Chapitre 1

Visibilité dans le plan

Dans ce chapitre nous indiquons où se situent nos travaux – utilisation de la cohérence spatiale et temporelle dans les calculs de visibilité – par rapport aux problèmes de visibilité dans le plan. Nous exposons ensuite les structures de données – graphe et complexe de visibilité – au centre de ces travaux et les techniques algorithmiques – balayage et dualité – principales que nous utilisons.

— Seurhăne, ma Seurhăne ! Ne vois tu rien venir ?
Astérix chez Rahăzade.

1.1 Nature des problèmes de visibilité dans le plan

Les problèmes de visibilité font partie des problèmes fondamentaux de la géométrie algorithmique et de synthèse d'image. Dans les problèmes de planification de trajectoire par exemple, les objets d'une scène sont des obstacles à éviter et le calcul d'une trajectoire fait intervenir les relations de visibilité entre les objets de la scène.

Si nous cherchons plutôt à visualiser une scène, seuls les objets visibles doivent apparaître à l'écran : il vaut mieux alors éviter de faire des calculs coûteux sur les objets cachés car le temps passé dans ces calculs n'aura pas d'incidences sur l'affichage de la scène.

Nous retrouvons aussi des problèmes de visibilité aussi au cœur des problèmes de surveillance ou d'éclairage.

Une grande partie des problèmes de planification de trajectoire concerne divers véhicules qui se déplacent au sol, c'est-à-dire sur un plan. Bien que l'image de synthèse concerne principalement des scènes dans l'espace 3D, certains problèmes de visualisation ont une base 2D : les calculs effectués pour afficher ce que voit un piéton se déplaçant dans une ville font intervenir le plan (2D) de la ville. Les problèmes de surveillance sont eux-aussi principalement des problèmes 2D. L'étude des relations de visibilité dans le plan n'est donc pas limitée à un seul intérêt théorique.

Les relations de visibilité dans le plan ont d'abord été étudiées sur des objets simples, notamment sur des ensembles de segments disjoints. Certains algorithmes ont même été adaptés à partir d'algorithmes opérant sur des ensembles de points (calcul de graphe de visibilité d'Edelsbrunner et Guibas [EG86] par exemple). Le cas de scènes polygonales a été aussi abordé, mais beaucoup d'algorithmes ne considèrent en fait que des scènes composées d'un seul polygone, où l'espace libre de la scène est limité à l'intérieur du polygone (calculs de visibilité de Chazelle et Guibas [CG89] ou de Heffernan et Mitchell [HM90] par exemple).

Notre travail porte sur des scènes polygonales, c'est-à-dire des scènes composées de polygones simples ayant deux à deux une intersection nulle.

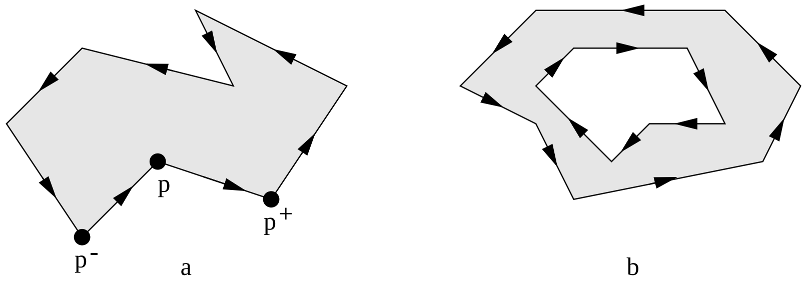


FIG. 1.1 – *Définition des polygones.* **a.** *Orientation des sommets.* **b.** *Polygone avec trou.*

Nous orientons les sommets d'un polygone de telle façon que le polygone soit localement à gauche de deux sommets consécutifs. Pour un sommet p nous notons p^+ (resp. p^-) le sommet suivant (resp. précédent) sur le polygone (figure 1.1a). Les polygones ont deux à deux une intersection nulle, mais un polygone peut en contenir un autre qui, s'il est correctement orienté, forme un trou (figure 1.1b). Ces trous peuvent à leur tour contenir d'autres polygones...

Un polygone peut représenter un mur délimitant un bâtiment, mur dont les pans peuvent avoir des caractéristiques différentes (couleur, matière, réflectance, etc.). Ces pans devront être considérés individuellement lors de l'affichage de la scène par exemple. Il est alors plus facile de visualiser les polygones si nous avons directement la liste des côtés visibles à afficher plutôt que de la calculer à partir de la liste des polygones. Un côté de polygone peut aussi représenter une fenêtre, et nous voulons alors calculer la partie de la scène visible depuis ce côté seulement et non pas la partie visible depuis tout le polygone.

Par conséquent, les objets élémentaires que nous considérons dans les relations de visibilité ne sont pas les polygones eux-mêmes, mais les côtés de polygone. Pour nous, un polygone est un objet composite composé d'éléments ayant chacun leurs propriétés propres. De plus, ce point de vue permet de ne pas se préoccuper de la convexité ou non des polygones dans les algorithmes que nous étudions.

Les divers problèmes de visibilité étudiés peuvent être regroupés en deux catégories générales : calculs globaux et calculs locaux de visibilité.

1.1.1 Calculs globaux de visibilité

Les calculs globaux de visibilité concernent les calculs où les données du problème sont constituées par la scène elle-même : le programme ne prend en entrée que les coordonnées des objets définissant la scène, effectue des calculs de visibilité sur les objets de la scène et renvoie les résultats du calcul global demandé.

Une grande catégorie de ces problèmes concerne les problèmes de gardiennage, appelés *problèmes de galerie d'art* en référence à la surveillance des œuvres d'art dans les musées. L'énoncé type de ces problèmes est :

> *étant donné un polygone simple de n côtés, combien faut-il mettre de gardes aux sommets du polygone pour que tout point de la frontière intérieure du polygone soit visible d'au moins un garde, et comment placer ces gardes ?*

La réponse à cette question a été donnée par Chvátal [Chv75] qui a montré que $\lfloor n/3 \rfloor$ gardes étaient toujours suffisants et parfois nécessaires. Avis et Toussaint [AT81] ont donné un algorithme de placement qui s'exécute en temps $O(n \log n)$.

Divers variantes de ce problème ont été étudiées : spécialisation pour des polygones particuliers (en étoile, orthogonaux...), gardes mobiles le long d'une arête, surveiller l'extérieur du polygone, etc. Le livre d'O'Rourke [O'R87] décrit en détails les divers problèmes de galerie d'art.

Des problèmes similaires se posent dans des calculs d'éclairage, où des sommets, des côtés de polygone, représentent des lampes illuminant l'intérieur du polygone.

Le calcul du graphe de visibilité, qui consiste à calculer toutes les paires de sommets de la scène mutuellement visibles, est un autre problème de calcul global de visibilité très étudié (il permet ensuite de faire des calculs de planification de trajectoire). Nous étudions ce calcul en détail dans la section 1.4.

1.1.2 Calculs locaux de visibilité

Les calculs locaux concernent plutôt le calcul de réponses à des requêtes locales de visibilité. La scène fait partie du contexte, et les données du problème sont constituées des données supplémentaires qui définissent la requête.

Les requêtes les plus courantes concernent le calcul des objets vus depuis un point de vue, que ce soit dans le contexte de calculs discrets comme dans le lancer de rayons ou dans le contexte de calculs continus comme dans le calcul d'une vue autour d'un point.

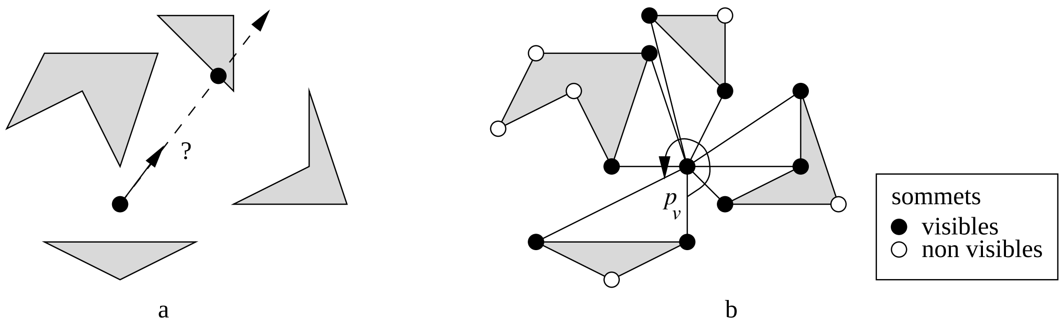


FIG. 1.2 – *Requêtes locales de calcul de visibilité.* **a.** *Calcul discret : lancer de rayons.* **b.** *Calcul continu : calcul de vue.*

L'énoncé d'une requête de lancer de rayons est :

> *soit un point et une direction de vision, quel est le premier objet de la scène intersecté par la demi-droite issue du point dans la direction donnée ?*(figure 1.2a)

Le lancer de rayons est utilisé en synthèse d'image pour visualiser une scène. L'image à calculer est déterminée par un point de vue et un écran de projection : c'est l'image de la scène obtenue sur l'écran quand elle est regardée à partir du point de vue. L'image est discrétisée en pixels (éléments de couleur uniforme) et pour la calculer, un rayon issu du point de vue et passant par le centre de chaque pixel est lancé dans la scène. La couleur résultante d'un pixel est celle du point de l'objet de la scène coupé par le rayon correspondant.

Ce calcul est discret : il ne concerne la visibilité que d'un rayon. Le calcul doit être refait à chaque fois que la direction de vision est modifiée, même si le point de vue reste le même.

La vue – ou polygone de visibilité – autour d'un point permet d'avoir une description continue dans l'espace objet de la visibilité autour du point de vue :

> *la vue autour d'un point p_v est l'ensemble des sommets de polygone de la scène vus depuis ce point et classés par ordre angulaire.* (figure 1.2b)

La vue code les limites de visibilité autour du point de vue : entre deux sommets consécutifs de la vue, l'objet vu depuis p_v est le même. Cette description est une description continue de la visibilité depuis ce point : elle ne fait intervenir aucune discrétisation arbitraire.

Les calculs continus permettent parfois de résoudre plus efficacement des problèmes d'abord résolus par une série de calculs discrets. Par exemple, le calcul d'une image en lancer de rayons nécessite de lancer un rayon par tous les pixels de l'image alors que l'image résultante peut directement être obtenue par un calcul de vue : il suffit de déterminer la projection sur l'écran des limites de visibilité.

Nous venons de décrire des calculs statiques, où tous les objets concernés sont immobiles. Les calculs de visibilité ont aussi une grande importance dans des situations dynamiques, où les données du problème (objets de la scène, point de vue) varient au cours du temps.

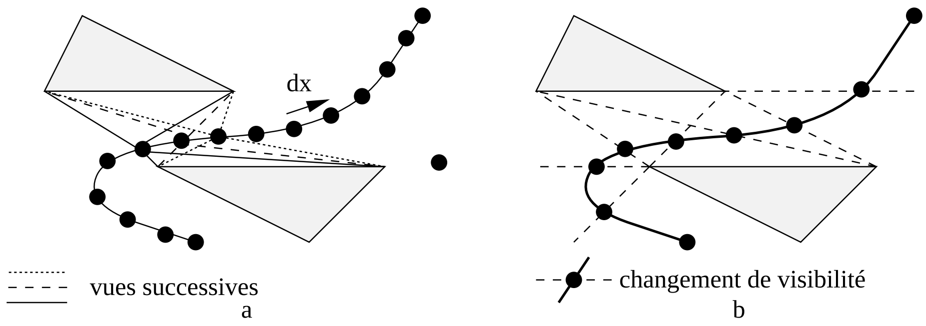


FIG. 1.3 – *Calcul de vue le long d'une trajectoire.* **a.** *Calcul discret : recalcul à chaque déplacement dx.* **b.** *Calcul continu : calcul des limites de visibilité.*

La simulation d'une personne en mouvement nécessite de pouvoir calculer la vue autour d'un point se déplaçant le long d'une trajectoire. Là aussi deux approches peuvent être envisagées pour ce calcul. Dans l'approche discrète, la vue est entièrement recalculée à chaque déplacement élémentaire dx du point de vue (figure 1.3a).

Cependant, pour accélérer les calculs il est préférable d'adopter une approche continue, c'est-à-dire de calculer les limites dans lesquelles l'ensemble d'objets vus reste le même et de faire, si possible, lors d'un changement de visibilité une mise à jour incrémentale moins chère qu'un recalcul complet (figure 1.3b).

L'utilisation de l'approche continue pour résoudre un problème de visibilité permet en général une résolution plus souple et plus économique. En effet, l'approche continue utilise la cohérence spatiale de la scène – comme dans le cas du calcul d'une vue – ou la cohérence temporelle du problème – comme dans le cas du calcul d'une vue le long d'une trajectoire – pour déterminer rapidement si la visibilité a changé et le cas échéant pour mettre à jour la visibilité sans devoir la recalculer entièrement. Les travaux que nous présentons dans cette thèse sont centrés sur l'utilisation de l'approche continue – utilisation de la cohérence spatiale et temporelle – pour répondre à des requêtes de calculs de visibilité statique et dynamique.

La cohérence spatiale ou temporelle ne peut cependant être pleinement utilisée que si certains calculs de visibilité ont déjà été effectués auparavant, notamment lors d'une phase de prétraitement.

1.1.3 Prétraitement

Dans la plupart des cas, le résultat d'une requête locale – calcul des objets vus depuis un point par exemple – ne fait intervenir qu'une petite partie des objets de la scène. Si ce calcul est fait en considérant tous les objets de la scène, alors le temps de calcul ne dépend que de la taille de la scène, quelle que soit la taille du résultat. Or tous ces calculs sont inutiles si au final un seul objet est vu ! Pour accélérer les calculs et obtenir des algorithmes *sensibles à la sortie*, c'est-à-dire dont le temps de calcul dépend de la taille du résultat calculé, il ne faut tenir compte si possible que des objets vus. Ceci n'est en général possible que si une certaine connaissance de la scène – répartition des objets, relations de visibilité, etc. – a déjà été acquise. Cette connaissance est construite lors d'une étape de prétraitement, où des calculs préliminaires sont effectués une fois pour toute avant toute requête. Ce prétraitement construit en général certaines structures de données qui permettent ensuite d'améliorer le temps de calcul des requêtes.

Le prétraitement s'adresse aux calculs locaux de visibilité, cependant un calcul global de visibilité peut servir d'étape de prétraitement pour ces calculs locaux.

Nous nous intéressons dans cette thèse à la construction et à l'utilisation de structures globales pour accélérer les calculs de visibilité.

1.2 Partitions dans le plan

Une grande catégorie de prétraitements est basée sur des schémas de découpage du plan dont le but est de subdiviser le plan en régions contenant moins d'objets. Lors d'un calcul de visibilité, une première série de tests de visibilité est effectuée sur les régions issues du découpage. Les régions non visibles, et tous les objets qu'elles contiennent, sont éliminés du calcul. Après ce premier calcul, qui ne doit pas être trop coûteux, le nombre d'objets restant à considérer est en général plus petit que le nombre total d'objets et prend moins de temps à traiter.

1.2.1 Boîtes englobantes

Une première méthode pour découper le plan est de regrouper les objets en groupes, et d'utiliser pour la phase de sélection la boîte englobante de ces groupes objets. Une boîte englobante est une région de l'espace, généralement un polygone, qui contient tous les objets du groupe. Ces boîtes englobantes peuvent avoir des côtés parallèles aux axes, être constituées par l'enveloppe convexe des objets, etc. La première étape d'un calcul de lancer de rayons par exemple, consiste à tester l'intersection entre le rayon et les boîtes englobantes. Ensuite un calcul normal de lancer de rayons est effectué, non pas sur tous les objets de la scène, mais seulement sur ceux contenus dans les boîtes englobantes coupées successivement par le rayon (figure 1.4).

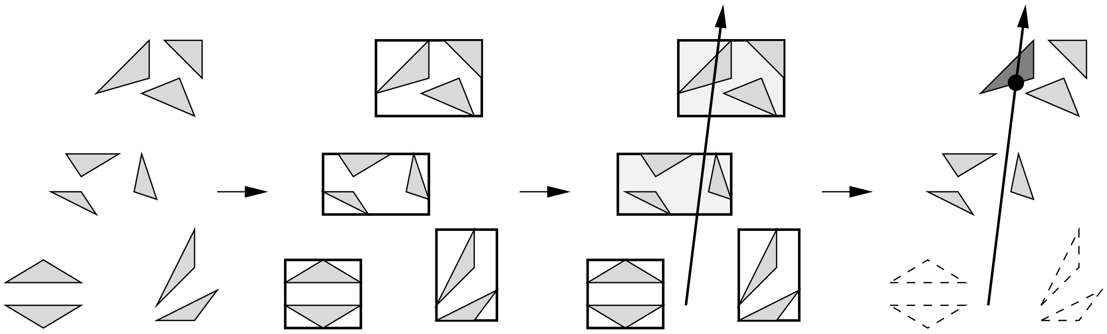


Fig. 1.4 – *Utilisation des boîtes englobantes dans le lancer de rayons.*

Les boîtes englobantes sont bien adaptées aux calculs discrets de visibilité. Cependant leur utilité dans des calculs continus est très limitée. De plus, la construction automatique de ces boîtes, ou plus précisément le regroupement optimal des objets en agrégats, n'est pas une tâche facile et fait toujours l'objet de recherches.

1.2.2 Grilles, grilles récursives

D'autres types de méthodes de découpage sont basées sur la subdivision de l'espace en régions et permettent de se déplacer d'une région à une autre sans considérer toute la scène.

La façon la plus simple de découper automatiquement la scène est d'utiliser une grille régulière. À chaque rectangle de la grille est associée la liste des objets qu'il contient. Lors d'un calcul de lancer de rayons, ne sont considérés que les objets contenus dans les rectangles de la grille coupés successivement par le rayon (figure 1.5). La structure de grille permet d'avoir directement les rectangles coupés par le rayon sans avoir à considérer les autres rectangles : il est en effet possible de passer d'un rectangle à un autre rectangle incident en temps constant.

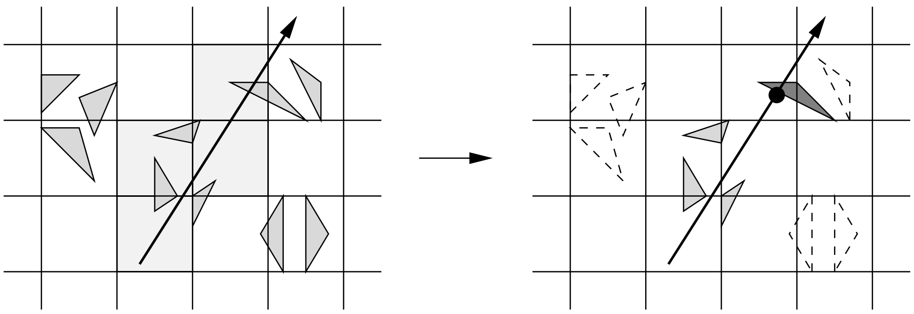


Fig. 1.5 – *Utilisation d'une grille dans le lancer de rayons.*

Dans le cas où les objets de la scène ne sont pas uniformément répartis, il est plus utile d'utiliser des grilles adaptatives qui tiennent compte de la répartition des objets et qui évitent le découpage de régions vides. Ces grilles sont souvent définies de façon récursive.

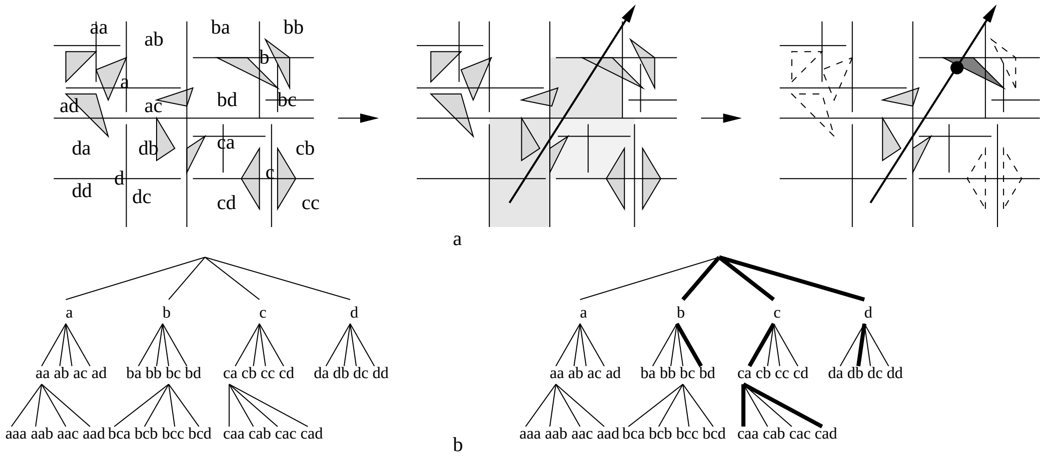


FIG. 1.6 – *Utilisation d'arbre quadtree dans le lancer de rayons.* **a.** *Découpage de la scène.* **b.** *Arbre correspondant.*

Dans les arbres *quadtree* par exemple, une région est subdivisée en quatre, et chacune des sous-régions est elle même subdivisée à son tour en quatre... La subdivision s'arrête quand une région devient vide (figure 1.6a). Ces subdivisions sont représentées sous la forme d'un arbre : chaque région est un nœud de l'arbre et a pour fils ses quatre sous-régions (figure 1.6b). De plus, à chaque région est associée la liste des objets qu'elle contient. Plusieurs méthodes peuvent être employées pour effectuer la subdivision : subdivision régulière, subdivision cherchant à répartir uniformément les objets dans les sous-régions... La structure d'arbre peut être utilisée pour rechercher en temps logarithmique les régions traversées par un rayon dans une requête de lancer de rayons.

Il existe d'autres types de grilles récursives, comme les *kd*-arbres qui subdivisent les régions en deux sous-régions selon une direction alternativement parallèle à l'axe des abscisses et à l'axe des ordonnées ; cependant le principe général d'utilisation reste le même (pour plus de détails, voir le survol de Samet [Sam84]).

Ces grilles ont l'avantage par rapport aux boîtes englobantes d'introduire une utilisation de la cohérence spatiale grâce à la possibilité de considérer pour une région donnée ses régions incidentes. Les grilles sont utiles dans le cadre de calculs discrets locaux ou dans le cadre de calculs plus globaux (comme déterminer l'ordre d'affichage des objets dans l'algorithme du peintre). En revanche, elles restent peu adaptées aux calculs continus de visibilité car elles permettent difficilement de calculer des limites de visibilité.

1.2.3 Triangulations

Découper la scène en régions ne signifie pas uniquement découper la scène en régions contenant des groupes d'objets. Le découpage de l'espace libre, le plan auquel on a ôté les objets de la scène, permet aussi d'accélérer les calculs. Un des découpages les plus connus est la triangulation, qui consiste à découper l'espace libre d'une scène polygonale en triangles dont les sommets sont les sommets des polygones de la scène (figure 1.7).

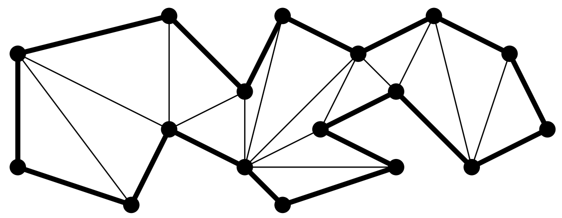


FIG. 1.7 – *Triangulation d'un polygone.*

Les triangulations servent notamment à résoudre des problèmes de visibilité quand l'espace libre est l'intérieur d'un polygone simple, comme par exemple des problèmes de lancer de rayons, de plus courts chemins ou d'illumination par un néon (voir par exemple [GHL+87]).

Bien qu'il existe un algorithme pour trianguler un polygone simple en temps linéaire, celui-ci est assez compliqué. D'autres découpages utilisant des formes polygonales plus grandes et générant moins de régions ont été étudiés pour résoudre ces problèmes, comme par exemple les profils de visibilité structurés d'Heffernan et Mitchel [HM90] ou les décompositions en trapèzes de Chiang, Preparata et Tamassia [CPT96].

Ces découpages introduisent une plus grande utilisation de la cohérence spatiale : les frontières des régions découpées s'appuient sur les côtés des objets. Cependant cela ne suffit pas toujours à réduire la complexité des calculs : un rayon peut traverser un nombre important de régions avant de heurter un des côtés du polygone. Pour éviter ces sauts de complexité, les auteurs greffent sur les structures de découpage des structures de recherche purement combinatoire (fractional cascading, $BB(\alpha)$-trees, etc.) et relativement compliquées afin d'obtenir des complexités intéressantes même dans les cas les pires.

1.3 Travaux présentés dans ce mémoire

Le schéma heuristique [1] de la figure 1.8 fait le point sur les problèmes de visibilité dans le plan. Il indique les relations entre ces problèmes, les structures de données, les algorithmes, etc., et souligne (en gras) les problèmes abordés dans cette thèse.

Plus précisément, nous cherchons dans ce travail de thèse à utiliser la cohérence spatiale et temporelle pour résoudre des problèmes de visibilité. Pour cela nous voulons utiliser des structures de données purement géométriques qui ne requièrent pas l'utilisation supplémentaire de structures combinatoires de recherche compliquées (pas plus compliquées qu'un arbre de recherche équilibré par exemple) pour obtenir de bonnes performances.

[1] Tony Buzan, *Une tête bien faite*, Éditions d'Organisation.

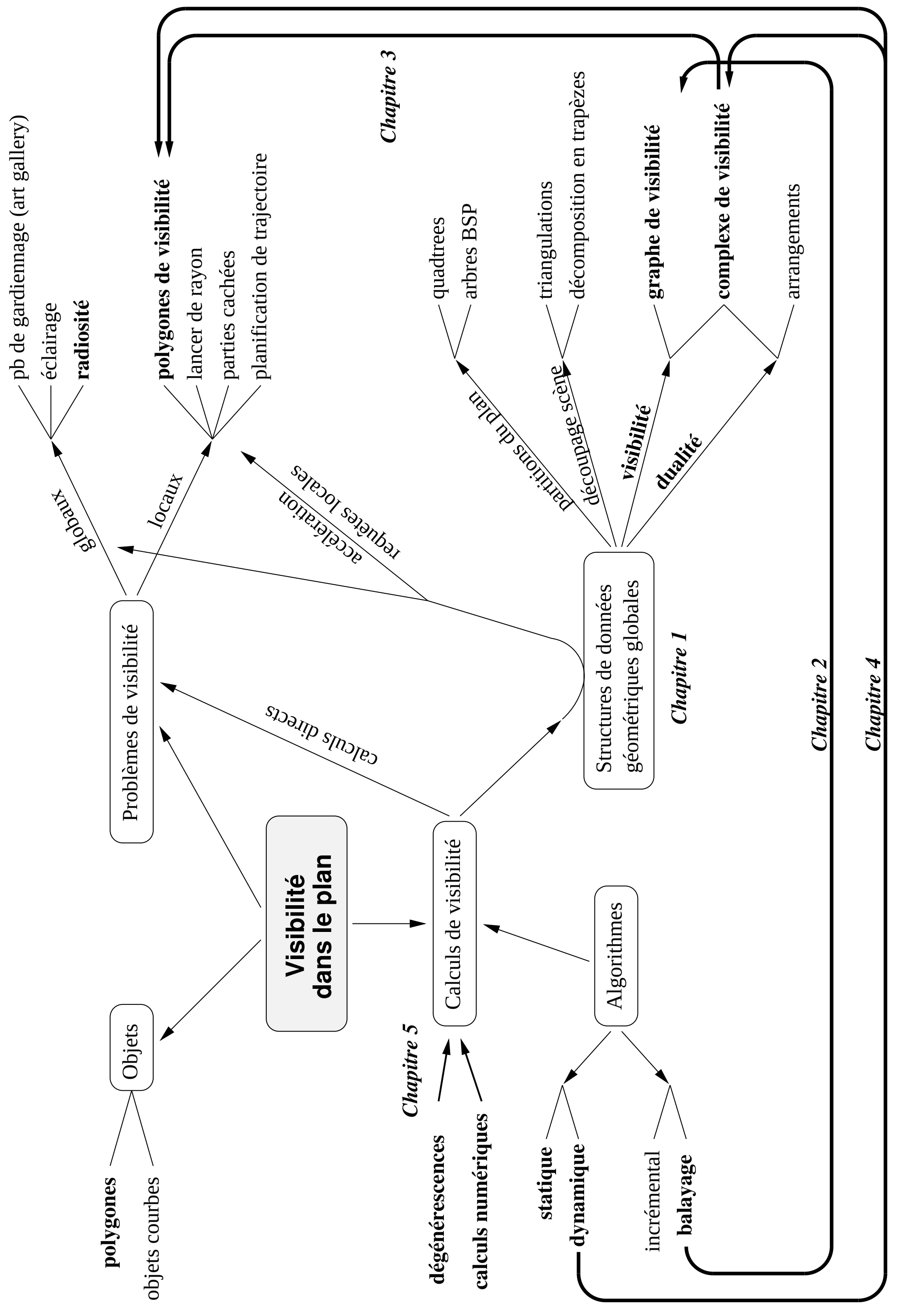


FIG. 1.8 – *Schéma heuristique décrivant les problèmes de visibilité dans le plan et indiquant les sujets traités dans ce mémoire.*

Le reste de ce chapitre 1 est consacré à l'étude des structures de données que nous allons utiliser : le graphe et le complexe de visibilité. Cette étude nous permettra aussi d'introduire deux outils de géométrie algorithme que nous utilisons presque constamment dans les algorithmes que nous étudions : le calcul par balayage et la dualité.

Dans le chapitre 2 nous décrivons un algorithme optimal pour construire ces structures de données.

Dans le chapitre 3 nous montrons comment utiliser le complexe de visibilité pour calculer des polygones de visibilité :

- le polygone de visibilité d'un point, qui représente la vue autour de ce point ;
- le polygone de visibilité d'un segment, qui représente la zone de la scène éclairée par le segment considéré comme un néon.

Dans le chapitre 4 nous abordons des problèmes de visibilité dynamique : nous étudions comment maintenir la vue d'un point qui se déplace et comment maintenir le complexe de visibilité quand les objets de la scène sont en mouvement. Les algorithmes de maintien de vue que nous donnons améliorent les temps de calculs des algorithmes déjà existants.

Tous les algorithmes que nous présentons sont développés dans le cadre de scènes d'objets en *position générale*, c'est-à-dire où les sommets des polygones ne sont pas alignés. Nous supposons donc que les scènes ne contiennent pas de configurations dégénérées telles que celles données en exemple dans la figure 1.9

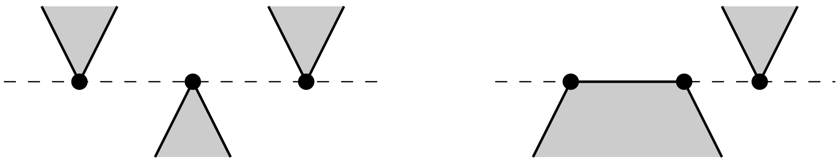


FIG. 1.9 – *Configurations dégénérées où des sommets de polygones sont alignés.*

Nous supposons aussi que tous les calculs effectués dans ces algorithmes sont faits de façon exacte. Les problèmes de dégénérescence et d'imprécision des calculs numériques ne sont pas oubliés pour autant : ces problèmes sont importants en pratique et font l'objet du chapitre 5 de ce mémoire.

Enfin, nous faisons dans le dernier chapitre un bilan des travaux présentés et indiquons des directions de recherches permettant de poursuivre et compléter ces travaux.

1.4 Structures de visibilité

Nous étudions ici des structures de données directement liées à la visibilité – graphe et complexe de visibilité – qui d'une certaine façon codent les relations de visibilité entre objets. Pour mener cette étude, nous utilisons notamment des relations de dualité entre droites et points qui permettent de mieux appréhender ces structures.

1.4.1 Graphe de visibilité

Une des structures globales de visibilité les plus étudiées est le *graphe de visibilité*.

Étant donné une scène composée de polygones simples disjoints, le graphe de visibilité de cette scène est le graphe $(\mathcal{V}, \mathcal{E})$ tel que

- $\mathcal{V}$, ensemble des sommets du graphe, est l'ensemble des sommets des polygones ;
- $\mathcal{E}$, ensemble des arêtes du graphe, est l'ensemble des paires $e = (p, q)$ de sommets de $\mathcal{V}$ tels que le segment $[p, q]$ a une intersection nulle avec l'intérieur des polygones.

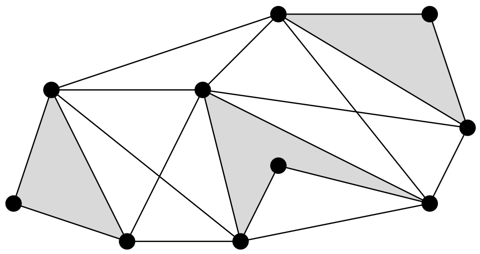
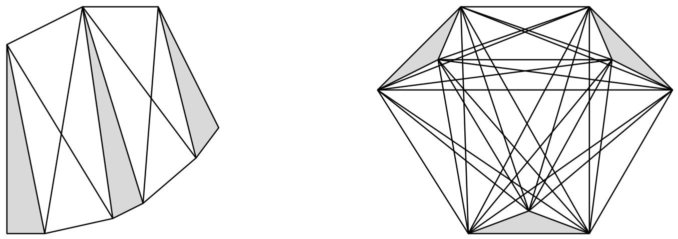


FIG. 1.10 – *Graphe de visibilité d'une scène.*

Les arêtes du graphe de visibilité relient donc les sommets de la scène mutuellement visibles. La figure 1.10 montre la représentation géométrique du graphe de visibilité, où les segments correspondant aux arêtes du graphe ont été tracés dans la scène. Le graphe de visibilité contient au moins les arêtes correspondant aux côtés des polygones et aux segments de l'enveloppe convexe de la scène. Ce graphe n'est pas orienté, mais les arêtes incidentes à un sommet sont classées dans l'ordre dans lequel sont répartis les segments correspondants dans la scène.

Le nombre de sommets de ce graphe est le nombre total n de sommets des polygones. Son nombre d'arêtes, noté m, varie en revanche d'une taille linéaire ($m = O(n)$) à une taille quadratique ($m = O(n^2)$). La taille minimale est obtenue quand chaque polygone ne voit qu'un seul polygone et qu'il n'y a que quatre arêtes de visibilité entre ces paires de polygones (figure 1.11a) ; la taille est alors $m = n + 4(n_p - 1)$, où n_p est le nombre de polygones. La taille maximale est obtenue quand tous les sommets sont mutuellement visibles (figure 1.11b) ; le graphe de visibilité est alors le graphe complet K_n de taille maximale $m = n(n-1)/2$.



a b

FIG. 1.11 – *Graphes de visibilité.* **a.** *De taille minimale.* **b.** *De taille maximale.*

Les graphes de visibilité servent par exemple en planification de trajectoire : dans une scène de polygones, le plus court chemin évitant les obstacles entre deux points p

et q emprunte les arêtes du graphe de visibilité (figure 1.12). Le calcul de ce chemin se ramène alors à un calcul de plus court chemin dans le graphe de visibilité, où le poids des arêtes du graphe est la longueur de l'arête géométrique (voir [LPW79]).

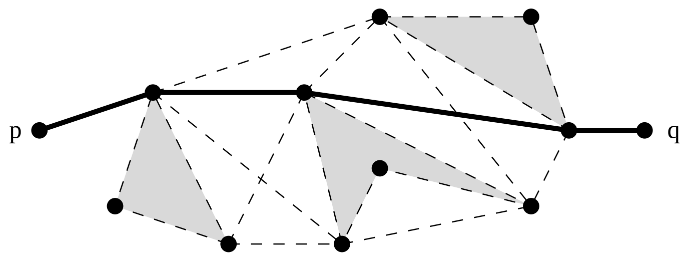


FIG. 1.12 – *Plus court chemin entre deux points en présence d'obstacles.*

L'algorithme naïf pour calculer le graphe de visibilité consiste à examiner toutes les paires (p, q) de sommets et à vérifier si les segments $[p, q]$ correspondants ne sont pas coupés par un côté de polygone. Pour chaque paire, la vérification prend un temps $O(n)$, soit un temps total de calcul en $O(n^3)$. Même si par chance il suffit de tester $O(1)$ côtés pour les segments ne correspondant pas à une arête du graphe de visibilité, il faut tester les n côtés avant de s'apercevoir qu'un segment correspond à une arête de visibilité, soit un coût total de $O(n^2 + (n-1)m)$. Que m soit linéaire ou quadratique, le temps de calcul est toujours n fois trop long par rapport au temps optimal $O(n \log n + m)$.

Grâce au balayage d'un espace de configuration annexe (un arrangement de droites), Edelsbrunner et Guibas [EG86] ont réduit le coût du calcul de graphe de visibilité d'une scène de n segments à un coût en $O(n^2)$, qui n'est optimal que pour le calcul de graphes denses. Overmars et Welzl [OW88] ont transposé ces techniques dans la scène de segments, ce qui leur a permis d'obtenir ensuite un algorithme sensible à la sortie, de complexité $O(m \log n)$. Ces algorithmes utilisent des notions et des techniques (dualité, balayage) au centre de nos travaux ; nous les détaillons dans les parties qui suivent, en les adaptant en général au cadre de scènes polygonales.

Entre-temps, Gosh et Mount [GM91] ont élaboré le premier algorithme de calcul de graphe de visibilité de polygones en temps optimal $O(n \log n + m)$. Ils utilisent un calcul incrémental (qui utilise donc un espace mémoire en $O(m)$) assez compliqué et reconnaissent eux-mêmes que leur algorithme est inutilisable en pratique, car les constantes impliquées dans le temps de calcul sont assez importantes. Nous ne détaillons pas cet algorithme dans ce mémoire.

Dans le cadre de scènes composées d'objets courbes convexes, Pocchiola et Vegter [PV96b] ont élaboré une nouvelle structure de données qui étend la notion de graphe de visibilité : le *complexe de visibilité*. Cette structure leur permet de calculer le graphe de tangence de la scène en temps optimal $O(n \log n + m)$, d'abord par un algorithme incrémental [PV96b], puis par un algorithme de balayage [PV96a].

En reprenant les concepts de complexe de visibilité, de dualité et de balayage dans le cadre de scènes polygonales, nous avons élaboré un nouvel algorithme de calcul de graphe de visibilité, le premier qui soit optimal en temps ($O(n \log n + m)$) et en espace de travail ($O(n)$). Nous l'exposons en détail dans le chapitre suivant.

Nous exposons jusqu'à la fin de ce chapitre les techniques et les structures de données que nous utilisons ensuite dans nos travaux.

1.4.2 Dualité et problèmes de visibilité

L'idée derrière l'utilisation de la dualité en géométrie est celle de l'adage « si quelque chose ne marche pas, essayez quelque chose de différent ». Il arrive que certains (ensembles d') objets et certaines propriétés géométriques soient difficiles à manipuler directement. L'utilisation d'une transformation géométrique peut permettre de transformer l'espace de départ en un autre espace, où les objets étudiés sont transposés en d'autres objets, où les propriétés de départ sont transposées en d'autres propriétés, le tout étant plus facile à manipuler. Cela permet de résoudre le problème dans ce nouvel espace – l'espace dual – puis de retransposer si nécessaire la solution dans l'espace de départ.

Il y a par exemple une correspondance naturelle entre les hyperplans et les points. La transformation la plus connue entre hyperplans et points est l'inversion de centre O, notée *, qui dans $\mathbb{R}^n$ associe à l'hyperplan H d'équation $\sum_{i=1}^{n} a_i x_i + 1 = 0$ le point de coordonnées $H^*(a_1, \ldots, a_n)$. Cette bijection a la propriété d'inverser la relation d'inclusion : en notant H^+ et H^- les deux demi-espaces délimités par l'hyperplan H (H^+ contenant O), alors pour tout point p

$$p \in H \text{ (resp. } H^+\text{, resp. } H^-\text{)} \iff H^* \in p^* \text{ (resp. } p^{*+}\text{, resp. } p^{*-}\text{).}$$

Cette transformation permet alors de manipuler un ensemble de points, plus facile à manier que l'ensemble d'hyperplans de départ.

D'une manière générale, on appelle *dualité* toute bijection entre points et hyperplans qui inverse la relation d'inclusion. Nous détaillons maintenant l'utilité et l'utilisation de relation de dualité dans le cadre de problèmes de visibilité dans le plan.

Quand nous travaillons sur des problèmes de visibilité dans le plan, nous manipulons en fait des droites orientées qui représentent les directions de vision, et les objets vus sont ceux coupés par ces droites. Pour accélérer les calculs de visibilité, nous voulons classer les droites en différents ensembles qui les regroupent en fonction des objets qu'elles coupent. Ainsi, pour savoir par exemple quel objet est vu dans telle direction, il suffit d'identifier à quel ensemble appartient la droite correspondante et de consulter les propriétés de visibilité de cet ensemble, propriétés communes à toutes les droites de l'ensemble.

Il est difficile de manipuler directement des ensembles de droites : comment peut-on les représenter, tester l'appartenance d'une droite à un ensemble, ou définir les frontières communes entre ces ensembles ? L'utilisation de relations de dualité nous permet de transformer ces ensembles de droites en ensembles de points, plus faciles à appréhender.

Nous considérons l'espace des droites orientées. Ainsi, les deux droites passant par un point p et de vecteur directeur respectif $\vec{v}$ et $-\vec{v}$, bien que géométriquement confondues, sont considérées comme étant deux droites distinctes.

Il y a plusieurs façons d'associer un point à une droite. Deux transformations souvent utilisées sont (figure 1.13a) :

- $d : y = ax + b \to d^*\ (a, b)$ (dualité (a, b))
- $d : y \cos\theta - x \sin\theta - u = 0 \to d^*\ (\theta, u)$ (dualité (θ, u))

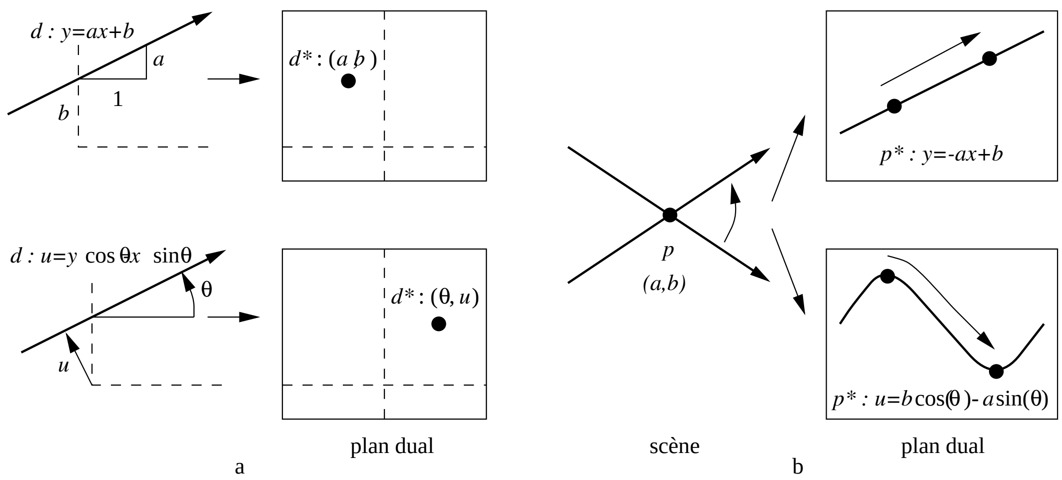


FIG. 1.13 – *Relations de dualité.* **a.** *Point dual d'une droite.* **b.** *Courbe duale d'un point.*

Ces deux relations ont beaucoup de propriétés communes, et leurs différences ne sont pas fondamentales (ce point est précisé plus loin). Dans la suite, la notation * désigne indifféremment l'une ou l'autre des deux relations quand les propriétés évoquées sont les mêmes ; s'il y a des différences, nous préciserons alors la relation concernée.

L'axe des abscisses du plan dual utilisé représente la pente (ou l'angle) des droites. Si d_1 et d_2 sont deux droites parallèles telles que d_1 soit au-dessus de d_2, alors les points duaux d_1^* et d_2^* ont même abscisse et d_1^* est au-dessus de d_2^*.

Pour résoudre les problèmes de visibilité que nous étudions, nous voulons, étant donné un côté de polygone (p, q), pouvoir considérer les droites qui coupent ce segment. Or une droite de direction θ coupe le segment (p, q) si et seulement si elle est située entre les droites de même direction θ et passant respectivement par p et q. Les droites passant par un sommet représentent des limites de visibilité, il est donc intéressant de considérer l'ensemble des droites passant par un sommet donné $p = (a, b)$. À chaque droite passant par p est associé un point dans l'espace dual. Quand la droite tourne autour de p, le point dual décrit une courbe duale p^* dans l'espace dual : la droite d'équation $y = -ax + b$ (dualité (a, b)) ou la sinusoïde d'équation $u(\theta) = b \cos\theta - a \sin\theta$ (dualité (θ, u)) (figure 1.13b).

Nous retrouvons avec ces courbes duales des propriétés d'inversion des relations d'inclusion. En effet, la courbe duale p^* d'un point p représente dans le plan dual le graphe d'une fonction qui sépare le plan en deux demi-espaces : p^{*+} (resp. p^{*-}) est le demi-espace ouvert situé au-dessus (resp. en dessous) de la courbe p^*. Nous avons alors les relations (figure 1.14a)

$$p \in d \text{ (resp. } d^+\text{, resp. } d^-\text{)} \iff d^* \in p^* \text{ (resp. } p^{*-}\text{, resp. } p^{*+}\text{)}.$$

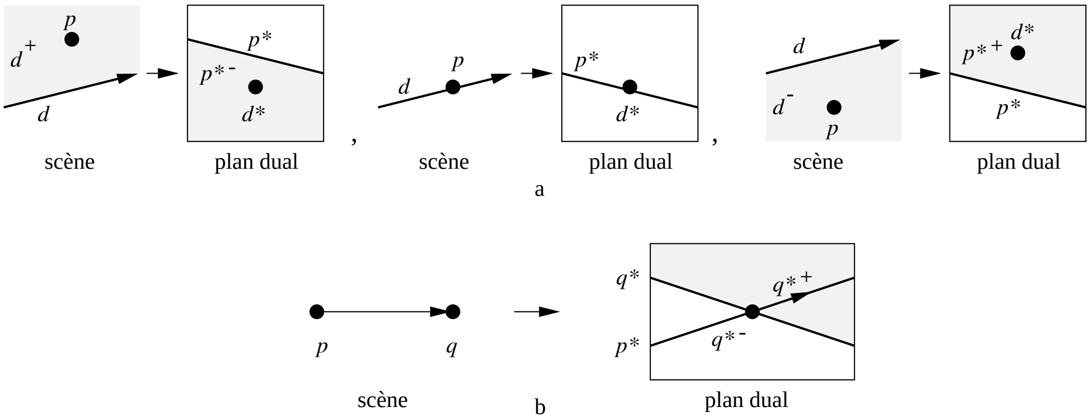


FIG. 1.14 – *Propriétés de la dualité.* **a.** *Inversion de la relation d'inclusion.* **b.** *Position relative et intersection dans le plan dual.*

Ces courbes duales ont d'autres propriétés. Si nous considérons la droite orientée (p, q), alors dans l'espace dual les courbes duales p^* et q^* se coupent en le point dual $(p, q)^*$, de telle façon que p^* passe de q^{*-} à q^{*+} (p^* traverse q^* de bas en haut) (figure 1.14b). Cette propriété permet d'obtenir directement dans l'espace dual des informations sur la position relative des objets par rapport à une direction de vision.

Reprenons notre côté de polygone (pq). Les deux courbes duales découpent l'espace dual en trois parties : au-dessus, entre et en dessous des deux courbes. Alors une droite d coupe le segment (pq) si et seulement si son point dual d^* est dans l'espace situé entre les deux courbes p^* et q^* (figure 1.15a). Si le point est au-dessus (resp. en dessous) de cette région, alors le segment est en dessous (resp. au-dessus) de la droite. Cette partition de l'espace dual correspond à une partition de l'ensemble des droites en fonction de leur position relative par rapport au segment.

Considérons maintenant une scène composée de polygones. Nous appliquons le découpage de l'espace dual précédent pour chaque côté de polygone. Nous obtenons alors un découpage de l'espace dual par les courbes duales des sommets des polygones : l'*arrangement dual* de la scène (figure 1.15b).

Cet arrangement dual est composé de faces, d'arêtes (portions de courbes comprises entre deux sommets) et de sommets (intersection de deux courbes). Les faces représentent les ensembles de droites intersectant une même liste d'objets (dans l'ordre de la liste).

Plus précisément, à chaque droite d nous associons une *étiquette*, constituée des objets intersectés dans l'ordre par la droite. Si la droite coupe le segment ouvert O_i, alors nous mettons O_i dans l'étiquette. Si la droite passe par un sommet p, alors nous mettons $p^{\pm\pm}$ dans l'étiquette : le premier signe de $p^{\pm\pm}$ est $+$ (resp. $-$) si le côté (p^-, p) du polygone est au-dessus (resp. en dessous) de la droite ; le deuxième signe de $p^{\pm\pm}$ est $+$ (resp. $-$) si le côté (p, p^+) du polygone est au-dessus (resp. en dessous) de la droite. La relation

$d \sim d'$ si et seulement si d et d' ont la même étiquette et on peut déplacer continûment d vers d' en gardant l'étiquette constante,

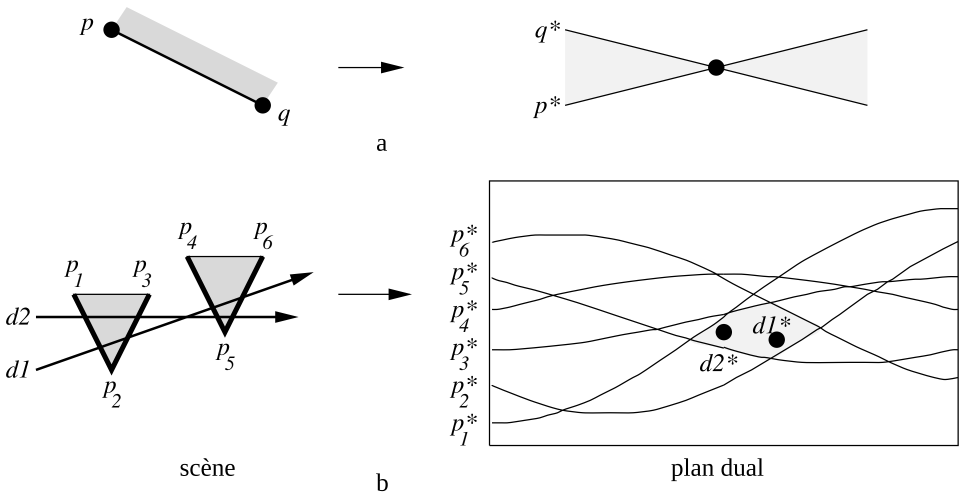


FIG. 1.15 – *Classification des droites grâce à la dualité.* **a.** *Zone d'un côté de polygone.* **b.** *Arrangement dual de la scène : droites correspondant à une face.*

est une relation d'équivalence. Les éléments de l'arrangement dual représentent les classes d'équivalence de cette relation.

