, 18

[127] Michael Ian SHAMOS : The early years of computational geometry—a personal memoir. *In* B. CHAZELLE, J. E. GOODMAN et R. POLLACK, éditeurs : *Advances in Discrete and Computational Geometry*, volume 223 de *Contemporary Mathematics*, pages 313–333. American Mathematical Society, Providence, 1999. iii, 20

[128] Michael Ian SHAMOS et D. HOEY : Closest-point problems. *In Proc. 16th Annual IEEE Symposium on Foundations of Computer Science*, pages 151–162, 1975. iii, 20

[129] Micha SHARIR et Mark H. OVERMARS : A simple output-sensitive algorithm for hidden surface removal. *ACM Transactions on Graphics*, 11(1):1–11, January 1992. 23

[130] Hayim SHAUL : Improved output-sensitive construction of vertical decomposition of triangles in three-dimensional space. Mémoire de D.E.A., Tel-Aviv University, 2001. 65, 70

[131] Richard SHOUP : SuperPaint : An early frame buffer graphics system. *IEEE Annals of the History of Computing*, 2001. 19

[132] Daniel D. SLEATOR et Robert Endre TARJAN : A data structure for dynamic trees. *In Proceedings of the thirteenth annual ACM symposium on Theory of computing (STOC)*, pages 114–122, New York, NY, USA, 1981. ACM Press. 87, 88

[133] Marc STAMMINGER et George DRETTAKIS : Perspective shadow maps. *In Proc. SIGGRAPH*, pages 557–562. ACM, ACM Press, 2002. 121

[134] Ileana STREINU : Pseudo-triangulations, rigidity and motion planning. *Discrete and Computational Geometry*, 34(4):587–635, novembre 2005. v

[135] Vitaly SURAZHSKY, Tatiana SURAZHSKY, Danil KIRSANOV, Steven J. GORTLER et Hugues HOPPE : Fast exact and approximate geodesics on meshes. *Transactions on Graphics (Proc. SIGGRAPH)*, 24(3), 2005. 20

[136] Ivan E. SUTHERLAND : *Sketchpad : A man-machine graphical communication system*. Thèse de doctorat, MIT, 1963. iii, 17

[137] Ivan E. SUTHERLAND, Robert F. SPROULL et Robert A. SCHUMACKER : A characterization of ten hidden-surface algorithms. *ACM Comput. Surv.*, 6(1):1–55, 1974. 19

[138] Seth J. TELLER : Computing the antipenumbra of an area light source. *In Proc. Computer Graphics (SIGGRAPH'92)*, pages 139–148, 1992. 28, 142

[139] Seth J. TELLER : *Visibility computation in densely occluded polyhedral environments*. Thèse de doctorat, UC Berkeley, CS department, 1992. iv

[140] Seth J. TELLER : *Visibility computation in densely occluded polyhedral environments*. Thèse de doctorat, UC Berkeley, CS department, 1992. 137, 140, 148, 150

[141] Seth J. TELLER et Pat HANRAHAN : Visibility computations for global illumination algorithms. *In Proc. Computer Graphics (SIGGRAPH'93)*, volume 27, pages 239–246. ACM Press, July 1993. 137, 139

[142] Seth J. TELLER et Carlo SÉQUIN : Visibility preprocessing for interactive walkthroughs. *In Proc. Computer Graphics (SIGGRAPH'91)*, volume 25, pages 61–69. ACM Press, 1991. iv, 139, 140

[143] Michiel van de PANNE et J. STEWART : Efficient compression techniques for precomputed visibility. *In Proc. Eurographics Rendering Workshop*, pages 305–316, June 1999. 140

[144] V. A. VASSILIEV : *Introduction to topology*. AMS, 2001. 7

[145] John E. WARNOCK : A hidden-surface algorithm for computer generated halftone pictures. Rapport technique TR 4-15, Computer Science Department, University of Utah, June 1969. 19

[146] Lance WILLIAMS : Casting curved shadows on curved surfaces. *In Proc. SIGGRAPH*, pages 270–274. ACM, ACM Press, 1978. 121

[147] Michael WIMMER et Jiri BITTNER : *GPU Gems 2 : Techniques for Graphics and Compute-Intensive Programming*, chapitre Hardware Occlusion Queries Made Useful, pages 91–108. Addison Wesley Professional, March 2005. ISBN : 0-321-33559-7. 139

[148] Michael WIMMER, Daniel SCHERZER et Werner PURGATHOFER : Light space perspective shadow maps. *In Proc. Eurographics Symposium on Rendering*. Eurographics, Eurographics Association, 2004. 121

[149] Alireza ZAREI et Mohammad GHODSI : Efficient computation of query point visibility in polygons with holes. *In Proceedings of the 21st annual symposium on Computational geometry (SOCG)*, pages 314–320, New York, NY, USA, 2005. ACM Press. 4, 21

Résumé

La notion de visibilité est centrale dans les domaines informatiques de la synthèse d'images et de la géométrie algorithmique. Un calcul de visibilité correspond au calcul de la forme, ou à la détermination de l'existence de l'ensemble des segments dans l'espace reliant deux objets spécifiés sans traverser aucun objet. Nous examinons le problème de la maintenance de la visibilité d'un point de vue mobile à travers le prisme d'un objet encodant la totalité des relations de visibilité existant dans une scène, le *complexe de visibilité*.

Nous effectuons deux approches différentes de ce problème. L'une s'attache à maintenir le polyèdre de visibilité d'un point mobile de façon exacte, et se voit appliquée à un algorithme de construction du complexe de visibilité d'un ensemble de polytopes disjoints.

L'autre approche a pour but de pouvoir dessiner une scène 3D complexe à une cadence élevée, en la décomposant en zones plus simples dont les relations de visibilité avec les autres zones sont connues et représentées par un graphe. On propose également un algorithme efficace pour le dessin d'ombres dures en temps réel.

Abstract

The notion of visibility is central in computer graphics and computational geometry. A visibility computation amounts to determining the shape, or merely the existence of the set of segments in space linking two specified objects without crossing any other. We examine the problem of maintaining the visibility of a moving viewpoint with the aid of a complex describing all the visibility relationships in a scene, at once : the *visibility complex*.

We take two different approaches to this problem. In one, we show how one can maintain the so-called visibility polyhedron of the moving point, in an exact fashion. We apply a variant of the algorithm to the construction of the visibility complex of a set of disjoint polytopes.

The second approach is motivated by the need to render complex 3D scenes at interactive rates. We propose an algorithm that decomposes a 3D scene into simple cells related together with simple visibility relationships represented as a graph, allowing pruning to speed up rendering. We also provide an efficient algorithm for real-time rendering of hard shadows.