Les deux relations de dualité évoquées jusqu'à maintenant sont très liées. Nous avons vu un certain nombre de propriétés communes, et elles sont liées de plus par la correspondance $a = \tan\theta$ et $b = u/\cos\theta$. Cependant la dualité (a, b) paraît plus limitée : elle ne permet de ne considérer que les droites orientées de direction comprise dans l'intervalle $]-\pi/2, \pi/2[$. Mais ces limitations ne sont pas importantes ici, car nous ne nous servons que des informations topologiques de l'arrangement dual, et les tests numériques sont tous retranposés sur les points et les droites de la scène.

De plus, la partie de l'arrangement dual de courbes (dualité (θ, u)) compris dans l'intervalle $]\theta_0 - \pi/2, \theta_0 + \pi/2[$ d'une scène et l'arrangement dual de droites (dualité (a, b)) de la scène à laquelle une rotation d'angle θ_0 a été appliquée sont topologiquement identiques (figure 1.16, où nous avons colorié des faces pour mieux faire apparaître les correspondances). Enfin, pour avoir toutes les informations de visibilité, il suffit de se restreindre à un intervalle de directions de longueur π, car les droites de direction θ et $\theta + \pi$ sont géométriquement identiques. Cela se traduit par des relations de symétrie dans l'arrangement dual de courbes (dualité (θ, u)). Pour visualiser un intervalle de directions de longueur 2π avec les arrangements de droites, il suffit de « coller » à l'arrangement son symétrique par rapport à l'axe des abscisses.

Nous utilisons ici dans nos représentations de l'espace dual la formulation (a, b), car elle est plus pratique pour nos illustrations. Cependant nous ne nous servons que des informations topologiques, et non pas géométriques, des droites duales. Nous n'excluons donc aucune droite (verticale) dans l'espace dual.

La formulation (θ, u) est plus pratique à utiliser dans le cadre de scènes composées d'objets courbes, où les courbes duales correspondent aux droites tangentes aux objets. Pocchiola [Poc90] utilise cette dualité pour construire l'arrangement dual d'un scène d'objets convexes courbes.

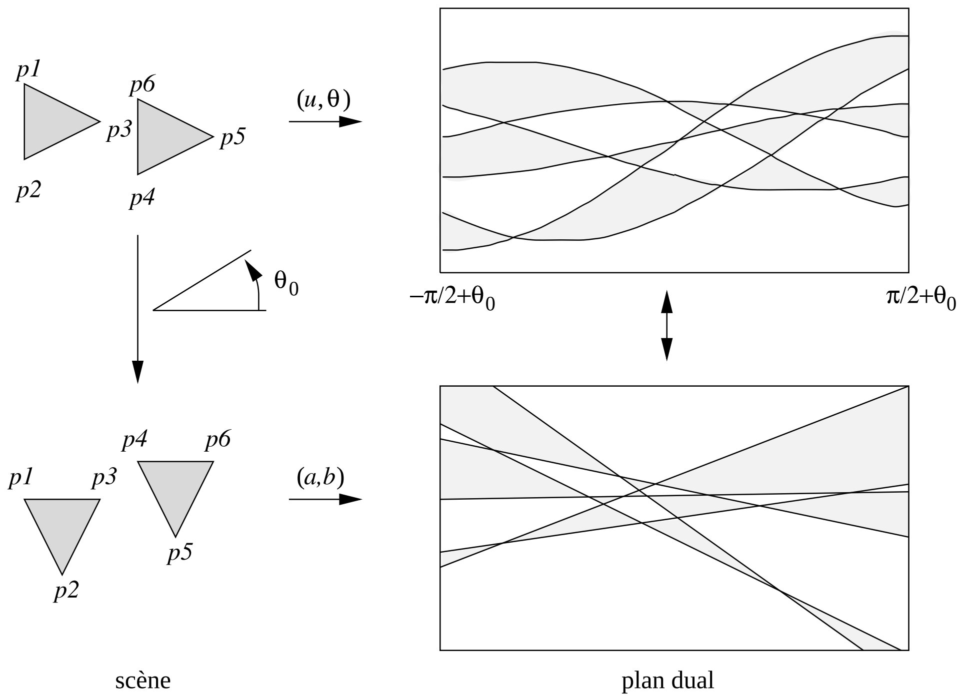


FIG. 1.16 – *Correspondance entre les deux relations de dualité.*

Pour savoir quels sont les objets traversés par une droite, il suffit de localiser l'élément de l'arrangement qui contient le point dual. Il existe deux tactiques pour construire cet arrangement de droites : construction incrémentale et balayage.

La construction incrémentale consiste à introduire les droites une par une. Supposons que nous avons déjà construit l'arrangement de $n-1$ droites. Nous introduisons la n^e droite, en cherchant les faces de l'arrangement traversées par cette droite, faces qui seront coupées en deux. Pour cela, il faut d'abord chercher la face semi-infinie gauche où commence cette droite. Ensuite, pour trouver le point de sortie de la face courante traversée, il suffit de parcourir les arêtes bordant la face – dans le sens trigonométrique direct par exemple – depuis l'arête d'entrée (figure 1.17).

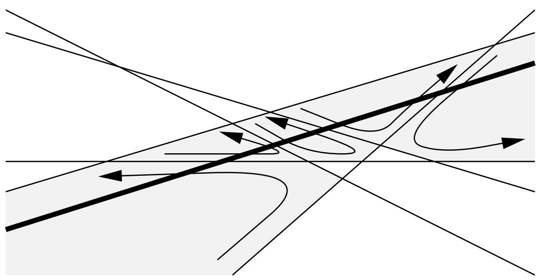


FIG. 1.17 – *Construction incrémentale d'un arrangement : faces traversées par la nouvelle droite insérée.*

Le théorème de la zone ([EOS86],[CGL85]) indique que le nombre total d'arêtes bordant les faces traversées par la droite est linéaire en fonction du nombre de droites de l'arrangement. La construction incrémentale d'un arrangement de n droites s'effectue donc en temps optimal $O(n^2)$.

Le balayage de l'arrangement consiste à parcourir les éléments de l'arrangement comme s'ils étaient balayés de la gauche vers la droite par une ligne. Le balayage est modélisé par la liste $(e_{i_k})_{1 \leq k \leq n}$ des arêtes « coupées » par la ligne de balayage, où il y a pour chaque sommet de la scène exactement une arête portée par la courbe duale de ce point (figure 1.18a).

Le processus de balayage est en fait discret : le balayage progresse en passant à chaque fois un sommet incident à deux arêtes consécutives e_{i_k} et $e_{i_{k+1}}$ de la coupe. La coupe est alors mise à jour en remplaçant ces deux arêtes par les arêtes suivantes e'_{i_k} et $e'_{i_{k+1}}$ respectivement sur les droites support de i_{k+1} et i_k (figure1.18b).

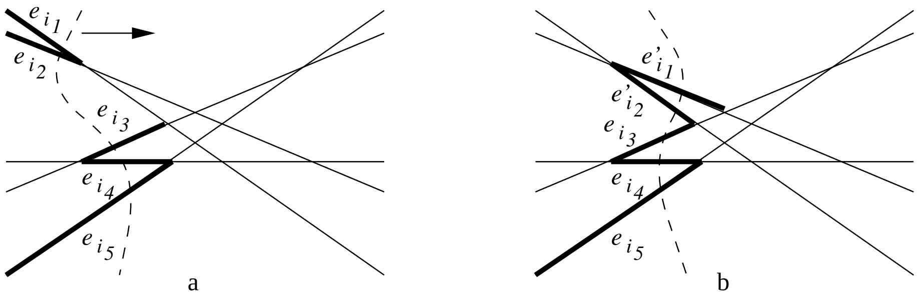


FIG. 1.18 – *Balayage d'un arrangement de droites.* **a.** *Coupe de balayage.* **b.** *Progression du balayage.*

La façon la plus directe d'effectuer ce balayage est de parcourir les sommets de l'arrangement dans l'ordre selon les abscisses croissantes. Cela revient à prendre une ligne de balayage verticale « droite ». Le balayage nécessite alors l'utilisation d'une queue de priorité et s'effectue en temps $O(n^2 \log n)$. Cependant ce type de balayage est trop strict et un balayage *topologique* plus souple suffit. Le balayage topologique parcourt les sommets de façon cohérente : il ne passe un sommet $v = d_i \cap d_j$ qu'après avoir déjà visité les sommets précédents sur d_i et d_j.

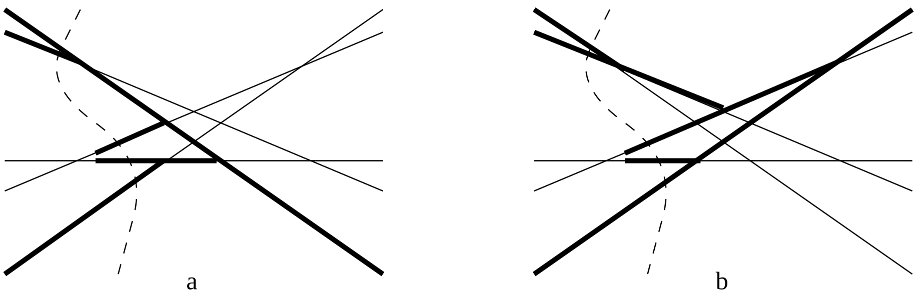


FIG. 1.19 – *Arbres d'horizon.* **a.** *Arbre supérieur.* **b.** *Arbre inférieur.*

Pour effectuer ce balayage, Edelsbrunner et Guibas [EG86] ont créé une structure de données auxiliaire : les *arbres d'horizon*. Étant donné une coupe de balayage, l'arbre d'horizon supérieur (resp. inférieur) est construit en étendant chaque arête de la coupe vers la droite jusqu'à ce qu'elle rencontre une droite de l'arrangement de pente plus petite (figure 1.19) (remarque : Edelsbrunner et Guibas [EG86] utilisent la dualité $d : y = ax + b \to d^*(a, b)$, qui inverse certaines relations par rapport à la dualité que nous utilisons. Nous avons retransposé ici la description de l'algorithme avec la dualité que nous utilisons). Alors le balayage peut progresser en passant un sommet v si et seulement si v est présent dans les deux arbres d'horizon : v est alors intersection de deux arêtes consécutives de la coupe.

Pour mettre à jour l'arbre d'horizon supérieur après le passage d'un sommet $v = e_{i_k} \cap e_{i_{k+1}}$, il faut trouver la nouvelle extrémité droite du segment porté par la droite i_{k+1}. Pour cela, il suffit de parcourir les arêtes de la baie formée par les arêtes successives de l'arbre d'horizon et commençant par l'arête $e_{i_{k-1}}$ de la coupe située juste au-dessus (figure 1.20).

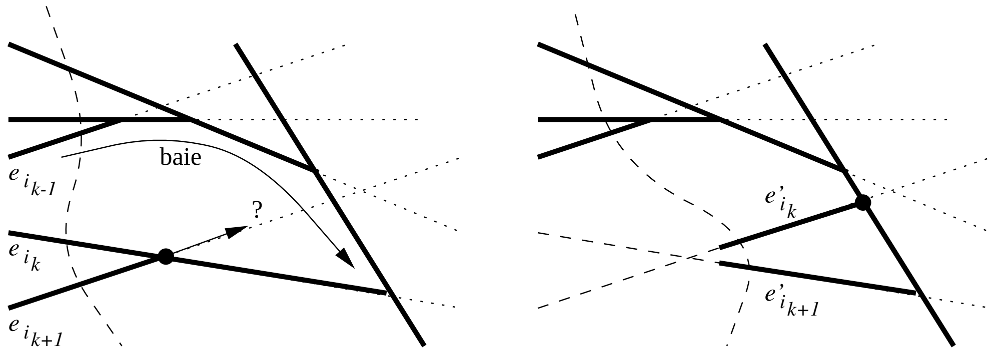


FIG. 1.20 – *Mise à jour de l'arbre d'horizon lors du passage d'un sommet.*

La mise à jour à chaque passage coûte *a priori* un temps $O(n)$, car une baie peut comporter $O(n)$ arêtes, soit pour les $O(n^2)$ passages un temps total $O(n^3)$. Cependant, grâce à un schéma de comptage global, les auteurs montrent dans [EG89] qu'au total seulement $O(n^2)$ arêtes sont balayées dans les baies. Le balayage topologique s'effectue alors en temps optimal $O(n^2)$.

Nous avons présenté cet arrangement dual comme permettant de résoudre des problèmes liés à la visibilité. Il permet en effet de calculer le graphe de visibilité de la scène, comme nous allons le voir ci-après, et de faire des calculs de vues, comme nous le verrons dans le chapitre 3.

Cependant cet arrangement de droites permet de résoudre d'autres problèmes géométriques, comme calculer le triangle d'aire minimale formé par 3 points parmi n points donnés, calculer une droite coupant le plus possible de segments parmi n segments donnés, tester la dégénérescence de $d+1$ points dans $\mathbb{R}^d$, etc. (voir par exemple [EG89] et [CGL85] pour d'autres applications).

1.4.3 Balayage de graphe de visibilité : arbres d'horizon et arbres de rotation

Les sommets de l'arrangement dual correspondent aux segments reliant deux sommets de la scène. En particulier, parmi ces sommets il y a ceux qui correspondent aux arêtes du graphe de visibilité. Puisque le balayage parcourt les sommets de l'arrangement un par un, il faut en profiter pour déterminer si le sommet visité correspond à une arête de visibilité ou non. Grâce à la cohérence du balayage, cette vérification peut se faire en temps constant.

En effet, la cohérence du balayage implique que lorsque le sommet v de l'arrangement correspondant à la droite (p, q) est visité, parmi les droites issues de q et passant par un autre sommet de la scène, toutes celles de pente inférieure à (p, q) ont déjà été parcourues et toutes celles de pente supérieure n'ont pas encore été parcourues (figure 1.21a).

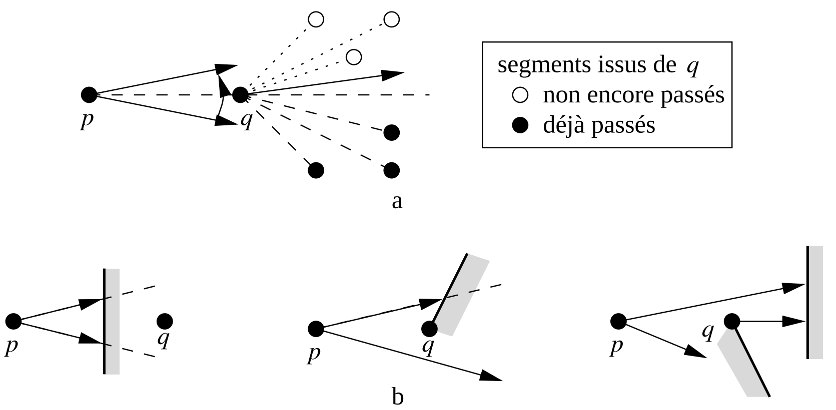


FIG. 1.21 – *Balayage de l'arrangement.* **a.** *Cohérence du balayage.* **b.** *Mise à jour de l'objet vu lors du passage d'un segment* (p, q).

À chaque sommet p de la scène est alors associé l'objet vu depuis p par tout rayon appartenant dans l'espace dual à l'arête e de la coupe associée à p. Lors du passage du sommet v correspondant au segment (p, q), la mise à jour de l'objet vu depuis p connaissant celui vu depuis q se fait alors en temps constant (la figure 1.21(b) montre les situations à considérer) et vérifier si (p, q) est une arête du graphe de visibilité est immédiat.

Overmars et Welzl [OW88] ont remarqué qu'un seul arbre d'horizon pouvait suffire pour effectuer le balayage : il suffit alors de toujours passer le sommet en bas à gauche de l'arbre d'horizon supérieur, sommet qui peut être mis à jour en temps constant à chaque passage. Ils ont alors transposé les arbres d'horizon de l'espace dual dans la scène, où ils deviennent des *arbres de rotation* : comme déplacer un point sur une droite d dans le plan dual revient à faire tourner dans la scène une droite passant par le point d^*, balayer l'arrangement de droites dual revient à faire un balayage *rotationnel* dans la scène, c'est-à-dire à faire pivoter des droites autour des sommets de la scène.

Overmars et Welzl innovent ensuite pour tenir compte directement des relations de visibilité : au lieu d'étendre une demi-droite, ils étendent un demi-rayon libre maximal, qui s'arrête donc sur le premier objet rencontré.

L'arbre de rotation est alors défini pour une direction θ : à chaque sommet p de la scène, est associé le premier sommet vu depuis p en tournant dans le sens trigonométrique direct depuis la direction θ (figure 1.22). Un point fictif d'abscisse plus grande que celles des points de la scène et d'ordonnée infinie est ajouté à la scène en tant que sommet susceptible d'être vu.

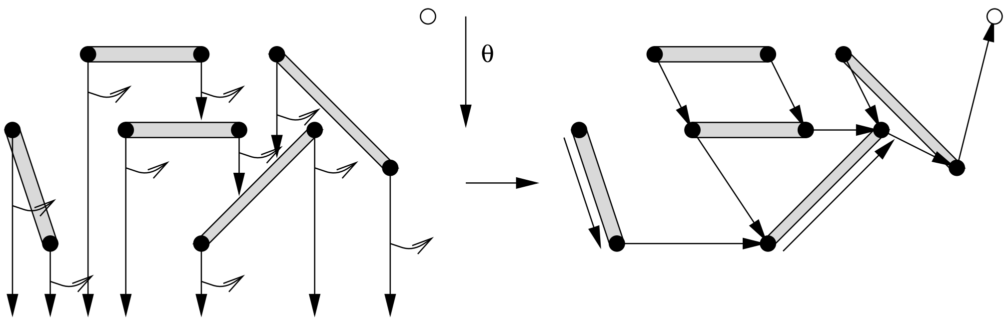


FIG. 1.22 – *Arbre de rotation d'une scène de segments.*

Ce graphe est un sous-graphe de taille n du graphe de visibilité. Le maintien de cet arbre quand θ varie de $-\pi/2$ à $\pi/2$ permet de trouver toutes les arêtes du graphe de visibilité. Le maintien se fait, comme pour les arbres d'horizon, de façon discrète : l'arête de plus petite pente est passée à chaque fois, ce qui assure en plus la cohérence du balayage.

L'étude des propriétés géométriques du graphe induites par le balayage droit conduit les auteurs à considérer des chaînes d'arêtes, suites maximales d'arêtes consécutives telles que les segments de la scène incidents aux sommets d'une chaînes soient au-dessus de la chaîne.

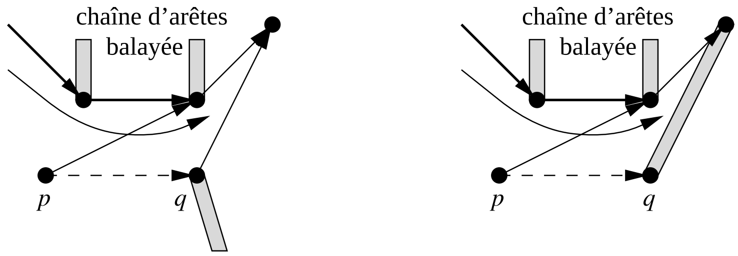


FIG. 1.23 – *Mise à jour de l'arbre de rotation lors du passage d'une arête.*

Lors du passage de l'arête (p, q), la nouvelle extrémité r de l'arête issue de p est le point de tangence des droites passant par p et d'une chaîne d'arêtes déterminée (figure 1.23). Ce point est trouvé en balayant les arêtes de la chaîne.

L'utilisation de la queue de priorité coûte un temps $O(m \log n)$. Le coût du balayage des arêtes lors des mises à jour pourrait être dans le pire cas $O(mn)$. Les auteurs annoncent (sans preuve détaillée) que le coût total n'est que de $O(\alpha(n)m)$. En fait le chapitre suivant montre implicitement que cette complexité de balayage n'est au total qu'en $O(m)$.

1.4.4 Complexe de visibilité

L'arrangement dual permet d'avoir une structure de données discrète qui classifie les droites selon les objets intersectés. Cependant, comme les droites traversent les obstacles, manipuler ces ensembles de droites a l'inconvénient de prendre en compte des événements entre objets non visibles ; les complexités obtenues ne dépendent alors que de n, même si le résultat obtenu est très petit. Pour ne tenir compte que des événements entre objets visibles et obtenir des algorithmes sensibles à la sortie, Pocchiola et Vegter ne manipulent plus des droites, mais des *segments libres maximaux*. Les segments libres maximaux (que nous appellerons aussi rayons) sont des segments de droite de longueur maximale dans l'espace libre de la scène. Ils peuvent être obtenus à partir d'un point p et une direction u : à partir de p un segment est étendu dans la direction u (vers l'avant et vers l'arrière) jusqu'à ce que ses extrémités rencontrent l'intérieur d'un objet. L'étiquette d'un rayon est définie de façon similaire à celle des droites, et est en fait constituée des objets de départ et d'arrivée du rayon et des points situés entre ces objets et par lesquels passe le rayon. Deux rayons sont équivalents si on peut passer continûment de l'un à l'autre en gardant l'étiquette constante. Le complexe de visibilité représente alors les classes d'équivalences des segments libres maximaux selon cette relation d'équivalence.

Une première version du complexe de visibilité a été donnée par Vegter [Veg90] pour des scènes de segments, sous le nom de *diagramme de visibilité*. Le complexe est ensuite pleinement étudié par Pocchiola et Vegter [PV96b] dans le cadre de scènes d'objets courbes convexes. Nous présentons ici (de manière moins formelle et moins combinatoire) le complexe de visibilité dans le cadre de scènes polygonales.

Le complexe de visibilité comporte trois sortes d'éléments : des faces, des arêtes et des sommets.

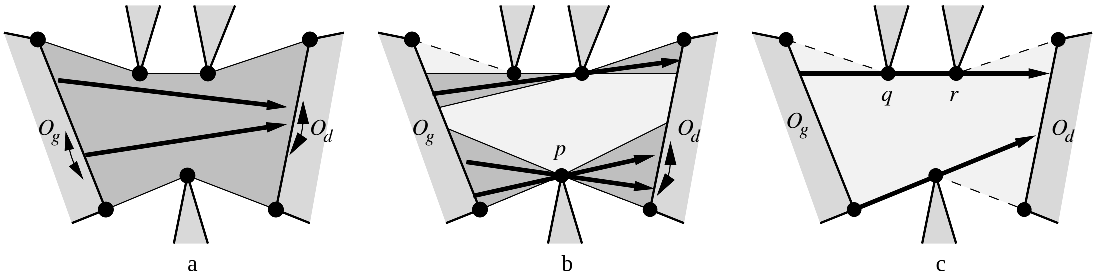


FIG. 1.24 – *Éléments du complexe de visibilité.* **a.** *Face.* **b.** *Arêtes.* **c.** *Sommets.*

Les faces sont des éléments de dimension deux : elles représentent les rayons d'étiquette (O_g, O_d), c'est-à-dire les rayons qui partent de O_g et arrivent à O_d sans toucher d'autre objet. Un tel rayon a donc deux degrés de liberté (figure 1.24a).

Les arêtes sont des éléments de dimension un : elle représentent les rayons d'étiquette $(O_g, p^{\pm\pm}, O_d)$, c'est-à-dire les rayons allant de O_g à O_d en passant par un sommet p. Ces rayons n'ont plus qu'un seul degré de liberté : ils doivent passer par p (figure 1.24b).

Les sommets sont des éléments de dimension zéro : ils représentent les rayons d'étiquette $(O_g, q^{\pm\pm}, r^{\pm\pm}, O_d)$, c'est-à-dire le rayon allant de O_g à O_d en passant par q et r. Ces rayons sont fixés : ils doivent passer par deux sommets (figure 1.24c).

De la même façon qu'il était plus facile de représenter les classes de droites dans l'espace dual grâce à l'arrangement de droites dual, nous représentons aussi les éléments du complexe de visibilité en utilisant cet espace dual. En effet, un segment libre maximal est inclus dans une droite support; nous représentons alors un tel segment par le point dual de sa droite support. L'espace dual nous permet de mieux visualiser les relations entre les divers éléments du complexe (figure 1.25).

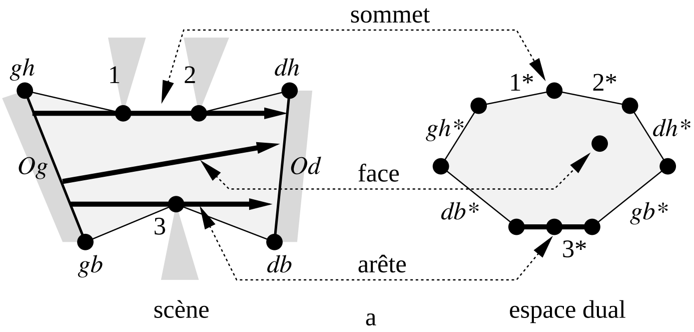


FIG. 1.25 – *Éléments du complexe de visibilité : visualisation dans l'espace dual.*

Un sommet représente un rayon passant par deux sommets p et q. Dans l'espace dual c'est donc un point, intersection des deux courbes duales p^* et q^* associées aux deux sommets. Une arête représente des rayons de visibilité constante passant par un sommet p, qui sera dit sommet associé à l'arête. Dans l'espace dual, une arête est donc une portion de la courbe duale p^* associée au sommet, délimitée par deux sommets du complexe. Une arête est incidente à deux sommets, appelés respectivement sommet gauche et sommet droit (figure 1.26a). Un sommet est incident à quatre arêtes, appelées arêtes gauche/droite-haute/basse (figure 1.26b).

Les arêtes sont classées en six catégories, en fonction de la géométrie locale du polygone autour du sommet p associé (figure 1.27) : arête λ-convexe, λ-concave, μ-convexe, μ-concave, ν-gauche et ν-droit. Selon sa catégorie, une arête peut être incidente à une, deux ou trois faces. D'une manière générale, nous désignons quatre faces incidentes à une arête par face gauche/droite-haute/basse, même si certaines de ces faces sont nulles ou confondues.

Considérons maintenant une face du complexe, associée à (O_g, O_d). Nous voyons qu'il existe un seul rayon de pente minimale (resp. maximale) dans cette face. Ces deux rayons correspondent à deux sommets extrêmes bordant la face, dits sommets gauche et droit de la face. Si nous prenons maintenant un rayon appartenant à cette face, alors nous pouvons le translater vers le bas (resp. vers le haut) en gardant la même visibilité, jusqu'à ce que nous rencontrions un sommet qui provoque, si nous continuons le déplacement, un changement de visibilité. Les deux rayons extrêmes appartiennent chacun à une arête bordant la face. En fait, la face est bordée par deux chaînes d'arêtes, une chaîne supérieure et une chaîne inférieure, toutes deux délimitées par les deux sommets extrêmes de la face. La face délimite dans la scène une région géométrique,

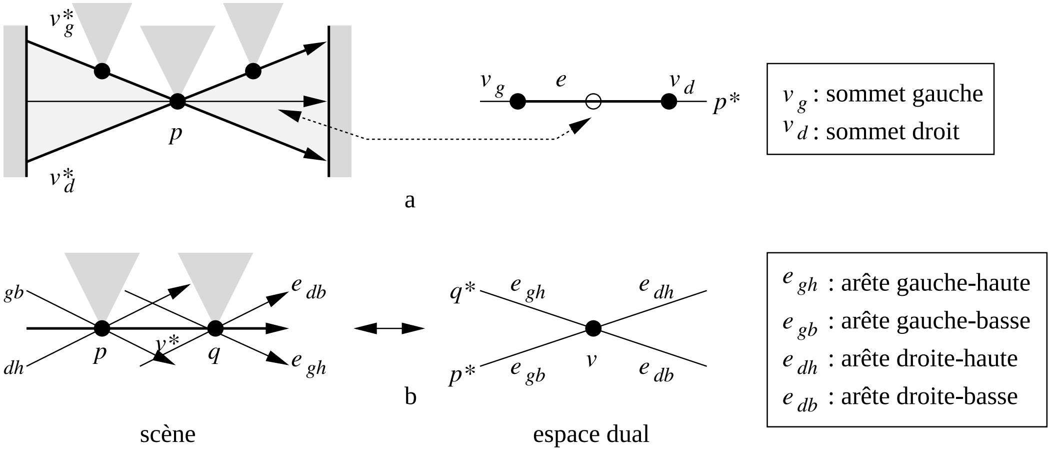


Fig. 1.26 – *Relations d'incidences entre* **a.** *arête et sommets,* **b.** *sommet et arêtes, dans le complexe de visibilité.*

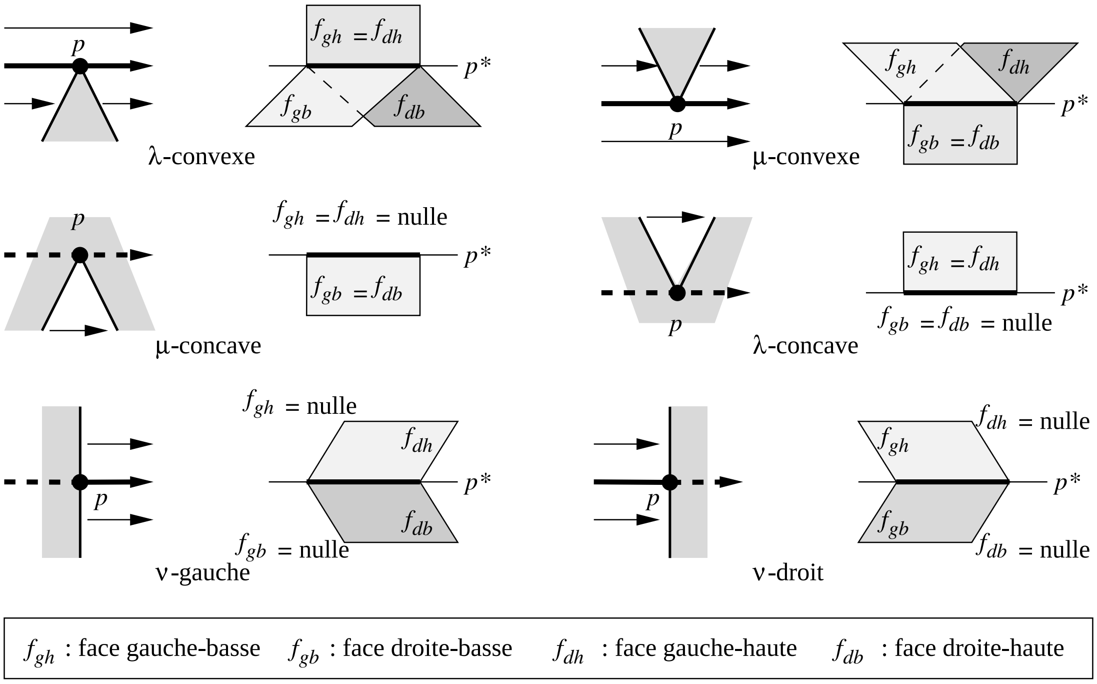


Fig. 1.27 – *Les différents types d'arêtes du complexe.*

limitée par les segments de droite associés aux sommets des chaînes d'arêtes (sommets extrêmes exclus). La figure 1.28 montre les éléments du complexe qui délimitent une face.

Si nous traçons la face dans l'espace dual en utilisant la dualité (a, b), nous voyons que c'est un polygone convexe. Cette caractéristique est due à une propriété plus générale des faces : si nous prenons dans l'espace dual deux points dans une face, alors la courbe duale passant par ces deux points est, entre ces deux points, entière-

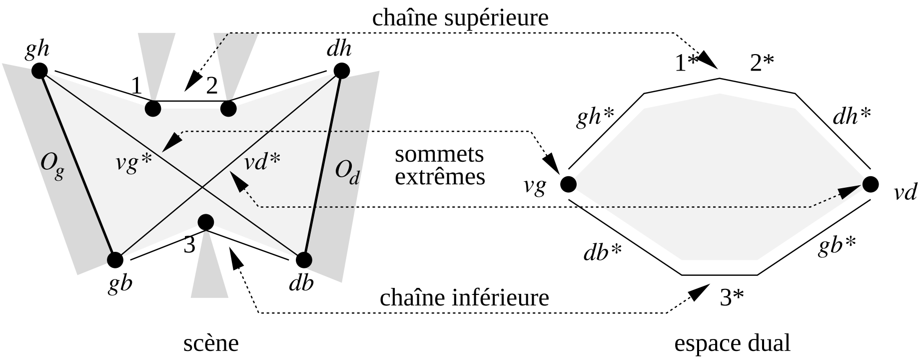


FIG. 1.28 – *Éléments délimitant une face du complexe.*

ment contenue dans la face. Cela signifie dans la scène que si nous prenons deux rayons r_1 et r_2 dans la face et notons $p = r_1 \cap r_2$ le point d'intersection des deux droites support des rayons, alors si nous déplaçons r_1 vers r_2 en faisant pivoter sa droite support autour de p, alors il ne quitte pas la face. En effet, les deux frontières de la face (les chaînes d'arêtes) sont convexes par visibilité. Cela signifie notamment que dans la scène, chaque sommet du bord haut de la face est visible des sommets du bord bas de la face et réciproquement.

Les arêtes peuvent encore être classées en deux catégories. En effet, une arête bordant une face (O_g, O_d) peut soit avoir aussi la visibilité (O_g, O_d), soit passer par l'une des extrémités de O_g ou O_d. Dans ce dernier cas, la face peut être bordée par une série d'arêtes consécutives portées par la même courbe duale (figure 1.29).

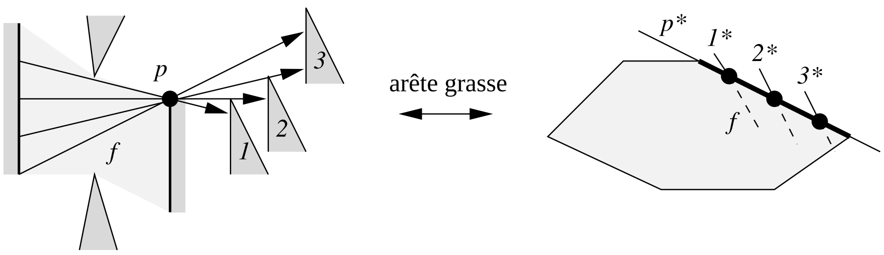


FIG. 1.29 – *Arêtes grasses.*

De telles arêtes sont dites *grasses* (terminologie empruntée à Pocchiola et Vegter [PV96b]), car une portion de courbe duale est décomposée en sous-arêtes bordant une même face. Chacune des deux chaînes d'arêtes d'une face peut avoir deux séries d'arêtes grasses : une en début de chaîne (arête grasse gauche) et une en fin de chaîne (arête grasse droite).

Contrairement à l'arrangement de courbes dual, le complexe de visibilité n'a pas une structure plane. En effet, des segments libres maximaux différents peuvent être contenus dans une même droite support. Ils sont représentés alors dans le plan dual par un même point dual. Nous plongeons ce plan dans $\mathbb{R}^3$, où ces points duaux sont « décollés » (figure 1.30).

La figure 1.31 montre la structure non plane du complexe au voisinage de l'un de

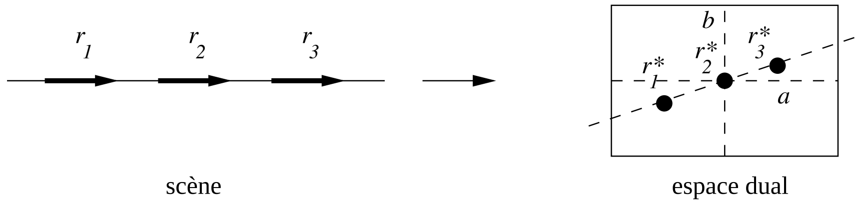


FIG. 1.30 – *Utilisation d'une dimension supplémentaire pour représenter les rayons confondus en projection dans le plan dual.*

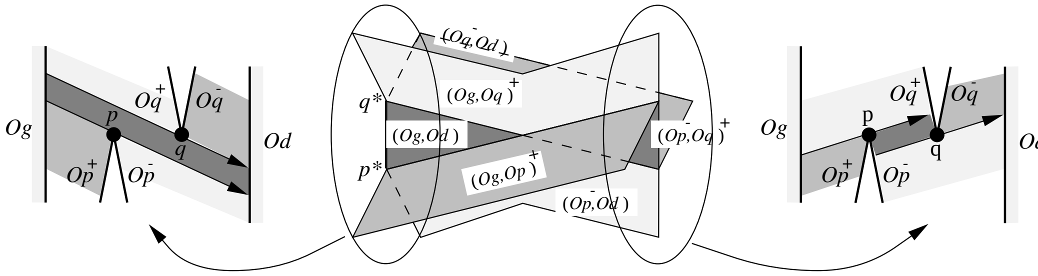


FIG. 1.31 – *Non planarité du complexe au voisinage d'un sommet.*

ses sommets et montre les deux situations dans la scène correspondant à la coupe du complexe avant (resp. après) le sommet. En particulier, deux courbes duales qui se coupent en projection dans le plan dual ne se coupent pas forcément en un sommet dans le complexe, comme c'était le cas dans l'arrangement dual : elles se coupent seulement si les deux points associés sont mutuellement visibles.

La configuration du complexe autour d'un sommet (nombre de faces incidentes, relations d'incidence) correspondant à l'arête de visibilité $[p, q]$ dépend alors de la configuration géométrique de la scène autour des points p et q. La figure 1.32 montre les 17 configurations géométriques possibles autour de ces sommets.

1.4.5 Relations avec le diagramme de visibilité

Vegter [Veg90] a introduit le diagramme de visibilité pour des scènes composées de segments de droite. Ce diagramme de visibilité est bien le complexe de visibilité de la scène, mais avec une représentation des données légèrement différente. Vegter met l'accent sur les rayons issus d'un même segment s.

Il définit d'abord pour chaque segment s une subdivision $Vis(s, S)$ (S désigne la scène) qui représente la classification en classe d'équivalences des rayons issus de s. $Vis(s, S)$ est donc l'ensemble des faces du complexe de visibilité qui ont dans leur étiquette pour objet gauche le segment s. Ces faces sont donc comprises dans l'espace dual entre les deux courbes duales p_1^* et p_2^* des deux extrémités p_1 et p_2 du segment s (figure 1.33a). Parmi les sommets de $Vis(s, S)$, ceux situés sur p_1^* (resp. p_2^*) ne sont incidents qu'à trois arêtes. Les arêtes ne sont incidentes qu'à la face (resp. qu'aux deux faces) qu'elles bordent dans $Vis(s, S)$. Le diagramme de visibilité est alors constitué

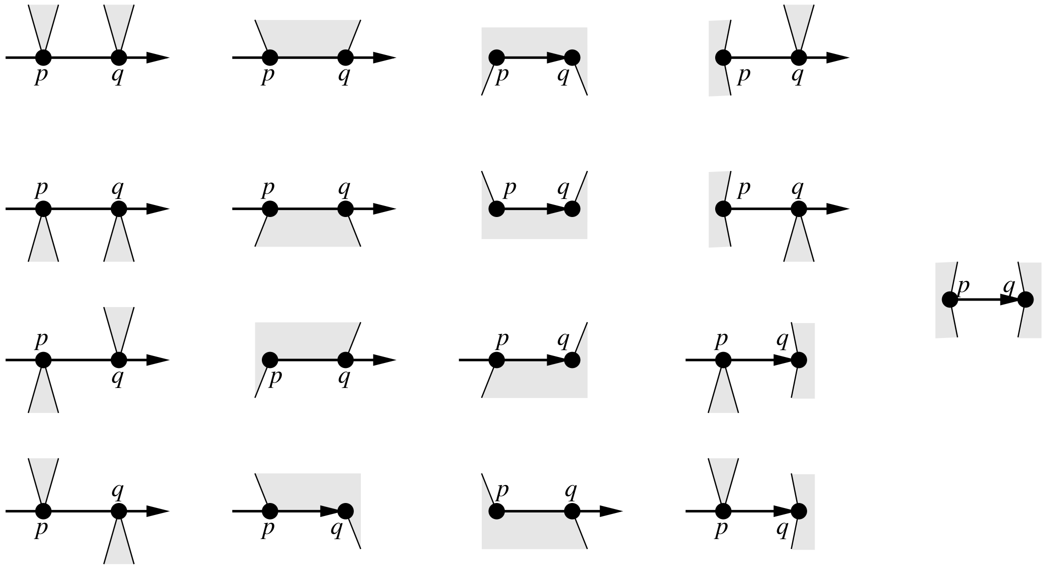


FIG. 1.32 – *Différentes configuration géométriques correspondant à un sommet du complexe.*

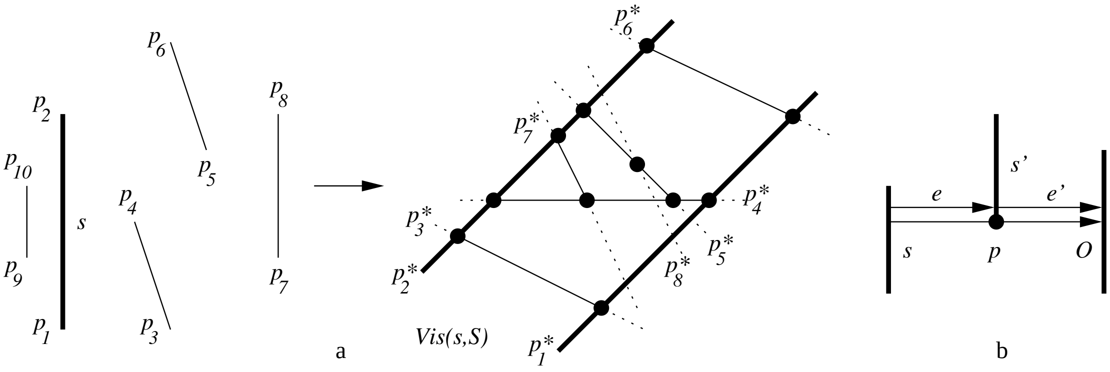


FIG. 1.33 – *Diagramme de visibilité.* **a.** *Décomposition $Vis(s, S)$ du diagramme de visibilité.* **b.** *Arêtes partenaires e et e'.*

de l'ensemble de ces subdivisions $Vis(s, S)$. Ces subdivisions sont reliées entre elles grâce aux arêtes partenaires. Un rayon issu du segment s et passant par le sommet p d'un segment s' avant d'être arrêté par un segment O peut-être associé soit à l'arête e bordant les faces $f(s, s')$ et $f(s, O)$ de $Vis(s, S)$, soit à l'arête e' bordant la face $f(s', O)$ de $Vis(s', S)$ (figure 1.33b). Les deux arêtes e et e' sont alors dites partenaires et ont chacune un pointeur sur l'autre.

Les calculs sont plutôt considérés dans une subdivision $Vis(s, S)$, avec occasionnellement un passage d'une subdivision à une autre par l'intermédiaire d'arêtes partenaires. Le diagramme de visibilité et le complexe de visibilité sont équivalents. La représentation du complexe que nous utilisons, adaptée de celle de Pocchiola et Vegter [PV96b], est un peu plus uniforme.

Nous montrons dans le chapitre suivant comment calculer de façon optimale le complexe de visibilité d'une scène polygonale.

Chapitre 2

Balayage de graphe de visibilité

Dans ce chapitre nous considérons le graphe et le complexe de visibilité de scènes polygonales. Nous exposons d'abord un algorithme optimal de calcul de graphe de visibilité et étudions ensuite ses performances expérimentales. Nous montrons ensuite comment utiliser cet algorithme, qui effectue en fait un balayage du complexe de visibilité, pour construire le complexe.

> — *Dis Astérix, nous ne sommes tout de même pas venus ici pour balayer leur pays !*
>
> *Astérix et les Goths.*

2.1 Calcul optimal de graphe de visibilité

Nous étudions dans cette section un algorithme optimal de calcul de graphe de visibilité. Nous formalisons d'abord le problème de balayage à l'aide de la notion de *filtre*. Nous introduisons ensuite *les arbres d'horizon*, outil qui permet d'effectuer ce balayage. Nous étudions enfin la complexité théorique de l'algorithme.

2.1.1 Balayage topologique

Comme nous l'avons indiqué dans le chapitre précédent, nous considérons des scènes composées de polygones simples, ayant deux à deux une intersection nulle. Nous considérons aussi que les polygones sont des objets composites et que les objets élémentaires considérés dans les relations de visibilité (appelés simplement « objets ») sont en fait les segments de droites qui forment les côtés des polygones. Pour rester cohérent dans les relations de visibilité, nous considérons qu'un rayon partant à l'infini voit en fait l'objet O_∞, qui peut être considéré comme le « ciel bleu » qui entoure la scène.

Nous notons $(\mathcal{V}, \mathcal{E})$ Le graphe de visibilité de la scène, n son nombre de sommets ($n = |\mathcal{V}|$) et m son nombre d'arêtes ($m = |\mathcal{E}|$). Nous étendons l'ensemble des arêtes géométriques du graphe de visibilité à l'ensemble des arêtes orientées u du graphe de visibilité : $u = ((p,q),\theta)$ est l'arête orientée de p à q de pente θ où $\theta \equiv \text{angle}(\overrightarrow{p,q})(2\pi)$. Cet ensemble – maintenant infini – est aussi noté $\mathcal{E}$ (le contexte permettra de distinguer l'ensemble des arêtes géométriques de l'ensemble des arêtes orientées).

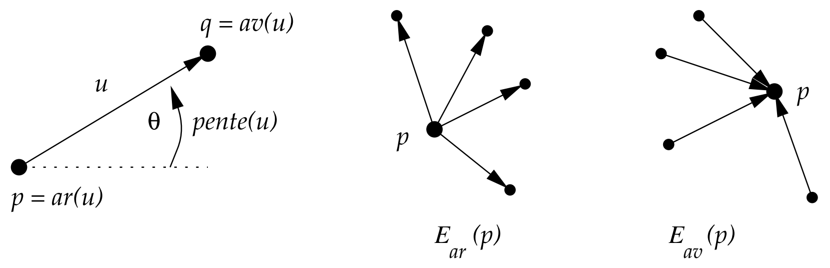


FIG. 2.1 – *Notations pour les arêtes.*

Pour une arête $u = ((p,q),\theta)$, nous notons (figure 2.1) :

- $ar(u) = p$, le sommet arrière de u ;
- $av(u) = q$, le sommet avant de u ;
- $pente(u) = \theta$, la pente de u.

Pour un point p de la scène nous notons aussi :

- $\mathcal{E}_{ar}(p) = \{u \in \mathcal{E} / ar(u) = p\}$, l'ensemble des arêtes partant de p ;
- $\mathcal{E}_{av}(p) = \{u \in \mathcal{E} / av(u) = p\}$, l'ensemble des arêtes arrivant en p ;
- $\mathcal{E}(p) = \mathcal{E}_{ar}(p) \cup \mathcal{E}_{av}(p)$, l'ensemble des arêtes ayant p pour extrémité.

Nous avons vu dans le chapitre précédent qu'Edelsbrunner et Guibas [EG89] et Overmars et Welzl [OW88] définissaient le balayage rotationnel du graphe de visibilité comme devant, pour chaque sommet du graphe, reporter les arêtes incidentes dans l'ordre de leur pente.

Nous nous sommes inspiré de la formalisation de Pocchiola et Vegter [PV96a] pour présenter ici la notion de balayage topologique. Nous introduisons d'abord un ordre sur l'ensemble $\mathcal{E}$ des arêtes du graphe.

Définition 2.1 *Soit un sommet p de la scène. La relation $\prec_p$ définie par*

$$\forall (u,v) \in \mathcal{E}(p)^2,\ u \prec_p v \iff \mathit{pente}(u) < \mathit{pente}(v),$$

est un ordre total sur $\mathcal{E}(p)$.

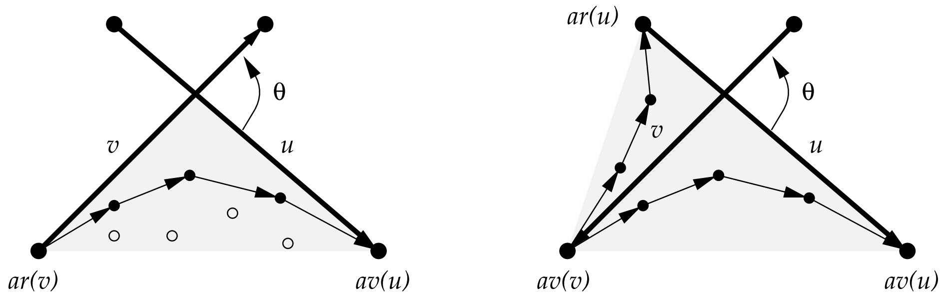
La relation $\prec$ définie par

$$\forall (u,v) \in \mathcal{E},\ u \prec v \iff \exists\, (u_i)_{0 \le i \le k},\ u_0 = u,\ u_k = v,\ u_{i-1} \text{ et } u_i \text{ ont une extrémité commune } p \text{ et } u_{i-1} \prec_p u_i,$$

est un ordre partiel sur $\mathcal{E}$.

Une arête u est donc inférieure à une arête v si on peut passer de u à v par une suite d'arêtes incidentes et de pente croissante. Il est facile de vérifier que cette relation est bien une relation d'ordre partiel sur $\mathcal{E}$, compatible avec l'ordre total induit par l'ordre sur les pentes des arêtes. Nous étendons cet ordre aux ensembles d'arêtes : pour deux sous-ensembles U et V d'arêtes de $\mathcal{E}$, $U \prec V$ signifie que pour toute paire d'arêtes $(u,v) \in U \times V$, soit $u \prec v$, soit u et v ne sont pas comparables.
Deux arêtes incidentes sont donc comparables, mais aussi

Propriété 2.2 *Deux arêtes sécantes sont comparables suivant l'ordre partiel $\prec$.*



a. *pente(v)=pente(u) +* θ   b. *pente(v)=pente(u) +* θ+ π

FIG. 2.2 – *Comparaison de deux arêtes sécantes.*

Démonstration. Soient deux arêtes sécantes u et v. Supposons que $\mathit{pente}(v)$ soit supérieure à $\mathit{pente}(u)$ mais inférieure à $\mathit{pente}(u) + \pi$ (figure 2.2a). Considérons les arêtes joignant les sommets de l'enveloppe convexe des points contenus dans le triangle formé par $av(u)$, $ar(v)$ et le point d'intersection des deux arêtes. Alors on peut passer de u à v en suivant ces arêtes, qui constituent une suite d'arêtes de pentes croissantes.

Si $\mathit{pente}(v)$ est supérieure à $\mathit{pente}(u) + \pi$ (figure 2.2b), alors nous passons de u à $v-\pi$ comme précédemment, puis de $v-\pi$ à $u+\pi$ par l'enveloppe convexe du triangle $(av(v), v \cap u, ar(u))$, puis de $u+\pi$ à v par la première enveloppe convexe. □

Faire un balayage topologique du graphe de visibilité revient à faire un tri topologique des arêtes du graphe selon l'ordre $\prec$ qui vient d'être défini. Pour exprimer ce tri nous utilisons la notion de *filtre* :

> *un filtre R d'un ensemble partiellement ordonné E est un sous-ensemble de E tel que $\forall x \in R$, $x \prec y \Rightarrow y \in R$.*

Nous définissons alors le balayage topologique en terme de filtre :

Définition 2.3 *Un balayage topologique du graphe de visibilité $(\mathcal{V}, \mathcal{E})$ est une suite de filtres $(R_i)_{0 \leq i \leq k}$ de $\mathcal{E}$ tels que $\forall\ 0 \leq i < k$, $R_{i+1} = R_i \backslash \{u\}$ où $u \in \min\limits_{\prec} R_i$.*

Le filtre R_i représente l'ensemble des arêtes qui restent à balayer alors que les arêtes de $\mathcal{E} \setminus R_i$ ont déjà été vues. Nous pouvons initialiser le balayage en prenant par exemple pour R_0 l'ensemble des arêtes de pente supérieure à un angle θ_0.
Pour faire progresser le balayage nous devons trouver une arête minimale u parmi les arêtes restantes du filtre, et nous disons alors que nous *passons* l'arête u.

Nous pouvons alors calculer le graphe de visibilité de la scène en balayant l'ensemble de ses arêtes géométriques, c'est-à-dire en balayant ses arêtes orientées de pente comprise entre θ_0 et $\theta_0 + \pi$.

2.1.2 Arbres d'horizon

Dans cette partie nous montrons comment calculer $\min_\prec R_i$, l'ensemble des arêtes minimales de R_i. À cet effet nous introduisons deux nouveaux graphes : l'*arbre d'horizon* inférieur (resp. supérieur). L'étude de leurs propriétés permet de donner un algorithme pour calculer $\min_\prec R_i$.

Étant donné que pour tout point p la relation $\prec_p$ est un ordre total sur $\mathcal{E}(p)$, nous pouvons désigner l'élément suivant et l'élément précédent selon cet ordre. Pour une arête u de $\mathcal{E}(p)$, nous notons alors

- $succ^s_p(u) = \min\limits_{\prec_p}\{v \in \mathcal{E}_{ar}(p) / u \prec_p v\}$: arête suivante de u dans $\mathcal{E}_{ar}(p)$;
- $pred^s_p(u) = \max\limits_{\prec_p}\{v \in \mathcal{E}_{ar}(p) / v \prec_p u\}$: arête précédente de u dans $\mathcal{E}_{ar}(p)$;
- $succ^i_p(u) = \min\limits_{\prec_p}\{v \in \mathcal{E}_{av}(p) / u \prec_p v\}$: arête suivante de u dans $\mathcal{E}_{av}(p)$;
- $pred^i_p(u) = \max\limits_{\prec_p}\{v \in \mathcal{E}_{av}(p) / v \prec_p u\}$: arête précédente de u dans $\mathcal{E}_{av}(p)$.

Une arête peut être le successeur de plusieurs arêtes ; en effet soit u une arête de $\mathcal{E}_{ar}(p)$, alors pour toute arête v de $\mathcal{E}(p)$ telle que $u \prec_p v \prec_p succ^s_p(u)$, on a $succ^s_p(v) = succ^s_p(u)$. En revanche, si l'on se restreint à $\mathcal{E}_{ar}(p)$, l'ensemble des arêtes comprises entre une arête u et son successeur $succ^s_p(u)$ est vide.

Étant donné un filtre R, les éléments de $\min_\prec R$ se trouvent parmi les successeurs des éléments de $\mathcal{E} \backslash R$. Dans cette optique nous définissons deux nouveaux graphes, les *arbres d'horizon* :

Définition 2.4 *Soit un filtre R de $\mathcal{E}$ et soit $B = \mathcal{E}\backslash R$. Le graphe $\mathcal{H}^s(R) = (\mathcal{V}^s, \mathcal{E}^s)$ défini par*

- $\mathcal{V}^s = succ^s(B) \setminus B = \{u \in R \;/\; \exists\, v \in B, u = succ^s_{ar(v)}\}$,
- $\mathcal{E}^s = \{(u,v) \in \mathcal{V}^{s2} / av(u) = ar(v) \text{ et } pente(u) < pente(v) + \pi\}$,

est un arbre de racine $\max_{\prec} \mathcal{V}^s$, appelé arbre d'horizon supérieur. Le parent de u dans cet arbre est noté $\mathcal{E}^s(u)$.

Similairement, l'arbre d'horizon inférieur est l'arbre défini par

- $\mathcal{V}^i = succ^i(B) \setminus B = \{u \in R \;/\; \exists\, v \in B, u = succ^i_{av(v)}\}$,
- $\mathcal{E}^i = \{(u,v) \in \mathcal{V}^{i2} / ar(u) = av(v) \text{ et } pente(u) < pente(v) + \pi\}$,

et le parent de u dans cet arbre est noté $\mathcal{E}^i(u)$.

Démonstration. S'il existe un cycle d'arêtes consécutives de $\mathcal{V}^s$, alors il existe deux arêtes consécutives u et v de ce cycle telles que $pente(u) \geq pente(v) + \pi$, et donc ce cycle n'est pas un cycle dans le graphe $\mathcal{H}^s$. □

La figure 2.3 montre les arbres d'horizon d'une scène où R est l'ensemble des arêtes du graphe de visibilité de pente supérieure à $-\pi/2$.

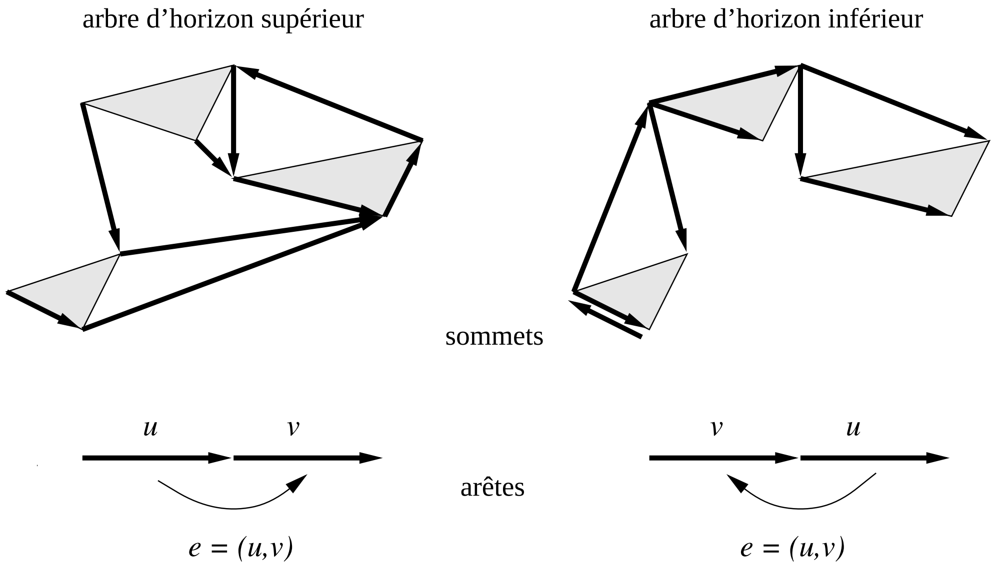


FIG. 2.3 – *Arbres d'horizon.*

Dans la suite de ce chapitre, une « arête de l'arbre d'horizon » désignera en fait – sauf mention contraire explicite – une arête du graphe de visibilité appartenant à l'arbre d'horizon, c'est-à-dire une arête de $\mathcal{E}$ appartenant à $\mathcal{V}^s$ (resp. $\mathcal{V}^i$).

Deux arêtes arrivant à un même point ont le même parent et sont dites sœurs. Elles sont classées par pente croissante, ce qui permet de désigner pour une arête sa sœur gauche (de pente plus petite) et sa sœur droite (de pente plus grande).

Les deux arbres d'horizon sont liés par une relation de symétrie : si nous inversons le sens des arêtes de l'arbre d'horizon inférieur associé à R, alors nous obtenons l'arbre d'horizon supérieur associé à $R' = R+\pi$, l'ensemble des arêtes de R auxquelles on a appliqué une rotation d'angle π. Dans la suite, nous ne démontrerons certaines propriétés que pour les arbres d'horizon supérieurs : les démonstrations sont à une symétrie près les mêmes pour les arbres d'horizon inférieurs.

Les arbres d'horizon permettent de caractériser les arêtes déjà balayées :

Propriété 2.5 *Soit une arête $u = (p, q)$ du graphe de visibilité, alors $\forall v \in \mathcal{E}_{ar}(p)$*

- *si $u \in R$, alors $u \prec_p v \Rightarrow v \notin \mathcal{V}^s$;*
- *si $u \in \mathcal{V}^s$, alors $v \prec_p u \iff v \notin R$.*

Démonstration. Si l'arête $u \in R$ est inférieure à l'arête $v \in \mathcal{E}_{ar}(p)$, alors $succ^s(u) \preceq v$, ce qui empêche l'existence d'une arête $w \notin R$ (et donc inférieure à u) telle que $v = succ^s(w)$. et donc v n'appartient pas à l'arbre d'horizon.

Soit $u \in \mathcal{V}^s$. D'après le point précédent on a $v \in R \Rightarrow u \preceq_p v$, soit $v \prec_p u \Rightarrow v \notin R$. La réciproque est une conséquence directe de la propriété des filtres. □

Ces propriétés combinatoires des arêtes des arbres d'horizon induisent dans la scène certaines propriétés géométriques :

Propriété 2.6 *Toutes les arêtes d'un arbre d'horizon ont deux à deux une intersection nulle ou dégénérée (extrémité commune ou support commun).*

Démonstration. Soient deux arêtes u et v du graphe de visibilité ayant une intersection non nulle et non dégénérée. Reprenons la première situation de la démonstration de la propriété 2.2 (figure 2.2a). Soit w l'arête l'enveloppe convexe issue de $ar(v)$. Si u appartient à l'arbre d'horizon, alors comme w est supérieure à u, w appartient au filtre R et donc d'après la propriété 2.5, v n'appartient pas à l'arbre d'horizon. La démonstration est similaire pour la deuxième situation. □

Certaines propriétés sont plus liées aux relations de visibilité entre objets :

Propriété 2.7 *Soient deux arêtes u et v consécutives appartenant à l'arbre d'horizon supérieur ($(u, v) \in \mathcal{V}^{s2}$, $av(u) = ar(v)$), alors*

- *$v \prec u \Rightarrow (ar(u), av(v)) \notin \mathcal{E}$;* *(figure 2.4a)*
- *$u \prec v \Rightarrow pred^s_{av(u)}(v) \prec_{av(u)} u$;*
- *Si $u \prec v$ et $av(u) \neq ar(u)^+$, alors les rayons issus de $av(u)$ et de directions respectives u et v sont stoppés par un même segment. De plus le triangle délimité par ces deux rayons et ce segment est vide de tout sommet de la scène.(figure 2.4b)*

Démonstration. Supposons que $v \prec u$ et notons $w = (ar(u), av(v))$. Comme w est supérieure à v, w appartient à R (propriété des filtres). Si w est une arête du graphe de visibilité alors, comme $w \prec_{ar(u)} u$, d'après la propriété 2.5 (2^e point) w n'appartient pas à R ce qui est absurde, w n'est donc pas une arête du graphe de visibilité.

Supposons que $u \prec v$ et notons $w = pred^s_{av(u)}(v)$. Comme $v \in \mathcal{V}^s$, d'après la

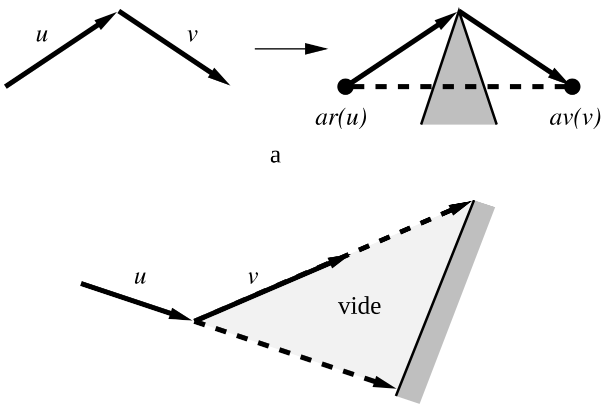


FIG. 2.4 – *Propriétés de visibilité des arbres d'horizon.*

propriété 2.5, w n'appartient pas à R. Si w était supérieure à u, alors w appartiendrait à R (propriété des filtres), ce qui est contradictoire, donc w est inférieure à u.

Enfin le dernier point est une conséquence directe du deuxième point. □

Nous remarquons que nous pouvons associer à chaque sommet de la scène l'unique arête de l'arbre d'horizon supérieur (resp. inférieur) partant de (resp. arrivant à) ce sommet :

Proposition 2.8 *Soit un filtre R et soit $B = \mathcal{E}\backslash R$. L'application e^s (resp. e^i) définie par les deux définitions équivalentes*

- $e^s(p) = \min_{\prec_p}(R \cap \mathcal{E}_{ar}(p))$ *(resp.* $e^i(p) = \min_{\prec_p}(R \cap \mathcal{E}_{av}(p))$*)*
- $e^s(p) = succ^s_p(\max_{\prec_p}(B \cap \mathcal{E}(p)))$ *(resp.* $e^i(p) = succ^i_p(\max_{\prec_p}(B \cap \mathcal{E}(p)))$*)*

est une bijection de $\mathcal{V}$ sur $\mathcal{V}^s$ (resp. $\mathcal{V}^i$).

Démonstration. Comme $e^s(p) \in \mathcal{E}_{ar}(p)$, e^s est injectif.

Soit $u \in \mathcal{V}^s$, et $p = ar(u)$. La propriété 2.5 permet d'affirmer que $\forall w \in \mathcal{E}_{ar}(p)$, $w \in R \Rightarrow u \preceq_p w$, ce qui signifie que $u = \min_{\prec_p}(R \cap \mathcal{E}_{ar}(p))$; u a donc $ar(u)$ pour antécédent par e^s : e^s est surjectif. □

Grâce à cette bijection nous connaissons la taille des arbres d'horizon : $|\mathcal{V}^s| = n$ et $|\mathcal{E}^s| = n$.

Nous pouvons aussi réexprimer la relation de parenté : le parent de u dans $\mathcal{H}^s$ est $\mathcal{E}^s(u) = e^s(av(u))$, et le parent de u dans $\mathcal{H}^i$ est $\mathcal{E}^i(u) = e^i(ar(u))$.

Nous énonçons maintenant le théorème qui montre comment les arbres d'horizon permettent de déterminer l'ensemble des arêtes pouvant être passées pour faire progresser le balayage :

Théorème 2.9 *Soit un filtre R, alors*

$$\min_{\prec} R = \{u \in \mathcal{V}^s \cap \mathcal{V}^i / u \prec \mathcal{E}^s(u) \text{ et } u \prec \mathcal{E}^i(u)\}.$$

Démonstration. Soit une arête $u = (p, q)$ appartenant à l'ensemble qui vient d'être défini. Comme u appartient à $\mathcal{V}^s$, alors $u = \min_{\prec_p}(R \cap \mathcal{E}_{ar}(p))$. D'autre part, $u \prec \mathcal{E}^i(u)$ se réécrit $u \prec e^i(p) \prec_p \min_{\prec_p}(R \cap \mathcal{E}_{av}(p))$, ce qui entraîne $u = \min_{\prec_p}(R \cap \mathcal{E}(p))$. Similairement nous montrons que $u = \min_{\prec_q}(R \cap \mathcal{E}(q))$, et donc finalement u appartient à $\min_\prec R$.

L'inclusion inverse est évidente. □

Nous pouvons effectuer le calcul de $\min_\prec R$ directement en étudiant chaque arête d'un des deux arbres d'horizon. Le temps de calcul est alors linéaire ($O(n)$). Cependant si nous utilisons cette méthode lors du balayage des arêtes du graphe de visibilité, le calcul complet de $\min_\prec R$ à chaque passage coûterait un temps total $O(mn)$, ce qui est trop coûteux pour calculer le graphe de visibilité. Pour abaisser ce coût, nous ne calculons directement $\min_\prec R$ que pour initialiser le balayage et nous utilisons ensuite le théorème seulement pour mettre à jour $\min_\prec R$ lors de chaque passage.

2.1.3 Mise à jour de $\min\limits_\prec R$ lors d'un passage

Nous supposons que nous passons du filtre R_t au filtre R_{t+1} en passant l'arête $u = (p, q)$ du graphe de visibilité. Nous supposons aussi que les arbres d'horizon ont été mis à jour et nous notons u^s (resp. u^i) l'arête qui remplace u dans l'arbre d'horizon supérieur (resp. inférieur) : $\mathcal{V}^s_{t+1} = (\mathcal{V}^s_t \setminus \{u\}) \cup \{u^s\}$ (resp. $\mathcal{V}^i_{t+1} = (\mathcal{V}^i_t \setminus \{u\}) \cup \{u^i\}$). Pour mettre à jour $\min_\prec R$, nous devons chercher les nouvelles arêtes à rajouter à $\min_\prec R_t \setminus \{u\}$.

Nous devons d'abord considérer les arêtes des nouveaux arbres d'horizon qui n'étaient pas présentes dans les anciens arbres, c'est-à-dire les nouvelles arêtes u^s et u^i. D'après l'égalité $\min_\prec R = \{u \in \mathcal{V}^s \cap \mathcal{V}^i / u \prec \mathcal{E}^s(u)$ et $u \prec \mathcal{E}^i(u)\}$, nous devons aussi considérer les arêtes des arbres d'horizon dont le père a changé, c'est-à-dire les arêtes qui avaient u pour père. Parmi celles-ci il suffit de ne considérer que les deux arêtes de pente minimale $w^s = \min_{\prec_p} {\mathcal{E}^s_{t+1}}^{-1}(p)$ et $w^i = \min_{\prec_q} {\mathcal{E}^i_{t+1}}^{-1}(q)$.

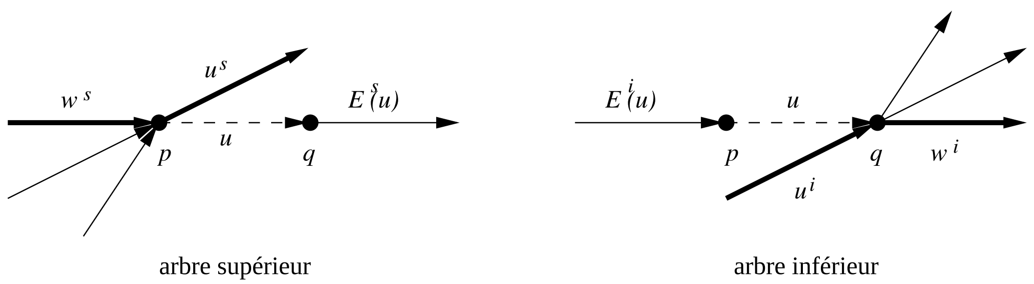


FIG. 2.5 – *Correspondance entre les mises à jour dans les arbres d'horizon.*

Si nous dessinons les deux arbres d'horizon au voisinage de ces arêtes (figure 2.5), nous voyons qu'il pourrait exister des correspondances entre les arêtes des deux arbres d'horizon, notamment entre w^s et $\mathcal{E}^i_t(u)$ (resp. entre w^i et $\mathcal{E}^s_t(u)$). En effet, $\mathcal{E}^i(u)$ est inchangée et donc $\mathcal{E}^i_t(u) = \min_{\prec_p}\{v \in R_{t+1} \cap \mathcal{E}_{av}(p)\}$; d'autre part, l'arête de pente minimale w^s vérifie $w^s = \min_{\prec_p}\{v \in \mathcal{V}^s_{t+1} \cap \mathcal{E}_{av}(p)\}$, ce qui entraîne $\mathcal{E}^i_t(u) \preceq w^s$. Si

$\mathcal{E}_t^i(u)$ appartient à $\mathcal{V}_{t+1}^s$, d'après la propriété 2.5 nous avons $w^s \preceq \mathcal{E}_t^i(u)$ et alors w^s et $\mathcal{E}_t^i(u)$ désignent la même arête. Le même raisonnement s'applique à w^i et $\mathcal{E}_t^s(u)$.

Au lieu de calculer w^s (resp. w^i), nous considérons directement $\mathcal{E}_t^i(u)$ (resp. $\mathcal{E}_t^s(u)$), et l'algorithme de mise à jour s'écrit :

$$
\begin{array}{l}
\min\limits_{\prec} R = \min\limits_{\prec} R \setminus \{u\} \\
\textbf{si } \mathcal{E}_t^i(u) \prec u^s \\
\quad \textbf{si } \mathcal{E}_t^i(u) \in \mathcal{V}_{t+1}^s \textbf{ et } \mathcal{E}_t^i(u) \prec \mathcal{E}_{t+1}^i(\mathcal{E}_t^i(u)) \\
\quad\quad \min\limits_{\prec} R = \min\limits_{\prec} R \cup \{\mathcal{E}_t^i(u)\} \\
\textbf{sinon} \\
\quad \textbf{si } u^s \in \mathcal{V}_{t+1}^i \textbf{ et } u^s \prec \mathcal{E}_{t+1}^s(u^s) \\
\quad\quad \min\limits_{\prec} R = \min\limits_{\prec} R \cup \{u^s\} \\
\textbf{si } \mathcal{E}_t^s(u) \prec u^i \\
\quad \textbf{si } \mathcal{E}_t^s(u) \in \mathcal{V}_{t+1}^i \textbf{ et } \mathcal{E}_t^s(u) \prec \mathcal{E}_{t+1}^s(\mathcal{E}_t^s(u)) \\
\quad\quad \min\limits_{\prec} R = \min\limits_{\prec} R \cup \{\mathcal{E}_t^s(u)\} \\
\textbf{sinon} \\
\quad \textbf{si } u^i \in \mathcal{V}_{t+1}^s \textbf{ et } u^i \prec \mathcal{E}_{t+1}^i(u^i) \\
\quad\quad \min\limits_{\prec} R = \min\limits_{\prec} R \cup \{u^i\}
\end{array}
$$

Nous devons montrer maintenant comment calculer u^s et u^i, c'est-à-dire comment mettre à jour les arbres d'horizon.

2.1.4 Mise à jour des arbres d'horizon

Nous étudions la mise à jour de l'arbre d'horizon supérieur lors du passage de l'arête $u = (p, q)$ (la mise à jour de l'arbre inférieur est similaire à une symétrie près).

Cette mise à jour s'effectue en considérant successivement les quatre cas décrits ci-après. Les deux premiers cas se traitent directement, les deux autres cas demandent de considérer une nouvelle structure : les *chaînes d'arêtes*.

Premier cas : $av(u) = ar(u)^+$

L'arête rentre dans le polygone, et l'arête suivante est l'arête $u^s = (ar(u), ar(u)^-)$ sortant du polygone (figure 2.6a).

Deuxième cas : $av(\mathcal{E}^s(u)) = ar(u)$

Nous en déduisons que le demi-plan supérieur délimité par la droite passant par u est vide. u est alors une arête de l'enveloppe convexe de la scène, et la prochaine arête u^s est l'arête de l'enveloppe convexe de la scène partant de $ar(u)$ (figure 2.6b).

Troisième cas : $av(u)^-$ est au-dessus de u

Considérons l'enveloppe convexe inférieure des points contenus dans le triangle $(ar(u), av(u), av(u)^-)$ ($av(u)$ exclus). La prochaine arête est l'arête de cette enveloppe

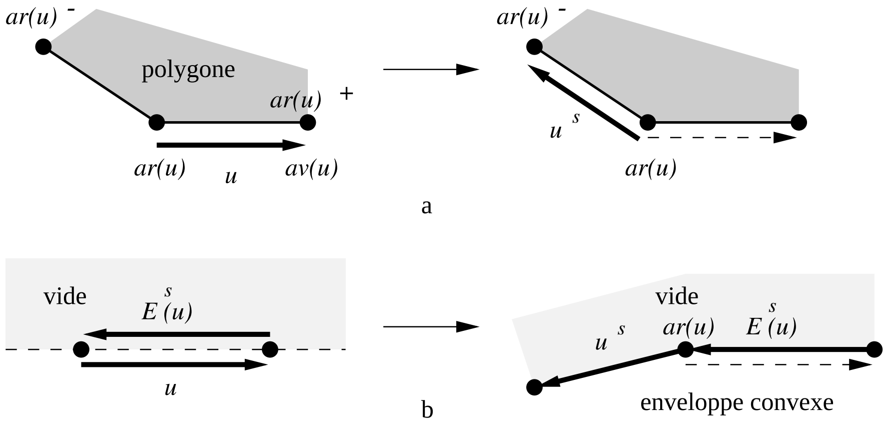


FIG. 2.6 – *Cas directs de mise à jour des arbres d'horizon.* **a.** *Rentrée dans un polygone.* **b.** *Sortie hors de la scène.*

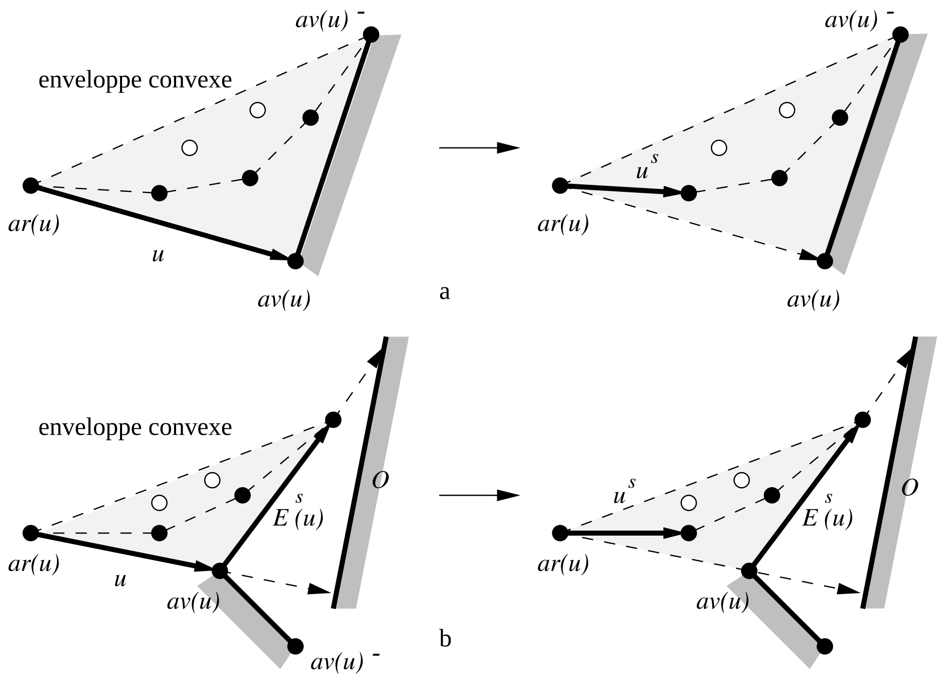


FIG. 2.7 – *Mise à jour des arbres d'horizon.* **a.** *Rayon passant par u stoppé par le segment incident à $av(u)$.* **b.** *Rayon passant par u stoppé après $av(u)$.*

partant de $ar(u)$ (figure 2.7a).

Quatrième cas : $av(u)^-$ est en-dessous de u

Soit O le segment de la scène vu depuis $av(u)$ dans la direction u. Le triangle, délimité par O et les deux rayons issus de $av(u)$, de directions respectives u et $\mathcal{E}^s(u)$

et stoppés par O, est vide (propriété 2.5).

Considérons l'enveloppe convexe inférieure des points contenus dans le triangle $(ar(u), av(u), av(\mathcal{E}^s(u)))$ ($av(u)$ exclus). La prochaine arête est l'arête de cette enveloppe partant de $ar(u)$ (figure 2.7b).

Nous avons une situation similaire dans les deux derniers cas : nous devons considérer une enveloppe convexe de points. Mais cette enveloppe convexe est en fait déjà calculée :

Proposition 2.10 *Toutes les arêtes de l'enveloppe convexe considérée dans le troisième ou le quatrième cas – exceptée celle partant de p – appartiennent à l'arbre d'horizon.*

Démonstration. Considérons un point r de l'enveloppe convexe (différent des deux points extrêmes), et notons $cv(r)$ l'arête de l'enveloppe convexe partant de r. Comme $cv(r)$ est supérieure à u, elle appartient à R et donc l'arête $e^s(r)$ de l'arbre d'horizon issue de r vérifie $e^s(r) \preceq cv(r)$.
Si $e^s(r)$ avait une pente inférieure à celle de $cv(r)$, d'après la géométrie de la situation, la pente de $e^s(r)$ serait inférieure à celle de u, ce qui est impossible ($u \in \min_{\prec} R$) ; les deux arêtes $e^s(r)$ et $cv(r)$ sont donc les mêmes. □

L'enveloppe convexe considérée fait partie de plus grands sous-ensembles d'arêtes de l'arbre d'horizon :

Définition 2.11 *On appelle chaîne d'arêtes d'un arbre d'horizon $\mathcal{H}^s$, toute suite maximale d'arêtes $(u_i)_{0\leq i\leq k}$ telle que*

- $\forall\, 0\leq i<k \quad u_i \in \mathcal{V}^s$ *et* $av(u_i) = ar(u_{i+1})$;
- $\forall\, 0\leq i<k \quad av(u_i)^-$ *est au-dessus de* u_i ;
- $\forall\, 0<i\leq k \quad ar(u_i)^+$ *est au-dessus de* u_i ;
- $\forall\, 0\leq i<k \quad u_i$ *est l'arête de plus petite pente de* $\mathcal{V}^s$ *arrivant en* $av(u_i)$.

De cette définition nous en déduisons les propriétés géométriques suivantes :

Propriété 2.12 *Soit une chaîne d'arêtes $(u_i)_{0\leq i\leq k}$, alors*

- $u_i \prec u_{i+1}$ *: une chaîne d'arêtes est convexe ;*
- *les rayons issus de $av(u_i)$ et de direction u_i sont stoppés par un même segment de droite O de la scène, excepté peut-être celui issu de la dernière arête u_k où alors $av(u_k)$ est une extrémité de O ;*
- *l'espace (ouvert) délimité par la demi-droite issue de u_0, par la chaîne et par O est vide de sommets.*

Démonstration. Il suffit d'appliquer la propriété 2.7 à chaque paire d'arêtes consécutives de la scène. □

Nous utilisons ces chaînes d'arêtes pour mettre à jour l'arbre d'horizon :

Proposition 2.13 *Après le passage de l'arête $u = (p, q)$, la nouvelle arête u^s partant de p de l'arbre d'horizon est l'arête issue de p et tangente à la chaîne d'arêtes se terminant par*

– *l'arête de plus petite pente située au-dessus de* (q^-, q) *si* q^- *est au-dessus de* (p, q) *;*
– *l'arête sœur gauche de* $e^s(q)$ *sinon.*

Démonstration. Toute arête de $\mathcal{E}$ arrivant en q^- dans le premier cas, ou en $av(e^s(q))$ dans le deuxième cas, et située en dessous de l'enveloppe convexe coupe forcément u, et donc ne peut appartenir à l'arbre d'horizon. L'enveloppe convexe est donc bien la partie droite de la chaîne d'arête considérée. □

Nous balayons les sommets r de la chaîne d'arêtes à partir de u_0 jusqu'à ce que nous trouvions le point de tangence avec cette chaîne (on a alors $e^s(r) \succ (p, r)$) ou que nous soyons arrivé en fin de chaîne (figure 2.8).

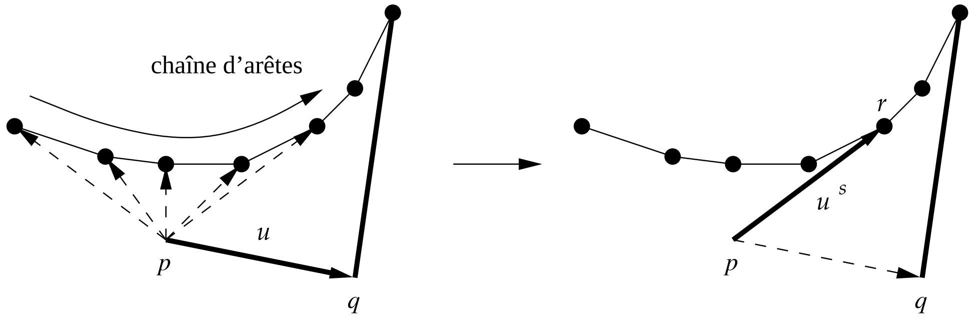


FIG. 2.8 – *Balayage d'une chaîne d'arêtes.*

Une fois le point r de tangence trouvé, la mise à jour des chaînes est directe : la chaîne est coupée en deux au niveau de r, ce qui donne une chaîne gauche et une chaîne droite, et l'arête u^s est ajoutée au début de la chaîne droite.

2.1.5 Complexité du balayage du graphe de visibilité

Nous étudions ici la complexité de notre algorithme pour balayer le graphe de visibilité.

Représentation des données

Chaque sommet p de la scène a un pointeur vers l'unique arête de l'arbre d'horizon qui en part et un pointeur sur l'arête de plus petite pente qui y arrive. Pour traiter le troisième cas de passage, chaque sommet a aussi un pointeur sur l'arête arrivante de plus petite pente supérieure à (p, p^+).

Chaque arête de l'arbre d'horizon a des pointeurs sur ses extrémités, un pointeur vers sa sœur droite et sa sœur gauche. Son père est en fait l'arête partant de son sommet droit. Les chaînes d'arêtes ne font pas l'objet d'une structure de données à part : seulement un double pointeur entre la première et la dernière arête de la chaîne.

Après un tri des sommets des polygones, la construction des arbres d'horizon et de $\min_{\prec} R_0$ se fait en temps linéaire $O(n)$.

Lors du passage d'une arête u, une fois la nouvelle extrémité de u^s trouvée, la mise à jour des différentes structures de données (arbres et chaînes d'arêtes) se fait en temps constant. Distinguer les quatre cas et trouver le cas échéant la chaîne d'arêtes à balayer se fait aussi en temps constant.

Il nous reste à calculer le coût des balayages des chaînes d'arêtes.

Balayage des sommets dans les chaînes d'arêtes

A priori, dans le cas le pire n sommets peuvent être balayés dans une chaîne, soit un coût $O(mn)$ pour le calcul du graphe de visibilité. Cependant, nous montrons dans le reste de cette partie que le nombre total des sommets balayés dans les chaînes est seulement en $O(m)$.

Nous montrons d'abord que les sommets balayés peuvent être reliés aux arêtes du graphe de visibilité :

Proposition 2.14 *Soit r un sommet d'une chaîne d'arête balayé lors de la mise à jour de l'arête u de l'arbre d'horizon. Alors l'arête $(ar(u), r)$ est une arête du graphe de visibilité.*

Démonstration. L'espace compris entre la demi-droite issue de u_0 (première arête de la chaîne), la chaîne et le segment vu O est vide. Cependant ce n'est pas suffisant pour montrer la proposition, car l'arête u est située strictement en-dessous de la demi-droite issue de u_0. Cependant l'espace de balayage compris entre la chaîne, le segment de droite $[ar(u_0), ar(u)]$ et la demi-droite issue de u et O est lui aussi vide (figure 2.9). En effet s'il y avait des sommets dans cet espace, alors il y aurait parmi ces sommets un sommet s tel que $(s, av(u))$ soit une arête du graphe de visibilité. Comme $(s, av(u)) \prec_{av(u)} u$, et comme les arêtes de l'arbre d'horizon ne peuvent se couper, l'extrémité de $e^s(s)$ est sur la chaîne d'arêtes, ce qui est impossible d'après le choix de cette chaîne.
Cet espace étant vide, tous les sommets balayés sont vus de $ar(u)$ et les arêtes correspondantes appartiennent au graphe de visibilité. □

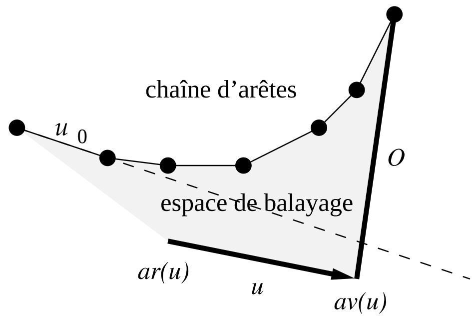


FIG. 2.9 – *Vacuité de l'espace de balayage.*

La complexité du balayage des chaînes se réduit déjà à $O(m^2)$. Nous montrons maintenant grâce à un schéma de comptage, que chaque arête du graphe de visibilité n'est en fait prise en compte seulement un nombre constant de fois lors des balayages des chaînes.

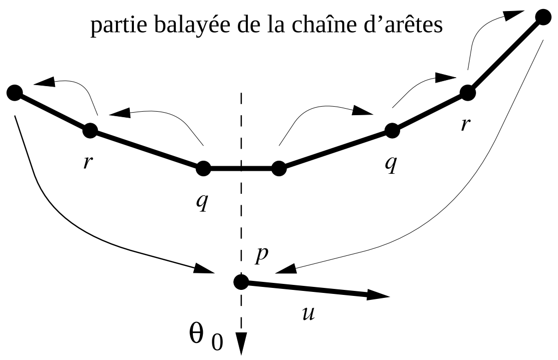


FIG. 2.10 – *Schéma de comptage : report des charges.*

Lors de la mise à jour de l'arête u de l'arbre d'horizon, considérons la partie balayée de la chaîne d'arêtes. La droite de direction θ_0 (début du balayage) passant par $ar(u)$ sépare les sommets de cette partie de chaîne en deux ensembles. Pour les sommets à droite, nous reportons la charge de balayage d'un sommet q sur le sommet suivant r de la chaîne. Pour les sommets à gauche, nous reportons la charge d'un sommet q sur le sommet précédent r de la chaîne. Enfin nous reportons la charge des deux sommets extrêmes de la partie de chaîne sur $ar(u)$ (figure 2.10).

La charge due aux sommets extrêmes est de $2m$. Nous montrons maintenant que si un sommet est chargé lors de la mise à jour d'une arête partant de p, alors il ne peut être chargé qu'une seule fois pendant le balayage entre θ_0 et $\theta_0 + \pi$ lors des mises à jour des arêtes partant de p.

Soit r un sommet chargé lors du balayage d'une partie droite de chaîne lors de la mise à jour d'une arête issue de p. Considérons la dernière fois où cette situation se produit et soit alors q le sommet qui charge r. Comme l'espace qui vient d'être balayé est vide (voir la démonstration de la proposition 2.14), avant cet événement toute arête issue de q coupe (p, r), et donc r est « caché » de p et ne peut être chargé lors de la mise à jour d'une arête issue de p.

Supposons maintenant que le sommet r est chargé lors du balayage d'une partie gauche de chaîne lors de la mise à jour d'une arête issue de p. Considérons la première fois où cette situation se produit et soit alors q le sommet qui charge r. Comme l'arête (r, q) est inférieure à la nouvelle arête u^s, elle est passée avant u^s. Si r était chargé ultérieurement par un sommet q' lors de la mise à jour d'une arête issue de p, alors r se retrouverait dans l'espace de balayage, ce qui est contradictoire avec le fait que cet espace est vide. r ne peut donc plus être chargé lors de la mise à jour d'une arête issue de p.

Lors de l'algorithme, un sommet r ne peut être chargé qu'une fois à cause de la mise à jour d'une arête issue de p, et (p, r) est une arête du graphe de visibilité. Le nombre total de charges est donc $O(m)$.

L'algorithme décrit, est donc un algorithme optimal de balayage du graphe de visibilité, qui s'exécute en temps $O(n \log n + m)$ en utilisant un espace de travail $O(n)$.

2.2 Résultats expérimentaux

Nous avons programmé cet algorithme et nous l'avons comparé expérimentalement à deux autres algorithmes de balayage de graphe de visibilité : l'algorithme d' Overmars et Welzl [OW88] qui calcule le graphe de visibilité d'une scène de segments de droite, et l'algorithme de Pocchiola et Vegter [PV93] qui calcule le graphe de visibilité d'une scène d'objets courbes quelconques, que nous avons adapté ici pour des scènes de segments.

Ces deux algorithmes, les seuls à notre disposition pour les tests, ont tous deux un temps de calcul en $O(n \log n + m \log n)$ et utilisent une queue de priorité, que nous avons implémentée avec un tas (en utilisant un tableau).

Les trois algorithmes utilisent comme opération élémentaire le calcul de la position relative de trois points. Les autres opérations sont de simples mises à jour de pointeurs, d'importance comparable dans les trois algorithmes.

La comparaison de temps d'exécution dans ce cadre n'a pas beaucoup de sens : les différences de temps dépendraient plus des différences d'optimisations de programmation de bas niveau (qui peuvent accélérer de 50% un programme par rapport à un programme non optimisé). Nous avons alors mesuré la complexité « algorithmique » des trois algorithmes : nous avons compté le nombre d'opérations élémentaires effectuées lors du calcul du graphe de visibilité (sans prendre en compte la phase d'initialisation des diverses structures de données).

Pour effectuer les tests, nous avons généré aléatoirement des scènes de segments. Pour chaque segment $[p_1, p_2]$, le premier point p_1 est tiré au hasard dans le carré centré en $(0, 0)$ et de côté $2a$. Le second point est tiré au hasard dans le carré centré en p_1 et de côté $2b$. Nous avons effectué les tirages en utilisant une loi de probabilité uniforme. Pour un nombre de segments n fixé, nous avons fait varier la taille du graphe de visibilité en jouant sur les paramètres a et b. Par exemple, les scènes générées avec b pas trop petit par rapport à a ont un graphe de visibilité peu dense ($m = O(n)$), alors que les scènes générées avec b très petit devant a ont un graphe de visibilité dense ($m = O(n^2)$).

Pour une valeur fixée de n nous avons généré 500 scènes aléatoires de façon à obtenir des graphes de visibilité à peu près uniformément répartis entre graphes peu denses et graphes denses. La complexité du temps de calcul des graphes en fonction de leur taille devrait alors être une droite d'équation $complexite = c(n) * m + m_0$. Les mesures obtenues sont effectivement réparties selon une droite comme le montre la figure 2.11 pour des scènes de $n = 500$ segments. Nous avons alors calculé la pente $c(n)$ de la droite en utilisant une régression linéaire.

Nous avons calculé cette constante de complexité $c(n)$ pour $n = 100, 200, \ldots, 1000$ et la figure 2.12 montre la courbe de complexité résultante où nous avons tracé les points $c(n)$ en fonction de n.

Nous observons que les algorithmes d'Overmars et Welzl [OW88] et de Pocchiola et Vegter [PV93] sont très proches et la nature logarithmique de leur complexité en $O(m \log n)$ est visible (les courbes d'interpolations tracées pour ces deux algorithmes sont des régressions logarithmiques).

L'algorithme de balayage topologique est meilleur et sa complexité est effectivement indépendante de n. La complexité d'implémentation du programme étant de même or-

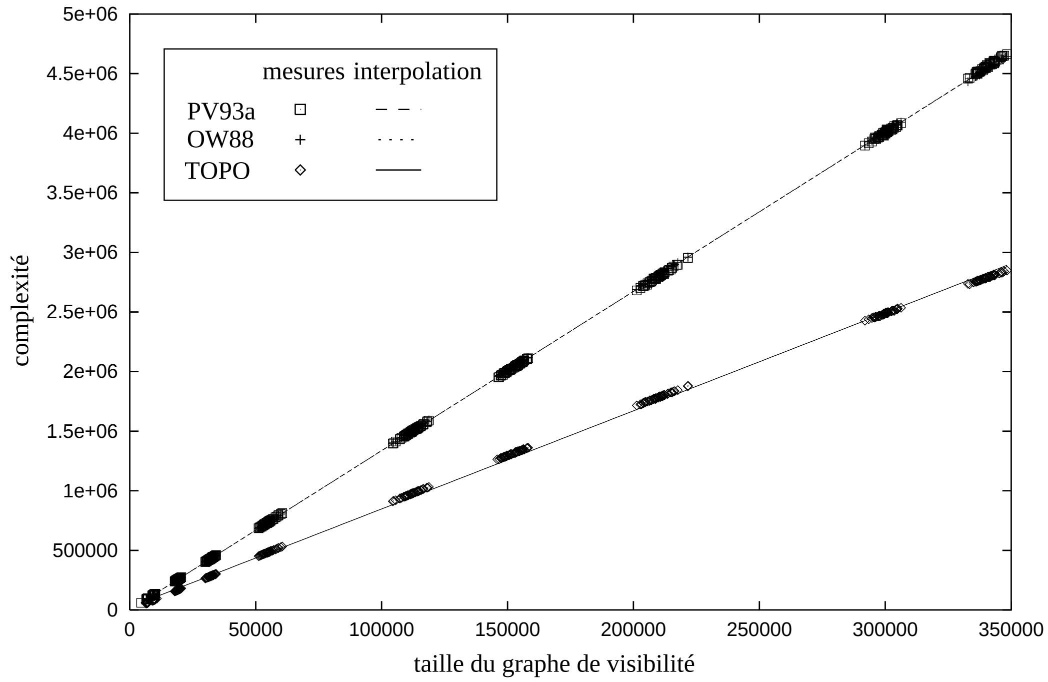


FIG. 2.11 – *Mesure de la complexité pour $n = 500$.*

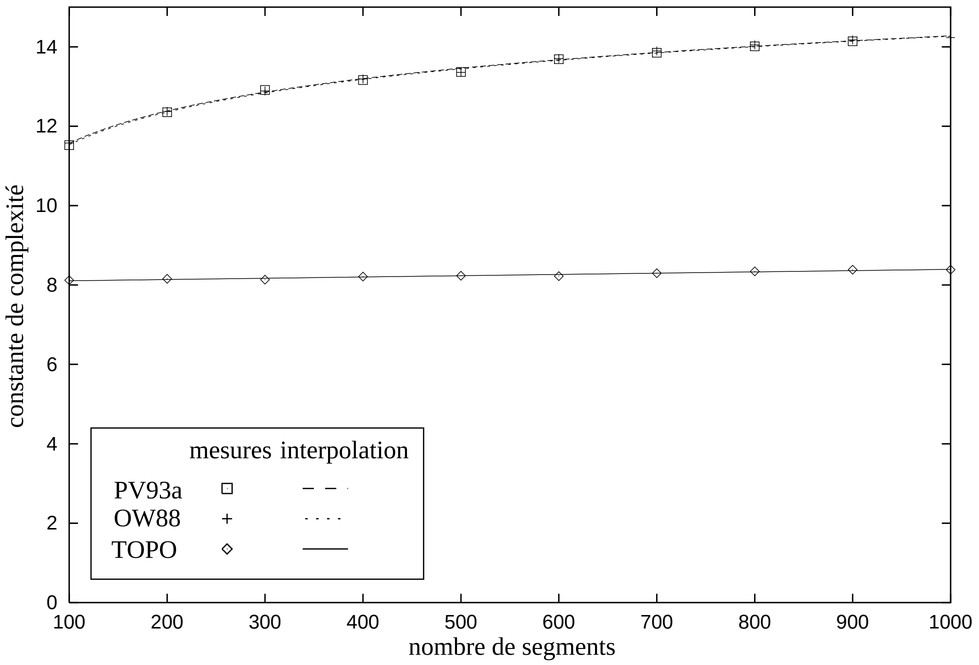


FIG. 2.12 – *Constante de complexité en fonction de n.*

dre que celles des deux autres programmes (nous avons même réutilisé une partie du code que nous avons écrit pour l'algorithme d'Overmars et Welzl dans l'implémentation de l'algorithme de balayage topologique), l'algorithme est aussi plus efficace en pratique.

2.3 Synthèse

Dans cette partie nous revenons sur les liens existants entre les algorithmes utilisant les arrangements de droites, l'algorithme d'Overmars et Welzl [OW88], le complexe de visibilité et notre algorithme de balayage topologique.

Comme nous l'avons déjà indiqué, le cadre combinatoire formel définissant le balayage à l'aide de filtres doit beaucoup à la formalisation qu'utilisent Pocchiola et Vegter [PV96a] dans la description de leur algorithme de calcul optimal de graphe de visibilité d'objets convexes.

Liens avec les arrangements de droites

On peut dire que notre algorithme de balayage de graphe de visibilité est au complexe de visibilité, ce que l'algorithme de balayage topologique d'Edelsbrunner et Guibas [EG86] est aux arrangements de droites.

Le balayage topologique du graphe de visibilité peut être vu comme le balayage topologique du squelette du complexe de visibilité dans l'espace dual. Puisque le squelette du complexe se projette sur l'arrangement de droites dual des points de la scènes, nous remarquons des analogies avec le balayage d'arrangements de droites. Cependant les différences entre les droites et les segments libres maximaux créent des différences fondamentales que nous devons prendre en compte ici. Ainsi, avec le formalisme utilisé dans ce chapitre, dans l'algorithme d'Edelsbrunner et Guibas [EG86] l'ensemble des arêtes à passer est simplement : $\min_{\prec} R = \mathcal{E}^s \cap \mathcal{E}^i$. Ici, comme nous l'avons vu, la condition doit être renforcée du fait d'une certaine indépendance entre arêtes dont les extrémités sont mutuellement cachées (figure 2.13).

Le schéma de comptage présenté dans la preuve de complexité est une adaptation de celui utilisé par Edelsbrunner et Guibas [EG89], transposé des arrangements de droites à la scène polygonale. Mais là aussi des précautions ont du être prises à cause de la nature différente du complexe.

Liens avec l'algorithme d'Overmars et Welzl [OW88]

Comme l'algorithme d'Overmars et Welzl [OW88] prend directement en compte les visibilités, il se démarque des algorithmes de balayage d'arrangements de droites. Cet algorithme reste cependant prisonnier de certaines limitations, notamment du sommet factice à l'infini correspondant à la droite verticale factice ajoutée à la fin de l'arrangement pour arrêter le balayage.

Ici, bien que nous ayons indiqué à chaque fois les rapports entre la situation dans la scène et dans l'espace dual, notre algorithme n'a pas été envisagé comme une « dédualisation » d'un algorithme lié à l'arrangement dual de droites. Le fait de considérer l'ensemble infini des arêtes orientées du graphe de visibilité et de faire abstraction de la relation de dualité utilisée nous a permis par exemple de faire disparaître le point factice. Nous pouvons alors faire un balayage infini, ce qui nous permet notamment

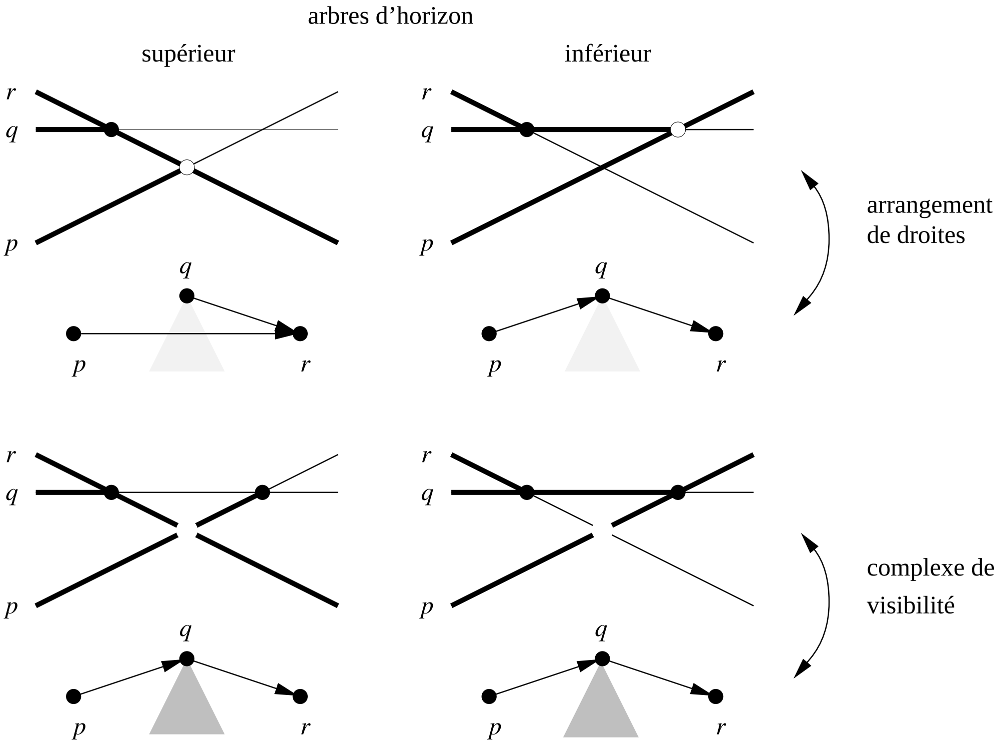


FIG. 2.13 – *Différences dues à la prise en compte de la visibilité.*

de calculer le graphe de visibilité de polygones avec trous (trous pouvant contenir eux-mêmes d'autres polygones...), ce qui n'était pas possible avant.

Retour à l'espace dual

Chaque point de vue apporte ses propres caractéristiques pour résoudre un même problème, ce qui permet d'avoir plus d'outils à sa disposition.

La vision duale permet elle aussi de rajouter des outils :

Proposition 2.15 *Si dans la représentation duale du complexe, on peut tracer un segment de droite d'une arête p^* à une arête q^*, tel que la pente de ce segment soit supérieure (resp. inférieure) à celles des deux arêtes et que ce segment ne coupe pas d'autres arêtes de pente supérieure (resp. inférieure), alors dans la scène, les deux points p et q correspondant aux deux arêtes sont mutuellement visibles.*

Démonstration. La droite support du segment est la droite duale d'un point r. Le segment représente des demi-droites pivotant autour de r, depuis p jusqu'à q. Si dans le complexe, ce segment ne rencontre pas d'arête de pente plus grande, cela signifie qu'il n'y a pas de points de la scène dans le cône $(\overrightarrow{rp}, \overrightarrow{rq})$ (restreint aux régions de visibilité correspondantes du complexe), et donc les points p et q sont mutuellement visibles (figure 2.14). □

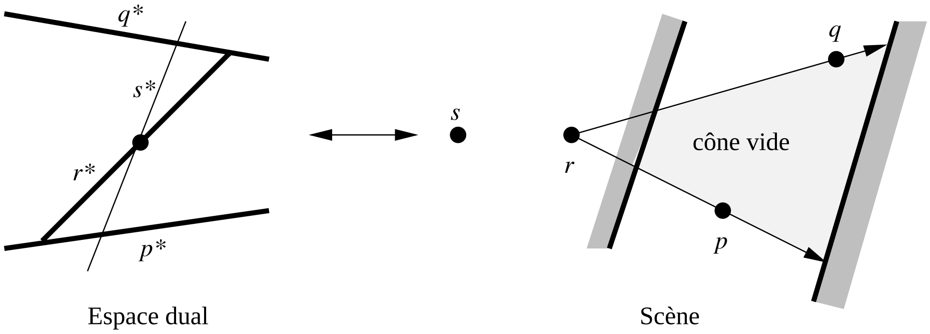


FIG. 2.14 – *Visibilité démontrée dans l'espace dual.*

2.4 Construction du complexe de visibilité

Nous montrons ici comment construire le complexe de visibilité en utilisant l'algorithme de balayage topologique.

Nous avons vu que l'algorithme de balayage topologique du graphe de visibilité effectuait en fait un balayage du squelette du complexe de visibilité. Grâce à la cohérence de ce balayage, nous pouvons construire les éléments du complexe.

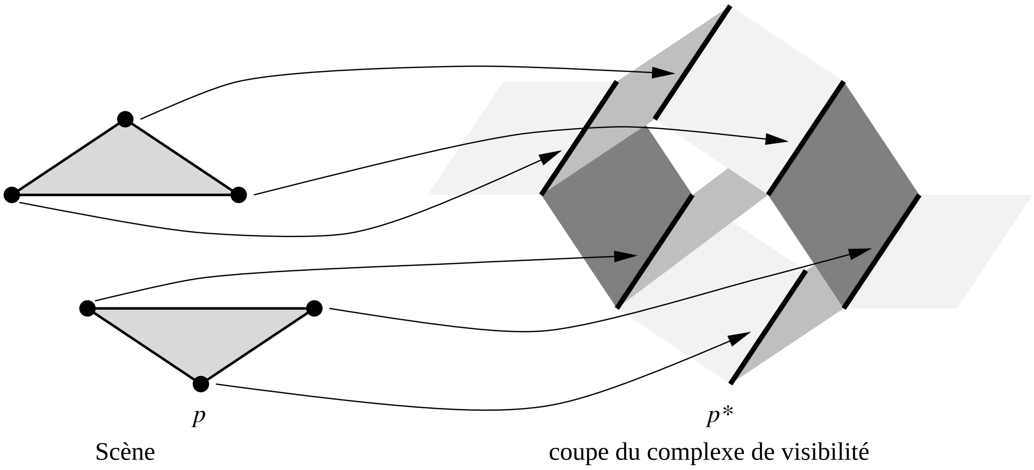


FIG. 2.15 – *Association d'arêtes de la coupe du complexe aux sommets de la scène.*

Lors de ce balayage, nous associons à chaque sommet p de la scène l'arête du complexe de visibilité « coupée » par la ligne de balayage et dont le support est la courbe duale p^* (figure 2.15). Plus précisément, chaque arête du graphe de visibilité correspond à un sommet du complexe. Nous associons au sommet p l'arête du complexe telle que l'arête correspondant à son sommet gauche a déjà été balayée et l'arête associée à son sommet droit appartient au moins à l'un des deux arbres d'horizon.

Les faces du complexe ne sont pas mises en correspondance directe avec un élément de la scène : nous y accédons par l'intermédiaire des arêtes du complexe déjà créées.

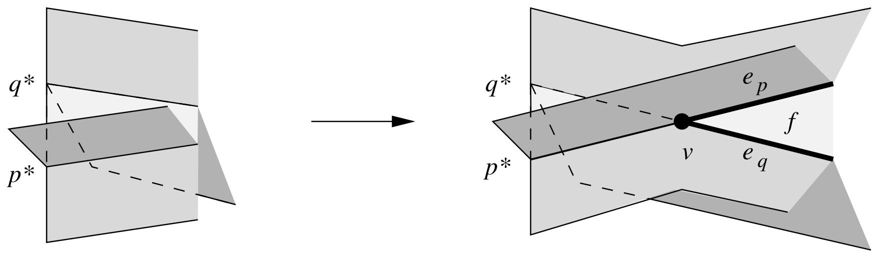


FIG. 2.16 – *Exemple de construction du complexe lors du passage d'une arête du graphe de visibilité.*

Le complexe est construit au fur et à mesure lors du passage d'une arête (p, q) du graphe de visibilité comme dans l'exemple de la figure 2.16. Ce passage indique le passage d'un sommet v dans le complexe. À ce moment là, les deux arêtes gauches incidentes au sommet et les faces incidentes à ces arêtes sont déjà créées, mais leur construction est incomplète. Lors du passage, nous créons le sommet v et mettons à jour les relations d'incidence entre le sommet et ses deux arêtes gauches associées à p et q. Nous créons les deux arêtes droites e_p et e_q de v, mettons à jour les relations d'incidences entre ces éléments et associons les arêtes e_p et e_q respectivement aux points p et q. Enfin, nous créons la face f « ouverte » par v (c'est-à-dire la face dont v est le sommet extrême gauche) et mettons à jour les relations d'incidences entre les nouvelles arêtes e_p et e_q et les faces incidentes aux arêtes droites de v (dont f fait maintenant partie).

La configuration locale du complexe autour d'un sommet (nombre de faces et relations d'incidences entre ces faces et les arêtes incidentes au sommet) dépend de la configuration géométrique de la scène autour des sommets de l'arête du graphe de visibilité correspondante. Comme nous l'avons vu au chapitre précédent, il y a 17 configurations différentes à traiter. Chaque configuration est uniquement déterminée par les types des arêtes incidentes gauches du sommet et finalement, la construction du « morceau » de complexe autour du sommet passé se fait en temps $O(1)$.

Il ne faut pas oublier de calculer les étiquettes des éléments nouvellement créés du complexe. Mais là aussi, grâce à la cohérence du balayage les visibilités sont mises à jour en temps constant : elles ne sont modifiées qu'en fonction de la configuration géométrique locale de la scène.

Grâce au balayage topologique, le complexe peut être construit en temps optimal $O(n \log n + m)$ et servir ensuite a effectuer des calculs de visibilité comme nous le montrons dans le chapitre suivant.

Chapitre 3

Calculs de polygones de visibilité

Dans ce chapitre nous montrons comment utiliser le complexe de visibilité pour calculer le polygone de visibilité d'un point situé dans une scène composée de polygones, c'est-à-dire calculer la vue autour du point considéré comme point de vue. Nous étudions deux algorithmes et comparons leurs performances expérimentales.

Nous montrons ensuite comment calculer le polygone de visibilité d'un segment, c'est-à-dire comment calculer les régions éclairées de la scène quand le segment est considéré comme une source de lumière.

— *Attendez que je trouve à tâtons le petit bureau du fond, et je vous en mettrai plein la vue.*

— *Aïe ! L'andouille a mis son doigt dans mon œil !*

Le cas Lagaffe.

3.1 Calculs de vues autour d'un point

Soit un point p_v, dit *point de vue*, dans une scène polygonale. Le polygone de visibilité de p_v, appelé *vue autour de* p_v, contient l'ensemble des points de la scène visibles depuis p_v. C'est un polygone étoilé (figure 3.1a) autour de p_v dont les côtés sont successivement *radiaux* et *transversaux* (figure 3.1b). Les côtés radiaux sont portés par les rayons issus du point de vue p_v et passant par les sommets de polygone visibles depuis p_v ; ils sont délimités par le sommet de polygone visibles et par le point d'intersection du rayon avec l'objet vu le long de ce rayon (qui peut être dans certains cas confondu avec le sommet de polygone). Les côtés transversaux relient les extrémités des côtés radiaux et sont portés par les côtés de polygones visibles depuis p_v.

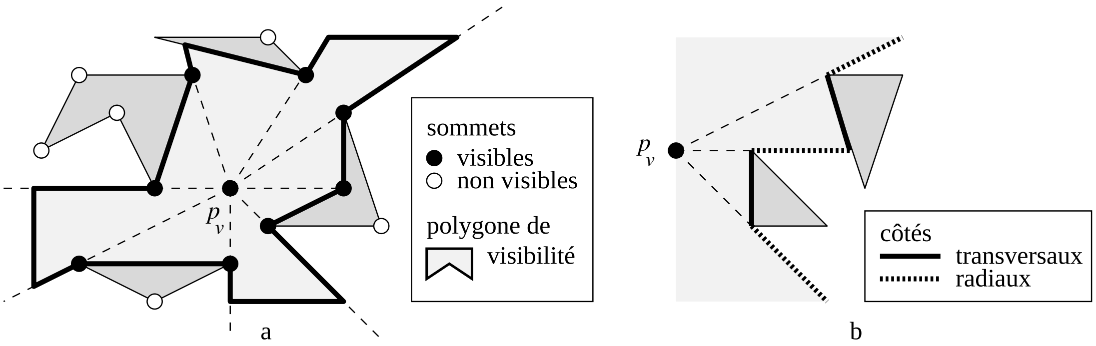


FIG. 3.1 – *Vue autour d'un point.* **a.** *Polygone de visibilité.* **b.** *Nature des côtés du polygone de visibilité.*

La vue autour de p_v calcule donc les objets vus depuis p_v, mais dans un certain ordre : les côtés radiaux apparaissent dans la frontière du polygone de visibilité selon leur ordre angulaire autour de p_v.

L'algorithme naïf pour calculer cette vue consiste à examiner tous les sommets de polygone de la scène, et pour chacun de ces sommets examiner tous les côtés de polygone pour vérifier s'il y en a un ou non qui cache le sommet de p_v. Cet algorithme s'exécute dans le cas le pire en temps $O(n^2)$ (n sommets à examiner, pour chaque sommet $O(n)$ segments à tester), et dans le meilleur des cas en temps $O(vn)$, où v est la taille de la vue, si la vérification des sommets cachés ne prend par chance qu'un temps $O(1)$.

Il faut enfin effectuer un tri angulaire de ces points. La recherche des objets stoppant les rayons radiaux peut se faire en même temps que le test de visibilité d'un sommet depuis p_v.

3.1.1 Calcul de vue par balayage

Puisque les sommets visibles doivent être ordonnés selon leur angle autour de p_v, autant se servir de cet ordre pour maintenir une cohérence dans les calculs de visibilité. C'est ce que fait l'algorithme classique de calcul de vue suivant, qui calcule une vue en temps $O(n \log n)$ grâce à un balayage rotationnel. Après avoir trié les points de la scène selon leur ordre angulaire autour de p_v, l'algorithme calcule la liste des objets intersectés (dans l'ordre) par la demi-droite issue de p_v et passant par le premier point du tri, et maintient cette liste quand la demi-droite issue de p_v passe successivement par les autres sommets de la scène dans l'ordre du tri angulaire.

La liste ne contient que les côtés susceptibles d'être vus, c'est-à-dire les côtés tels que p^+ soit au-dessus de (p_v, p). Les autres côtés sont cachés du point de vue par le polygone et ne sont pas considérés (figure 3.2). Certains sommets p incidents à de tels côtés peuvent même être déjà écartés lors du tri initial.

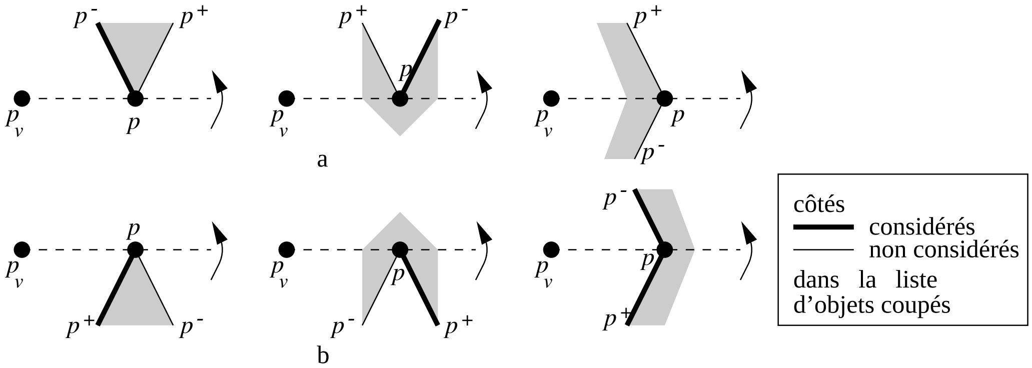


FIG. 3.2 – *Configurations lors du passage d'un sommet lors du calcul de vue.*

Une fois initialisée, la liste est maintenue à chaque fois que la demi-droite issue de p_v passe par l'un des sommets de la scène (dans l'ordre du tri) :

si $[p, p^+]$ n'est pas dans la liste (figure 3.2a)
  si p^- est au-dessus de (p_v, p)
    ajouter $[p^-, p]$ dans la liste
    si $[p^-, p]$ est le premier segment de la liste, p est vu depuis p_v
sinon (figure 3.2b)
  si p^- est au-dessus de (p_v, p)
    remplacer $[p, p^+]$ par $[p^-, p]$
    si $[p^-, p]$ est le premier segment de la liste, p est vu depuis p_v
  sinon
    enlever $[p, p^+]$ de la liste

Le tri initial se fait en temps $O(n \log n)$. La recherche dans la liste ordonnée et son maintien se fait en temps logarithmique à chaque passage de l'un des $O(n)$ points examinés. La complexité finale de l'algorithme est donc en $O(n \log n)$.

L'Utilisation de l'arrangement dual de la scène, construit lors d'une phase de pré-traitement, permet de supprimer le tri initial et l'utilisation de la queue de priorité.

L'ensemble des droites passant par le point de vue p_v devient dans l'espace dual une courbe p_v^*. Comme chaque face de l'arrangement correspond à un ensemble de droites traversant la même liste d'objets, pour avoir les listes successives des objets traversés par la demi-droite issue de p_v quand celle ci pivote autour de p_v, il suffit de calculer les faces successives de l'arrangement traversées par p_v^*.

Ces faces sont trouvées en reprenant la méthode utilisée dans la construction incrémentale de l'arrangement (exposée au chapitre 1) : parcours des arêtes bordant une face pour trouver l'arête de sortie et la face suivante. La localisation de la première face (qui est l'une des faces extrêmes gauche de l'arrangement) coupée par la courbe duale se fait en temps $O(\log n)$. Trouver le premier objet vu se fait aussi en temps $O(\log n)$ en faisant une recherche dans la liste des objets associée à cette première face. À chaque nouvelle face rencontrée, la liste des objets coupés et le premier objet vu sont maintenus en temps constant : le ou les segments qui disparaissent ou apparaissent dans la liste sont ceux incidents au sommet de polygone associé à l'arête de sortie de la face ; la liste des objets vus est donnée directement par la nouvelle face traversée ; la mise à jour de l'objet vu se fait alors en temps constant en reprenant l'examen de cas de l'algorithme précédent.

Cet algorithme utilise une droite passant par p_v et non plus une demi-droite issue de p_v comme dans l'algorithme précédent, et des changements dans la liste de segments intersectés peuvent alors concerner des objets situés derrière p_v. Le balayage s'effectue alors seulement de $-\pi/2$ à $\pi/2$ et un changement derrière p_v dans la direction θ correspond à un changement dans la direction $\theta + \pi$.

Grâce à des structures persistantes, les listes d'objets associées aux faces de l'arrangement n'occupent un espace mémoire qu'en $O(n^2)$, et d'après le théorème de la zone, le calcul de la vue s'effectue en temps $O(n)$. Cet algorithme a été notamment décrit par Pocchiola [Poc90] dans le cadre de scènes composées d'objets courbes convexes où la complexité du calcul de vue devient $O(2^{\alpha(n)}n)$, où $\alpha(n)$ est la fonction inverse d'Ackerman (fonction constante en pratique).

3.1.2 Calculs de vue avec le complexe de visibilité

Comme pour l'algorithme de calcul de graphe de visibilité en temps $O(n^2)$ d'Edelsbrunner et Guibas [EG86], la complexité des deux algorithmes de calcul de vue que nous venons de décrire ne dépend que de n. Le dernier algorithme n'est efficace que dans le cas le pire, c'est-à-dire quand (presque) tous les objets de la scène sont visibles depuis le point de vue. Cependant la taille v de la vue calculée est souvent beaucoup plus petite que le nombre total d'objets ($v \ll n$). Pour accélérer le calcul et obtenir un algorithme sensible à la sortie (c'est-à-dire dépendant aussi de v), il faut aussi pour ce problème tenir compte directement des relations de visibilité pour ne traiter, si possible, que les objets visibles depuis le point de vue.

Nous utilisons dans cette partie le complexe de visibilité, qui « contient » déjà les calculs de visibilité entre objets de la scène, comme outil pour améliorer la complexité du calcul d'une vue.

L'avantage de l'utilisation de l'arrangement dual est d'introduire une cohérence dans le calcul de vue, cohérence qui simplifie le calcul en évitant de refaire à chaque fois des recherches. L'inconvénient de l'arrangement est qu'il fait prendre en compte dans le calcul des changements de visibilité non visibles depuis le point de vue. Ceci est du, comme nous l'avons déjà indiqué, au fait que l'arrangement représente des droites qui traversent les objets.

Nous avons aussi vu que le complexe de visibilité avait été créé pour résoudre ce problème : il ne représente plus des ensembles de droites, mais des ensembles de segments libres maximaux. L'utilisation du complexe permet alors de ne considérer que les points visibles depuis p_v.

Le calcul de la vue autour de p_v revient à calculer les changements de visibilité le long du *rayon*, et non plus de la droite, issu de p_v quand ce rayon pivote autour de p_v. Cela revient dans l'espace dual à trouver les faces successives du complexe de visibilité traversées par la courbe duale p_v^* du point de vue et à noter les arêtes séparant ces faces.

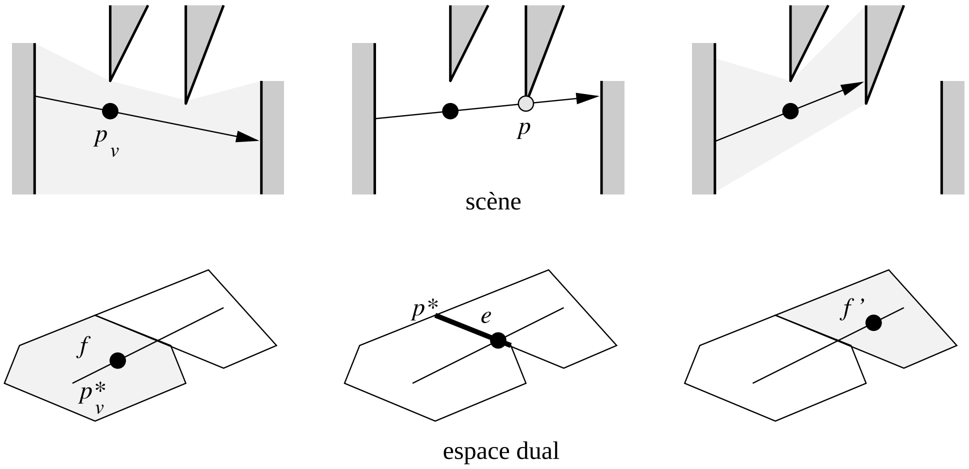


FIG. 3.3 – *Changement de visibilité autour d'un point : passage d'une face du complexe à une autre.*

En effet, lorsque le rayon pivote autour de p_v, un changement de visibilité le long de ce rayon se produit quand celui-ci quitte la face f du complexe où il se trouve pour entrer dans une autre face f'. Le rayon effectue ce franchissement de face en traversant l'arête e qui sépare les deux faces (figure 3.3) et le sommet de polygone correspondant à l'arête traversée est alors visible depuis p_v et fait partie de la vue.

L'algorithme de calcul de vue consiste à maintenir la face du complexe contenant le rayon passant par p_v quand il pivote autour de p_v depuis la direction θ_0 jusqu'à la direction $\theta_0 + \pi$ (car les changements de visibilité à l'arrière du rayon dans la direction θ sont ceux se produisant à l'avant du rayon dans la direction $\theta + \pi$) :

calculer la face f du complexe contenant le rayon de pente θ_0 passant par p_v
tant que la pente du rayon est inférieure à $\theta_0 + \pi$
    calculer l'arête de sortie de la face f
    ajouter le sommet correspondant à l'arête dans la vue
    calculer la nouvelle face f où se trouve le rayon

La première étape de l'algorithme consiste à trouver la face contenant le rayon de direction θ_0. Pour cela, nous considérons la décomposition en trapèzes de la scène selon

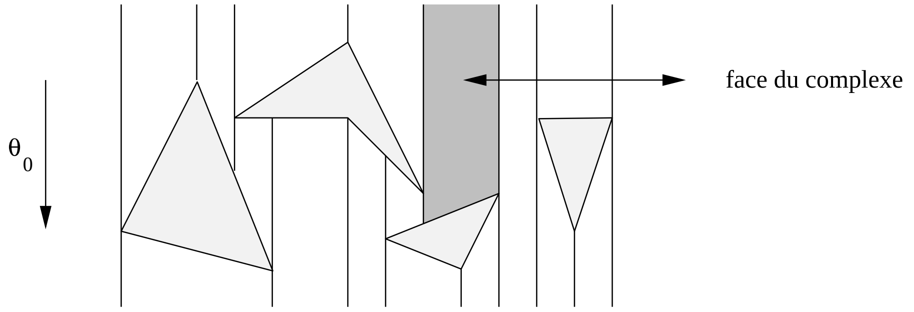


FIG. 3.4 – *Décomposition en trapèzes de la scène.*

la direction θ_0. Cette décomposition est obtenue en étendant à partir de chaque sommet de polygone un segment libre maximal de direction θ_0 (figure 3.4).

Elle correspond à une coupe du complexe de visibilité selon la direction θ_0 et chaque trapèze de cette décomposition correspond à une face du complexe présente dans la coupe. Trouver la face initiale revient donc à trouver le trapèze de la décomposition contenant le point de vue. Comme la décomposition contient $n_{sommets} + n_{polygones} + 1 = O(n)$ trapèzes, en utilisant l'algorithme de localisation de point dans une subdivision du plan de Sarnak et Tarjan [ST86] (algorithme basé sur une structure d'arbre rouge–noir persistant) la recherche s'effectue en temps $O(\log n)$. La structure d'arbres persistant se construit en temps $O(n \log n)$ et en espace $O(n)$ pendant le calcul de la coupe initiale du complexe (et donc de la décomposition en trapèzes) lors de la phase d'initialisation de la construction du complexe.

Une fois la face initiale calculée, nous devons chercher à chaque étape l'arête par laquelle le rayon passant par p_v sort de la face où il se trouve ainsi que la nouvelle face où il pénètre.

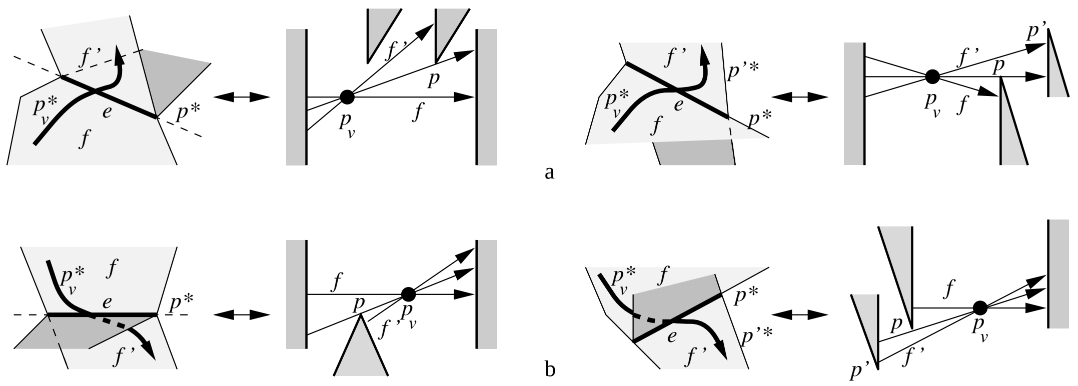


FIG. 3.5 – *Nouvelle face traversée par* p_v^*. **a.** *Sortie par une arête de la chaîne inférieure.* **b.** *Sortie par une arête de la chaîne supérieure.*

Nous avons vu qu'une face était délimitée par deux chaînes d'arêtes. Si le rayon sort de la face f où il se trouve par une arête e de la chaîne supérieure de f, soit e est une arête grasse, soit e est une arête de type μ-convexe ; dans les deux cas, la nouvelle face

f' contenant le rayon est la face incidente gauche-haute de e (figure 3.5a). Si l'arête de sortie e appartient à la chaîne inférieure de f, soit e est une arête grasse, soit e est une arête de type λ-convexe ; dans les deux cas, la nouvelle face f' contenant le rayon est la face incidente droite-basse de e (figure 3.5b).

Pour savoir si l'arête de sortie se trouve dans la chaîne inférieure ou dans la chaîne supérieure, il suffit de tester la position relative du point de vue par rapport à la droite v^* correspondant au sommet v extrême droit de la face : si p_v est au-dessus de cette droite, alors le changement de visibilité se produit à l'avant du rayon et l'arête de sortie se trouve dans la chaîne supérieure (figure 3.6a) ; si p_v est en dessous de v^*, alors le changement de visibilité se produit à l'arrière du rayon et l'arête de sortie se trouve dans la chaîne inférieure (figure 3.6b).

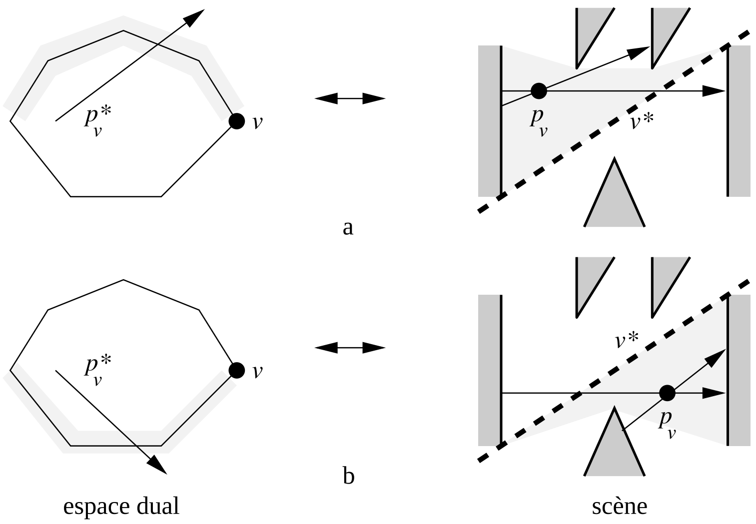


FIG. 3.6 – *Sortie de la face.* **a.** *Sortie par la chaîne inférieure.* **b.** *Sortie par la chaîne supérieure.*

Ainsi nous n'avons qu'une seule chaîne à considérer. De plus, nous remarquons aussi que le rayon passant par p_v ne peut sortir de la face par une arête grasse gauche ; nous écartons donc aussi ces arêtes de la recherche.

Soit une arête $e = (v_g, v_r)$ de la chaîne de sortie. Pour savoir si le rayon sort de la face avant, après, ou par cette arête, il suffit de tester la position de p_v par rapport aux droites v_g^* et v_r^* correspondant aux deux sommets v_g et v_r délimitant l'arête. Par exemple, si l'arête e est dans la chaîne supérieure, alors si p_v est en dessous de v_d^*, la sortie se fait par une arête située après e (figure 3.7a) ; si p_v est au-dessus de v_g^*, la sortie se fait par une arête située avant e (figure 3.7b) ; enfin si p_v est au-dessus de v_d^* et en dessous de v_g^*, la sortie se fait par l'arête e (figure 3.7c).

Pour trouver l'arête de sortie, nous pouvons nous inspirer de l'algorithme de calcul de vue utilisant l'arrangement de droites : après avoir trouvé la chaîne d'arêtes de sortie, nous parcourons les arêtes de cette chaîne de la gauche vers la droite jusqu'à trouver l'arête de sortie. La recherche commence à partir de l'arête d'entrée si celle-ci

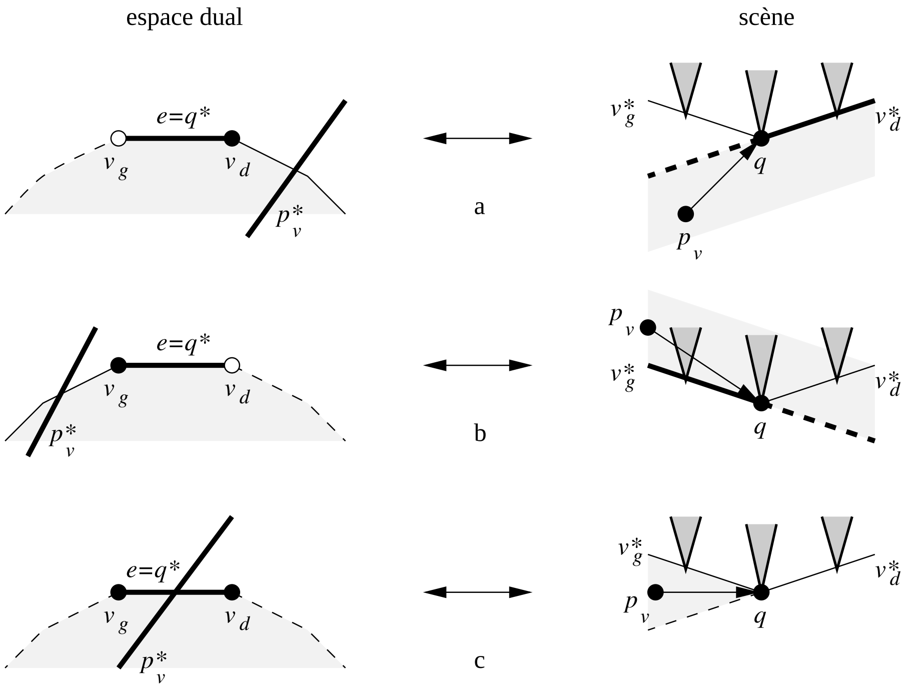


FIG. 3.7 – *Sortie d'une face par la chaîne supérieure : test d'une arête non grasse.* **a.** *Sortie après l'arête testée.* **b.** *Sortie avant l'arête testée.* **c.** *Sortie par l'arête testée.*

se trouve dans la chaîne de sortie, à partir de la première arête de la chaîne de sortie ne faisant pas partie de l'arête grasse gauche sinon.

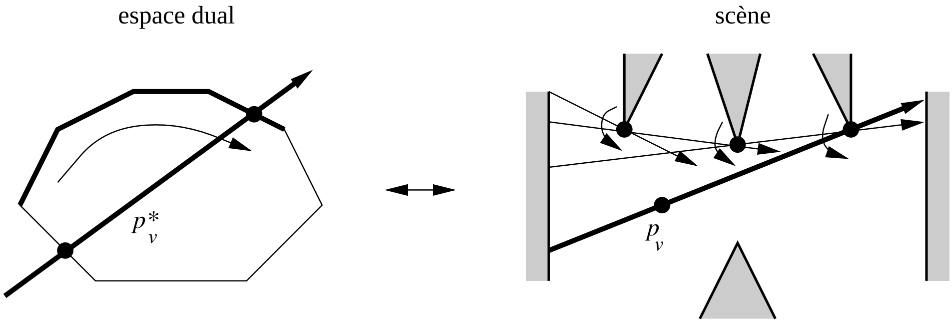


FIG. 3.8 – *Signification géométrique du parcours des arêtes d'une chaîne pour trouver la sortie.*

Dans la scène, cela revient à faire pivoter une droite autour des sommets bordant la face géométrique jusqu'à ce que cette droite passe par le point de vue (figure 3.8).

Grâce à cette méthode, nous obtenons un premier algorithme de calcul de vue :

Proposition 3.1 *Une vue autour d'un point p_v se calcule par la méthode qui vient d'être décrite en temps $\Omega(\log n + v)$ et $O(vn)$, où v est la taille de la vue calculée. De plus, il existe des situations où la borne inférieure (resp. supérieure) est atteinte.*

Démonstration. La propriété 2.13 de visibilité indirecte du chapitre précédent permet de montrer que tous les points correspondant aux arêtes balayées sont visibles du point de vue. Si aucune arête grasse n'est balayée, des techniques similaires à celles employées dans le chapitre précédent pour démontrer la complexité du balayage du graphe de visibilité permettent de montrer que le nombre d'arêtes balayées est en $O(v)$. La borne inférieure v peut même être atteinte dans certains cas.

Les arêtes grasses peuvent augmenter fortement la complexité du balayage. Ces arêtes peuvent être décomposées par les courbes duales de sommets cachés de p_v, que ce soit à leur début ou à leur fin (figure 3.9). De plus, un sommet caché de p_v peut décomposer toutes les arêtes grasses des faces traversées dans le calcul de vue. La complexité maximale en $O(vn)$ est alors obtenue comme dans l'exemple de la figure 3.10. □

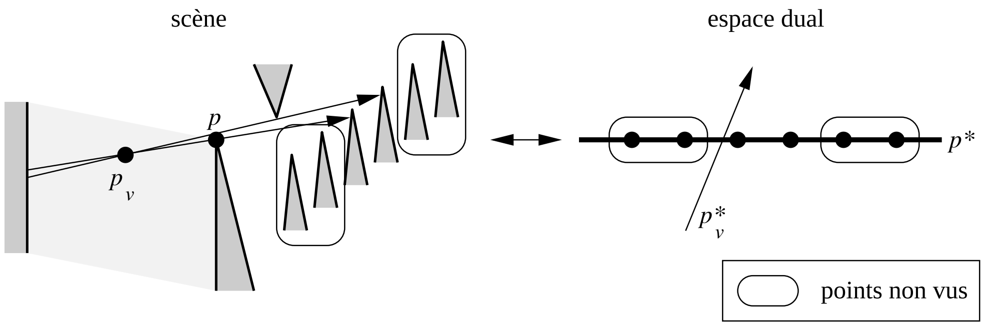


FIG. 3.9 – *Découpage d'une arête grasse par des sommets non vus.*

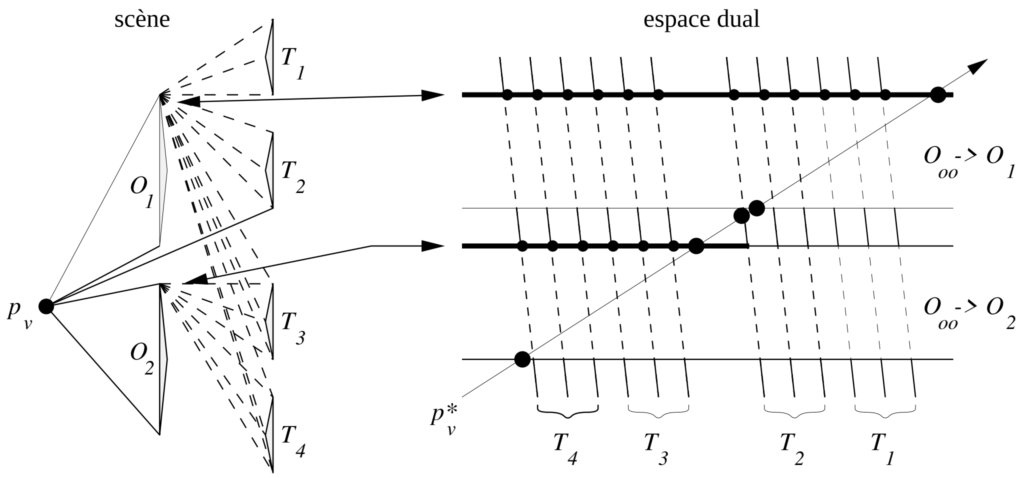


FIG. 3.10 – *Sommets non vus présents dans plusieurs arêtes grasses.*

La complexité de l'algorithme varie d'une complexité optimale à une complexité pire que celle de l'algorithme classique. Des études expérimentales montrent dans la partie suivante comment cette complexité varie en pratique.

Comme les arêtes sont ordonnées dans une chaîne d'arêtes, nous pouvons effectuer une recherche dichotomique dans ces chaînes pour trouver l'arête de sortie :

Proposition 3.2 *Une vue autour d'un point p_v peut se calculer en temps $O(v \log n)$, où v est la taille de la vue calculée, en utilisant le complexe de visibilité.*

Démonstration. Après une initialisation en temps $O(\log n)$, l'algorithme comporte v étapes. À chaque étape, une recherche dichotomique en temps $O(\log n)$ est effectuée pour trouver l'arête de sortie, soit une complexité totale en $O(v \log n)$. □

Nous pouvons construire une structure de données de recherche pour chaque chaîne d'arêtes en temps total $O(m)$. En effet, lors du balayage du complexe, les arêtes des chaînes sont examinées dans l'ordre où elles apparaîssent dans ces chaînes. Nous pouvons, en utilisant par exemple des arbres rouge–noir avec un pointeur sur le plus grand élément, insérer chaque arête en temps amorti $O(1)$ dans les chaînes correspondantes. Le temps de construction du complexe augmenté de structures de recherche reste donc inchangé (en $O(n \log n + m)$).

3.1.3 Étude expérimentale

Nous venons d'étudier deux algorithmes de calcul de vue autour d'un point de complexités théoriques différentes. Le premier, l'algorithme de *parcours*, a une complexité en $\Omega(\log n + v)$ et en $O(vn)$, et le deuxième, l'algorithme de *recherche*, a une complexité en $O(v \log n)$.

Nous étudions maintenant le comportement en pratique de chacun des algorithmes, puis nous comparons les performances expérimentales des deux algorithmes.

Protocole expérimental

Nous avons effectué des calculs de vue sur des scènes aléatoires de triangles disjoints. Chaque triangle $p_1p_2p_3$ est généré de façon aléatoire. Le premier sommet p_1 est tiré aléatoirement dans le disque de centre $(0,0)$ et de rayon a. Le deuxième sommet p_2 est tiré aléatoirement dans le disque centré en p_1 et de rayon b. Enfin le troisième sommet du triangle est tiré aléatoirement dans le rectangle de côté $[p_1, p_2]$ et de largeur c. Les trois paramètres a, b et c permettent de faire varier les caractéristiques de la scène, notamment la taille du graphe de visibilité.

Nous utilisons la taille du graphe de visibilité pour mesurer une *densité de visibilité* de la scène, que nous notons d_v. Cette densité est calculée par la formule $d_v = (2m - 2n)/n(n-3)$ et représente la fraction de sommets (autres que les deux sommets adjacents sur le polygone) vus en moyenne par un sommet de la scène. Elle varie théoriquement de 0 (aucun sommet vu) à 1 (tous les $n-3$ autres sommets vus), cependant les densités maximales obtenues en pratique sont plutôt de l'ordre de 0.6.

Pour une scène donnée, nous avons tiré aléatoirement 500 points de vue dans le disque centré en $(0,0)$ et de rayon a. Ces points de vue se situent donc en général à l'intérieur de la scène. Dans une deuxième série de tests, nous avons tiré aléatoirement 500 points de vue dans l'anneau compris entre les disques centrés en $(0,0)$ et de rayons respectifs a et $2a$. De cette façon, les points de vue se situent en général en dehors de la scène.

Complexité mesurée

Comme pour le chapitre précédent, la complexité mesurée n'est pas le temps d'exécution des programmes, mais une complexité plus algorithmique. Pour l'algorithme de parcours, nous avons mesuré le nombre d'arêtes parcourues. Pour l'algorithme de recherche, nous avons utilisé comme structure de recherche des arbres binaires équilibrés AVL. Nous avons alors compté non pas le nombre d'arêtes testées, mais le nombre de comparaisons réellement effectuées. En effet, une arête peut donner lieu à un ou deux tests (sortie avant – sortie après). Ainsi nous mesurons bien pour les deux algorithmes, le nombre de tests effectués.

Dans les diagrammes présentant les résultats, nous avons mis en abscisse la taille des vues calculées, et en ordonnée la complexité correspondante du calcul.

Résultats

La figure 3.11 montre les résultats obtenus avec l'algorithme de parcours pour des scènes de 600 triangles, de densités respectives faible ($d_v = 0.10$), moyenne ($d_v = 0.27$) et élevée ($d_v = 0.50$). Ces résultats sont typiques de ceux obtenus avec des scènes de taille différentes et de même densité. Nous observons que la taille minimale des vues calculées à l'extérieur de la scène est plus grande que celle des vues calculées à l'intérieur. En effet un point à l'intérieur a plus de chances d'être situé très près d'un ou de plusieurs triangles qui lui cachent alors une grande partie de la scène. Nous observons aussi que la complexité du calcul de vue depuis des points situés à l'extérieur est plus élevée que celle de vues calculées à l'intérieur. Pour des points extérieurs, la situation de la figure 3.10 se produit avec la majeure partie de la scène derrière O_1, $O_2 \ldots$, alors que pour des points intérieurs, seule une fraction de la scène est située derrière ces obstacles. La complexité quadratique du découpage des arêtes grasses explique alors la complexité plus élevée pour des points extérieurs.

Nous voyons dans la figure 3.11(a) que la borne inférieure de la complexité est bien linéaire en fonction de v. En revanche, la limite supérieure est moins marquée. Pour des point de vue extérieurs (figure 3.11b), la différence entre les limites inférieure et supérieure de complexité est assez élevée et les résultats sont uniformément répartis entre ces deux limites.

La figure 3.12 montre les résultats obtenus avec l'algorithme de recherche sur les mêmes scènes et avec les mêmes points de vue. Cette fois-ci, la distribution des complexités est beaucoup moins dispersée et la dépendance de la complexité en fonction de la taille de la vue calculée apparaît clairement.

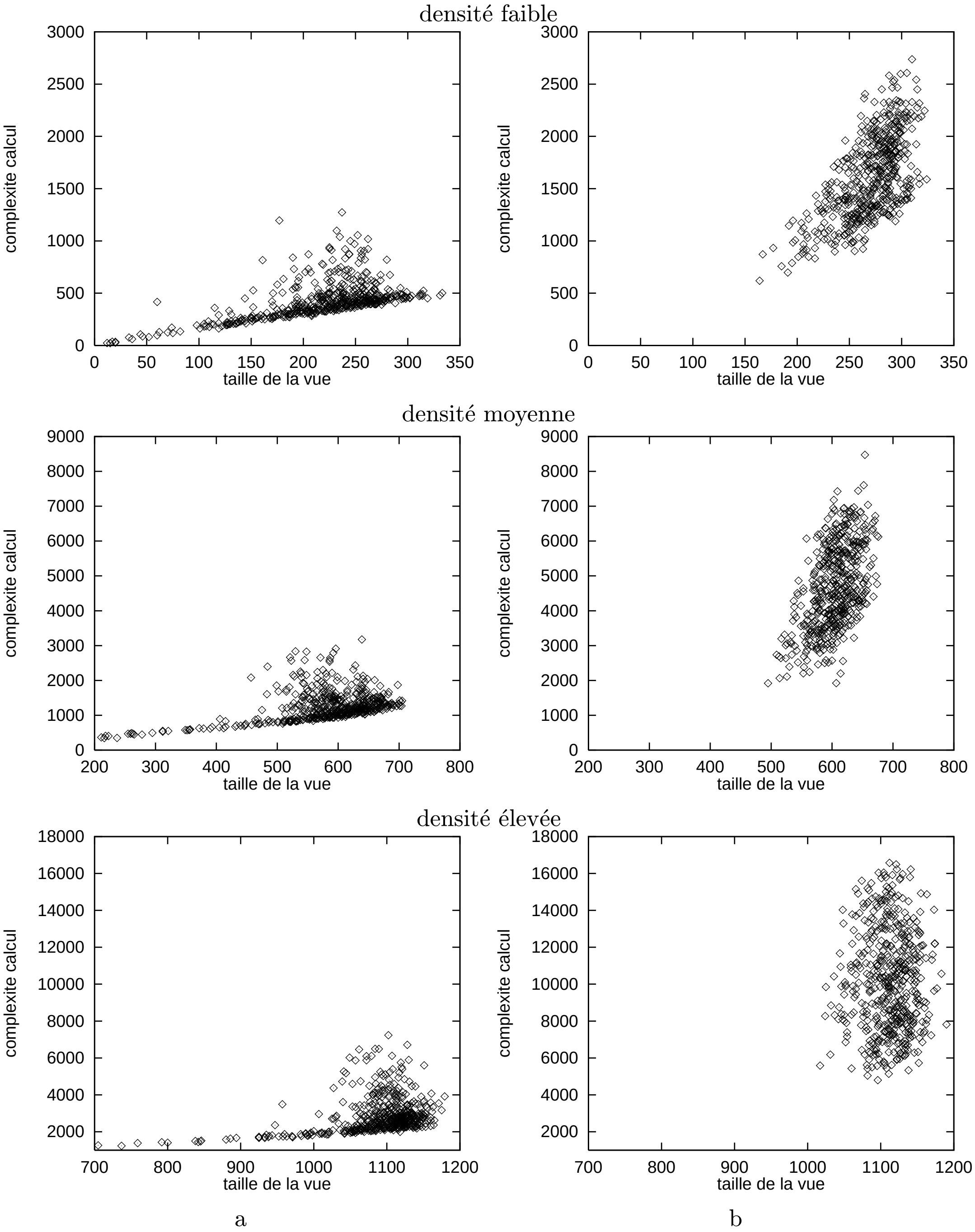


FIG. 3.11 – *Résultats de l'algorithme de parcours sur une scène de 600 triangles.* **a.** *Points de vue dans la scène.* **b.** *Points de vue à l'extérieur de la scène.*

Pour faciliter la comparaison entre les deux algorithmes, nous avons regroupé dans la figure 3.13 les résultats des deux algorithmes pour la scène de 600 triangles densité moyenne sur les mêmes diagrammes. L'algorithme de recherche semble meilleur que l'algorithme de parcours quand les points de vue sont situés à l'extérieur de la scène (figure 3.13b). En revanche, quand les points de vue sont situés à l'intérieur de la scène,

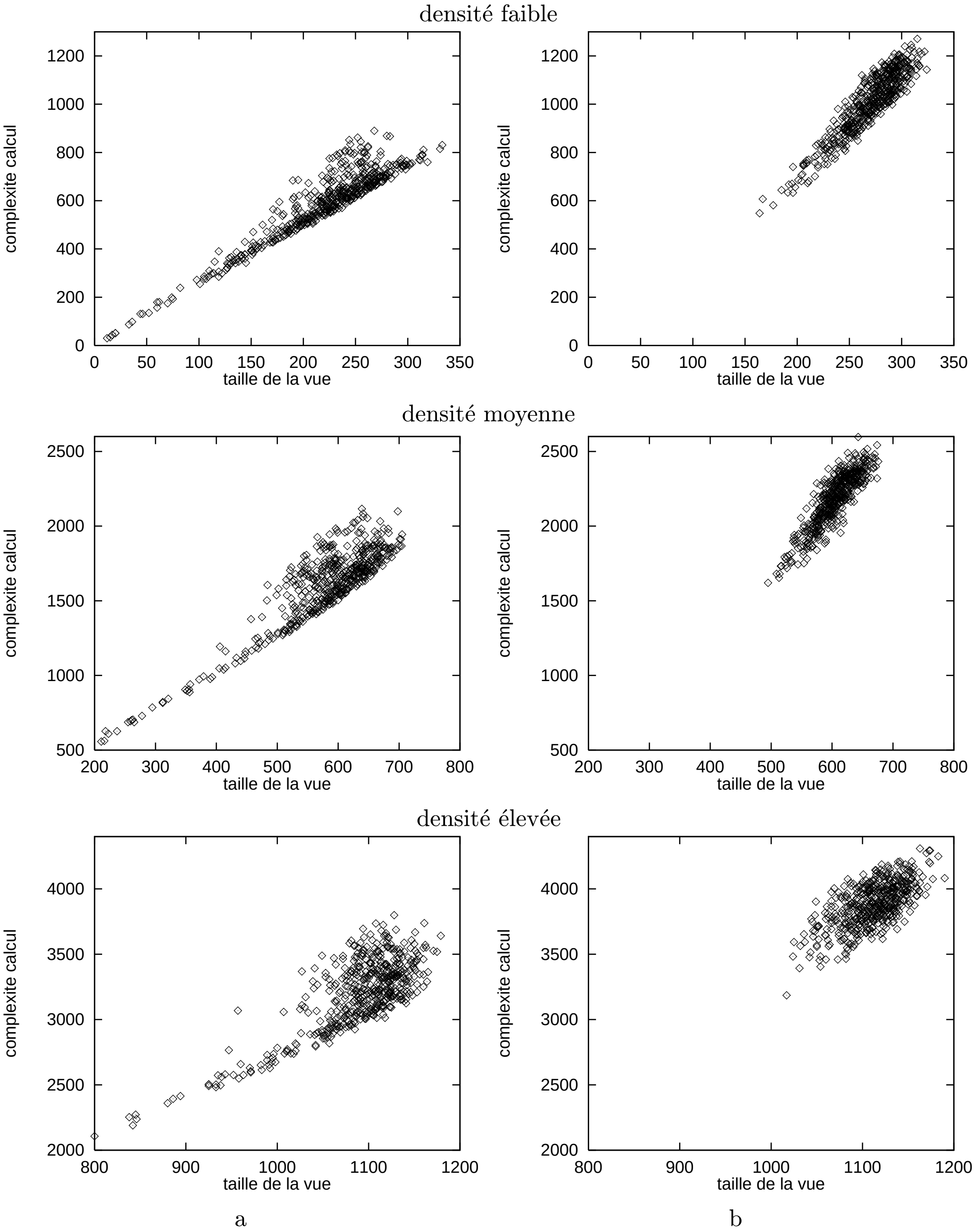


FIG. 3.12 – *Résultats de l'algorithme de recherche* **a.** *Points de vue dans la scène.* **b.** *Points de vue à l'extérieur de la scène.*

l'algorithme de parcours semble meilleur en moyenne (figure 3.13a).

Les mesures de complexité peuvent guider le choix de l'algorithme à utiliser. Cependant, il ne faut pas oublier l'aspect concernant la taille prise en mémoire par les structures de données. L'ajout de structure de données de recherche (arbres binaires par

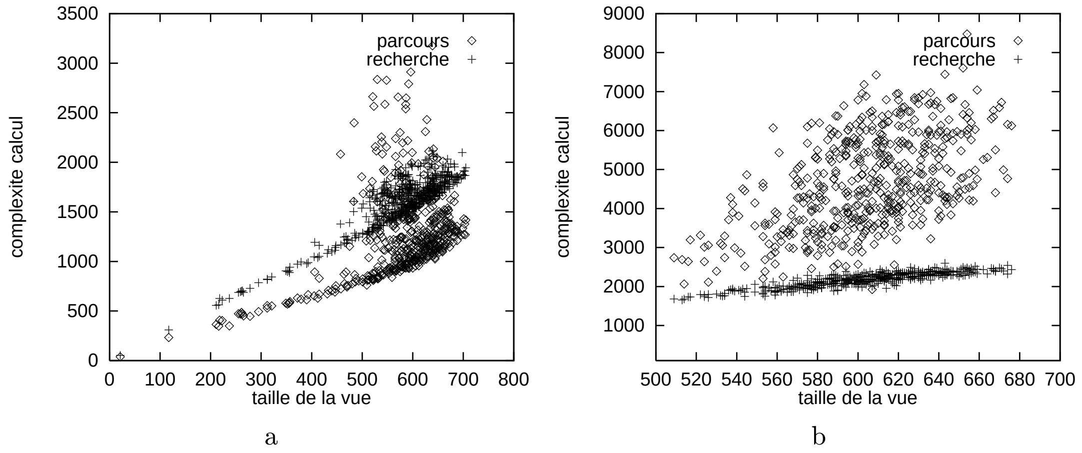


FIG. 3.13 – *Comparaison des deux algorithmes sur une scène de densité moyenne.* **a.** *Points de vue dans la scène.* **b.** *Points de vue à l'extérieur de la scène.*

exemple) augmente d'environ 20 % la place prise en mémoire par le complexe de visibilité, ce qui peut s'avérer être beaucoup.

Ainsi, le complexe de visibilité de la scène de 600 triangles et de densité élevée, a une taille m proche du million de sommets. Il prend sur une station de travail une place mémoire d'environ 80 Mo, et d'environ 100 Mo avec des arbres de recherche. Cette place mémoire est déjà importante pour une petite scène. Pour des scènes plus complexes, la place mémoire occupée peut devenir une vraie préoccupation. Dans ce cas, un algorithme un peu moins efficace, mais tenant dans la mémoire de la machine peut être préféré si on veut faire tourner le programme sur une machine usuelle (PC par exemple) et non seulement sur des stations de calculs très hautes performances.

3.2 Calcul du polygone de visibilité d'un segment

Nous montrons dans cette section comment calculer le polygone de visibilité d'un segment. Ce polygone délimite les parties éclairées de la scène quand le segment est considéré comme un néon illuminant la scène.

Nous décrivons d'abord un algorithme de calcul du polygone de visibilité. Nous étudions ensuite la complexité de l'algorithme et de la taille du polygone même. Nous montrons enfin que l'algorithme peut aussi servir à calculer toutes les discontinuités d'éclairage dues au néon en temps quasi-optimal.

Nous supposons dans un premier temps que la ligne support du segment est située totalement en dehors de l'enveloppe convexe de la scène. Nous montrons alors comment l'algorithme et sa complexité sont modifiés lorsque le segment est situé dans un endroit quelconque de la scène.

3.2.1 Structure du polygone de visibilité

Soit un segment $s = [p, q]$, dont la droite support est située en dehors de l'enveloppe convexe de la scène. Nous considérons que s est une source lumineuse étendue (un néon par exemple) éclairant le demi-plan contenant la perpendiculaire directe du segment $[p, q]$. Le polygone de visibilité du segment s représente la partie de la scène éclairée par s, et les parties situées en dehors du polygone de visibilité sont les zones d'ombres, qui ne reçoivent aucune lumière de s (figure 3.14a).

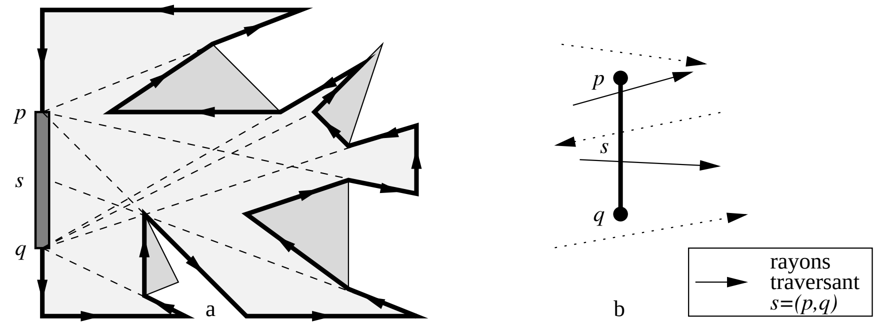


FIG. 3.14 – *Éclairage de la scène par un néon.* **a.** *Polygone de visibilité du segment* $s = (p, q)$. **b.** *Rayons traversant* s.

Un point de la scène est éclairé s'il existe un rayon partant de s et passant par ce point. La zone éclairée de la scène est donc celle couverte par l'ensemble des rayons libres maximaux passant par s. Plus précisément, nous appelons *rayon passant par s* un segment libre maximal de la scène (s n'étant pas considéré comme faisant partie de la scène) traversant s depuis son côté non éclairant vers son côté éclairant (figure 3.14b).

Dans l'espace dual, l'ensemble des rayons éclairant la scène est représenté par l'ensemble des faces (ici externes) du complexe comprises entre les courbes duales p^* et q^* associées aux extrémités de s. Ces faces, dont une partie des rayons associés traversent s, sont dites appartenant à la *zone de s*.

Examinons plus en détails la structure du polygone de visibilité. Comme pour le polygone de visibilité d'un point, les côtés du polygone de visibilité d'un segment peuvent être classés en deux catégories : côtés transversaux et côtés radiaux.

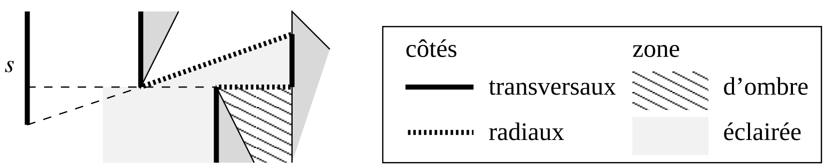


FIG. 3.15 – *Détail de la nature des côtés du polygone de visibilité de s.*

Les côtés transversaux du polygone sont les bords d'obstacle directement éclairés par le segment s. Les côtés radiaux relient les extrémités des côtés transversaux et délimitent les zones d'ombre (figure 3.15). De plus, les côtés successifs du polygone sont alternativement radiaux ou transversaux, les côtés radiaux pouvant parfois avoir leur deux extrémités confondues.

Pour calculer ce polygone, nous parcourons ses côtés transversaux successifs un par un, dans l'ordre où ils apparaissent dans sa frontière. Par exemple, les côtés du polygone de visibilité de la figure 3.14(a) seront parcourus dans l'ordre indiqué par les flèches. Ce parcours revient à visiter les faces du complexe appartenant à la zone du segment s dans un certain ordre.

3.2.2 Calcul du polygone de visibilité

Supposons que nous soyons en train de parcourir un côté transversal du polygone de visibilité porté par l'objet $O = [O_p, O_p^+]$ de la scène, en considérant qu'il est éclairé par les rayons de la face f du complexe de visibilité traversant s.

Si l'extrémité O_p de O n'est pas visible depuis s, alors la limite de la partie éclairée de O est donnée par l'intersection de O et du rayon d de pente maximale parmi les rayons de f traversant s. Ce rayon est « bloqué en haut » par l'extrémité O'_p d'un objet O' et « bloqué en bas » soit par l'extrémité q de s (figure 3.16a), soit par un objet de la scène (figure 3.16b).

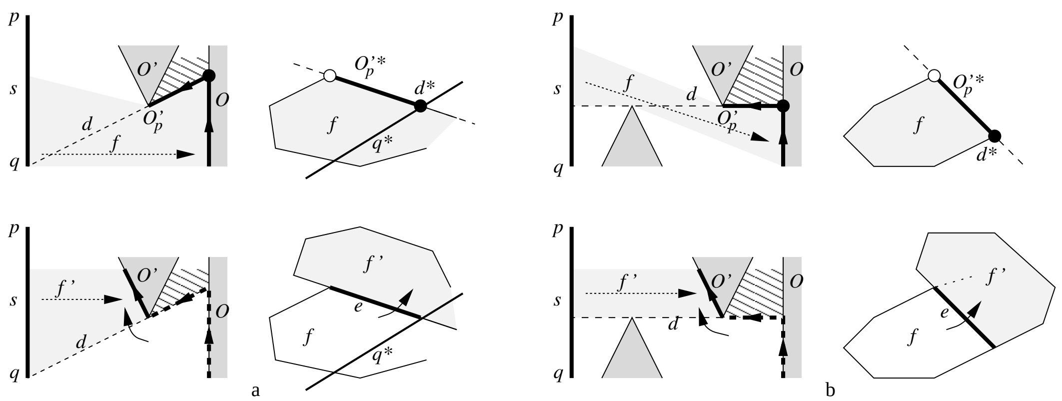


FIG. 3.16 – *Limite d'éclairage sur un côté transversal quand son extrémité n'est pas éclairée.* **a.** *Éclairage limité par q.* **b** *Éclairage limité par un objet de la scène.*

Dans le complexe de visibilité, nous avons deux configurations correspondantes. Soit la courbe duale q^* coupe la face f en une arête e non grasse de la chaîne supérieure de f, auquel cas nous notons d^* le sommet intersection entre l'arête e et q^*. Soit, dans le cas contraire, la dernière arête e de la chaîne supérieure de f n'est pas une arête grasse, et nous notons alors d^* le sommet droit de cette arête. Alors, l'extrémité de la partie éclairée de O est l'intersection de O et du rayon d associé au sommet d^*. Ce rayon passe par O'_p, sommet associé à la courbe duale support de l'arête e. De plus, en notant f' la face incidente gauche-haute de e, le prochain côté transversal du polygone de visibilité est porté par O', objet droit de l'étiquette (O_∞, O') de f', et ce côté est éclairé par les rayons de f' passant par s.

Si l'extrémité O_p de O est visible depuis s, alors l'extrémité de la partie éclairée de

O est O_p. Pour trouver le prochain segment transversal du polygone de visibilité, nous considérons le rayon de pente minimale passant par O_p, traversant s et appartenant à f.

Si le rayon est de type ν-droit en O_p, c'est-à-dire si le rayon est stoppé en O_p par le polygone dont O_p est un sommet, alors le côté suivant du polygone de visibilité est porté par O', côté précédent de O du polygone. Le côté radial est dans ce cas dégénéré : ses deux extrémités sont confondues (et égales à O_p). Le rayon peut être bloqué en bas soit par q (figure 3.17a), soit par un objet de la scène (figure 3.17b).

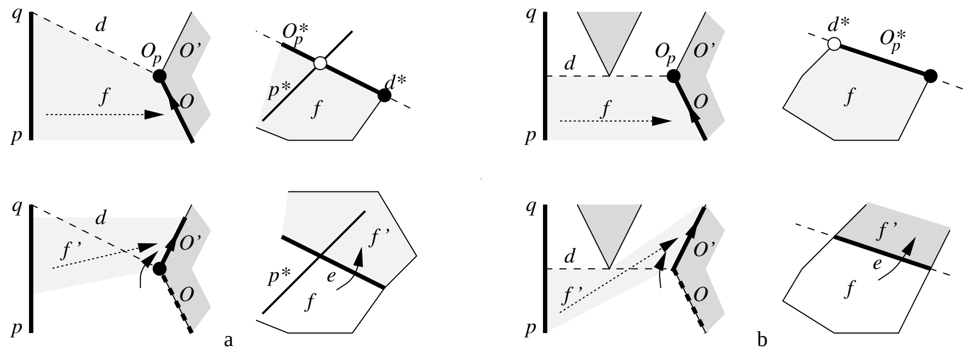


FIG. 3.17 – *Situation où le côté radial du polygone de visibilité est dégénéré.* **a.** *Côté bloqué par q.* **b.** *Côté bloqué par un objet de la scène.*

Dans le complexe de visibilité, la dernière arête e de la chaîne supérieure est de type ν-droit. Le rayon d associé au sommet droit d^* de e est considéré comme support du côté radial dégénéré du polygone de visibilité. En notant f' la face incidente gauche-haute de e, le prochain côté transversal du polygone de visibilité est O', objet droit de l'étiquette (O_∞, O') de f' (et aussi côté précédent de O sur le polygone), éclairé par les rayons de f' passant par s.

Enfin, si le rayon est de type λ-convexe, alors O crée derrière lui une zone d'ombre dont la limite est donnée par le rayon de pente minimale passant par O_p, appartenant à f et traversant s. Ce rayon est bloqué soit par q (figure 3.18a), soit par un objet de la scène (figure 3.18b). Ce rayon est stoppé par un objet O' qui porte le prochain côté du polygone de visibilité et la première extrémité de ce nouveau côté est l'intersection entre O' et le rayon.

Dans le complexe de visibilité, la dernière arête de la chaîne supérieure de f est une arête grasse dont la première sous-arête est de type λ-convexe. Si p^* coupe une sous-arête de cette arête grasse, soit e cette sous-arête et soit d^* le sommet intersection entre e et p^* ; sinon, soit e la première sous-arête de cette arête grasse et soit d^* son sommet gauche. En notant f' la face incidente gauche-haute de e, le prochain côté transversal du polygone de visibilité est porté par O', objet droit de l'étiquette de f', et son extrémité droite est le point d'intersection entre O' et d, le rayon associé au sommet d^*.

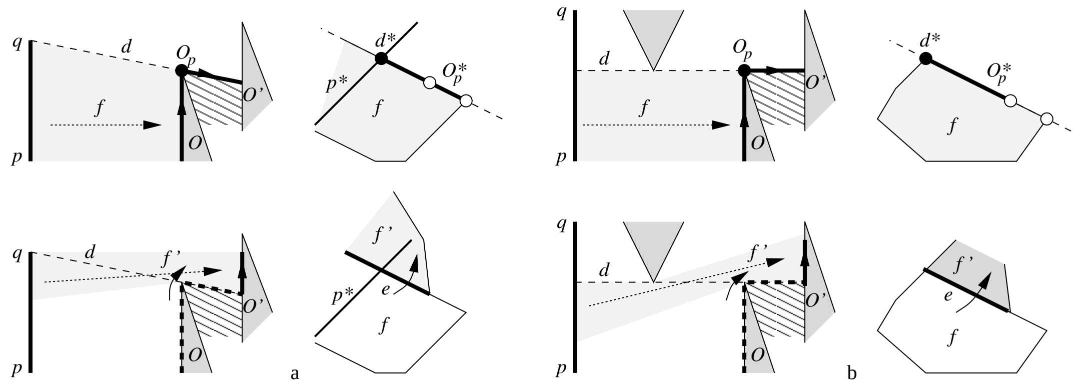


FIG. 3.18 – *Zone d'ombre délimitée par un objet dont son extrémité est éclairée.* **a.** *Éclairage limité par q.* **b.** *Éclairage limité par un objet de la scène.*

La méthode que nous venons de décrire permet de passer d'un côté transversal à un autre. Nous montrons dans la partie suivante la correction de l'algorithme et étudions sa complexité. Nous résumons maintenant l'algorithme de façon complète, en indiquant comment synchroniser le calcul des vues de p et q avec le parcours des faces du complexe. La première face f_d (resp. dernière face f_a) examinée par l'algorithme est la face du complexe de visibilité contenant le rayon (p,q) (resp. (q,p)). Comme nous avons supposé que la droite support de s était en dehors de l'enveloppe convexe de la scène, f_d (resp. f_a) est la face semi-infinie inférieure (resp. supérieure) du complexe de visibilité. L'algorithme complet s'écrit alors :

$f = f_d$
e_p=première arête du complexe coupée par p^*
e_q=première arête du complexe coupée par q^*
tant que $f \neq f_a$
  si e_p est incidente à f **et** est une arête grasse
    $e = e_p$
    si e est de type ν-droit
      $v =$ sommet droit de e
    sinon
      $v = e \cap p^*$
  sinon
  si e_q est incidente à f **et** n'est pas une arête grasse
    $e = e_q$
    $v = e \cap q^*$
  sinon
  si la dernière arête de la chaîne supérieure de f est une arête grasse
    $e =$ première sous-arête de l'arête grasse
    si e est de type ν-droit
      $v =$ sommet droit de e
    sinon
      $v =$ sommet gauche de e

sinon
  $e =$ dernière arête de la chaîne supérieure de f
  $v =$ sommet droit de e
le côté radial est porté par la droite associée à v
si e_p est incidente à f
  $e_p =$ prochaine arête coupée par p^*
si e_q est incidente à f
  $e_q =$ prochaine arête coupée par q^*
$f =$face incidente gauche-haute de e

3.2.3 Correction et complexité de l'algorithme

L'algorithme que nous venons de définir parcourt les faces du complexe appartenant à la zone de s selon un certain ordre : à une face f de la zone de s, nous associons une unique face f', que nous appelons *face suivante de f selon s*. Un examen des relations d'incidence entre f' et f montre aussi que pour toute face f', il existe une unique face f telle que f' soit la face suivante de f selon s ; f est alors dite *face précédente de f' selon s*.

Toute face du complexe comprise dans la zone de s et différente de f_a (resp. f_d) a une unique face suivante (resp. précédente) selon s. Cela montre que si nous partons de f_d et visitons à chaque fois la face suivante selon s, alors nous arrivons à f_a en ayant traversé chaque face du complexe comprise dans la zone de s une fois et une seule. L'algorithme est donc correct.

Considérons l'ensemble des arêtes e traversées par l'algorithme lors du parcours des face de la zone de s, c'est-à-dire les arêtes séparant une face de la zone de s de sa face suivante selon s. Pour chacune de ces arêtes, nous avons retenu un sommet d^* qui est soit une extrémité propre de l'arête, soit une intersection entre l'arête et une des deux courbes duales p^* ou q^*. Ces sommets permettent de définir dans la scène les côtés radiaux du polygone de visibilité de s. Chacun de ces sommets n'est visité qu'une fois par l'algorithme.

Dans la scène, les rayons correspondant à ces sommets font soit partie de la vue de p (resp. q), soit partie de l'ensemble des arêtes du graphe de visibilité qui (une fois étendues) traversent s. En notant m_s (resp. m_p, resp. m_q) le nombre d'arêtes du graphe de visibilité traversant s (resp. la taille de la vue autour de p, resp. q), alors la taille $Visi(s)$ du polygone de visibilité vérifie $Visi(s) = O(m_s + m_p + m_q)$.

Cependant, la taille du polygone peut être plus petite. La figure 3.19 montre des situations où des arêtes de la vue de p (resp. des arêtes du graphe de visibilité coupant s) dont les sommets correspondant dans le complexe ne sont incidents à aucune arête traversée par l'algorithme.

Précisons maintenant la complexité de l'algorithme. En associant pour chaque face un pointeur sur la première sous-arête de l'arête grasse droite de sa chaîne supérieure, le passage d'une face à la face suivante se fait en temps constant si l'arête de passage e n'est coupée ni par p^* ni par q^*.

Pour rester dans la zone de s, nous devons rester entre les courbes duales p^* et q^*

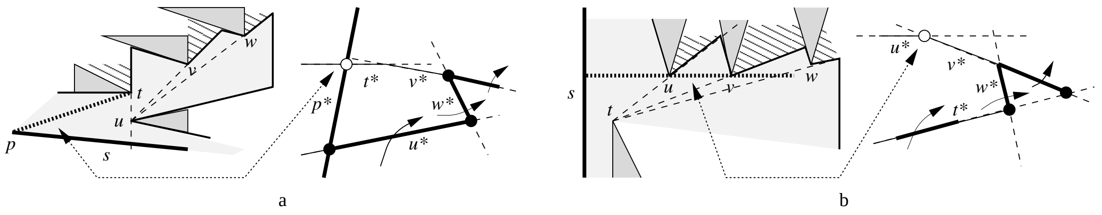


FIG. 3.19 – *Arêtes non visitées dans le calcul du polygone de visibilité.* **a.** *Arête de la vue de p.* **b.** *Arête du graphe de visibilité.*

qui peuvent restreindre les faces visitées. Pour cela, nous calculons au fur et à mesure la vue autour de ces deux points, c'est-à-dire les arêtes du complexes coupées par les courbes duales. Le temps total de calcul de ces faces successives est donc le temps de calcul des deux vues autour de ces points.

Comme la droite support de s est située en dehors de l'enveloppe convexe de la scène, la face f_d de départ (resp. f_a d'arrivée) du parcours de faces est en fait la face semi-infinie délimitant le bas (resp. le haut) du complexe de visibilité, et est accessible directement.

Selon la méthode utilisée pour calculer la vue autour de p et q, le calcul du polygone de visibilité du segment s se fait donc en temps $\Omega(Visi(s)_s + \log n + m_p + m_q)$ et $O(Visi(s)_s + (m_p + m_q)n)$ (méthode de parcours), ou en temps $O(Visi(s)_s + (m_p + m_q)\log n)$, où $Visi(s)_s$ désigne le nombre de côtés radiaux du polygone de visibilité portés par une arête du graphe de visibilité. Avec la première méthode, la borne optimale $O(\log n + Visi(s))$ peut être atteinte dans certains cas.

3.2.4 Limites de discontinuité de l'éclairage

Les côtés radiaux du polygone de visibilité calculés correspondent aux limites d'ombre portées par les objets. Cependant, ces limites d'ombre ne sont que locales. Pour un point O_p, différentes faces f peuvent avoir une arête portée par la courbe duale O_p^* dans leur chaîne supérieure, et plusieurs de ces arêtes peuvent être traversées pendant le parcours des faces de la zone de s. Le polygone de visibilité n'est donc pas forcément un polygone simple et les limites d'ombre indiquées peuvent n'être alors que des limites de pénombre.

La figure 3.20 nous montre un exemple de scène illuminée par le segment s. Le polygone de visibilité correspondant n'est pas un polygone simple (figure 3.20(a), le sens des flèches et la numérotation indiquent le sens de parcours des côtés du polygone), et le parcours des faces dans le complexe (figure 3.20b) nous montre effectivement que plusieurs limites d'ombres locales sont considérées au voisinage d'un même point (points 3,4,5,6 de la figure 3.20c).

Pour avoir la vraie limite d'ombre derrière un objet O, il faut prendre la limite de plus grande pente (ou de plus petite pente selon le cas) parmi les ≪ limites locales d'ombres ≫ (figure 3.20c). La structure de polygone est alors perdue.

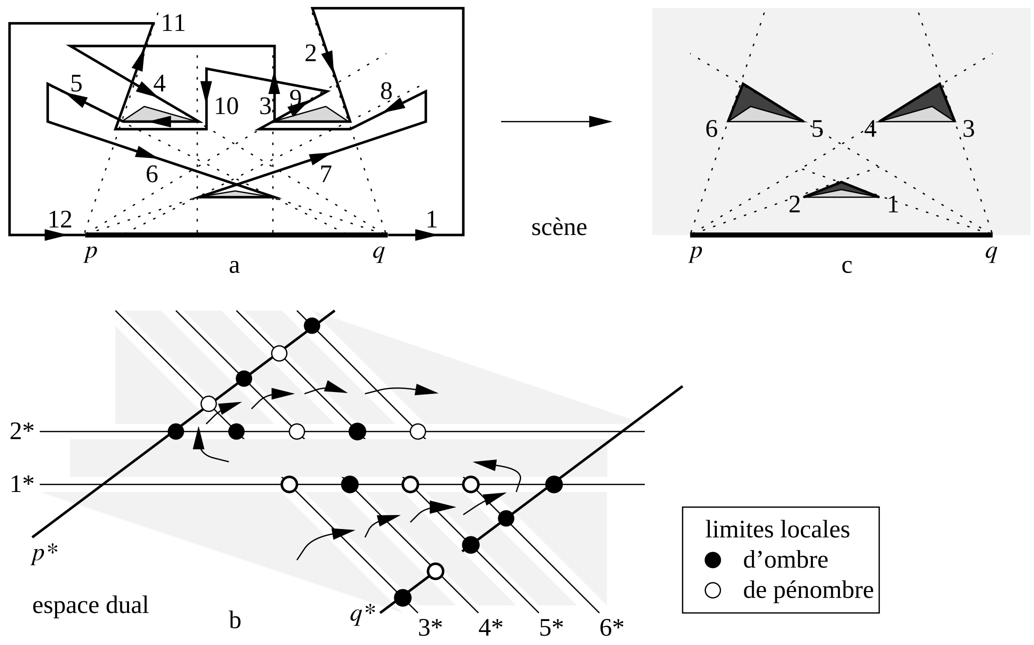


FIG. 3.20 – **a.** *Polygone de visibilité non simple de la scène.* **b.** *Parcours correspondant des faces du complexe.* **c.** *Limites d'ombre résultantes dans la scène.*

Pour certaines applications (calcul d'illumination globale par exemple), il est intéressant d'avoir en plus des limites d'ombre les limites de pénombre. Nous venons de voir que les limites d'ombre calculées ne sont que locales et peuvent n'être alors que des limites de pénombre. Cependant, il existe aussi des « vraies » limites de pénombre : ces limites séparent deux zones qui sont toutes deux (localement) éclairées, mais dont la partie de s qui les éclaire est différente.

D'une manière générale, les limites d'éclairage (ombre et pénombre) sont données par les arêtes qui relient p (resp. q) aux points de sa vue et par les arêtes (étendues) du graphe de visibilité qui coupent le segment s. Notre algorithme calcule déjà la vue autour de p et q. Nous devons donc visiter en plus toutes les arêtes du graphe coupant s et non plus seulement celles qui portent un côté du polygone de visibilité. Nous vérifions facilement que les sommets du complexe associés à ces arêtes sont toujours incidents à une chaîne supérieure d'arêtes d'une face de la zone de s.

Comme notre algorithme parcourt chaque face f du complexe de la zone de s, il suffit de visiter tous les sommets extrémité des arêtes de la chaîne supérieure de f. Ainsi, chaque limite de zone ou de pénombre est reportée, et l'est au plus deux fois. Cela permet de calculer l'éclairage de la scène en temps optimal $O(\log n + limvisi(s))$, où $limvisi(s)$ est le nombre de limites d'ombre et de pénombre.

3.2.5 Segment compris dans la scène

Nous avons supposé que la droite support de s était en dehors de l'enveloppe convexe de la scène. Ainsi tous les rayons éclairant la scène provenaient en fait de l'objet

O_∞ situé à l'infini. De cette façon, aucun événement de visibilité ne se produisait derrière s. Si s est situé n'importe où dans (l'espace libre de) la scène, alors les faces traversées peuvent être délimitées par des changements de visibilité situés derrière s. Ces changements ne modifient pas le polygone de visibilité de s (s n'éclaire que d'un côté) mais peuvent compliquer l'algorithme et augmenter le temps de calcul.

Examinons d'abord le cas où le segment éclairant s est situé hors de l'enveloppe convexe de la scène, sans que sa droite support ait nécessairement une intersection vide avec l'enveloppe convexe. Dans ce cas, des objets de la scène peuvent être situés « derrière » s, c'est-à-dire dans la partie non éclairée, et des changements de visibilité dans la scène peuvent alors survenir « derrière » s. Mais dans ces situations, le devant de s est vide : les rayons correspondant à ces changement de visibilité traversent s pour se perdre ensuite vers O_∞. Nous pouvons alors restreindre le balayage du polygone de visibilité aux directions où les objets de la scène sont situés devant s.

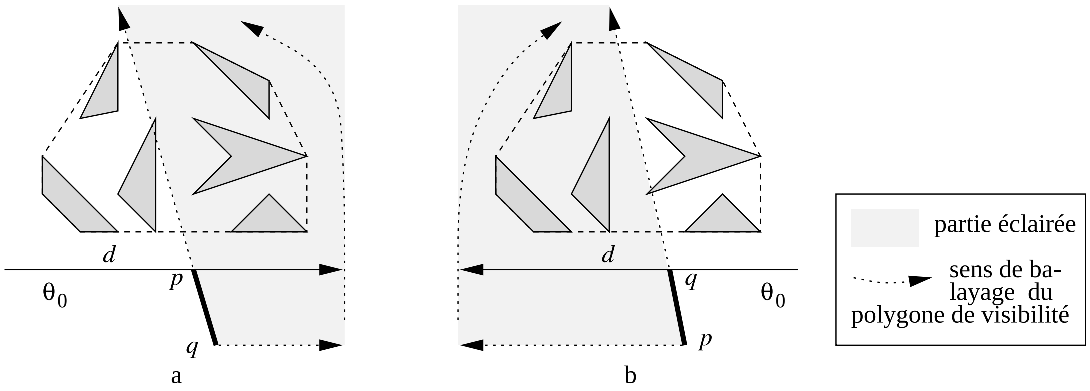


FIG. 3.21 – *Calcul du polygone de visibilité quand le segment est en dehors de l'enveloppe convexe de la scène.* **a.** *Segment en dessous.* **b.** *Segment au-dessus.*

Comme le segment s est situé en dehors de l'enveloppe convexe de la scène, il existe une droite passant par une des extrémité de s et séparant s de la scène. Si cette droite passe par p, alors les rayons de même direction que la droite et passant par s appartiennent à la la face semi-infinie inférieure du complexe. Nous prenons alors cette face comme face de départ f_d dans l'algorithme de calcul de polygone de visibilité (figure 3.21a). La face d'arrivée f_a de l'algorithme, face contenant le rayon (p, q), n'est plus la face semi-infinie supérieure du complexe. Nous rajoutons alors dans l'algorithme un test pour vérifier si la face en train d'être parcourue contient le rayon (p, q). Ce test est effectuable en temps constant une fois connues les arêtes de sorties de p^* et q^*.

Si la droite séparatrice passe par q, alors les rayons de même direction que la droite et passant par s sont dans la face semi-infinie supérieure du complexe (figure 3.21b). Pour simplifier les calculs, nous utilisons l'algorithme qui parcourt les faces de la zone de s en sens inverse de celui de l'algorithme que nous avons détaillé (les deux algorithmes sont similaires). Nous prenons alors la face semi-infinie supérieure comme face de départ et nous rajoutons un test pour vérifier si la face traversée n'est pas la face contenant le rayon (q, p).

Dans le cas où le segment s est en position quelconque en dehors de l'enveloppe convexe de la scène, les modifications apportées à l'algorithme de calcul de polygone de visibilité sont mineures : seules les faces de départ et d'arrivée sont modifiées. Il n'en va pas de même si le segment est situé à l'intérieur de la scène : lors du balayage, nous rencontrons des situations où des objets sont situés de part et d'autre du segment s, et où des changements de visibilité se produisent derrière s et ne doivent pas être pris en compte dans le calcul du polygone de visibilité.

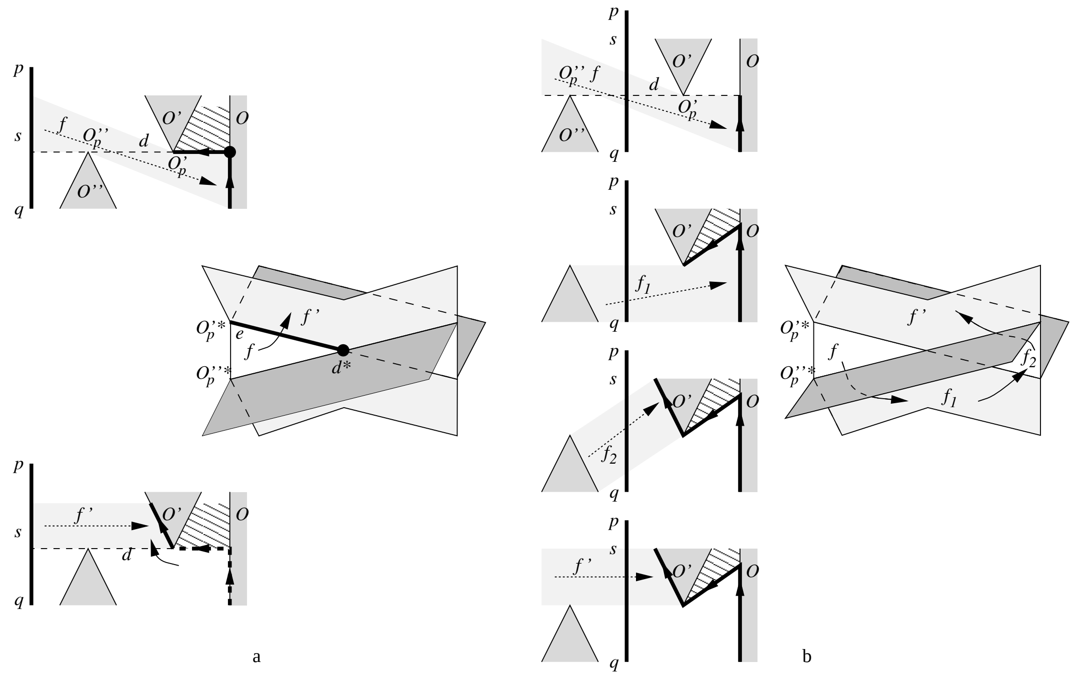


FIG. 3.22 – *Détermination de la zone d'ombre créée par le bas d'un objet.* **a.** *Changement de visibilité devant le néon.* **b.** *Changement de visibilité derrière le néon.*

Ainsi, la situation de la figure 3.16(b) où nous cherchions la limite supérieure d'éclairage sur un segment n'est plus qu'un cas particulier de la situation générale où nous cherchons la limite de l'ombre créée par le bas de l'objet O' sur l'objet O (figure 3.22). S'il y a un objet O'' entre le néon s et O', alors O'' cache le néon et la limite d'ombre est donnée par le rayon passant par (O''_p, O'_p), extrémités respectives de O'' et O' (figure 3.22a). Nous passons alors de la face f qui éclairait l'objet O à la face f' qui éclaire O' en considérant la dernière arête e de la chaîne supérieure de f : f' est la face incidente gauche-haute de e.

Si maintenant le segment s est situé entre O'' et O', bien que le complexe de visibilité de la scène soit toujours le même, nous devons agir autrement. En effet, O'' ne cache plus le néon et nous devons passer de f à f_1 pour chercher la limite d'ombre, où cette fois-ci f_1 est la face incidente droite-basse de la dernière arête de la chaîne inférieure de f. La figure 3.22(b) montre les faces alors parcourues dans le complexe avant de traverser f'.

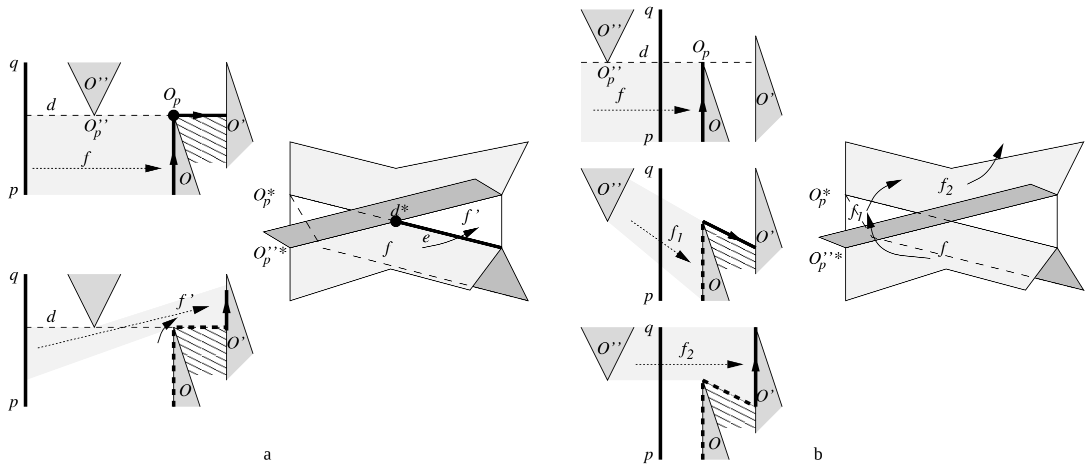


FIG. 3.23 – *Détermination de la zone d'ombre créée par le haut d'un objet.* **a.** *Changement de visibilité devant le néon.* **b.** *Changement de visibilité derrière le néon.*

Les mêmes précautions doivent être prises avec la situation de la figure 3.18(b) où nous cherchions la limite inférieure d'éclairage d'un segment : elle n'est plus qu'un cas particulier de la situation générale où nous cherchons la limite de l'ombre créée par le haut de l'objet O sur l'objet O' (figure 3.23). S'il y a un objet O'' entre le néon et O, alors O'' cache le néon et la limite d'ombre est donnée par le rayon passant par (O''_p, O_p), extrémités respectives de O'' et O (figure 3.23a). Nous passons alors de la face f qui éclairait l'objet O à la face f' qui éclaire O' en considérant la première sous-arête e de l'arête grasse droite de la chaîne supérieure de f : f' est la face incidente gauche-haute de e.

Si maintenant le segment s est situé entre O'' et O', comme pour la situation décrite précédemment, nous devons aussi agir autrement. O'' ne cache plus le néon et nous devons passer de f à f_1 pour chercher la limite d'ombre, où cette fois-ci f_1 est la face incidente droite-haute de l'arête précédent la première sous-arête grasse droite de la chaîne inférieure de f. La figure 3.23(b) montre les faces alors parcourues dans le complexe. Nous remarquons aussi que dans ce cas la face f' n'est pas parcourue.

Ceci est du au fait que les propriétés du chemin de parcours des faces de la zone de s étudiées quand s était en dehors de la scène ne sont plus valables. En particulier, la figure 3.24 montre un cas où deux faces f_1 et f_2 ont la même face suivante selon s. L'algorithme étendu de calcul de polygone de visibilité ne parcourt donc plus toutes les faces de la zone de s. Cependant les faces oubliées n'ont pas d'incidence sur le calcul du polygone de visibilité. La figure 3.23 permet de voir que la face f' de la première situation (figure 3.23a) est coupée en deux faces dans la deuxième situation (figure 3.23b) : les faces f' (similaire à la première situation) et f_2. Ce découpage est du à un obstacle O'' situé derrière le néon, obstacle qui n'a aucune influence sur l'éclairage de la scène : il ne détermine aucune limite d'ombre ou de pénombre.

La figure 3.24 montre aussi que f_2 n'a pas vraiment de face précédente selon s. L'algorithme modifié de calcul de polygone de visibilité parcourt correctement toutes les faces utiles au calcul d'éclairage de la scène et ne se perd pas dans des faces inutiles.

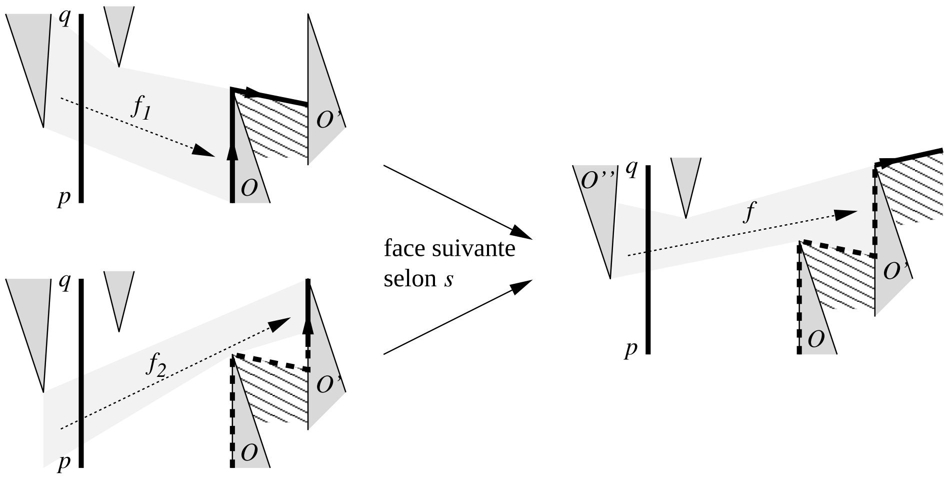


FIG. 3.24 – *Exemple où deux faces ont la même face suivante selon s.*

Il faut simplement calculer la face f_d de départ contenant le rayon (q, p). Cette face doit cette fois-ci réellement calculée : ce n'est plus une des deux faces semi-infinies du complexe. Elle peut cependant être trouvée simplement en calculant la vue autour de p et q.

L'algorithme de calcul du polygone de visibilité d'un segment s se complique fortement quand s est situé dans la scène : quand un changement de visibilité se produit, il faut vérifier s'il se produit dans la partie éclairée de la scène et agir en conséquence sinon. La complexité de l'algorithme s'en ressent. Elle est toujours en $O(m_s + (m_p + m_q) \log n)$, où m_s est toujours le nombre d'arêtes (étendues) du graphe de visibilité coupant s, mais parmi les arêtes coupant s seules celles situées dans la partie éclairée de la scène interviennent dans les limites d'éclairage.

L'algorithme perd aussi la propriété de parcours des faces de la zone de s : toutes les faces de la zone ne sont plus parcourues. Il faut alors calculer le polygone de visibilité de $[p, q]$ et de $[q, p]$ pour visiter toutes les faces de la zone.

Chapitre 4

Calculs de visibilité dynamique

Dans ce chapitre nous montrons comment maintenir la vue autour d'un point se déplaçant dans la scène. Nous donnons deux algorithmes qui ont chacun leur spécificité : le premier s'applique plutôt pour des petits déplacements successifs, le deuxième pour un long déplacement le long d'une trajectoire connue à l'avance.

Nous montrons ensuite comment maintenir le complexe de visibilité quand les objets se déplacent dans la scène et comment maintenir la vue autour de points quand les points de vue et les objets de la scène sont tous en mouvement.

— *Arrêtez-vous Lagaffe !! On ne voit rien !!*
— *Ben ! On n'y verra pas plus si on s'arrête, malin !...*
Un gaffeur sachant gaffer.

4.1 Maintien d'une vue autour d'un point mobile

Nous étudions dans cette section comment, après avoir calculé une vue autour d'un point p_v, nous pouvons maintenir cette vue quand le point de vue se déplace dans la scène.

4.1.1 Changements de visibilité

Soit une scène de polygones. Si nous étendons dans l'espace libre de cette scène les arêtes de son graphe de visibilité, nous obtenons une partition de la scène en régions de visibilité constante : tous les points situés dans une même région ont la même vue. (figure 4.1).

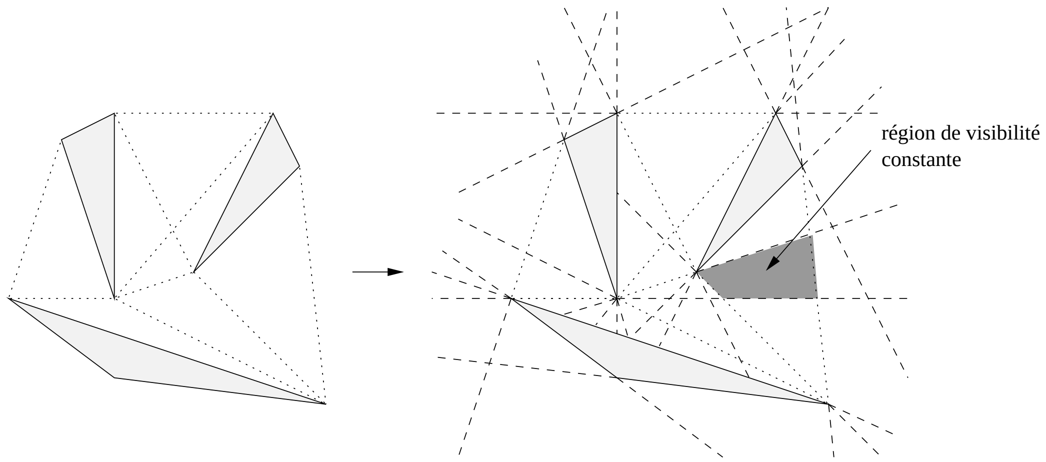


FIG. 4.1 – *Découpage en régions de visibilité constante.*

Plus précisément, nous disons que deux points p_v et p'_v ont une même vue si la structure combinatoire de leur polygone de visibilité est la même, c'est-à-dire si les côtés transversaux d'un polygone de visibilité et ceux correspondants de l'autre polygone de visibilité sont portés par les mêmes côtés d'obstacle, et si aux côtés radiaux du type (p_v, p) de l'un correspondent les côtés radiaux (p'_v, p) de l'autre ; de plus les extrémités des côtés sont donnés par les mêmes sommets de la scène ou par les points d'intersection avec les mêmes droite correspondantes. Seules les données numériques (coordonnées des sommets) peuvent être différentes, mais elles sont déduites automatiquement de la structure combinatoire du polygone de visibilité.

Même si la vue autour d'un point ne change pas réellement quand celui-ci traverse une arête non étendue du graphe de visibilité, de tels changements sont intéressants (ils peuvent signaler le passage d'une porte par exemple) et surtout ils correspondent à un changement de situation dans l'espace dual. Ces changements seront appelés aussi abusivement changements de visibilité.

L'idée générale derrière les algorithmes de maintien de vue que nous exposons, est de calculer les frontières de la région de visibilité où se trouve le point, de calculer le côté de sortie de cette région quand le point se déplace et de mettre à jour les frontières de la nouvelle région de visibilité.

Le complexe de visibilité permet d'accéder directement au découpage induit par les arêtes du graphe de visibilité. Dans le chapitre précédent, nous calculons une vue en cherchant la liste des arêtes du complexe traversées par la courbe duale p_v^* du point de vue p_v. Soit e une des arêtes traversées et soit p le point (visible depuis p_v) de la scène correspondant. Les deux sommets v_g et v_d délimitant cette arête déterminent deux arêtes du graphe de visibilité incidentes à p dont les droites support v_g^* et v_d^* encadrent (p_v, p).

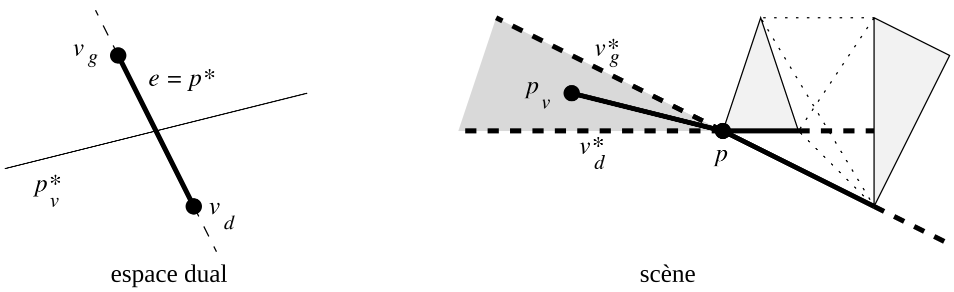


FIG. 4.2 – *Limites de changement de visibilité.*

Ces arêtes, une fois étendues, déterminent deux limites de changement de visibilité pour p_v (figure 4.2) : si le point de vue sort du coin délimité par v_g^* et v_d^*, alors sa vue change. Le coin délimité par v_g^* et v_d^* est en fait l'intersection des deux demi-plans délimités par v_g^* (resp. v_d^*) et contenant p_v. À chaque arête du complexe traversée par la courbe duale p_v sont alors associés deux demi-plans, et la région de visibilité constante associée au point de vue p_v est constituée par l'intersection de tous ces demi-plans.

De façon plus détaillée, considérons pour chaque point p de la vue l'arête e du complexe traversée par p_v^* et associée à p. Chacune des deux droites associées aux sommets délimitant e peut être classée inférieure ou supérieure : si p_v est au dessus de la droite v^*, alors la droite est dite inférieure ; si p_v est en dessous de la droite v^*, alors la droite est dite supérieure. Nous obtenons alors deux enveloppes de droites qui délimitent la région de visibilité : l'enveloppe inférieure, composée des droites inférieures, et l'enveloppe supérieure, composée des droites supérieures.

Dans l'espace dual, si le sommet du complexe v correspondant à la droite v^* est situé au-dessus (resp. en dessous) de la courbe duale du point de vue, alors la droite appartiennent à l'enveloppe supérieure (resp. inférieure), comme illustré par l'exemple de la figure 4.3.

Pour maintenir la vue, deux stratégies différentes peuvent être mises en œuvre, selon que l'on effectue des petits déplacement successifs, ou que l'on effectue un grand déplacement le long d'une trajectoire.

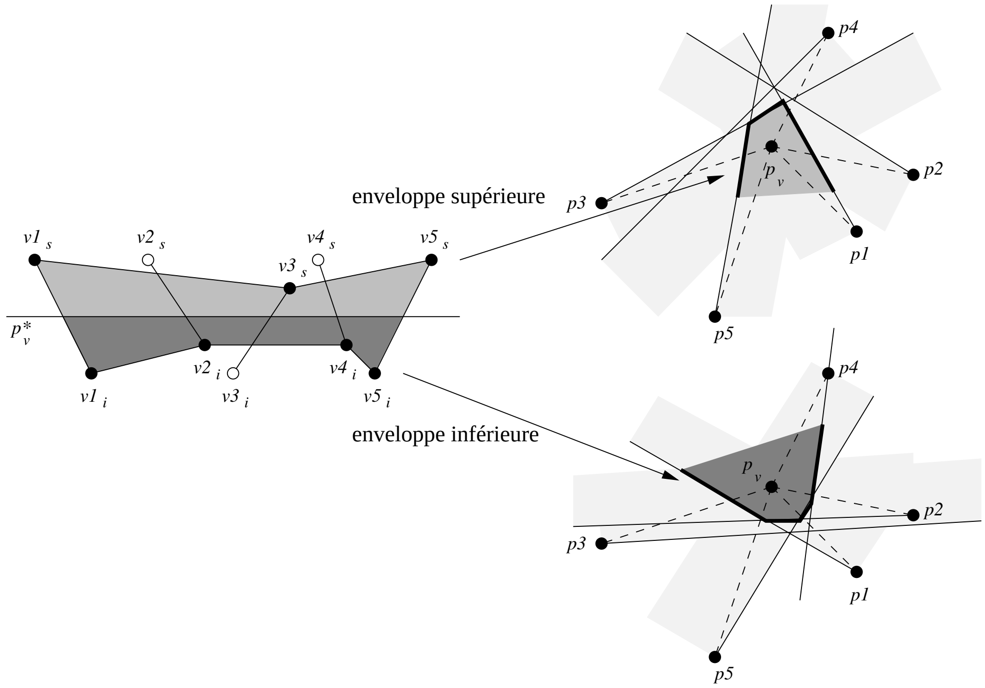


FIG. 4.3 – *Limites supérieure et inférieure de visibilité.*

4.1.2 Maintien de la vue lors d'un petit déplacement

Supposons que le point de vue se déplace en ligne droite de p_v à p'_v. Les deux enveloppes de droites délimitant la visibilité se coupent en deux points. En testant la position de p'_v par rapport à la droite passant par ces deux points (droite qui sépare en quelque sorte les deux enveloppes), nous savons par quelle enveloppe le point risque de sortir. Nous vérifions ensuite si le point de vue est vraiment sorti de cette enveloppe grâce à une recherche dichotomique utilisant la convexité de l'enveloppe.

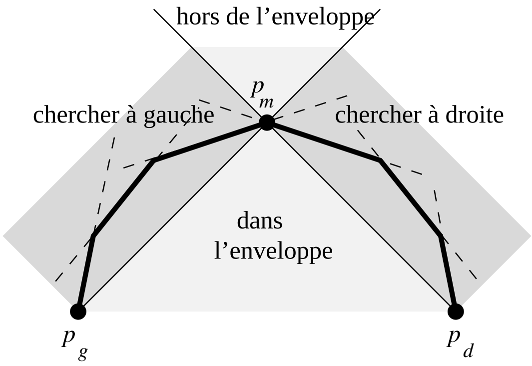


FIG. 4.4 – *Tester la sortie du point de vue de la région de visibilité.*

En effet, considérons le sommet milieu p_m de l'enveloppe et les droites passant par ce sommet et les points extrêmes p_g et p_d de l'enveloppe (figure 4.4). Selon la position de p'_v par rapport à ces deux droites, nous savons si le point est dans l'enveloppe, hors de l'enveloppe, ou s'il faut poursuivre la vérification dans la partie gauche (resp. droite) de l'enveloppe. Cette vérification prend un temps logarithmique $O(\log v)$.

Nous pouvons de plus utiliser cette vérification pour trouver le cas échéant la droite de sortie : si (p_v, p'_v) coupe le segment $[p_g, p_m]$, alors p'_v sort par la partie gauche de l'enveloppe. Si (p_v, p'_v) coupe le segment $[p_m, p_d]$, alors p'_v sort par la partie droite. Nous pouvons donc trouver la droite de sortie aussi en temps $O(\log v)$.

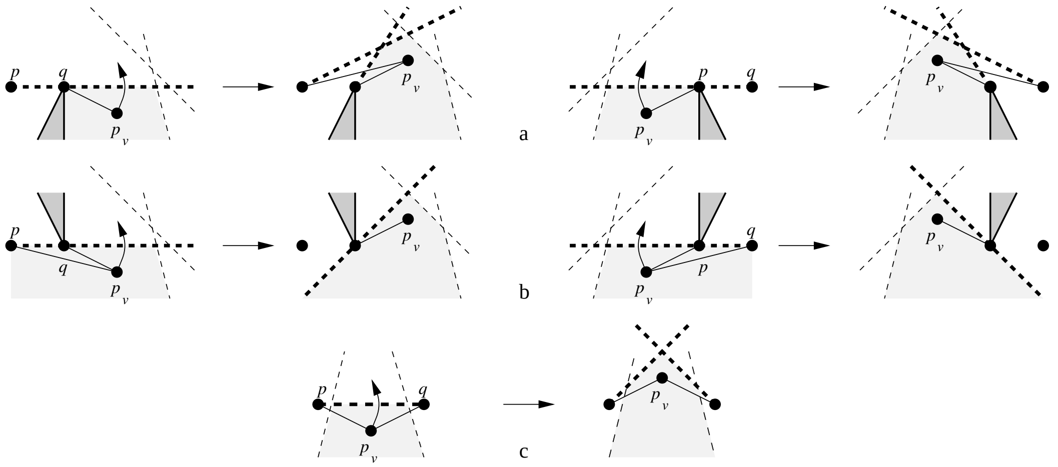


FIG. 4.5 – *Mise à jour de la vue et de l'enveloppe supérieure.*

Les modifications à effectuer pour mettre à jour la vue et l'enveloppe lors du franchissement d'une arête étendue du graphe de visibilité sont locales : si un sommet de l'arête devient visible (figure 4.5a), il est ajouté à la vue et deux nouvelles limites de visibilité sont insérées dans l'enveloppe de sortie ; si un sommet de l'arête devient invisible (figure 4.5b), il est enlevé de la vue et seule la nouvelle limite de visibilité associée à l'autre sommet de l'arête est insérée dans l'enveloppe de sortie ; enfin si le point de vue passe par l'arête même (non étendue) du graphe de visibilité (figure 4.5c), les deux sommets de l'arête restent visibles et les nouvelles limites de visibilité associées aux sommets remplacent dans l'enveloppe de sortie les anciennes limites.

Les modifications correspondantes – lors du franchissement de l'arête étendue (p, q) du graphe de visibilité – dans le complexe de visibilité concernent la liste des arêtes du complexe traversées par la courbe duale du point de vue. Ces modifications sont aussi locales. Si p (resp. q) reste visible, alors l'arête correspondante dans la liste doit être remplacée par l'arête suivante (ou précédente) portée par la même courbe duale p^* (resp. q^*) (figure 4.6c). Si p (resp. q) devient invisible, alors l'arête correspondante est enlevée de la liste (figure 4.6b). Si p (resp. q) devient visible, alors l'arête correspondante doit être ajouté dans la liste (figure 4.6a). Notons que si les deux arêtes associées aux deux points sont coupées par p_v^*, alors elles sont consécutives dans la liste d'arêtes coupées par p_v^*.

La mise à jour de la vue (et de la liste des arêtes du complexes coupées par p_v^*) se fait en temps constant grâce à l'utilisation par exemple d'une liste doublement chaînée. Le maintien de l'enveloppe de droites s'effectue en utilisant des algorithmes de maintien dynamique d'enveloppe convexe.

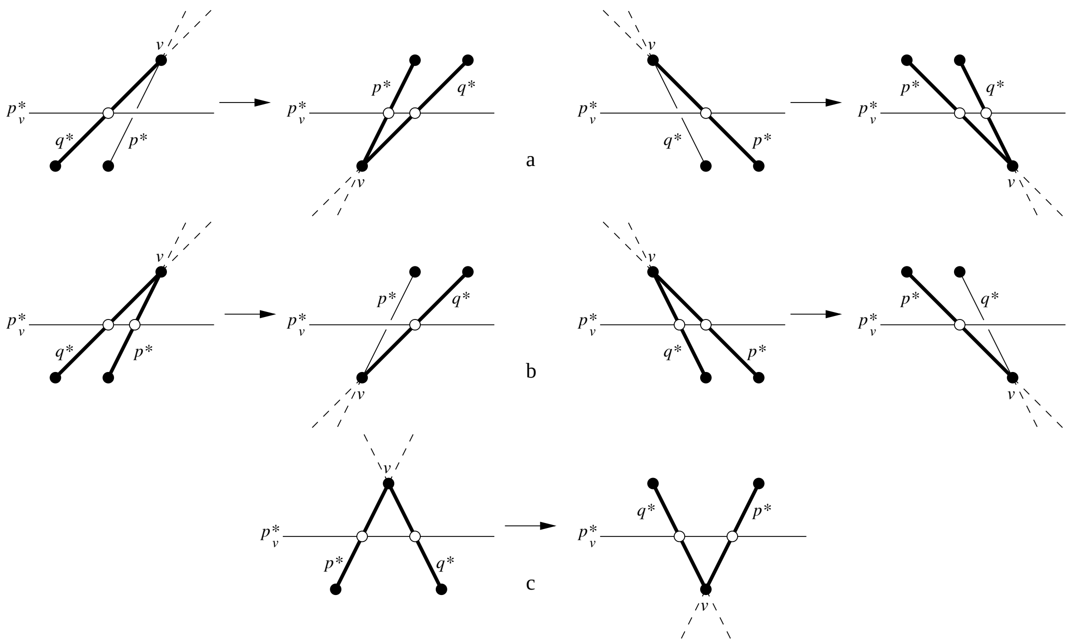


FIG. 4.6 – *Cas de mise à jour de la vue dans le complexe correspondant aux cas de la figure précédente.*

Proposition 4.1 *Après un calcul initial effectué en temps $O(v_0 \log v_0)$ (v_0 taille de la vue initiale), la vue autour d'un point peut être maintenue lors du déplacement du point de vue en temps $O(\log^2 v)$, ou en temps randomisé $O(\log v)$, à chaque changement élémentaire de visibilité, v désignant la taille de la vue à l'étape considérée.*

Démonstration. Vérifier si le point sort de l'enveloppe convexe et calculer le cas échéant l'arête de sortie se fait en temps $O(\log v)$. Le maintien dynamique de l'enveloppe s'effectue en temps $O(\log^2 v)$ en utilisant l'algorithme de maintien d'enveloppe convexe d'Overmars et van Leeuwen [OvL81], ou en temps randomisé $O(\log v)$ en utilisant l'algorithme de Dobrindt et Yvinec [DY93]. La construction initiale de l'enveloppe prend alors un temps $O(v_0 \log v_0)$, et un espace $O(v_0)$ (resp. $O(v_0 \log v_0)$ dans le cas randomisé). □

Cet algorithme est plutôt adapté à des petits déplacements, successifs, où peu de changements de visibilité interviennent et où l'important est d'avoir rapidement la vue finale. Nous avons supposé que le point de vue s'était déplacé en ligne droite. Nous pouvons, sans modification de l'algorithme, considérer d'autres mouvements pas trop ≪ tordus ≫ , c'est-à-dire qui ne faussent pas les test de localisation par rapport aux droites de l'enveloppe.

4.1.3 Maintien de la vue le long d'une trajectoire

Si nous considèrons maintenant que le problème est :

étant donnée une trajectoire sur laquelle se déplace le point de vue, calculer tous *les changements de visibilité qui interviennent le long de cette trajectoire,*

alors une autre stratégie peut être utilisée.

Nous supposons que le point de vue se déplace de p_v à p'_v sur une trajectoire telle que nous puissions savoir en temps constant si une droite coupe cette trajectoire et calculer alors (en temps constant) le point d'intersection. Nous supposons aussi que nous pouvons comparer en temps constant deux points sur cette trajectoire, c'est-à-dire savoir quel est le point situé avant l'autre sur la trajectoire.

Après avoir calculé la vue autour de p_v, nous considérons pour chaque point de la vue les deux droites associées qui délimitent les frontières de visibilité. Nous calculons les points d'intersection (s'ils existent) entre ces droites et la trajectoire du point de vue. Nous mettons ces points dans une queue de priorité en fonction de leur ordre de position sur la trajectoire et passons à chaque fois le premier point de la queue. À chaque passage d'un point d'intersection, nous mettons à jour la vue en y insérant ou supprimant un sommet. Lors de cette mise à jour, de nouvelles droites frontières de visibilité peuvent apparaître. Nous calculons alors leur intersection éventuelle avec la trajectoire et mettons à jour la liste ordonnée des points d'intersection (figure 4.7).

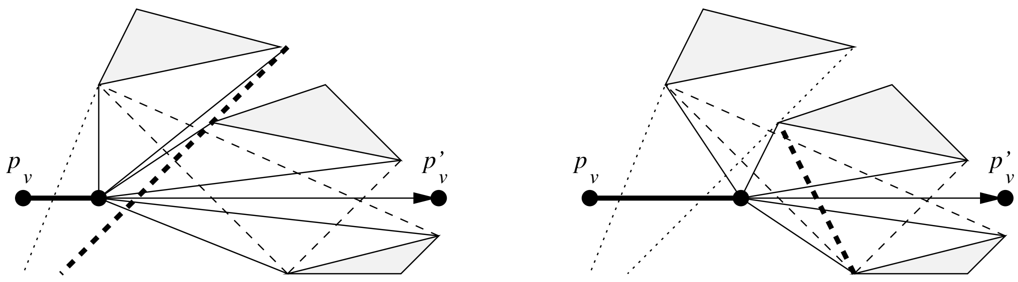


FIG. 4.7 – *Passage d'un changement de visibilité.*

Proposition 4.2 *Soit un point de vue se déplaçant suivant une trajectoire déterminée à l'avance. Après un calcul initial en $O(v_0 \log v_0)$, la vue autour de ce point peut être maintenue en temps $O(\log v)$ à chaque changement de visibilité le long de cette trajectoire. Les k changements de visibilité sont donc calculés en temps $O(v_0 \log v_0 + k \log(v_0 + \frac{k+1}{2}))$.*

Démonstration. Le maintien de la liste des points d'intersection a un coût logarithmique à chaque étape, et la vue augmente d'au plus un sommet à chaque mise à jour. □

Le premier algorithme est plutôt adapté à des petits déplacement successifs. En effet, le calcul de l'enveloppe de droites (en $O(v_0 \log v_0)$) n'est fait qu'une fois au début, et ensuite chaque changement de visibilité ne coûte qu'un temps $O(\log^2 v)$ (ou $O(\log v)$ randomisé), quels que soient les déplacements successifs. Le deuxième algorithme doit recalculer et trier (en temps $O(v \log v)$) les points d'intersection entre la trajectoire et les limites de visibilité à chaque changement de trajectoire.

En revanche, sur un long déplacement donné, le coût plus faible de mise à jour à chaque changement de visibilité du deuxième algorithme ($O(\log v)$) devient plus intéressant que celui du premier algorithme ($O(\log^2 v)$), surtout que les algorithmes de maintien dynamique d'enveloppe convexe sont sensiblement plus complexes que le simple maintien d'une queue de priorité.

4.2 Maintien dynamique du complexe de visibilité

Nous étudions dans cette section le maintien du complexe de visibilité quand les objets de la scène sont en mouvement. Nous étudions d'abord la mise à jour du complexe lors d'un changement élémentaire de visibilité, puis nous montrons comment gérer ces changements élémentaires pour maintenir le complexe.

4.2.1 Changements élémentaires de visibilité

Quand les coordonnées des sommets des polygones de la scène changent, les données numériques du complexe – coordonnées des sommets (c'est-à-dire pente des arêtes du graphe de visibilité) par exemple – sont modifiées. Cependant, les informations les plus importantes contenues dans le complexe sont les informations concernant les relations de visibilité entre les objets de la scène : une fois ces relations connues, les calculs numériques sont immédiats. Or ces informations de visibilité ne dépendent que de la topologie du complexe. Nous ne devons prendre en compte les changements dans la scène que lorsqu'ils produisent des modifications dans les relations de visibilité entre objets, modifications qui se traduisent par une modification de la topologie du complexe de visibilité.

Toute modification de la structure du complexe dans une scène dynamique se ramène à une succession de *changements élémentaires de visibilité* – changements de visibilité les plus simples – successifs, qui ne produisent qu'une modification topologique locale dans la structure du complexe.

Ces changements élémentaires de visibilité surviennent au voisinage de trois sommets p_1, p_2 et p_3 de la scène reliés successivement par les arêtes (p_1, p_2) et (p_2, p_3) du graphe de visibilité. Les changements se produisent quand, lors du déplacement, les trois points deviennent alignés puis changent de position relative : le sommet p_2 initialement au-dessus (resp. en dessous) de la droite (p_1, p_3) passe en dessous (resp. au-dessus) de cette droite (figure 4.8). La visibilité entre les points extrêmes p_1 et p_3 est alors modifiée : s'ils étaient mutuellement visibles, alors ils deviennent cachés par l'obstacle en p_2 ; s'ils étaient cachés par cet obstacle, ils deviennent alors mutuellement visibles.

La figure 4.9 montre les modifications qui interviennent dans la topologie du squelette du complexe lors des changements élémentaires de visibilité de la figure 4.8. Dans le premier cas, les courbes duales p_1^* et p_3^* qui se coupaient en projection (resp. en le sommet v_{13} du complexe) en dessous de p_2^* glissent et se coupent maintenant au-dessus de p_2^* en le sommet v_{13} du complexe (resp. en projection).

Considérons maintenant plus en détail tous les éléments du complexe au voisinage de ces trois courbes. Différentes modifications interviennent lors de la mise à jour.

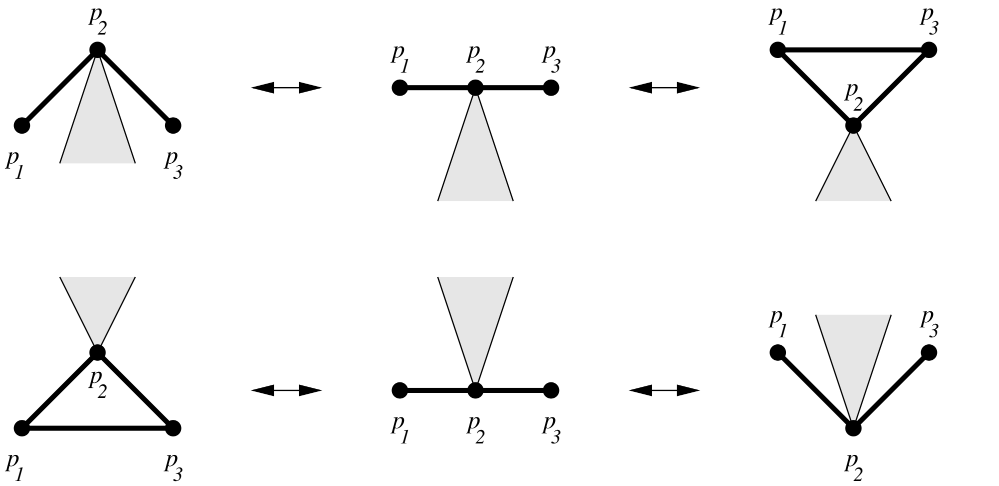


FIG. 4.8 – *Changements élémentaires de visibilité au voisinage de trois points.*

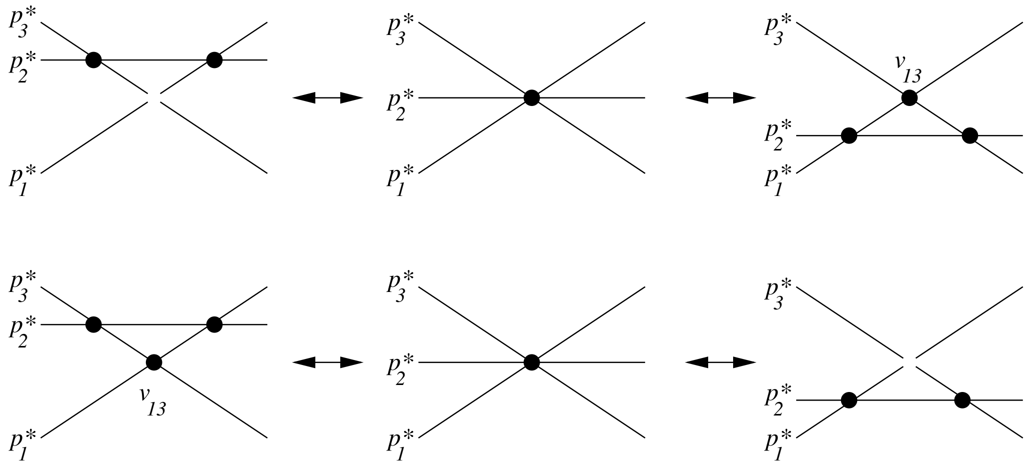


FIG. 4.9 – *Changements élémentaires de topologie du squelette du complexe.*

La figure 4.10 montre les changements de base communs à toutes les configurations géométriques correspondant aux deux situations de la figure 4.8. Dans la première situation, l'arête e_2 du complexe est détruite ; la face f, les trois arêtes qui la bordent et le sommet v_{13} sont créés ; les relations d'incidences entre les arêtes et les sommets v_{12} et v_{23} sont modifiées ; enfin les relations d'incidences entre les arêtes et les faces sont elles aussi modifiées.

Ce schéma commun doit cependant être précisé et complété, car les modifications complètes sont différentes pour chacune des 50 configurations géométriques possibles autour des trois points, montrées dans la figure 4.11. Ainsi, le type des arêtes créées dépend de la configuration géométrique. De plus d'autres faces incidentes aux arêtes portées par les courbes duales p_1^*, p_2^* et p_3^* que celles indiquées dans la figure 4.10 sont affectées par la suppression (resp. création) de l'arête e_2 et de celles bordant la face f. Le nombre et la position de ces faces sont eux aussi différents pour chacune des

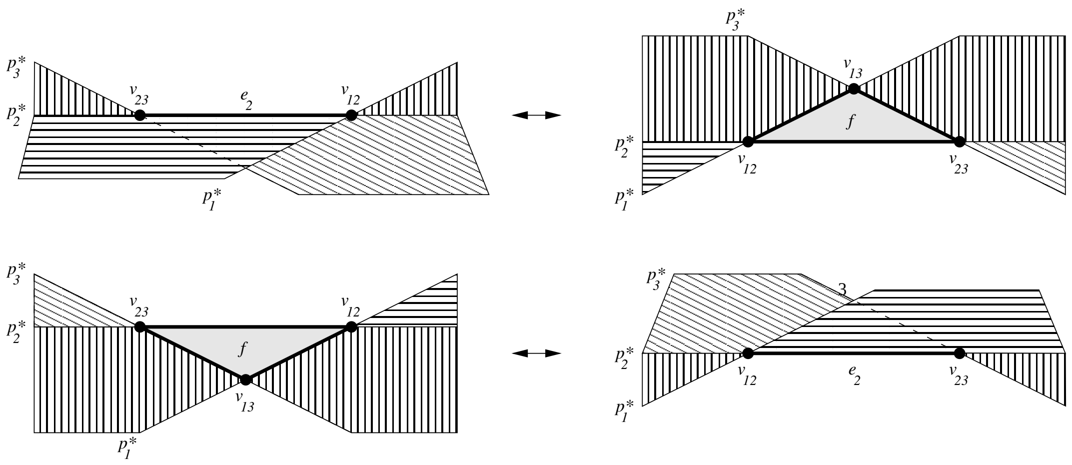


FIG. 4.10 – *Modifications de base dans le complexe lors d'un changement de visibilité.*

configurations géométriques.

Cependant, toutes ces configurations ne sont pas entièrement indépendantes. Nous avons regroupé ces configurations dans la figure 4.11 par famille : chaque ligne comporte des configurations proches de la première configuration de la ligne. Les différences au sein d'une famille se traduisent principalement par l'absence de quelques faces par rapport à la configuration de base.

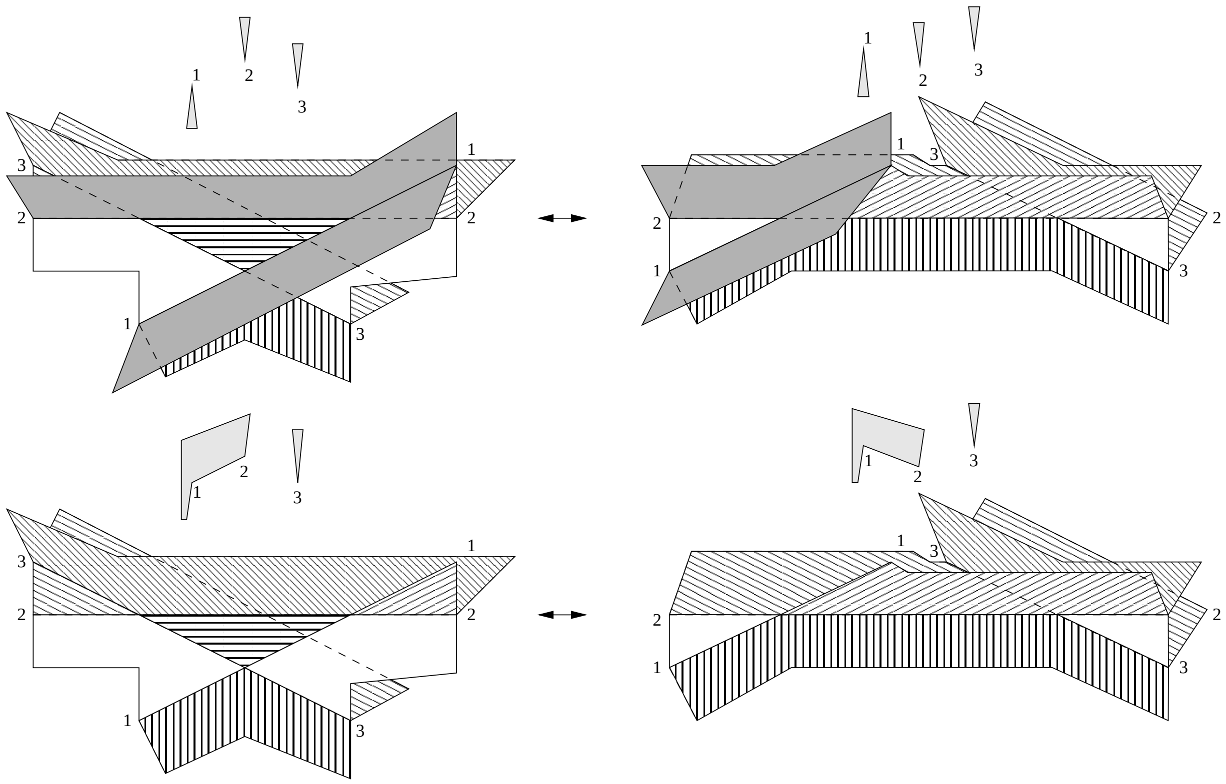


FIG. 4.12 – *Exemple de différences entre deux configurations.*

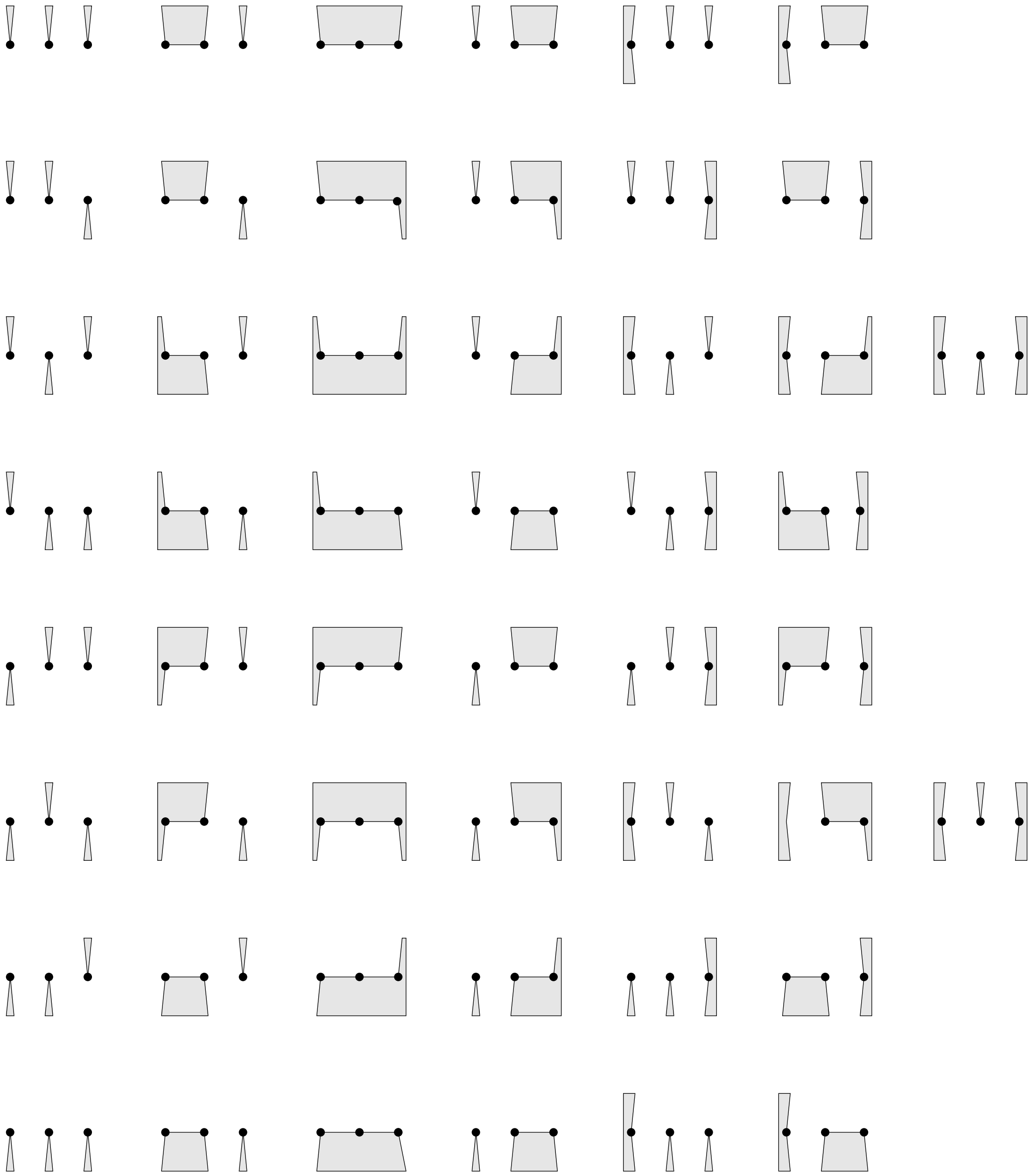


FIG. 4.11 – *Les 50 configurations géométriques au voisinage de trois points.*

Ainsi, dans la figure 4.12 nous avons indiqué en gris uniforme les deux faces manquantes entre deux configurations d'une même famille choisies en exemple.

Ces changements élémentaires servent d'opération de base pour maintenir le complexe : tout changement dans la structure du complexe se produisant lors du déplacement des points se ramène à une suite de changements élémentaires.

4.2.2 Maintien du complexe pour une scène mouvante

Nous supposons que les sommets des polygones ont une trajectoire connue à l'avance, déterminée par la position $p_i(t)$ du point p_i, où t varie de 0 à 1. Nous cherchons à maintenir le complexe depuis $t = 0$ jusqu'à $t = 1$. Nous pouvons pour cela maintenir la liste des changements élémentaires de visibilité à venir.

Les changements élémentaires se produisent quand, dans le complexe, une arête devient réduite à un sommet. Cependant, dans le cas où les trois points p_1, p_2 et p_3 sont mutuellement visibles, trois arêtes deviennent dégénérées lors de ce changement. Nous décidons alors de ne considérer que les arêtes $e = (v_g, v_d)$ qui sont arête haute (resp. basse) de leurs sommets extrêmes v_g et v_d. Ainsi une seule arête est associée à un changement élémentaire de visibilité, que les points extrêmes p_1 et p_3 soient mutuellement visibles ou non. Pour chacune de ces arêtes e, nous calculons une « date de mort », qui est la date t_e où les droites associées aux sommets $v_g(t_e)$ et $v_d(t_e)$ de l'arête seront confondues et où le changement de visibilité aura lieu.

Il suffit alors d'effectuer les changements associés aux arêtes dans l'ordre de leur date de mort. À chaque changement, il faut mettre à jour les arêtes à considérer ainsi que leur date de mort. Cependant, comme le changement est local, ces modifications sont elles aussi locales.

La figure 4.13 montre un changement de visibilité autour de l'arête e_{2m}. Cette arête est détruite et parmi les trois arêtes créées, seule e_{2m} doit être considérée. Les arêtes e_{3g} et e_{1d} ont (si elles sont à considérer) une nouvelle date de mort qu'il faut calculer. Les arêtes e_{1g} et e_{3d} ont un statut et une (éventuelle) date de mort inchangés. Enfin, les arêtes e_{2g} et e_{2d} changent de statut : si elle étaient à considérer, elles ne le sont plus ; si elles n'étaient pas à considérer, il faut vérifier si elles le deviennent et calculer le cas échéant leur date de mort.

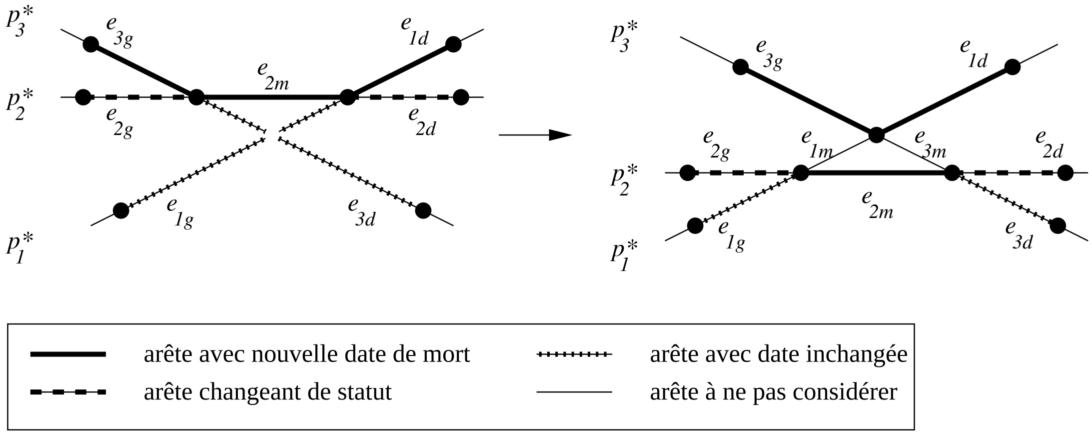


FIG. 4.13 – *Mise à jour des dates de mort : arêtes à considérer.*

Grâce à l'utilisation d'une queue de priorité, nous pouvons effectuer les changements élémentaires de visibilité dans l'ordre et maintenir le complexe de visibilité lors du mouvement des objets :

Proposition 4.3 *Après un calcul initial en temps $m_0 \log m_0$, où m_0 est la taille du* *complexe à $t = 0$, le complexe de visibilité peut être maintenu de $t = 0$ à $t = 1$ en temps $O(\log n)$ à chaque changement élémentaire de visibilité.*

Démonstration. Il y a $O(m)$ arêtes à considérer à chaque instant. L'initialisation de la queue de priorité prend un temps $O(m_0 \log m_0)$. À chaque traitement d'un changement de visibilité, les changements à effectuer sont locaux et se font en temps constant. La mise à jour de la queue de priorité prend un temps $O(\log m) = O(\log n)$. □

Dans le pire cas, il peut y avoir effectivement $O(n^2)$ arêtes à considérer (n triangles presque alignés).

4.2.3 Maintien d'une vue dans une scène mouvante

Nous cherchons maintenant à maintenir la vue autour d'un point mobile dans une scène où les objets sont aussi sujets à modifications. Si nous voulions réutiliser les algorithmes de maintien de vue dynamique du chapitre précédent, il faudrait remettre à jour les structures de données utilisées lors de ce maintien (enveloppes convexes par exemple) à chaque changement de topologie dans le complexe.

Il est plus simple d'inclure les changements de visibilité du point de vue avec les changements de visibilité entre objets de la scène. Nous avons vu que le calcul d'une vue consistait à calculer les arêtes du complexe de visibilité coupée par la courbe duale du point de vue, et que la vue changeait quand le courbe duale passait par l'un des sommets des arêtes coupées. Nous calculons donc pour chaque arête de la coupe les dates t où le point de vue $p(t)$ sera aligné avec les points correspondant à chaque sommet de l'arête. Ces dates sont rajoutées dans la queue de priorité avec les dates de changement élémentaires de visibilité.

La mise à jour lors d'un changement dans la vue du point $p(t)$ s'effectue comme décrit dans la section précédente. Nous devons juste ensuite calculer les dates de passage avec les nouvelles arêtes coupées par $p^*(t)$. La mise à jour et le calcul des nouvelles dates se fait en temps constant. L'insertion des nouvelles dates dans la queue de priorité se fait en temps logarithmique.

Lors d'un changement élémentaire de visibilité, en plus de la mise à jour du complexe, il faut vérifier si les arêtes modifiées ne font pas partie des arêtes coupées par le point de vue. Le cas échéant, il faut recalculer les dates de franchissement de ces arêtes. Vérifier si une arête est dans la coupe du point de vue peut se faire en temps constant en maintenant un tableau de taille n indexé par les points de la scène dont chaque case contient un pointeur sur l'arête de la coupe associée au point indice (le pointeur est NULL s'il n'y a pas d'arête associée dans la coupe). Ce surplus de mise à jour se fait aussi en temps constant, et la mise à jour de la queue de priorité en temps logarithmique.

Le maintien d'une vue dans une scène dynamique se fait en temps $O(\log n)$ à chaque changement élémentaire de visibilité dans la scène ou à chaque changement élémentaire dans la vue.

Si maintenant nous voulons maintenir la vue autour de n_p points, en ne tenant pas compte de la visibilité entre ces points de vue, alors avec un espace mémoire supplémentaire en $O(n_p n)$, nous pouvons maintenir les visibilités en temps $O(log(n^2 + nn_p))$ à chaque changement.

Chapitre 5

Dégénérescences et imprécisions des calculs numériques

Dans ce chapitre nous montrons comment traiter les cas dégénérés dans les divers calculs de visibilité étudiés dans les chapitres précédents, grâce à l'idée de *simulation de simplicité guidée par la visibilité*. Nous étudions ensuite les problèmes posés par les imprécisions de calculs numériques et proposons plusieurs solutions pour y remédier.

— *C'est une machine qui joue au bilboquet sans jamais rater son coup. [...] La force et l'ampleur de son mouvement sont calculées en fonction du poids exact de la boule, dont la trajectoire est instantanément décidée avec une précision...Hou làà ! Au micropoil ! [...]*
— *Fou ! Il est fou !*
— *[...]*
— *Clic ! Djiiiiii !* **KRAK !** *Tchink !*
— *Lorsque la brave machine, dans un éclair de lucidité, a compris combien elle était moche et mal fichue, elle décida, courageusement, de se supprimer ...*

Le gang des gaffeurs.

5.1 Traitement des dégénérescences

Nous avons supposé dans les chapitres précédents que les points de la scène (sommets des polygones ou points de vue) n'étaient jamais alignés. L'hypothèse « nous supposons que les objets sont en *position générale* » est couramment rencontrée dans l'étude théorique d'un problème géométrique. Cette hypothèse permet d'éliminer les cas dégénérés qui pourraient se présenter (points alignés, points cocycliques, droites concourantes ...). En effet, des arguments de perturbation montrent qu'en général la complexité théorique des structures de données et des algorithmes relatifs au problème étudié est maximale pour les *configurations générales* (c'est-à-dire les scènes où les objets sont en position générale). De plus, si la complexité théorique n'est pas modifiée par la présence de scènes dégénérées, il n'en est pas de même pour la complexité pratique : il faut alors considérer un nombre parfois important de situations particulières propres aux situations dégénérées.

Cette abondance de cas particuliers n'apporte en général rien à l'étude théorique du problème géométrique abordé et détailler tous les cas particuliers aurait plutôt tendance à rallonger et obscurcir l'exposé. Le traitement de ces cas est donc laissé de côté. Cependant les cas dégénérés sont souvent la règle en pratique, et les programmes qui ne les prennent pas en compte terminent souvent de façon brutale et prématurée leur exécution lorsqu'ils traitent des exemples concrets (figure 5.1).

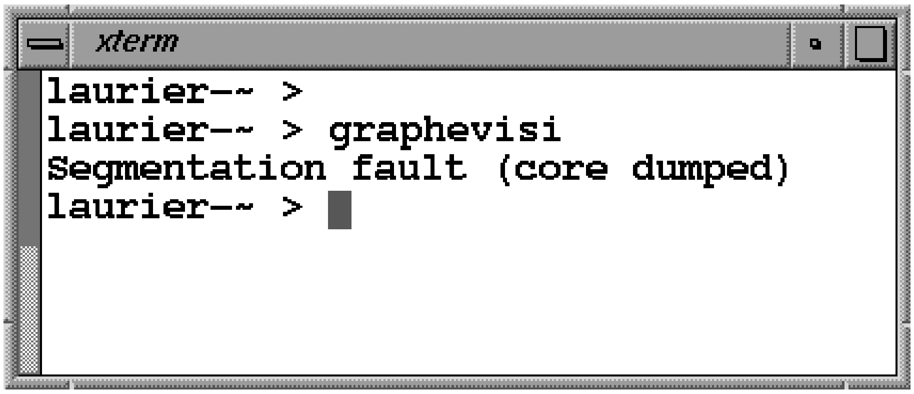


FIG. 5.1 – *Résultat d'un programme traitant une scène dégénérée.*

Dans cette section nous étudions la nature des problèmes posés par les dégénérescences. Nous étudions ensuite la possibilité d'appliquer un schéma de perturbation symbolique pour traiter ces problèmes. Nous montrons enfin comment améliorer ces schémas pour traiter les problèmes liés à la visibilité dans le plan.

5.1.1 Problèmes posés par les dégénérescences

Une configuration dégénérée est une configuration limite entre deux configurations non dégénérées. Lorsqu'un programme écrit pour des scènes générales rencontre un cas dégénéré, il l'assimile à l'une des deux configurations générales voisines. Or les traitements à appliquer aux deux configurations générales sont différents et souvent incompatibles. De plus, le choix de la configuration générale assimilée au cas dégénéré est souvent différent selon l'endroit du programme où le cas dégénéré est rencontré. Un même cas dégénéré peut donc recevoir des traitements différents et incompatibles au cours du déroulement du programme ; la cohérence est alors perdue et le programme devant faire face à des informations contradictoires s'interrompt alors brutalement.

Prenons comme exemple le balayage du graphe de visibilité d'une scène polygonale. La situation où trois points p_1, p_2 et p_3 sont alignés est une situation limite entre la configuration *complète* où les trois points sont mutuellement visibles et la configuration *simple* où les points extrêmes p_1 et p_3 sont cachés par le polygone en p_2 (figure 5.2).

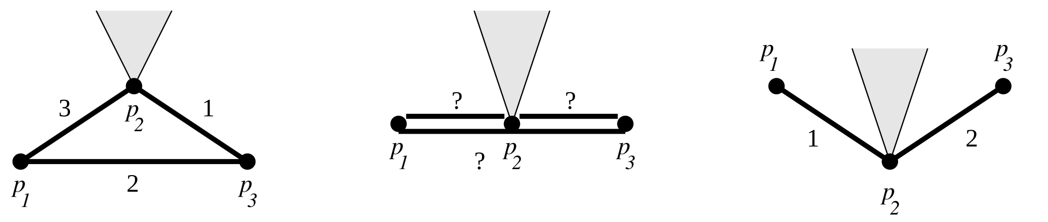


FIG. 5.2 – *Deux situations générales et leur cas dégénéré limite : ordre de passage des arêtes.*

Nous avons indiqué dans la figure 5.2 près des arêtes du graphe de visibilité l'ordre relatif dans lequel celles-ci doivent être balayées : dans la configuration complète, les arêtes doivent être passées dans l'ordre (p_2, p_3), (p_1, p_3) et (p_1, p_2) ; dans la configuration simple, elles doivent être passées dans l'ordre (p_1, p_2), (p_2, p_3). L'ordre de passage est différent dans les deux situations générales pour les arêtes (p_1, p_2) et (p_2, p_3). Si le programme assimile à un moment la situation dégénérée à la situation simple puis à un autre moment l'assimile à la situation complète, l'ordre de passage des arêtes devient incohérent Ce scénario catastrophe a de grandes chances de se produire par exemple pour un algorithme de balayage *droit* qui passe les arêtes dans l'ordre de leurs pentes : le choix d'une arête parmi plusieurs de même pente dans la queue de priorité risque de ne pas être cohérent.

La méthode la plus directe pour résoudre ces problèmes de dégénérescence est de traiter directement les cas dégénérés, c'est-à-dire de tenir compte de chaque cas dégénéré pouvant se produire. Cela suppose le recensement de toutes les configurations dégénérées, l'écriture du code spécifique pour chaque configuration et l'écriture de tests particuliers dans le programme pour repérer la présence de ces dégénérescences. Cette méthode, bien que directe, n'est pas la plus simple : il faut parfois étudier un grand nombre de configurations dégénérées. De plus, les solutions apportées sont spécifiques au problème traité et ne sont pas applicables à d'autres problèmes pour lesquels tout est à refaire.

Plusieurs méthodes générales ont été étudiées pour proposer des techniques pouvant s'appliquer à une grande variété de problèmes et dont l'emploi permet de ne pas bouleverser le code déjà écrit.

5.1.2 Schémas de perturbation symbolique

L'idéal pour le programmeur est d'avoir un moyen automatique de transformer (de façon purement syntaxique) un programme qui résout un problème pour toute entrée non dégénérée en un programme qui résout le problème pour toutes les entrées. Les *schémas de perturbation symbolique* permettent de transformer le programme en un programme qui « fonctionne » toujours, même sur des données dégénérées, mais qui cependant ne résout pas le problème initial mais un problème perturbé « proche » du problème initial. Nous rappelons ici quelques grandes lignes de la technique de perturbation (pour plus de détails, lire par exemple l'exposé de Seidel [Sei94] dont nous nous sommes inspirés pour cette présentation).

Un programme est formellement modélisé par un arbre de décision : le programme prend en entrée un vecteur de données $\mathbf{p} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ et, à chaque branche de son exécution, choisit la branche suivante à exécuter en fonction du signe de $f(\mathbf{p})$, où f est une fonction de décision de $\mathbb{R}^n$ dans $\mathbb{R}$ associée à la branche en cours d'exécution.

Un schéma de perturbation est défini par une fonction π, qui à tout vecteur de données $\mathbf{p}$ de $\mathbb{R}^n$ associe une fonction continue π_p de $\mathbb{R}^+$ dans $\mathbb{R}^n$ telle que $\pi_p(0) = \mathbf{p}$. Dans le programme, le calcul de $f(\mathbf{p})$ est alors remplacé par le calcul de $\overline{f}(\mathbf{p}) = \lim_{\varepsilon \to 0^+} f(\pi_p(\varepsilon))$. Si f est continue (en tout point non dégénéré), alors $\overline{f}(\mathbf{p}) = f(\mathbf{p})$ et le programme perturbé se comporte comme le programme initial pour des données non dégénérées. L'emploi de perturbations a pour but d'éviter d'exécuter les branches associées à des données dégénérées ($f(\mathbf{p}) = 0$) ; le schéma de perturbation π est alors dit valide si $\lim_{\varepsilon \to 0^+} \mathrm{sgn} f(\pi_p(\varepsilon))$ existe et est différent de 0 pour toutes les données $\mathbf{p}$ et toutes les fonctions de décision f. Grâce à l'emploi de perturbations, les cas dégénérés n'existent plus à l'intérieur du programme.

Le schéma de perturbation est symbolique, c'est-à-dire que le signe limite est en fait calculé sans calculs numériques impliquant ε (ce point est développé plus loin sur un exemple).

Enfin un schéma de perturbation doit être consistant pour être utilisable : si une fonction f n'est en fait qu'une fonction à m variables, alors l'ensemble des résultats obtenus pour toutes les permutations de taille m des coordonnées de $\mathbf{p}$ doivent être cohérents.

Plusieurs auteurs ont élaboré différents schémas particuliers de perturbation. Yap [Yap90] a conçu un schéma basé sur l'évaluation des dérivées partielles successives des fonctions de décision et a démontré un théorème de consistance. Ce schéma est cependant assez difficile à mettre en œuvre.

Seidel [Sei94] a montré qu'on pouvait toujours utiliser un schéma de perturbation linéaire, c'est-à-dire une fonction de perturbation $\pi_p(\varepsilon) = \mathbf{p} + \varepsilon\mathbf{a}$ où $\mathbf{a}$ est un vecteur de $\mathbb{R}^n$ non dégénéré ($f(\mathbf{a}) \neq 0$ pour toutes les fonctions de décision). Ce schéma est assez simple, cependant le calcul du vecteur $\mathbf{a}$ utilisé n'est pas toujours facile. De plus, un désavantage de cette méthode est que le résultat obtenu dépend du vecteur $\mathbf{a}$ choisi : le programme de calcul de graphe de visibilité calculerait, pour une même scène, différents graphes de visibilité suivant le vecteur $\mathbf{a}$ utilisé.

Edelsbrunner et Mücke [EM90] ont élaboré un schéma de perturbation appelé *simu-*

lation de la simplicité, surtout utilisé pour le calcul de déterminants. Nous le détaillons ici, car en plus d'être simple à mettre en œuvre, il nous permet d'illustrer concrètement l'utilisation d'un schéma de perturbation sur une opération – le calcul d'un déterminant – qui est une opération de base de nos algorithmes de calculs de visibilité. En effet, le signe du déterminant de trois points indique l'orientation relative des trois points, et permet par exemple de faire le test $u \prec \mathcal{E}^s(u)$ et de balayer les chaînes d'arêtes dans le calcul du graphe de visibilité (chapitre 2).

La simulation de la simplicité effectue la perturbation symbolique suivante : les coordonnées (x_i, y_i) d'un point p_i sont transformées en coordonnées perturbées $(x'_i, y'_i) = (x_i + \varepsilon^{2i}, y_i + \varepsilon^{2i+1})$, et le déterminant de trois points devient le déterminant perturbé

$$\begin{aligned}
\det{}'(p_i, p_j, p_k) &= \begin{vmatrix} x'_i & x'_j & x'_k \\ y'_i & y'_j & y'_k \\ 1 & 1 & 1 \end{vmatrix} \\
&= \begin{vmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ 1 & 1 & 1 \end{vmatrix} + \varepsilon^{2i} \begin{vmatrix} y_j & y_k \\ 1 & 1 \end{vmatrix} - \varepsilon^{2i+1} \begin{vmatrix} x_j & x_k \\ 1 & 1 \end{vmatrix} - \varepsilon^{2j} \begin{vmatrix} y_i & y_k \\ 1 & 1 \end{vmatrix} \\
&\qquad + \varepsilon^{2j+1} \begin{vmatrix} x_i & x_k \\ 1 & 1 \end{vmatrix} + \varepsilon^{2k} \begin{vmatrix} y_i & y_j \\ 1 & 1 \end{vmatrix} - \varepsilon^{2k+1} \begin{vmatrix} x_i & x_j \\ 1 & 1 \end{vmatrix}.
\end{aligned}$$

Le déterminant perturbé est un polynôme en ε, dont le signe est celui du premier coefficient non nul (en supposant ici $i < j < k$). Si les trois points sont différents alors, même si le déterminant initial est nul, au moins un des sous-déterminants extraits est non nul et le déterminant perturbé est toujours non nul. La perturbation reste symbolique : la procédure qui calcule le signe du déterminant perturbé n'utilise pas ε, mais elle calcule le déterminant et les sous-déterminants non perturbés jusqu'à trouver le premier non nul et renvoie alors son signe.

L'inconvénient de cette méthode est qu'elle augmente parfois inutilement la complexité des situations dégénérées. Ainsi, dans notre calcul de graphe de visibilité, si quatre points sont alignés toutes les arêtes entre paires de points risquent d'être calculées (figure 5.3a) alors que nous voulons seulement considérer les arêtes entre points successifs (figure 5.3b).

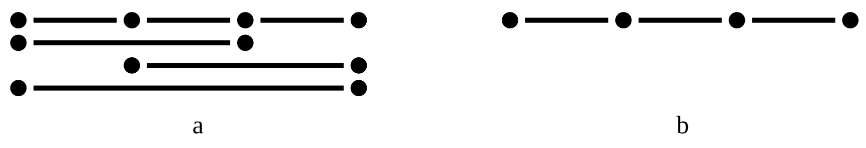


FIG. 5.3 – *Situation dégénérée.* **a.** *Situation complète.* **b.** *Situation simple.*

De plus, quand le déterminant initial est nul, le signe du déterminant perturbé dépend fortement de l'ordre relatif des indices i, j et k de numérotation des trois points considérés. L'existence et l'ordre de passage des arêtes entre points alignés sont alors dictés par la numérotation de ces points, et nous avons une situation surprenante : le programme de calcul de graphe de visibilité utilisant ce déterminant perturbé calcule

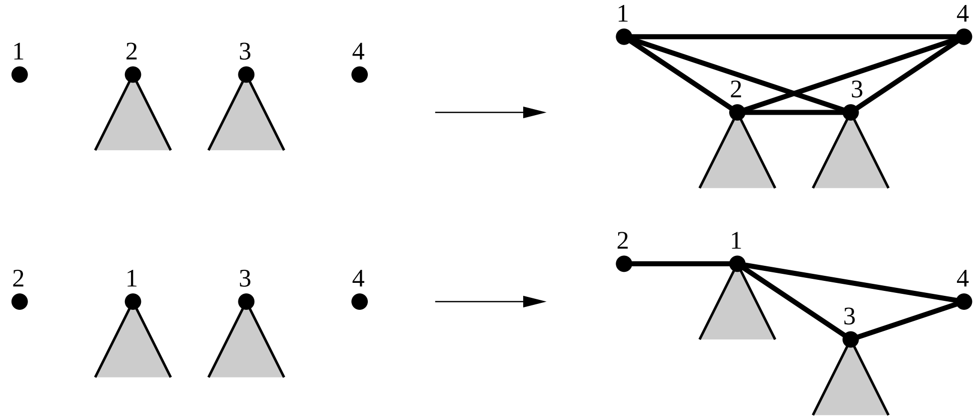


FIG. 5.4 – *Graphes de visibilité d'une même scène dégénérée calculés avec des déterminants perturbés : différences dues à la numérotation des points.*

pour une même scène des graphes différents suivant l'ordre dans lequel les points qui lui sont donnés sont numérotés (figure 5.4) !

En fait, le problème de cette méthode de perturbation et des autres méthodes vues précédemment est qu'elles utilisent des données extérieures au problème pour enlever les dégénérescences (le point **a** pour les perturbation linéaires, l'ordre de numérotation des points pour la simulation de simplicité). Pour éviter cette dépendance artificielle, nous nous servons des relations de visibilité pour guider la simulation de la simplicité.

5.1.3 Simulation de simplicité guidée par la visibilité

Nous retenons l'idée de faire comme si les cas dégénérés n'existaient pas, mais nous voulons en plus garder la « simplicité » géométrique de la situation. Considérons les situations dégénérées où trois points p_1, p_2 et p_3 sont alignés. Nous assimilons alors la configuration dégénérée à la configuration générale où les deux points extrêmes p_1 et p_3 sont cachés l'un de l'autre par le polygone en p_2 (figure 5.5).

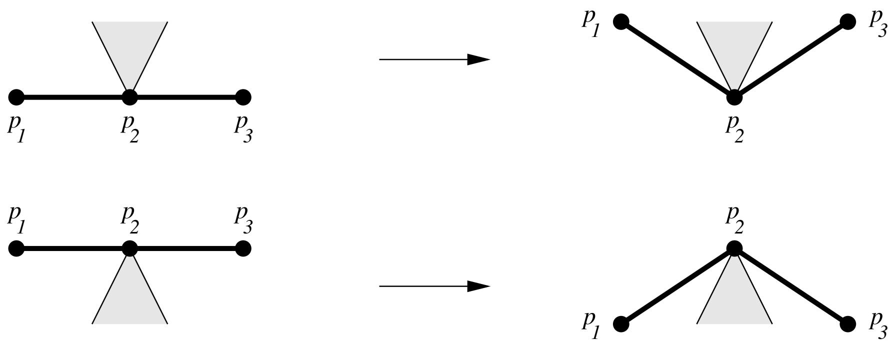


FIG. 5.5 – *Configurations dégénérées et configurations générales simulées.*

Cette méthode s'applique directement dans les tests du programme où la géométrie locale autour des trois points testés est connue. Le test

si $\det(p_1, p_2, p_3) > 0$ **alors** `traitement1`
sinon `traitement2`

est laissé tel quel si le cas dégénéré ($\det(p_1, p_2, p_3) = 0$) doit être assimilé à la situation où p_3 est au-dessus de (p_1, p_2). Le test doit être modifié en

si $\det(p_1, p_2, p_3) < 0$ **alors** `traitement2`
sinon `traitement1`

si le cas dégénéré doit être assimilé à la situation où p_3 est en dessous de (p_1, p_2).

Nous montrons maintenant comment appliquer cette règle simple aux calculs de visibilité étudiés dans les chapitres précédents.

5.1.4 Application aux calculs de visibilité

Reprenons tout d'abord le calcul de vue exposé au chapitre 3, et plus particulièrement le test d'une arête de la chaîne supérieure pour trouver l'arête de sortie de la face. La figure 5.6 montre les situations dégénérées – où le point de vue est aligné avec deux sommets de la scène – et les situations générales simulées correspondantes.

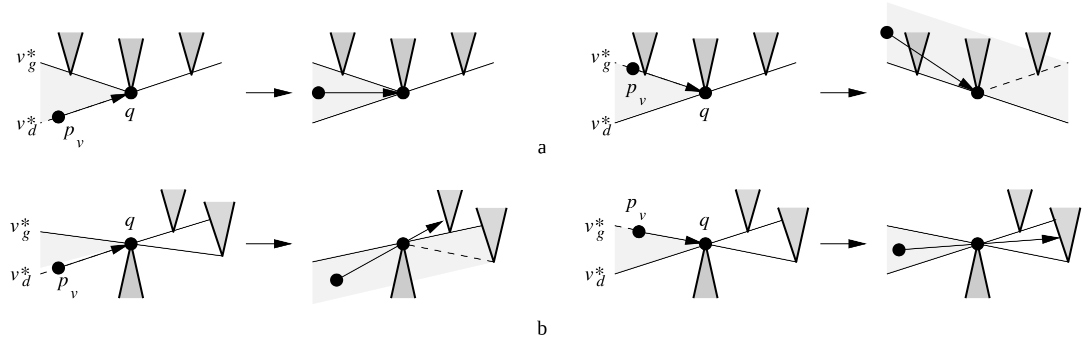


FIG. 5.6 – *Configurations dégénérées pour le test de sortie d'une face dans le calcul de vue et configurations générales associées.* **a.** *arête normale.* **b.** *arête grasse.*

Les tests de comparaison entre le point de vue p_v et l'arête $e = (v_g, v_r)$ qui s'écrivaient

sortie après l'arête $e \iff p_v$ est en dessous de v_d^*
sortie avant l'arête $e \iff p_v$ est au-dessus de v_g^*

deviennent maintenant pour une arête normale

sortie après l'arête $e \iff p_v$ est **strictement** en dessous de v_d^*
sortie avant l'arête $e \iff p_v$ est au-dessus de **ou sur** v_g^*

et pour une arête grasse

sortie après l'arête $e \iff p_v$ est en dessous de **ou sur** v_d^*
sortie avant l'arête $e \iff p_v$ est **strictement** au dessus de v_g^*

Les tests de comparaison pour une sortie par la chaîne basse sont modifiés de façon similaire.

Reprenons maintenant le test $u \prec \mathcal{E}^s(u)$ utilisé dans notre algorithme de balayage de graphe de visibilité (exposé au chapitre 2). En notant $u = (p, q)$ et $\mathcal{E}^s(u) = (q, r)$, le test s'écrit pour des scènes de polygones en position générale

$$u \prec \mathcal{E}^s(u) \iff r \text{ est au-dessus de } (p, q).$$

La figure 5.7 montre les configurations dégénérées possibles de u et $\mathcal{E}^s(u)$ quand p, q et r sont alignés et les configurations générales simulées correspondantes.

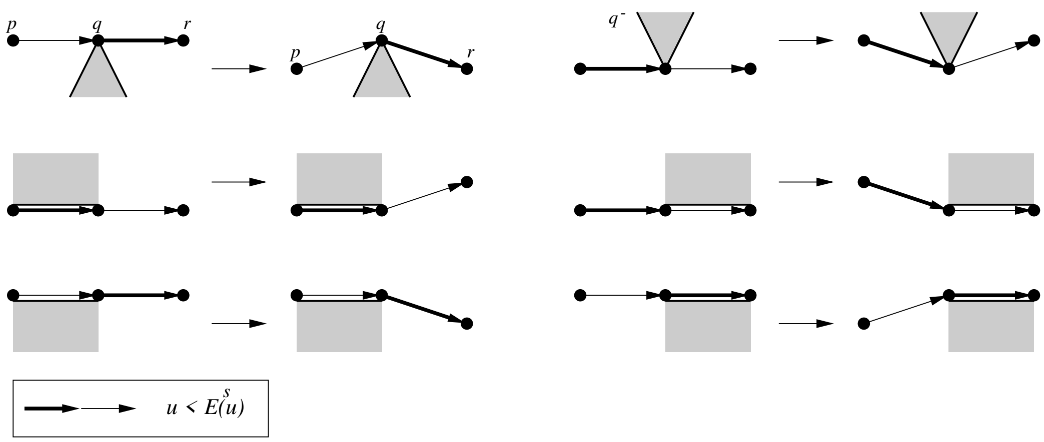


FIG. 5.7 – *Configurations dégénérées pour le test $u \prec \mathcal{E}^s(u)$ et configurations générales associées.*

Pour le test des arêtes de sortie dans le calcul de vue, la situation de test impose la géométrie de la configuration et la modification à apporter au test est immédiate. Ici, la situation (c'est-à-dire la donnée de u et $\mathcal{E}^s(u)$) ne nous donne pas toutes les informations géométriques nécessaires pour modifier immédiatement le test. Nous devons faire quelques tests supplémentaires pour identifier la configuration, et le test $u \prec \mathcal{E}^s(u)$ s'écrit alors

$$u \prec \mathcal{E}^s(u) \iff \begin{array}{l} r \text{ est strictement au-dessus de } (p, q) \\ \textbf{ou} \\ p = q^- \\ \textbf{ou} \\ q^- \text{ est strictement au-dessus de } (p, q) \end{array}$$

En revanche, pour le balayage des chaînes d'arêtes dans le même algorithme, la géométrie locale est connue et il suffit de faire attention au signe du déterminant (positivité ou positivité stricte) de façon similaire aux modifications faites pour le calcul de vue.

L'avantage de notre méthode est qu'elle ne nécessite pas de remaniements profonds des programmes. Lors des tests des signes des déterminants, nous devons juste savoir à quel cas général le cas dégénéré doit être assimilé, et parfois préciser le test ; mais la structure même du programme reste inchangée. Notre méthode est aussi consistante, puisqu'elle s'appuie directement sur la géométrie de la configuration.

Cette méthode est très liée aux problèmes de visibilité et quand la visibilité ne peut être utilisée pour guider le choix (par exemple pour choisir la chaîne d'arêtes de sortie dans le calcul d'une vue), un choix arbitraire doit être fait et de façon consistante. Cependant, la philosophie derrière cette méthode – assimiler un cas dégénéré au cas général le plus « simple » – pourrait trouver des applications à d'autres problèmes.

5.2 Traitement des imprécisions numériques

Les algorithmes étudiés sont programmés avec l'hypothèse implicite que les calculs sont effectués de façon exacte. Cependant les nombres ne sont représentés qu'avec une précision limitée dans les processeurs. Cette limitation peut ne pas avoir beaucoup d'importance pour les calculs effectués avec des nombres entiers : le risque principal encouru est un débordement des calculs qui peut être évité si les entiers considérés ne sont pas trop grands. Les calculs effectués avec des réels en virgule flottante posent en revanche des problèmes plus importants. En effet, les réels représentés en machine ne sont qu'une approximation des nombres « réels » et des problèmes d'arrondis se posent à la fois dans la représentation des nombres et dans les résultats des calculs.

Ces problèmes d'arrondis vont parfois très loin : deux expressions représentant la même expression mathématique mais qui diffèrent par l'ordre dans lequel les opérations sont effectuées, le degré de factorisation ..., peuvent donner deux résultats numériques différents.

Les imprécisions numériques peuvent se propager et s'amplifier dans le programme jusqu'à donner des résultats aberrants. Ainsi l'instabilité numérique des systèmes chaotiques est bien connue : par exemple, les deux suites définies par récurrence $p_{n+1} = p_n + rp_n(1-p_n)$ et $q_{n+1} = (1+r)q_n - rq_n^2$ (où r est un paramètre réel) sont mathématiquement identiques ; cependant au bout d'une dizaine d'itérations les valeurs de p_n et q_n diffèrent de plus en plus jusqu'à devenir décorrélées.

Mais des erreurs plus subtiles peuvent se produire, comme pour le calcul de la suite définie par

$$u_0 = 2,\ u_1 = -4 \text{ et } u_{n+1} = 111 - \frac{1130}{u_n} + \frac{3000}{u_n u_{n-1}},$$

dont la limite est $\lim u_n = 6$, et qui cependant converge numériquement vers 100 sur presque toutes les machines.

Nous exposons d'abord les problèmes spécifiques posés par les imprécisions des calculs numériques à nos applications. Nous exposons ensuite plusieurs méthodes pour palier à ces problèmes sans avoir à recourir à des techniques de calculs exacts.

5.2.1 Calculs d'un déterminant

Après avoir modifié notre programme de calcul de graphe de visibilité pour qu'il traite correctement les dégénérescences, celui-ci calcule correctement le graphe de visibilité de la scène dégénérée de la figure 5.8(a). Cependant, si nous appliquons préalablement par calcul numérique une rotation à la scène (figure 5.8b) le programme se plante.

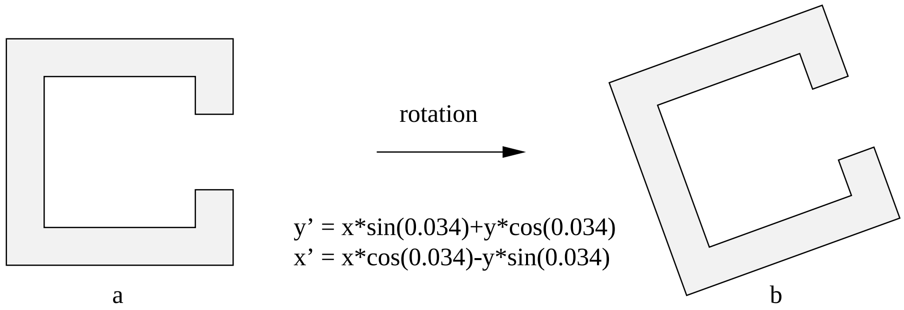
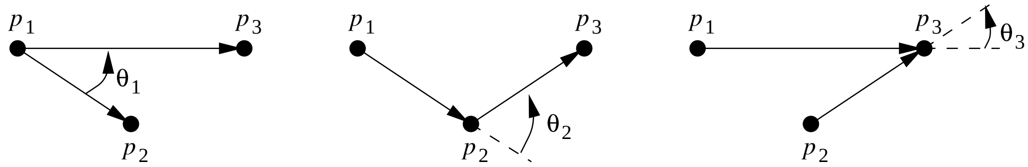


FIG. 5.8 – *Rotation numérique d'une scène test.*

Nous avons vu que le calcul du déterminant de trois points du plan était une opération de base de cet algorithme et du programme qui en résulte. Or le déterminant de trois points p_1, p_2 et p_3 peut s'écrire de trois façons différentes selon les deux côtés du triangle (p_1, p_2, p_2) qui sont choisis. Plus précisément, nous pouvons écrire ce déterminant selon les trois expressions suivantes (figure 5.9) :



$$
\begin{aligned}
D_1(p_1,p_2,p_3) = \left|\overrightarrow{p_1p_2}, \overrightarrow{p_1p_3}\right| &= (x_2-x_1)(y_3-y_1)-(y_2-y_1)(x_3-x_1)\\
&= \|\overrightarrow{p_1p_2}\|\,\|\overrightarrow{p_1p_3}\|\sin\theta_1\\
D_2(p_1,p_2,p_3) = \left|\overrightarrow{p_1p_2}, \overrightarrow{p_2p_3}\right| &= (x_2-x_1)(y_3-y_2)-(y_2-y_1)(x_3-x_2)\\
&= \|\overrightarrow{p_1p_2}\|\,\|\overrightarrow{p_2p_3}\|\sin\theta_2\\
D_3(p_1,p_2,p_3) = \left|\overrightarrow{p_1p_3}, \overrightarrow{p_2p_3}\right| &= (x_3-x_1)(y_3-y_2)-(y_3-y_1)(x_3-x_2)\\
&= \|\overrightarrow{p_1p_3}\|\,\|\overrightarrow{p_2p_3}\|\sin\theta_3
\end{aligned}
$$

FIG. 5.9 – *Signification géométrique des déterminants.*

Bien que mathématiquement les trois quantités exprimées par ces déterminants soient égales, si les calculs sont effectués en arithmétique flottante alors, comme les opérations sont effectuées sur des variables différentes et dans des ordres différents, les résultats numériques sont différents pour chacun des déterminants. Ces différences deviennent critiques quand les points sont presque alignés, comme le montre l'expérience suivante où nous avons effectué les calculs avec des réels en virgule flottante double précision.

Nous avons programmé les trois déterminants, soit par exemple

```
double D1(double x1,double y1,double x2,double y2,double x3,double y3)
{
  return (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
}
```

pour le déterminant D_1. Nous avons ensuite pris trois points alignés ($p_1 = (-3.2, 2)$, $p_2 = (1.3, 2)$, $p_3 = (5.9, 2)$) et nous leurs avons appliqué numériquement une rotation de i degrés :

```
double co = cos(i*0.01745329252), si = sin(i*0.01745329252);
xr1 = x1*co-y1*si; yr1 = x1*si+y1*co;
xr2 = x2*co-y2*si; yr2 = x2*si+y2*co;
xr3 = x3*co-y3*si; yr3 = x3*si+y3*co;
```

Enfin, nous avons calculé et affiché les trois déterminants :

```
cout<<D1(xr1,yr1,xr2,yr2,xr3,yr3)<<endl;
cout<<D2(xr1,yr1,xr2,yr2,xr3,yr3)<<endl;
cout<<D3(xr1,yr1,xr2,yr2,xr3,yr3)<<endl;
```

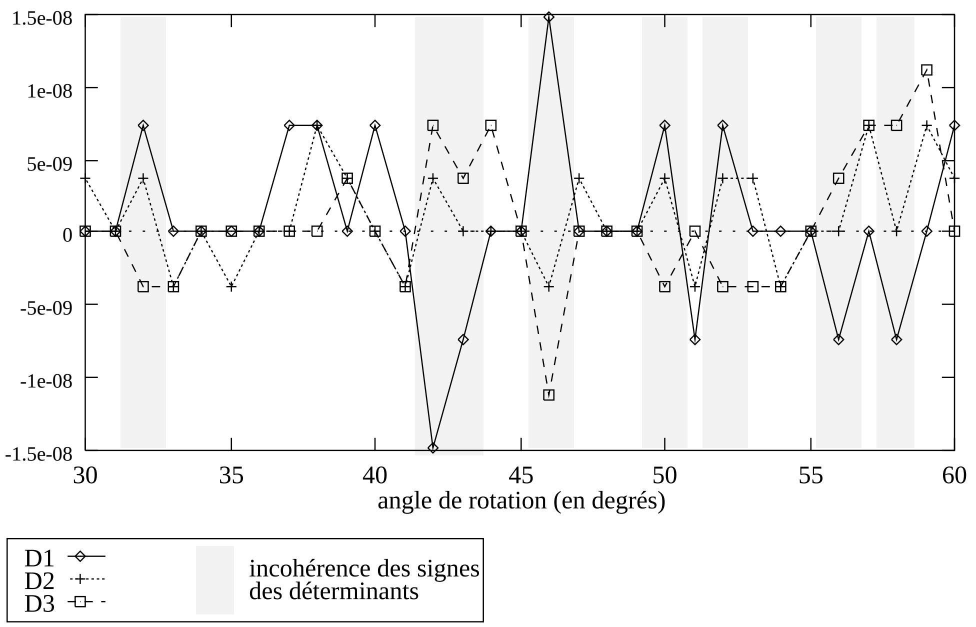


Fig. 5.10 – *Valeurs calculées par les trois déterminants : inversion du signe des résultats.*

Le diagramme de la figure 5.10 affiche les valeurs des déterminants obtenues (avec une arithmétique flottante double précision à la norme IEEE 754) en fonction de i, pour i variant de 30° à 60° (par pas de 1°).

Nous voyons que les valeurs calculées sont différentes pour les diverses rotations appliquées, mais surtout que les valeurs sont différentes entre les trois déterminants pour une même rotation et, ce qui est plus grave, que les signes des trois déterminants sont différents dans certains cas (parties grisées de la figure 5.10). Cela signifie que pour certains points p_1, p_2 et p_3, le signe du déterminant est différent selon que nous utilisons la formule de D_1, de D_2 ou de D_3. Ce problème subsiste toujours lorsque nous employons les expressions développées des déterminants à la place des expressions factorisées utilisées ici.

La cohérence géométrique de la situation est perdue : pour le programme, à certains endroits p_3 est au-dessus de (p_1, p_2), et à d'autres endroits p_3 est en dessous de (p_1, p_2). Ces incohérences expliquent l'échec de notre programme sur la scène à laquelle nous avions appliqué une rotation.

5.2.2 Calculs exacts

La solution la plus directe pour résoudre les problèmes d'imprécision des calculs numériques est d'effectuer des calculs exacts. Il est possible de définir exactement les nombres algébriques et de faire des calculs algébriques exacts. En effet un nombre algébrique α peut être représenté par une paire $\alpha = (P, I)$ telle que α soit la racine du polynôme P dans l'intervalle I. Il est alors possible de comparer ces nombres et d'effectuer dessus les quatre opérations arithmétiques ([Yapre]).

Cependant ces calculs font appel à des techniques lourdes et sont très coûteux (la théorie montre même que ces calculs sont doublement exponentiels en temps). Ils sont donc inutilisables en pratique et plutôt réservés à des programmes de calcul formel.

Cette notion de calcul exact étant trop forte, Yap et Dubé [YD95] ont défini un *paradigme de calcul exact*. Ce paradigme garantit que

- toutes les valeurs numériques sont représentées exactement ;
- toutes les décisions de branchement se font sans erreur.

Ce paradigme nécessite l'emploi de calculs multi-précision, c'est-à-dire de calculs mettant en jeu des nombres réels dont la taille de la mantisse (resp de l'exposant) peut être arbitrairement grande. Cependant ces calculs doivent être menés avec précaution pour éviter d'avoir des nombres représentés avec une précision trop grande, c'est-à-dire avec des chiffres non significatifs. Les nombres sont alors assortis d'un intervalle d'erreur.

Les manipulations de nombres s'accompagnent de manipulations d'expressions, qui sont représentées par des arbres dont les nœuds contiennent soit directement une valeur, soit une opération s'appliquant aux valeurs représentées par les sous-arbres du nœud. Les intervalles d'erreurs associés aux nombres sont alors redéfinis dynamiquement en fonction des besoins de précision.

Dans les calculs « paresseux » de l'équipe de St Etienne [BJMM93], les intervalles d'erreur sont ajustés depuis les feuilles vers la racine de l'arbre, jusqu'à ce que l'erreur sur la valeur finale (racine de l'arbre) satisfasse la précision demandée. Yap et Dubé [YD95] défendent plutôt des calculs « dirigés par la précision », où la précision, c'est-à-dire la taille de l'intervalle d'erreur, est fixée à la racine et est propagée vers les feuilles de manière à ce qu'elle soit garantie.

Yap et Dubé [YD95] défendent vivement le paradigme de calcul exact, car ce paradigme a une portée générale et permet de résoudre directement tous les problèmes de robustesse dans les programmes, alors qu'il serait selon eux impensable de recréer pour chaque programme une version robuste *ad hoc* de ce programme. Ils reconnaissent cependant que les calculs multi-précision ont un coût, qui peut être trop important pour certaines applications.

5.2.3 ε-géométrie

D'autres chercheurs, remarquant que dans certaines applications les données elles-mêmes sont imprécises, ont élaboré une géométrie « approximative », qu'ils ont appelée ε-géométrie. Dans cette géométrie, Guibas, Salesin et Stolfi [GSS89] modélisent les erreurs en supposant que les coordonnées des points ne sont connues qu'à une précision ε près. Pour un prédicat P défini sur un ensemble $\mathcal{O}$ de points, ils définissent le prédicat ε-P tel que

- $\varepsilon > 0$: ε-$P(p)$ est vrai $\iff \exists p' \in \mathcal{O}\ \|pp'\| < \varepsilon,\ P(p')$ est vrai ;
- $\varepsilon < 0$: ε-$P(p)$ est vrai $\iff \forall p' \in \mathcal{O}\ \|pp'\| < |\varepsilon|, P(p')$ est vrai.

Alors chaque test géométrique $P(p)$ ne renvoie plus `vrai` ou `faux`, mais renvoie deux limites f et v telles que le test ε-$P(p)$ est faux pour $\varepsilon < f$, vrai pour $\varepsilon \geq v$ et a un résultat non défini sinon. En particulier, si f est positive, alors $P(p)$ est « vraiment » fausse, et si v est négative, alors $P(p)$ est « vraiment » vraie.

Les limites renvoyées dépendent de la géométrie de la situation. Par exemple, un point r est strictement en dessous de (p, q) si le cercle centré en r et de rayon ε n'intersecte pas la zone au-dessus de (p, q) limitée par les bitangentes aux deux cercles centrés respectivement en p et q et de rayon ε (figure 5.11).

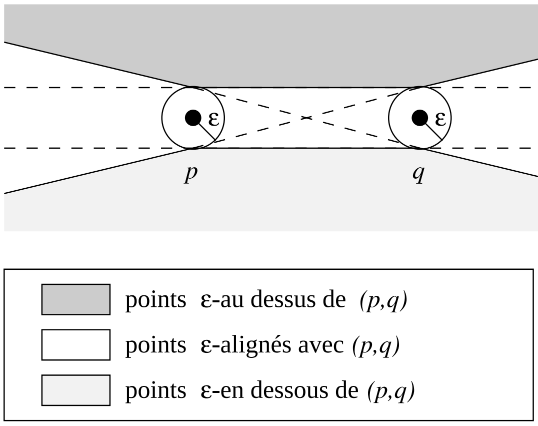


FIG. 5.11 – *ε-position relative par rapport à deux points.*

Les auteurs font aussi intervenir dans ces limites des encadrements d'erreurs liés aux problèmes d'arrondis des calculs effectués.

Nous cherchons ici à mettre en œuvre les méthodes les plus simples possibles pour traiter les problèmes d'imprécisions. Ces méthodes, bien qu'étudiées dans le cadre de notre algorithme de calcul de graphe de visibilité, ont une portée générale qui dépasse le seul cadre de notre programme et pourraient être réutilisées. Il s'agit en fait d'étudier le calcul des déterminants et leur utilisation.

5.2.4 Cohérence des calculs

Nous savons que des erreurs de calculs vont se glisser lors de l'exécution de notre programme de calcul de graphe de visibilité. Nous ne cherchons pas alors à obtenir un résultat exact, mais plutôt un résultat cohérent : nous voulons que le graphe de visibilité calculé soit un graphe de visibilité valide, même s'il correspond à une scène légèrement différente de la scène initiale.

Les problèmes d'inconsistance proviennent des résultats contradictoires obtenus par les différentes formules du déterminant. Pour être sûrs de ne plus avoir ces inconsistances, nous devons vérifier que pour trois points donnés la même formule de déterminant sera toujours utilisée. Nous devons faire alors attention à l'ordre dans lequel les points sont donnés dans le calcul d'un déterminant. En effet, nous avons les égalités suivantes entre *formules* des déterminants :

$$\begin{array}{lll}
D_1(p,r,q)=-D_1(p,q,r) & D_2(p,r,q)=-D_3(p,q,r) & D_3(p,r,q)=-D_2(p,q,r) \\
D_1(q,p,r)=-D_2(p,q,r) & D_2(q,p,r)=-D_1(p,q,r) & D_3(q,p,r)=-D_3(p,q,r) \\
D_1(q,r,p)=\ \ D_2(p,q,r) & D_2(q,r,p)=\ \ D_3(p,q,r) & D_3(q,r,p)=\ \ D_1(p,q,r) \\
D_1(r,p,q)=\ \ D_3(p,q,r) & D_2(r,p,q)=\ \ D_1(p,q,r) & D_3(r,p,q)=\ \ D_2(p,q,r) \\
D_1(r,q,p)=-D_3(p,q,r) & D_2(r,q,p)=-D_2(p,q,r) & D_3(r,q,p)=-D_1(p,q,r)
\end{array}$$

Dans notre programme de calcul de graphe de visibilité, nous avons inversé le sens des arêtes de l'arbre d'horizon inférieur : de cette façon, l'arbre inférieur est vraiment égal à l'arbre supérieur de la scène tournée de 180°. Nous avons alors utilisé le même code pour les deux arbres.

FIG. 5.12 – *Tests de position relative de trois points dans les arbres d'horizon.*

Considérons maintenant trois points p, q et r reliés successivement par une arête du graphe de visibilité. Alors à un moment le programme effectue le test $u \prec \mathcal{E}^s(u)$ avec $u = (p,q)$ et $\mathcal{E}^s(u) = (q,r)$; ce test est effectué par la comparaison $D_1(p,q,r) > 0$. Le programme effectue aussi à un autre moment le test $v \prec \mathcal{E}^i(v)$ avec $v = (r,q)$ et $\mathcal{E}^i(v) = (q,p)$; comme nous utilisons le même code pour les deux arbres d'horizon, ce test est effectué par la comparaison $D_1(r,q,p) > 0$ (figure 5.12).

Or nous avons vu que la formule $D_1(r,q,p)$ est la même que la formule $D_3(p,q,r)$. Pour ce triplet de points, différentes formules sont utilisées (au moins D_1 et D_3) pour les calculs de déterminant, ce qui explique les incohérences observées.

Nous avons alors écrit un code spécifique pour les arbres d'horizon inférieurs, de façon à ces que les formules utilisées pour les calculs de déterminant soient effectivement les mêmes que celles utilisées pour les arbres supérieurs. Par exemple, le test $v \prec \mathcal{E}^i(v)$ a été réécrit $D_3(r,q,p) > 0$, pour calculer effectivement $D_1(p,q,r)$ comme dans l'arbre supérieur.

Cependant cette cohérence ne peut être obtenue directement que si une certaine position relative des points est connue. Par exemple, dans le test $u \prec \mathcal{E}^s(u)$ nous savons que q est « entre » p et r. Cette information n'est malheureusement pas connue quand nous balayons les chaînes d'arêtes pour trouver la nouvelle arête de l'arbre d'horizon. Pour tester si l'arête partant de p est tangente à la chaîne en q, nous comparons la position relative des points p, q et $r = e^s(q)$. Or la position de p n'est pas connue : p peut être entre q et r, ou peut être derrière q et r. Suivant la position de p, le déterminant calculé ne sera pas le même que celui utilisé dans les tests $u \prec \mathcal{E}^s(u)$ et une nouvelle incohérence géométrique apparaîtra (figure 5.13).

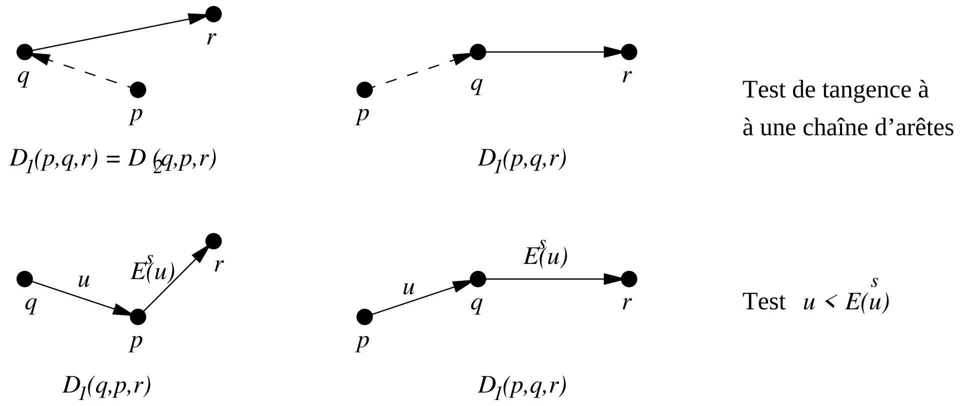


FIG. 5.13 – *Différences de déterminants dues à la différence de position du point p avec l'arête balayée de la chaîne.*

Nous pourrions, pour être sûrs de toujours utiliser la même formule de déterminant, utiliser un déterminant *lexicographique*, c'est-à-dire calculer pour trois points p_i, p_j et p_k le déterminant $D_1(p_i, p_j, p_k)$ où les points sont ordonnés selon l'ordre de leur numérotation (dans cet exemple $i < j < k$). Nous retrouvons alors les problèmes rencontrés dans l'utilisation de la simulation de la simplicité pour ôter les dégénérescences : le graphe de visibilité obtenu est différent selon les numérotations des points d'une même scène.

Nous voulons garder une certaine robustesse vis à vis de l'ordre dans lequel les données sont rentrées. Nous ne voulons pas faire de tests de position supplémentaires pour choisir le bon déterminant à calculer : ces tests seraient sujets eux-mêmes aux imprécisions de calcul et pourraient provoquer des incohérences. Nous cherchons alors à calculer un déterminant « robuste », qui reste cohérent dans tous ses emplois dans le programme.

5.2.5 Opérations unanimes

Nous voulons qu'à chaque requête concernant la position relative de trois points

p, q et r effectuée pendant le déroulement du programme, la réponse donnée soit toujours la même. Comme ces calculs de position relative peuvent faire intervenir trois formulations différentes de déterminant (D_1, D_2 et D_3), pour obtenir l'unanimité entre ces trois formulations nous disons que r est (strictement) au-dessus de (p, q) si et seulement si les signes des trois déterminants sont tous positifs, sinon r est supposé être en dessous de ou aligné avec (p, q). Le test `au_dessus(p,q,r)`, qui indique si r est au-dessus de (p, q), s'écrit maintenant

```
au_dessus(p,q,r) :
  si D1(p,q,r)<=0.0 → faux
  sinon
    si D2(p,q,r)<=0.0 → faux
    sinon
      → D3(p,q,r)>0.0
```

et la même modification est faite au test `en_dessous(p,q,r)` (qui indique si r est en dessous de (p, q)).

Les problèmes sont posés par les triplets de points (p, q, r) pour lesquels les signes des déterminants sont différents. Pour ces triplets, les tests `au_dessus(p,q,r)` et `en_dessous(p,q,r)` sont maintenant tous les deux faux. Cela signifie implicitement que les trois points sont alignés. Or comme notre programme traite correctement les cas dégénérés, il traite maintenant correctement les triplets qui causaient des incohérences locales.

Plus précisément, nous n'avons pas besoin de modifier notre programme car nous avons pris soin de ne pas particulariser les cas dégénérés mais de les assimiler directement à des cas non dégénéres. En effet, rappelons que dans notre programme, les tests de la forme

```
si au_dessus(p,q,r) alors traitement1
sinon                     traitement2
```

signifient que `traitement1` concerne le cas non dégénéré où r est **strictement** au-dessus de (p, q), et que `traitement2` concerne les autres cas – cas dégénéré inclus – où r est en dessous de **ou aligné avec** (p, q). Si le triplet (p, q, r) provoque des incohérences, alors `au_dessus(p,q,r)` est faux et le triplet, implicitement considéré comme dégénéré, reçoit bien le traitement adéquat.

Grâce à ces déterminants « unanimes », nous obtenons une cohérence *locale*. Malheureusement, les imprécisions numériques peuvent provoquer des incohérences *globales*. Par exemple, après avoir effectué une rotation de 11° aux points $p_1 = (-7.3, 2.2)$, $p_2 = (-2.1, 2.2)$, $p_3 = (1.4, 2.2)$ et $p_4 = (3.2, 2.2)$, les calculs de déterminants unanimes donnent pour résultat :

```
au_dessus(2,3,4) : vrai ; en_dessous(2,3,4) : faux
au_dessus(1,3,4) : faux ; en_dessous(1,3,4) : faux
au_dessus(1,2,4) : faux ; en_dessous(1,2,4) : vrai
au_dessus(1,2,3) : faux ; en_dessous(1,2,3) : faux
```

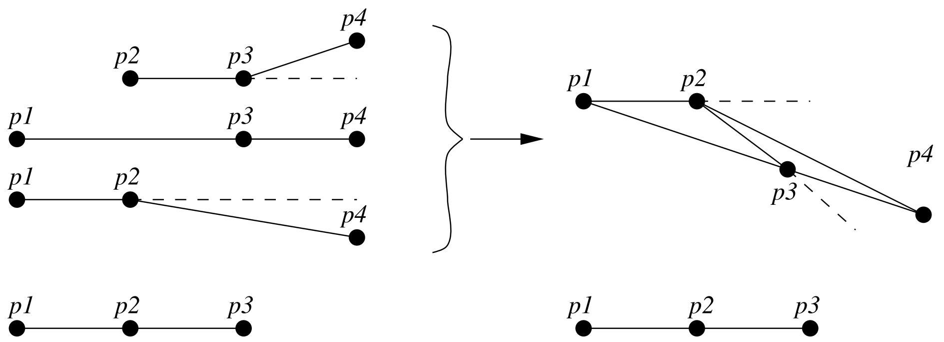


FIG. 5.14 – *Incohérences des résultats locaux de déterminant sur une situation globale.*

La figure 5.14 montre pour chacun des triplets de points les situations géométriques correspondant aux résultats des calculs. Nous voyons que ces résultats ne sont pas cohérents : il n'existe pas de configuration de quatre points compatible avec ces résultats.

5.2.6 Calculs d'erreurs

Nous devons, pour assurer une cohérence globale, mieux étudier les imprécisions numériques qui se produisent dans le calcul d'un déterminant. La norme IEEE 754 sur les calculs en virgule flottante définit un standard sur la représentation des nombres réels en virgule flottante double précision et sur les arrondis effectués sur les résultats d'opérations sur de tels nombres. Cette norme spécifie notamment que le résultat d'une opération algébrique $(+,-,*,/,\sqrt{})$ est arrondi à une demie *ulp* (Unit in Last Position) près, et que deux nombres x et x' consécutifs, c'est-à-dire ne différant que d'une ulp, vérifient

$$1.1\ 10^{-16} < 0.5^{53} < \frac{(x'-x)}{x} < 0.5^{52} < 2.3\ 10^{-16}\ .$$

Cela signifie que lorsque le processeur effectue une opération algébrique sur deux nombres, il renvoie une valeur arondie x^* qui vérifie par rapport au résultat exact x de l'opération : $x^* = (1+\varepsilon)x$, où l'erreur est bornée par $|\varepsilon| < u = 2.3\ 10^{-16}$. La valeur exacte x se situe alors dans l'intervalle $[x^* - \varepsilon|x^*|, x^* + \varepsilon|x^*|]$.

Si nous appliquons ces bornes (en ne gardant à la fin que les termes du premier ordre en u) à toutes les opérations effectuées lors du calcul du déterminant

$$D_1(p1,p2,p3) = (x2-x1)(y3-y1) - (y2-y1)(x3-x1)$$

alors nous obtenons l'encadrement

$$D_1 \in D_1^* \pm (u|D_1^*| + 3uM_1^*)$$

où $M_1 = |x2-x1||y3-y1| + |y2-y1||x3-x1|$.

De façon générale, nous notons $D_i^{min} = D_i - (u|D_i| + 3uM_i)$ et $D_i^{max} = D_i + (u|D_i| + 3uM_i)$ les bornes d'erreurs du déterminant D_i. Nous disons alors que le déterminant D_i est strictement positif (resp. négatif) si les deux bornes de l'intervalle sont strictement positives (resp. négatives), c'est-à-dire si $D_i^{min} > 0$ (resp. $D_i^{max} < 0$). Comme les encadrements d'erreur du determinant D_i certifient que la « vraie » valeur du déterminant D est comprise dans l'intervalle $[D_i^{min}, D_i^{max}]$, alors si D_i^{min} est strictement positif, nous pouvons en déduire que le déterminant D est strictement positif. En revanche, si l'intervalle d'erreur contient 0, cela ne nous permet pas d'affirmer que D n'est pas positif.

En effet, nous avons repris l'expérience de calculs de déterminants de la partie 5.2.1. Nous avons calculé les déterminants D_1, D_2 et D_3 sur les points $p_1 = (-3345671, 1)$, $p_2 = (1, 1.000000001)$ et $p_3 = (6654312, 1)$ après leur avoir appliqué des rotations (figure 5.15a). Nous avons aussi calculé les bornes d'erreur de ces déterminants (figure 5.15b).

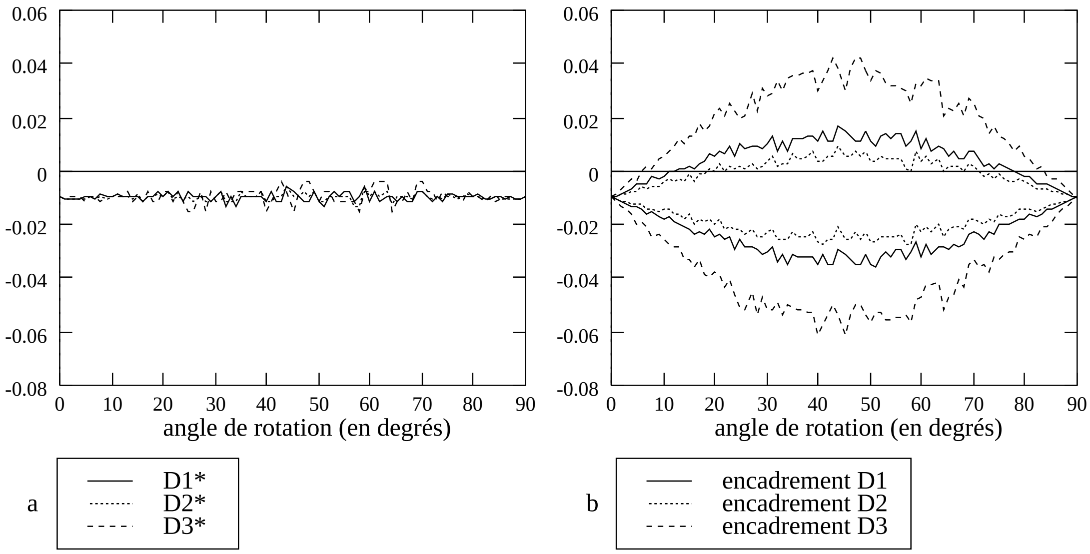


FIG. 5.15 – *Précision des calculs numériques.* **a.** *Déterminants calculés.* **b.** *Encadrement calculé des déterminants.*

Nous voyons que les valeurs des encadrement sont différentes selon les déterminants. En effet, si les points sont relativement alignés, alors $M_1 \approx \|\overrightarrow{p_1,p_2}\|\|\overrightarrow{p_1,p_3}\|\sin 2\theta$, $M_2 \approx \|\overrightarrow{p_1,p_2}\|\|\overrightarrow{p_2,p_3}\|\sin 2\theta$ et $M_1 \approx \|\overrightarrow{p_1,p_3}\|\|\overrightarrow{p_2,p_3}\|\sin 2\theta$, où θ est l'angle de la droite passant par les trois points. Nous voyons que pour certains angles, certains encadrements contiennent 0, et les autres non.

En fait nous devons avoir ici une unanimité des déterminants, non pas pour affirmer que le déterminant est positif (resp. négatif), mais pour affirmer qu'il ne l'est pas et qu'il doit être considéré comme nul. Nous n'avons donc plus

$$D \text{ positif} \iff D_1 \text{ positif } \textbf{et } D_2 \text{ positif } \textbf{et } D_3 \text{ positif}$$

comme pour les déterminants unanimes, mais plutôt

$$D \text{ positif} \iff D_1 \text{ positif } \textbf{ou } D_2 \text{ positif } \textbf{ou } D_3 \text{ positif}$$

et les fonctions de position relative s'écrivent alors :

au_dessus(p,q,r) :
  si D_1^{min}(p,q,r)>0.0 → vrai
  sinon
    si D_2^{min}(p,q,r)>0.0 → vrai
    sinon
      → D_3^{min}(p,q,r)>0.0

en_dessous(p,q,r) :
  si D_1^{max}(p,q,r)<0.0 → vrai
  sinon
    si D_2^{max}(p,q,r)<0.0 → vrai
    sinon
      → D_3^{max}(p,q,r)<0.0

En reprenant la formule de calcul d'erreur, nous remarquons que l'erreur absolue commise sur le déterminant peut être assez grande : uM est de l'ordre de grandeur de ud^2, où d est la longueur du plus grand côté du triangle (p_1, p_2, p_3). Il ne faudrait pas que l'encadrement calculé soit tellement grand qu'il contienne alors 0 et que le déterminant soit considéré comme nul, alors que le déterminant est *a priori* légitimement positif (resp. négatif).

Nous avons choisi les points utilisés pour les calculs de la figure 5.15 obtenir une distance d importante. La figure 5.15(a) montre que les valeurs D^* calculées des déterminants se répartissent autour de -0.01. Le déterminant peut être alors raisonnablement considéré comme négatif. Cependant, quand nous examinons les valeurs encadrant les déterminants (figure 5.15b), nous voyons que dans certains cas les trois encadrements contiennent 0, et que d'après nos décisions, ils doivent être considérés comme nuls.

Notre encadrement des erreurs de calculs est-il alors voué à l'échec ? Remettons certaines choses en perspective. Si nous reprenons l'interprétation géométrique du déterminant, nous voyons que la valeur du déterminant est de l'ordre de grandeur de $\sin\theta\, d^2$, où θ est l'un des angles du triangle (p_1, p_2, p_3). Or nous calculons une erreur possible de ud^2 sur cette valeur de $\sin\theta\, d^2$, avec $u = 2.3\ 10^{-16}$. Cette erreur uM ne devient importante par rapport à la valeur du déterminant que quand θ est très petit – de l'ordre de 10^{-15} radians –, c'est-à-dire quand les trois points doivent être considérés comme alignés. Si nous reprenons notre exemple « fautif », nous nous apercevons en fait que les angles du triangle formé par les trois points sont de l'ordre de 10^{-15} radians, et donc que les points peuvent être légitimements considérés comme alignés, malgré des déterminants autour de -0.01.

Si nous reprenons le calcul du graphe de visibilité de la scène de la figure 5.8, qui nous posait problème à cause des incohérences globales entre quatre points (figure 5.14), grâce à la maitrise des erreurs de calculs, ces point sont tous considérés comme alignés et le programme s'exécute correctement.

5.2.7 Coût du traitement des imprécisions numériques

Nous avons mis en place deux méthodes pour lutter contre les erreurs de calculs numériques des déterminants simples : les déterminants unanimes et les déterminants avec bornes d'erreur. Nous avons mis en place ces méthodes pour essayer de lutter contre les imprécisions de calcul avec un surcoût ausis faible que possible.

Nous avons effectué des mesures sur nos programmes pour évaluer ces surcoûts. Nous avons pris 10 scènes aléatoires de 600 triangles, de densité de visibilité croissante (voir chapitre ?), et nous avons exécuté notre programme de calcul de graphe de visibilité, en utilisant respectivement une seule formule de déterminant (déterminant simple), les déterminants unanimes et les déterminants tenant compte des bornes d'erreur de calcul.

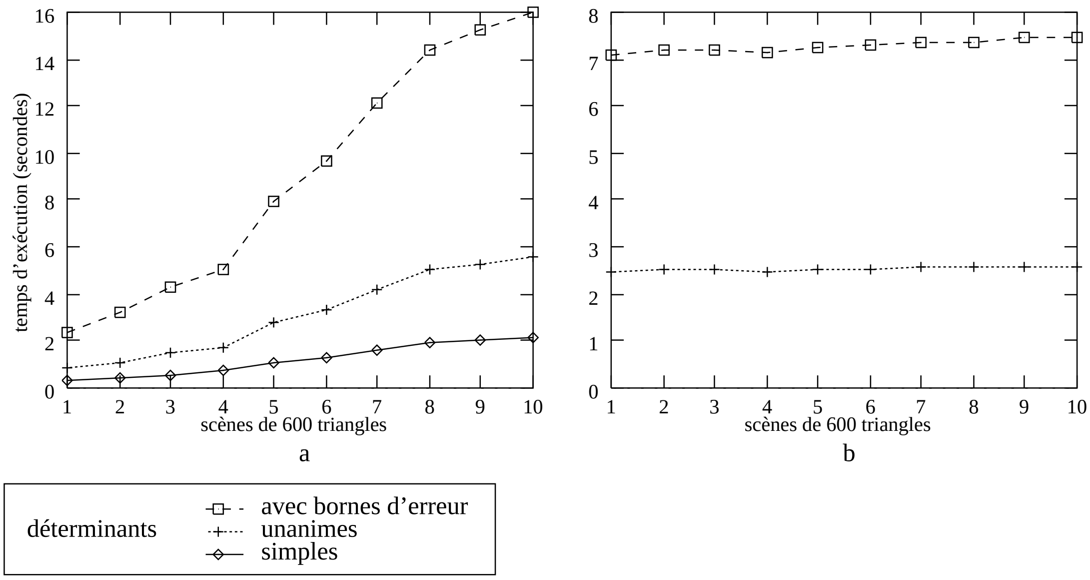


FIG. 5.16 – *Coûts de calcul des déterminants.* **a.** *Temps passé par le programme de calcul de graphe de visibilité à calculer des déterminants.* **b.** *Rapport de ces temps avec celui des déterminants simples.*

Nous avons tout d'abord isolé le temps passé par le programme dans le calcul des déterminants. La figure 5.16(a) affiche ce temps (en secondes) pour chacune des scènes tests, et la figure 5.16(b) affiche le rapport du temps utilisé par les déteminants avec borne d'erreur (resp. unanimes) avec le temps utilisé par les déterminants simples. Nous voyons que les déterminants unanimes coûtent environ 2.5 fois plus de temps que les déterminants simples, et que les calculs de bornes d'erreur coûte environ 7 fois plus de temps.

Ce surcoût est assez important. Cependant nous devons aussi examiner ce surcoût de manière plus relative, en regardant le surcoût occasionné par rapport à l'ensemble du programme. Le calcul des déterminants prend environ 18 % du temps total d'exécution du programme. Le temps d'exécution du programme devrait être donc multiplié par $1+(r-1)*0.18$, où r est le rapport entre les déterminants (figure 5.16b), soit un surcoût d'environ 1.3 (resp 2.1) pour l'utilisation de déterminants unanimes (resp. avec bornes d'erreur).

La figure 5.17(a) montre le temps d'exécution total des programmes, et la figure 5.17(b) les rapports de ces temps par rapport à celui du programme utilisant les déterminants simples. Nous voyons qu'effectivement les surcoûts mesurés sont concordants avec ceux que nous venons de calculer, et que le calcul de bornes d'erreur double le temps d'exécution du calcul de vues.

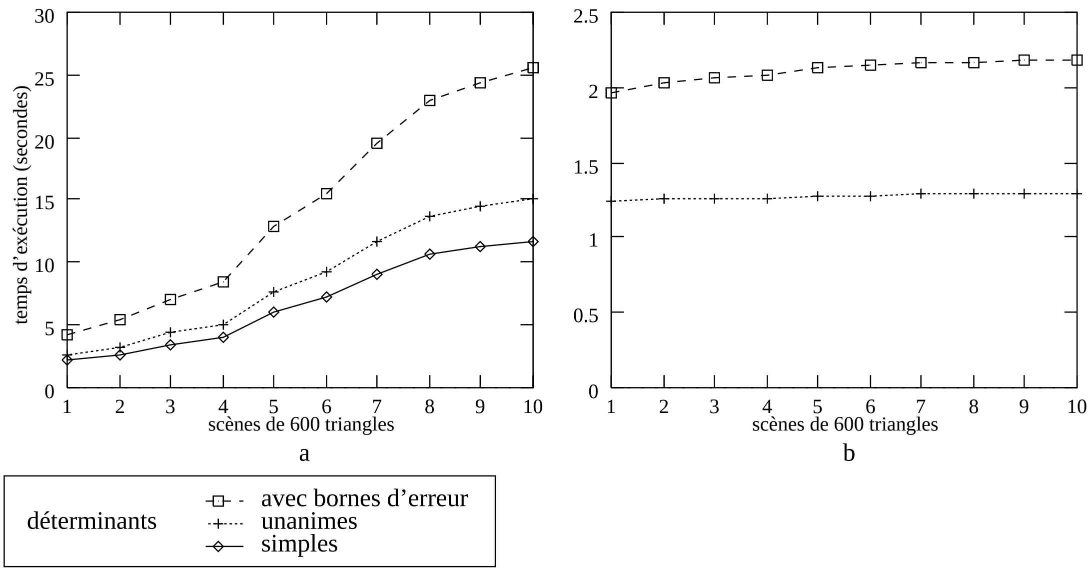


FIG. 5.17 – *Calculs de graphe de visibilité avec plusieurs déterminants.* **a.** *Temps d'exécution des programmes.* **b.** *Rapport de ces temps avec le programme utilisant les déterminants simples.*

5.2.8 Conclusion

Le traitement des problèmes posés par les imprécision des calculs en virgule flottante ne peut pas être séparé des problèmes de dégénérescences.

En effet, les configurations pour lesquelles se produisent des incohérences numériques sont des configurations qui doivent être considérées comme des configurations dégénérées, puisque les calculs ne permettent pas de faire la différence entre les situations générales auxquelles elles appartiennent. Ceci est encore plus marqué dans la ε-géométrie, où les ensembles de points ε-dégénérés sont continus et donc ont une certaine importance. Cependant, le traitement des dégénérescences ne peut se faire en utilisant les schémas de perturbation symbolique, car les erreurs de calculs entravent la gestion des perturbations. Par exemple, pour le calcul de déterminant utilisant la simulation de simplicité, non seulement le déterminant principal est souvent non nul, mais les imprécisions touchent aussi le calcul des sous-déterminants.

En revanche, notre simulation de simplicité guidée par la visibilité, qui n'utilise que des décisions basées sur la géométrie, s'adapte bien aux traitements des imprécisions numériques.

L'idée directrice majeure que nous avons utilisée pour lutter contre les problèmes d'imprécision de la façon la moins pénalisante possible est la *cohérence.*

Nous avons d'abord voulu être cohérents en essayant d'utiliser, parmi les diverses expressions correspondant à un même calcul, une seule de ces expressions. Malheureusement, nous avons vu que dans notre programme de calcul de graphe de visibilité, nous manquions d'informations pour décider directement quelle expression utiliser.

Nous avons ensuite assuré une cohérence locale en requérant l'unanimité entre les expressions pour donner une réponse positive. Cependant cette cohérence locale est insuffisante dans certains cas pour assurer une cohérence globale.

Nous avons alors utilisé la norme IEEE 754 pour déterminer les marges d'erreurs dans le calcul des déterminants, et tenir compte dans les décisions non pas seulement de la valeur calculée des déterminants, mais aussi des encadrements de cette valeur.

Certain auteurs poussent encore plus loin la modélisation des imprécisions pour garantir la robustesse des programmes ($\varepsilon-$géométrie par exemple). Enfin il reste la possibilité de recourir à des calculs exacts pour éliminer plus directement ces problèmes.

Nous pouvons cependant modérer un peu cette chasse aux imprécisions numériques. Les problèmes que nous avons rencontrés étaient construits de façon artificielle, en effectuant une rotation numérique à des points alignés. Ainsi les coordonnées de ces points avaient vraiment 16 chiffres après la virgule significatifs (en fait 33 bits), c'est-à-dire que le dernier chiffre de la mantisse est important et peut faire basculer le résultat d'un calcul.

Si la précision peut être importante à prendre en compte dans le cadre de simulations numériques par exemple, elle peut être excessive pour d'autres problèmes, comme nos programmes de visibilité par exemple. Les cordonnées des points de la scène ne nécessitent pas une précision excessive et quelques chiffres significatifs après la virgule suffisent. Avec une précision utile plus faible, les problèmes d'imprécision sont alors beaucoup moins importants. Un bon traitement des problèmes d'imprécision des calculs numériques passe peut-être d'abord par un bon conditionnement (si le cadre le permet) des données utilisées.

Chapitre 6

Bilan et perspectives

Dans ce chapitre nous passons d'abord en revue les algorithmes que nous avons affectivement programmés. Nous examinons ensuite les liens et les différences qui existent entre les algorithmes de calcul de graphe de visibilité de scènes polygonales et de scènes composées d'objets courbes.

Enfin, nous faisons le point sur l'utilisation du complexe de visibilité pour résoudre des problèmes de visibilité et proposons quelques directions de recherche pour améliorer la résolution de ces problèmes.

— *Et maintenant en route vers de nouvelles aventures !*
Le commissaire Bougret et son fidèle adjoint l'inspecteur Charolles.

6.1 Réalisations

Comme le chapitre 5 l'a notamment montré, nous ne nous sommes pas arrêtés à l'aspect théorique des problèmes, mais nous avons aussi étudié l'aspect pratique des algorithmes et nous les avons notamment programmés. Nous détaillons dans cette section quelques aspects de la programmation des algorithmes et de leur réutilisation dans d'autres programmes.

Les programmes ont été écrits en C++, ce qui a permis d'utiliser la programmation orientée objet pour écrire les programmes. La programmation orientée objet permet de regrouper les différents éléments des structures de données en classes, ce qui les rend plus facile à manipuler et à intégrer dans les programmes. De plus l'utilisation de l'héritage permet de mieux gérer des éléments ayant des propriétés de bases communes. Nous avons utilisé le langage C++ tel qu'il est décrit dans la (future) norme ANSI et nos programmes ont été compilés avec le compilateur GNU `g++`.

6.1.1 Programmation du complexe de visibilité

Nous avons tout d'abord programmé l'algorithme de balayage optimal du graphe de visibilité d'une scène polygonale. Nous avons détaillé les structures de données utilisées pour représenter un arbre d'horizon dans le chapitre 2, lors de l'étude de la complexité de l'algorithme.

Pour profiter des relations de symétrie entre les deux arbres d'horizon et éviter de recopier inutilement des fonctions similaires, nous avons représenté l'arbre d'horizon inférieur par l'arbre d'horizon supérieur de la scène pivotée de 180°. Les deux arbres utilisent les mêmes fonctions définies dans une classe de base et les quelques fonctions différentes utilisées lors de l'initialisation ont été définies dans les deux classes dérivées correspondant aux deux types d'arbres. La figure 6.1 montre les deux arbres d'horizon initiaux d'une scène et fait apparaître ces liens entre les deux arbres.

Pour que le programme de balayage soit facilement réutilisable par un utilisateur, nous avons mis à la disposition de ce dernier des fonctions de rappel (callback). Lorsque le programme passe une arête e du graphe de visibilité, il appelle la fonction `passEdge(e)` qui peut être redéfinie par l'utilisateur. L'utilisateur adapte alors cette fonction selon ses besoins ; elle peut servir à construire effectivement le graphe de visibilité, ou simplement servir à afficher le graphe comme dans la figure 6.2 lorsqu'elle est redéfinie comme suit par exemple :

```
window w;
void passEdge(const edge* e)
{
  w.MoveTo(e->coord()->x(),e->coord()->y());
  w.LineTo(e->vis()->coord()->x(),e->vis()->coord()->y());
}
```

Nous avons ensuite utilisé le programme de balayage de graphe pour programmer la construction du complexe de visibilité, comme indiqué à la fin du chapitre 2. Les structures de données utilisées par le complexe de visibilité sont simples : chaque élément du complexe – sommet, arête et face – possède un pointeur vers les éléments

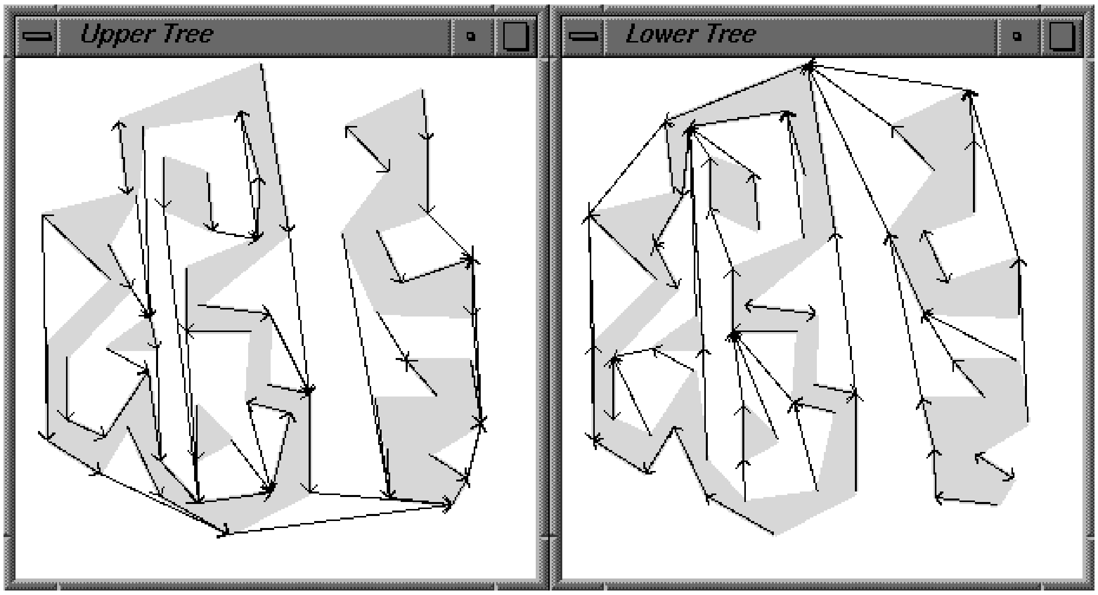


FIG. 6.1 – *Programme de calcul de graphe de visibilité : arbres d'horizon initiaux d'une scène.*

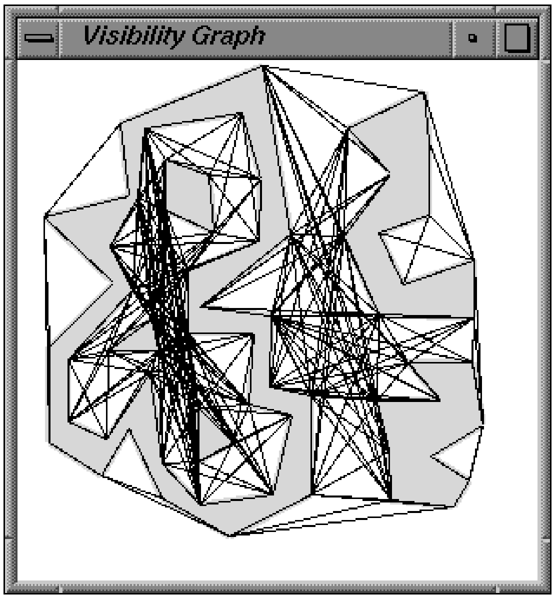


FIG. 6.2 – *Programme de calcul de graphe de visibilité : affichage du graphe.*

qui lui sont incidents : un sommet possède quatre pointeurs vers ses arêtes incidentes, une arête possède deux pointeurs vers ses sommets extrémités et des pointeurs vers ses faces incidentes, et une face possède deux pointeurs vers ses sommets extrêmes. Les six différents types d'arêtes sont dérivés à partir d'une classe de base qui fournit l'interface commune à toutes les arêtes.

Pour que le complexe puisse être utilisé ensuite dans des calculs de visibilité, il doit permettre l'accès à ses éléments. À cet effet, notre structure de données permet d'accéder aux deux faces semi-infinies du complexe et permet d'accéder pour chaque point de la scène à l'arête de la coupe initiale du complexe associée à ce point. Ces points d'accès permettent ensuite de faire des calculs de vue par exemple.

Pour les utilisateurs qui font des calculs globaux de visibilité et qui ont besoin de parcourir tous les éléments du complexe, nous avons programmé un itérateur qui permet d'effectuer un balayage topologique du complexe déjà construit : cet itérateur permet d'accéder successivement à tous les sommets du complexe de façon cohérente.

Bien que le complexe calculé sur un intervalle de directions de longueur π contienne toutes les informations nécessaires pour faire les calculs de visibilité, il n'est pas très pratique à utiliser : ses éléments sont coupés en deux aux bornes de l'intervalle. Nous avons alors calculé le complexe sur un intervalle de longueur 2π et « recollé » les morceaux coupés aux bornes de l'intervalle. Ainsi les calculs se font de façon continue dans $\mathbb{R}/2\pi$ sans problèmes de continuité aux limites.

6.1.2 Programmation des algorithmes de calculs de visibilité

La première utilisation du complexe que nous avons programmée est le calcul d'une vue autour d'un point. Pour rendre l'utilisation du programme flexible, nous avons défini la classe `view_iterator` qui calcule la vue selon le modèle d'un itérateur qui permet d'accéder successivement aux arêtes du complexe coupées par la courbe duale du point de vue. L'utilisateur décide ensuite de ce qu'il veut faire des arêtes fournies par l'itérateur.

Pour afficher par exemple la vue comme sur la figure 6.3, il suffit d'écrire le morceau de programme suivant :

```
window w;
complex c(nv,np,tabp);               //construction complexe
..........
point p(x,y);                        //point de vue
view_iterator it(p,c);               //iterateur de calcul de vue
while (it)
{
  w.MoveTo(x,y);
  w.LineTo((*it).x(),(*it).y());
  ++it;
}
```

Nous avons ensuite programmé les deux algorithmes de maintien dynamique d'une vue autour d'un point. Les programmes réutilisent le programme de calcul de vue pour calculer la liste initiale des arêtes du complexe coupées par la courbe duale du point de vue ; ils maintiennent ensuite cette liste quand le point de vue se déplace. L'algorithme de maintien lors de petits déplacements et l'algorithme de maintien le long d'une trajectoire ont été respectivement mis en œuvre par les classes `dynamic_view` et `traject_view`. Ces deux classes ont une fonction membre `moveTo(x,y)` qui permet de

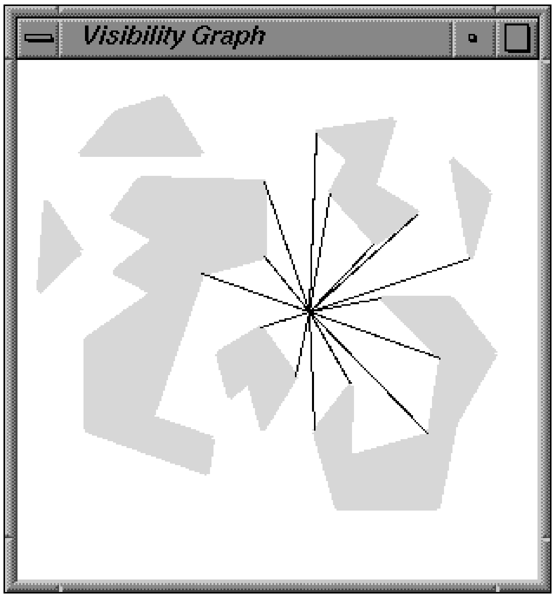


FIG. 6.3 – *Programme de calcul de vue : affichage de la vue.*

déplacer le point de vue.

Elles fournissent à l'utilisateur des itérateurs pour parcourir à un instant donné les sommets de la vue courante. Comme le but de l'algorithme de maintien le long d'une trajectoire est de signaler les changements de visibilité le long de cette trajectoire, la méthode `moveTo(x,y)` de la classe `traject_view` appelle une fonction de rappel à chaque changement de visibilité élémentaire. Cette fonction de rappel peut être redéfinie par l'utilisateur selon ses besoins.

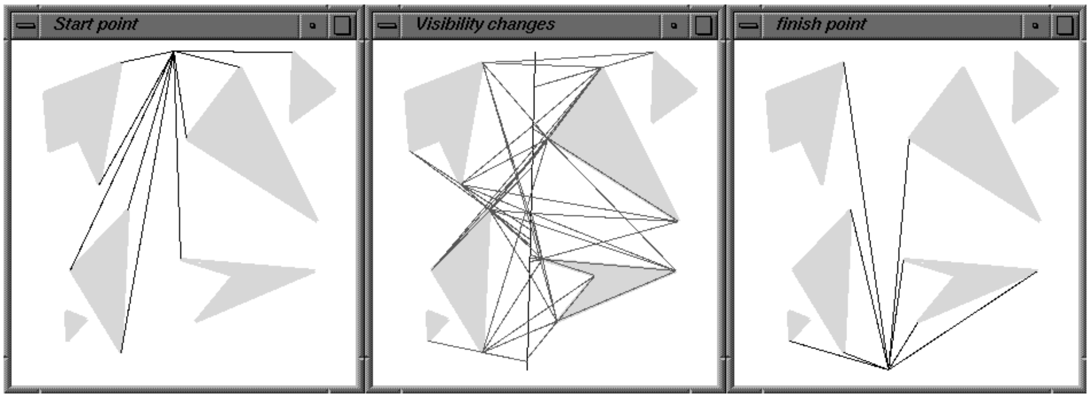


FIG. 6.4 – *Programme de maintien dynamique de vue le long d'une trajectoire.*

La figure 6.4 illustre l'exécution de l'algorithme de maintien de vue le long d'une trajectoire. Dans la première (resp. dernière) fenêtre, nous avons affiché la vue de départ (resp. d'arrivée) du point de vue, et dans la fenêtre du milieu, nous avons indiqué les limites de changement de visibilité traversées par le point de vue lorsqu'il se déplace en ligne droite depuis le point de départ jusqu'au point d'arrivée.

6.1.3 Utilisation du complexe de visibilité dans le calcul de radiosité

Les programmes que nous avons développés, notamment le programme de calcul du complexe de visibilité, ont été utilisés en synthèse d'images pour calculer l'éclairage d'une scène par la méthode de radiosité.

La méthode de radiosité est une méthode de calcul d'illumination globale de la scène. Dans cette méthode, tous les éléments d'une scène sont considérés comme émettant de l'énergie lumineuse : ils peuvent émettre une énergie lumineuse propre (source de lumière) et surtout ils renvoient une partie de l'énergie lumineuse qu'ils reçoivent des autres éléments de la scène.

Pour effectuer les calculs, la scène est divisée en éléments simples P_i. L'hypothèse est faite que l'énergie lumineuse totale émise en chaque point de la surface d'un élément est constante. Ce flux d'énergie – appelé radiosité – est noté B_i. L'énergie propre émise par un élément est notée E_i (figure 6.5a). Un élément est aussi considéré comme étant une surface diffuse, renvoyant la lumière de façon uniforme dans l'espace. Les éléments P_i sont alors caractérisés par le coefficient ρ_i de réflectance, coefficient indiquant la fraction de l'énergie renvoyée par la surface (figure 6.5b).

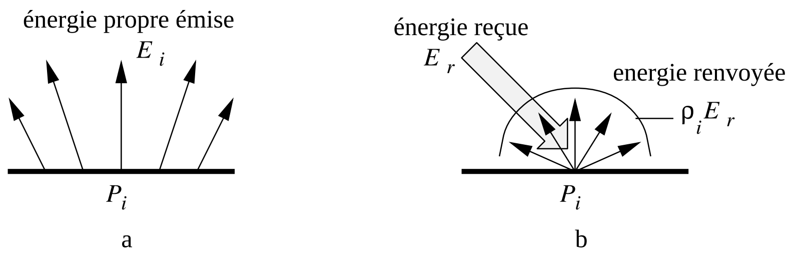


FIG. 6.5 – *Calcul de radiosité.* **a.** *Énergie propre émise par un élément.* **b.** *Énergie renvoyée par une surface diffuse.*

La méthode de radiosité cherche à calculer les valeurs de radiosité B_i de chaque élément. Ces valeurs vérifient l'équation de radiosité

$$B_i = E_i + \rho_i \sum_j F_{ij} B_j,$$

où F_{ij} est le facteur de forme entre les éléments P_j et P_j défini par

$$F_{ij} = \frac{1}{A_i} \int_{x \in P_i} \int_{y \in P_j} \frac{\cos\theta \cos\theta'}{\pi r^2} V(x,y)\, dy\, dx \ ,$$

où A_i désigne la surface de l'élément P_i, r est la distance $r = \|x, y\|$, θ_i (resp. θ_j) désigne l'angle entre le vecteur $\overrightarrow{xy}$ et la normale à P_i (resp. P_j) (figure 6.6a) et $V(x,y)$ est la fonction de visibilité entre les points : $V(x,y) = 1$ si x et y sont mutuellement visibles, $V(x,y) = 0$ si x et y sont cachés l'un de l'autre (figure 6.6b).

Le facteur de forme est une quantité purement géométrique et représente en quelque sorte l'ensemble des rayons allant de P_i à P_j. Une fois les facteurs de forme calculés,

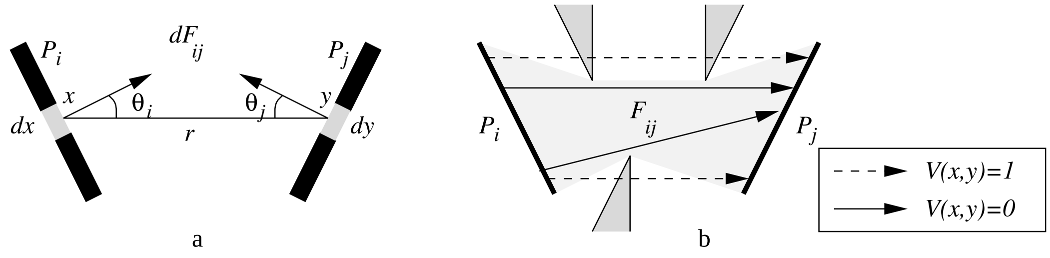


FIG. 6.6 – *Calcul de facteur de forme.* **a.** *Facteur de forme élémentaire entre deux éléments de surface sur P_i et P_j.* **b.** *Fonction de visibilité.*

il ne reste plus qu'à résoudre un système d'équations linéaires pour obtenir les valeurs de radiosité.

Le coût le plus important des algorithmes de calcul de radiosité est celui du calcul des facteurs de forme, ou plus précisément du calcul de la fonction de visibilité lors du calcul des facteurs de forme : il faut tester si les éléments d'intégration sont mutuellement visibles, et ce test demande des calculs de visibilité coûteux. Or le complexe de visibilité contient déjà ces calculs de visibilité. En particulier, les rayons allant de P_i à P_j à prendre en compte dans le calcul du facteur de forme F_{ij} sont ceux de la face du complexe d'étiquette (P_i, P_j). Le calcul des facteurs de forme se fait alors en considérant les faces du complexe de visibilité. Orti [ORDP96] a étudié comment calculer efficacement les facteurs de forme en utilisant le complexe de visibilité et a réutilisé nos programmes pour développer un environnement de calcul de radiosité de scènes polygonales 2D.

6.2 Polygones et objets courbes

Pocchiola et Vegter ont créé et étudié le complexe de visibilité dans le cadre de scènes composées d'objets courbes convexes. Nous examinons ici les rapports et les différences qui existent entre polygones et objets courbes dans les calculs de visibilité, notamment au niveau du calcul du graphe de visibilité.

Un polygone convexe peut être considéré comme un objet courbe convexe un peu particulier : l'objet n'est pas défini par une courbe mais par plusieurs morceaux de courbe. Dans certains travaux par exemple, les polygones sont considérés comme des approximations de courbes. En effet, les calculs de visibilité à effectuer sur les objets courbes, notamment les calculs de tangente, peuvent être délicats à mener en pratique : il faut résoudre des équations algébriques de degré élevé. Les calculs sont alors effectués sur des suites de polygones approximant de plus en plus les objets courbes : le calcul d'une tangente à un polygone convexe prend au plus un temps logarithmique en fonction du nombre de sommets du polygone.

Le parti que nous avons pris dans nos travaux est donc différent de ces approches. L'objet de base n'est pas le polygone mais le côté de polygone. Cela permet si nécessaire d'associer des propriétés différentes (couleur par exemple) aux côtés d'un même polygone. Un polygone ne peut plus alors être considéré comme *un* objet courbe. Par exemple, étant donné une direction θ il y a deux tangentes à un objet courbe, tangentes qui délimitent deux changements de visibilité (figure 6.7a). En revanche, pour un polygone à n sommets il y a n droites délimitant un changement de visibilité : ce sont les droites passant par les sommets du polygone (figure 6.7b).

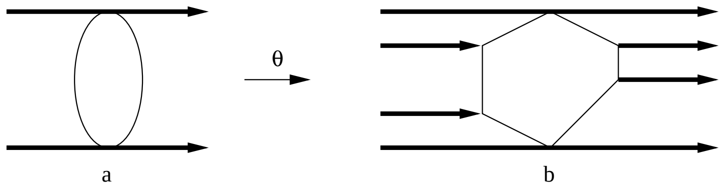


FIG. 6.7 – *Changements de visibilité selon une direction θ.* **a.** *Objet courbe : deux tangentes.* **b.** *Polygone : n droites passant par ses n sommets.*

Étant donné ces différences entre objets courbes et polygones, peut-on quand même réutiliser des techniques développées dans un cadre pour les appliquer dans l'autre cadre ? Nous discutons ici de la transposition des arbres d'horizon dans le cadre d'objets courbes et de la transposition des pseudo-triangulations dans le cadre de polygones.

De la même façon que nous associons à chaque segment de droite $[p,q]$ deux courbes duales p^* et q^* dans l'espace dual (figure 6.8a), il est possible d'associer deux courbes duales à un objet courbe convexe.

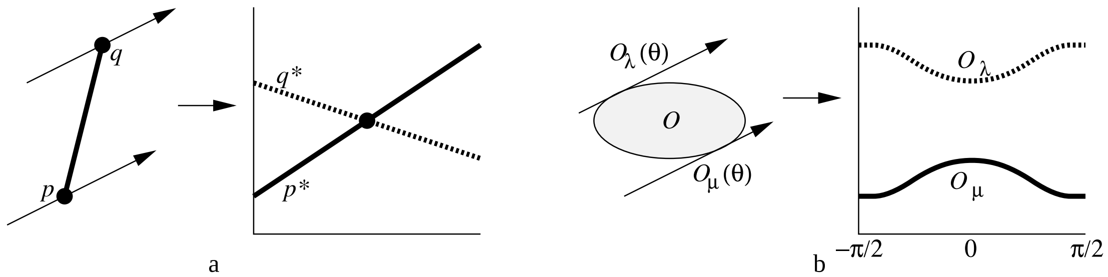


FIG. 6.8 – *Courbes duales.* **a.** *D'un segment de droite.* **b.** *D'un objet courbe convexe.*

Les courbes duales p^* et q^* représentent dans la scène l'ensemble des droites passant par p et q, c'est-à-dire l'ensemble des tangentes au segment $[p,q]$. Pour un objet courbe convexe O, il existe pour chaque direction θ deux tangentes $O_\mu(\theta)$ et $O_\lambda(\theta)$ à O de direction θ. La tangente $O_\mu(\theta)$ (resp. $O_\lambda(\theta)$) est telle que l'objet est au-dessus (resp. en dessous) de la tangente et est dite de type μ (resp. λ). Dans l'espace dual, quand θ varie les deux points duaux associés aux deux tangentes $O_\mu(\theta)$ et $O_\lambda(\theta)$ décrivent deux courbes duales O_μ et O_λ (figure 6.8b).

La dualité associe alors à une scène d'objets courbes un arrangement dual et Pocchiola [Poc90] s'en est servi par exemple pour faire des calculs de visibilité de façon similaire à ceux effectués avec l'arrangement dual d'une scène de polygones. Les courbes duales d'objets courbes convexes sont aussi les courbes support des arêtes du complexe de visibilité de la scène d'objets courbes.

Malgré ces similitudes, les arrangements duaux d'objets courbes ont une nature différente des arrangements duaux de polygones. L'arrangement dual d'une scène de polygones (pour un intervalle de directions de longueur π) est un arrangement de droites, ou de pseudo-droites, c'est-à-dire de courbes ayant deux à deux une seule intersection. L'arrangement dual d'une scène d'objets courbes convexes (pour un intervalle de directions de longueur 2π) est en revanche un arrangement de pseudo-cercles, c'est-à-dire de courbes ayant deux à deux deux points d'intersection, qui ne peut être ramené à un arrangement de pseudo-droites : l'intervalle séparant les deux points d'intersections de deux courbes est de longueur quelconque.

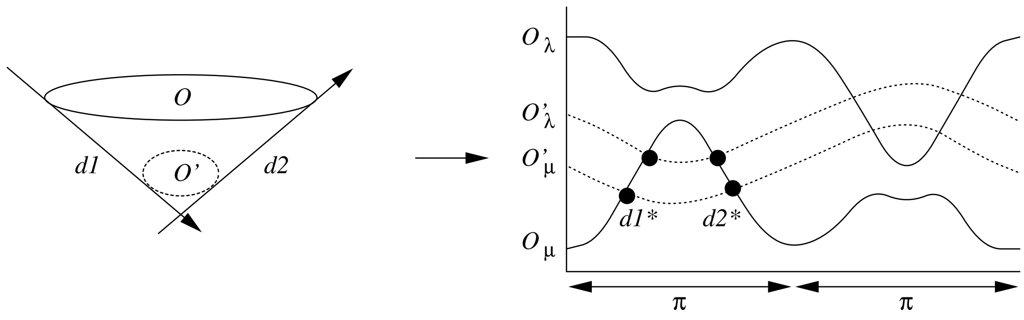


FIG. 6.9 – *Arrangement de courbes duales d'objets convexes.*

La figure 6.9 montre par exemple l'arrangement d'une scène de deux objets O et O' où les courbes O_μ et O'_μ ont leur deux points d'intersection dans l'intervalle $[-\pi/4, \pi/4]$.

Cette différence de nature entre les deux types d'arrangements modifie les propriétés des arbres d'horizon lorsque nous les transposons dans le cadre d'objets courbes convexes. Pour construire l'arbre d'horizon supérieur à partir d'une direction θ_0, nous étendons pour chaque objet O sa tangente $O_\mu(\theta_0)$ (resp. $O_\lambda(\theta_0)$) et la faisons tourner autour de O (dans le sens trigonométrique direct) jusqu'à ce qu'elle devienne tangente à un autre objet situé devant O. Chaque arête de l'arbre d'horizon est donc une bitangente, et la bitangente parent d'une bitangente de type O_μ (resp. O_λ) arrivant en O est la bitangente de type O_μ (resp. O_λ) partant de O.

Le graphe obtenu n'est malheureusement plus un arbre. La figure 6.10 montre l'arbre d'horizon supérieur d'une petite scène d'objets et la représentation combinatoire (figure 6.10b) montre que l'« arbre » d'horizon n'est plus connexe et comporte des cycles.

Les propriétés géométriques sont elles aussi modifiées. Par exemple, en voulant passer la bitangente $\lambda_1\mu_2$ de l'arbre d'horizon de la figure 6.10(a), nous voyons que les arêtes de l'enveloppe convexe inférieure des objets 3,4 et 5 ne font pas toutes partie de l'arbre d'horizon.

Des problèmes interviennent aussi à cause d'une autre différence entre les deux courbes duales d'un segment de droite et celles d'un objet courbe : la figure 6.8 nous montre que les courbes duales p^* et q^* des extrémités du segment se croisent alors que les courbes duales O_μ et O_λ de l'objet courbe restent séparées.

Les problèmes apparaissent quand nous devons passer une arête qui fait partie de l'enveloppe convexe de la scène. Dans le cas de polygones, l'arête suivante de cette arête

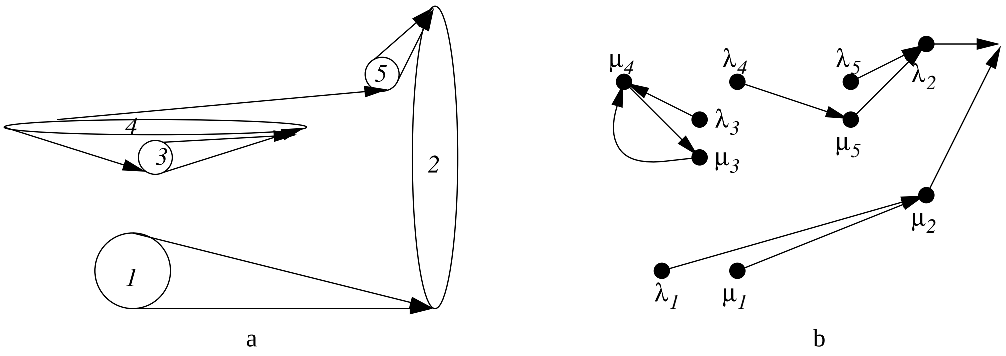


FIG. 6.10 – « *Arbre d'horizon* » *pour une scène d'objets courbes.* **a.** *Représentation géométrique.* **b.** *Représentation combinatoire.*

(p, q) est l'arête précédente de l'enveloppe convexe partant de p (figure 6.11a), alors que dans le cadre d'objets courbes, l'arête suivante ne fait pas partie de l'enveloppe convexe (figure 6.11b) : dans le premier cas, l'arête « pivote » autour de p alors que dans le deuxième cas, l'arête « glisse » autour de O.

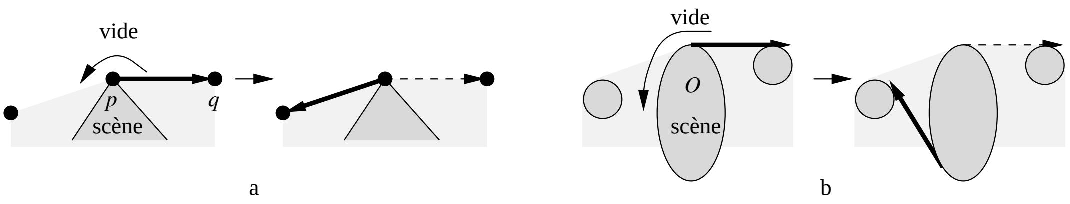


FIG. 6.11 – *Passage d'une arête sur l'enveloppe convexe.* **a.** *Scène de polygones.* **b.** *Scène d'objets courbes convexes.*

Pour transformer l'arrangement dual d'une scène d'objets courbes en arrangement de pseudo-droites, Pocchiola et Vegter [PV94] considèrent de nouveaux objets, les pseudo-triangles :

un pseudo-triangle est un ensemble de $\mathbb{R}^2$ borné et simplement connexe tel que :

- *sa frontière est composée d'une suite de trois courbes convexes tangentes en leurs extrémités ;*
- *l'ensemble est contenu dans le triangle formé par les trois extrémités de ces courbes.*

À chaque pseudo-triangle est associé une courbe duale, et l'ensemble des courbes duales des pseudo-triangles forme alors un arrangement de pseudo-droites facilement balayable topologiquement. Le lien entre les pseudo-triangles et la scène est donné par le concept de pseudo-triangulation :

une pseudo-triangulation d'un ensemble d'objets courbes convexes est un ensemble maximal de bitangentes libres non sécantes. (figure 6.12)

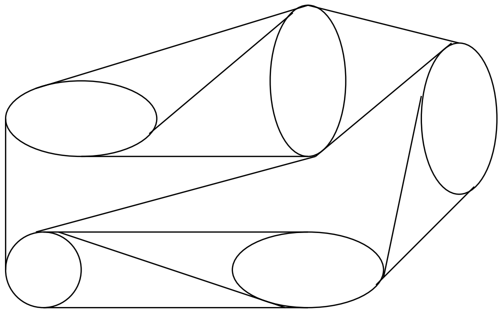


FIG. 6.12 – *Pseudo-triangulation d'une scène d'objets convexes.*

Les pseudo-triangulations découpent l'espace libre de la scène en pseudo-triangles et le balayage topologique de l'arrangement dual d'une pseudo-triangulation permet de balayer toutes les bitangentes de la scène.

Pocchiola et Vegter [PV96a] utilisent ensuite les pseudo-triangulations pour balayer le complexe, et donc le graphe, de visibilité de la scène en temps optimal. Deux pseudo-triangles adjacents forment un pseudo-quadrangle dont la bitangente commune est une diagonale du pseudo-quadrangle ; en basculant la bitangente dans l'autre diagonale du pseudo-quadrangle on obtient deux autres pseudo-triangles (figure 6.13). Si les deux pseudo-triangles considérés font partie d'une pseudo-triangulation, on obtient alors après la bascule de la bitangente une nouvelle pseudo-triangulation.

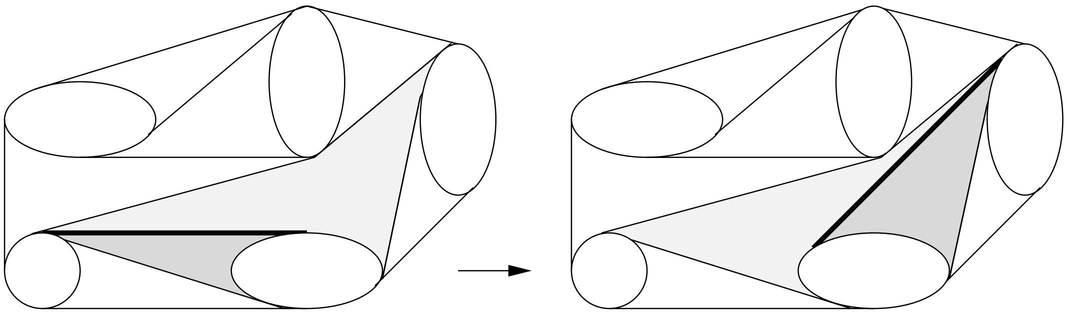


FIG. 6.13 – *Bascule d'une bitangente incidente à deux pseudo-triangles.*

L'algorithme de balayage de graphe de visibilité consiste à passer d'une pseudo-triangulation à une autre en basculant une bitangente : l'algorithme guide le choix de la bitangente à basculer et permet ainsi de basculer une à une toutes les bitangentes du graphe de visibilité de la scène. Cet algorithme peut être vu comme un algorithme de parcours des sommets du complexe de visibilité. Ce parcours est cependant effectué d'une façon différente de celui effectué par notre algorithme : quand un sommet v du complexe est passé, le sommet suivant considéré est le sommet extrême droit de la face dont v est le sommet extrême gauche.

Les différences entre objets courbes et polygones se font aussi sentir quand nous voulons transposer les pseudo-triangulations dans des scènes polygonales : une pseudo-triangulation devient alors une « vraie » triangulation avec des propriétés différentes.

Une pseudo-triangulation a moins de bitangentes qu'une triangulation : une pseudo-triangulation de n objets courbes contient $3n-3$ bitangentes alors qu'une triangulation de n_p polygones ayant au total n sommets contient $2n - 3 + 3n_p - n_{conv}$ arêtes, où n_{conv} est le nombre d'arêtes de l'enveloppe convexe.

Ces différences sont visibles dans la figure 6.14 où nous avons affiché une pseudo-triangulation d'une scène de trois ellipses et la triangulation « correspondante » quand les ellipses sont remplacées par des segments de droite.

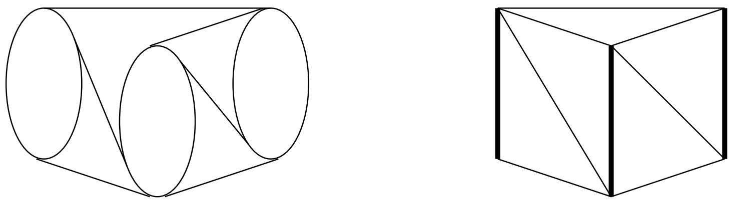


FIG. 6.14 – *Différences entre* **a.** *Une pseudo-triangulation, et* **b.** *Une triangulation.*

Ces différences sont dues au fait que toutes les arêtes ayant un point p pour extrémité sont non sécantes (figure 6.15a) alors que des tangentes à d'un objet O et de même type (μ ou λ) peuvent être sécantes (figure 6.15b).

FIG. 6.15 – *Tangentes* **a.** *Passant par un même point et non sécantes.* **b.** *À un même objet courbe et sécantes .*

Si nous appliquons quand même l'algorithme de bascule des diagonales des quadrangles à une triangulation, nous ne parcourons pas toutes les arêtes du graphe de visibilité. En effet, une arête (p, q) du graphe de visibilité peut, dans toutes les triangulations possibles de la scène, séparer deux triangles dont l'union forme un quadrangle concave. La seconde diagonale du quadrangle est alors située à l'extérieur du quadrangle et soit coupe un obstacle de la scène (figure 6.16a), soit coupe une arête de la triangulation (figure 6.16b), soit est déjà une arête de la triangulation (figure 6.16c).

L'arête (p, q) ne peut alors jamais être basculée et n'est donc pas visitée par l'algorithme.

Les ensembles de tangentes à un objet courbe convexe et les ensembles de « tangentes » aux côtés d'un polygone, c'est-à-dire de droites passant par les sommets du polygone, ont des propriétés fondamentalement différentes (même si le polygone est réduit à un segment de droite). Ces différences fondamentales viennent notamment du

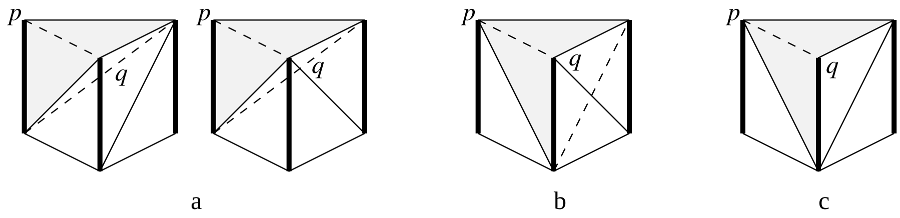


FIG. 6.16 – *Problèmes avec la diagonale extérieure d'un quadrangle.* **a.** *Diagonale coupant un obstacle.* **b.** *Diagonale coupant une arête de la triangulation.* **c.** *Diagonale déjà présente dans la triangulation.*

fait que les tangentes à un objet courbe « glissent » autour de l'objet alors que les tangentes d'un segment passent par une extrémité fixe du segment.

Les structures de données créées dans le cadre d'objets courbes (resp. polygones) dont les propriétés permettent de résoudre des problèmes de visibilité perdent alors leur propriétés lorsqu'elles sont transposées dans le cadre de polygones (resp. d'objets courbes) et y deviennent inutilisables.

6.3 Complexe de visibilité : bilan et perspectives

Nous faisons dans cette partie le point sur l'utilisation du complexe de visibilité pour effectuer des calculs de visibilité plus rapidement et évoquons des points qui restent à améliorer.

6.3.1 Utilité du complexe de visibilité

Le complexe de visibilité est un outil puissant pour résoudre certains problèmes de visibilité, en particulier des problèmes de mise à jour de visibilité. Le complexe est une structure de données relativement simple : il ne dépend que de la géométrie de la scène et ne contient pas de structures combinatoires annexes (excepté si nécessaire des arbres binaires de recherche). C'est notamment pour cela qu'il ne permet pas d'améliorer directement les temps de calcul de requêtes discrètes telles que le lancer de rayons. Il est d'ailleurs moins pratique à utiliser dans ce type de requêtes, car il prend une place mémoire proportionnelle à la taille du graphe de visibilité de la scène alors que les structures classiques utilisées pour faire du lancer de rayon n'occupent qu'une place linéaire en fonction du nombre de segments de la scène.

Le complexe de visibilité est en revanche très bien adapté aux calculs continus de visibilité, car il permet d'exploiter la cohérence spatiale de la scène ou la cohérence temporelle des déplacements dans la scène. Il permet d'avoir en plus une description de la visibilité dans l'espace objet (la scène) de façon indépendante de toute visualisation ultérieure.

Le complexe est aussi très utile dans des calculs faisant intervenir toutes les relations de visibilité entre objets de la scène car il contient déjà toutes ces relations. Nous avons par exemple évoqué son utilisation dans des calculs de radiosité.

Malgré tous ses avantages, le complexe de visibilité doit être employé avec précaution, notamment à cause de sa taille. Nous avons déjà évoqué le problème de la taille du complexe de visibilité : sa taille est proportionnelle à la taille du graphe de visibilité, soit une taille en $\Omega(n)$ et $O(n^2)$. L'utilisation du complexe de visibilité sur de grandes scènes peut se révéler problématique. Si les scènes sont relativement denses et que la majorité des objets sont cachés les uns des autres alors le complexe est très utile : il prend un espace mémoire raisonnable et permet d'effectuer des calculs de visibilité de façon vraiment plus rapide que s'il fallait dans ces calculs considérer à chaque fois tous les objets de la scène.

En revanche si les objets sont espacés et se voient pratiquement tous les uns les autres, le complexe de visibilité occupe une place mémoire quadratique et la performance gagnée sur les calculs de visibilité est moindre.

6.3.2 Complexe de visibilité hiérarchique

Tous les calculs de visibilité effectués avec le complexe de visibilité sont exacts, c'est-à-dire que tous les changements de visibilité sont pris en compte. Or un calcul exact peut être trop précis. Dans un calcul de vue par exemple, deux changements de visibilité intervenant dans les directions θ et $\theta + \varepsilon$ peuvent devenir indistinguables et être confondus en un seul changement de visibilité. Ce phénomène est du à l'existence d'une limite de résolution (ici angulaire) en dessous de laquelle les choses sont confondues. Cette limite de résolution est aussi bien matérielle (résolution en pixels d'un écran), physique (résolution d'une pellicule photo) que biologique (résolution de l'oeil). Il devient alors inutile de calculer des détails en deçà de cette résolution, détails qui ne seront pas perçus.

Pour éviter de faire des calculs « visuellement inutiles », il ne faut plus tout calculer, mais seulement ce qui est pertinent. Il ne faut cependant pas perdre des informations qui se révéleraient indispensables par la suite. Pour cela, il ne faut pas faire d'approximations dans le calcul même mais plutôt dans la modélisation de la scène : pour avoir des calculs rapides à un niveau de détails fixé, les calculs sont toujours menés de façon précise, mais ils sont effectués sur une autre scène moins détaillée que la scène originale (figure 6.17). Cependant il doit toujours être possible à partir d'une scène « d'approximation » de remonter à la scène détaillée originale.

La création de scènes à différents niveaux de détails s'effectue grâce à un regroupement hiérarchique des objets. Un groupe d'objets est assimilé à un seul *méta-objet* au niveau de hiérarchie supérieur, les méta-objets peuvent à leur tour être regroupés en un seul objet au niveau supérieur... En haut de la hiérarchie se trouve la version la plus simplifiée de la scène, et tout en bas de la hiérarchie se trouve la scène la plus détaillée, c'est-à-dire la scène originale. Le niveau de hiérarchie à choisir pour un groupe d'objets donné lors d'un calcul de visibilité dépend du niveau de détail adopté pour ce calcul.

Nous exposons des idées sur le calcul et l'utilisation du complexe de visibilité de scènes hiérarchiques.

Comme nous manipulons des polygones, nous pouvons regrouper un ensemble $(O_{i_j})_j$ de polygones en un polygone O_i dont les sommets ont été pris parmi les sommets des

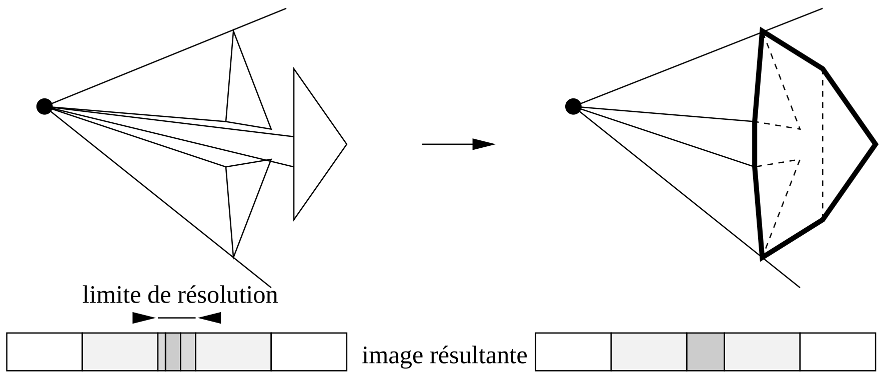
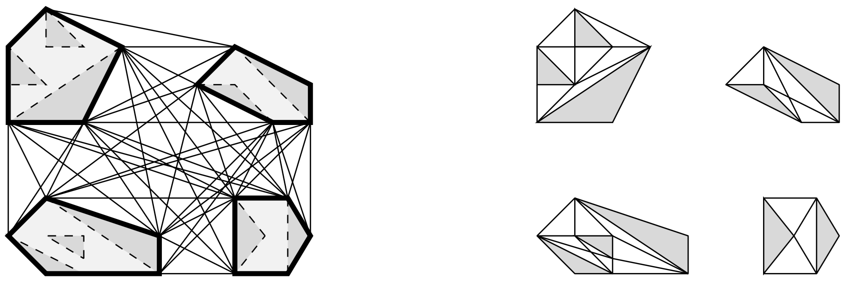


FIG. 6.17 – *Limite de résolution et calcul de vue à différents niveaux de détails.*

polygones O_{i_j}. Considérons maintenant une scène formée d'un ensemble de polygones $(O_i)_i$ où chaque polygone O_i représente un groupe de polygones $(O_{i_j})_j$. Nous calculons alors le complexe de visibilité de la scène composée des polygones O_i (figure 6.18a) et nous calculons le complexe de visibilité de chaque groupe de sous-polygones $(O_{i_j})_j$ (figure 6.18b).



a b

FIG. 6.18 – *Complexe hiérarchique.* **a.** *Calcul du complexe sur les méta-polygones.* **b.** *Calcul du complexe poour chacun des groupes de polygones.*

Comme les sommets des polygones sont des sommets de sous-polygones, les arêtes du complexe de visibilité de la scène sont portées par des courbes duales qui portent aussi les arêtes des sous-complexes. Ces courbes duales permettent alors de relier le complexe aux sous-complexes. Une arête e du complexe de la scène correspondant à un sommet de O_i est découpée dans le sous-complexe des polygones $(O_{i_j})_j$ en sous-arêtes. Nous associons alors à e la liste des sous-arêtes la décomposant.

Cela permet de faire par exemple un calcul de vue hiérarchique. La vue est effectuée en utilisant le complexe de la scène, et la liste des arêtes du complexe correspondant à la vue est gardée en mémoire. Si la visibilité entre deux points consécutifs p et q de la vue situés sur un même polygone doit être calculée avec plus de détails, alors grâce à liste de sous-arêtes des arêtes e_p et e_q associées à p et q, nous calculons les arêtes associées à p et q dans le sous-complexe du polygone, et calculons la vue entre ces deux arêtes (figure 6.19).

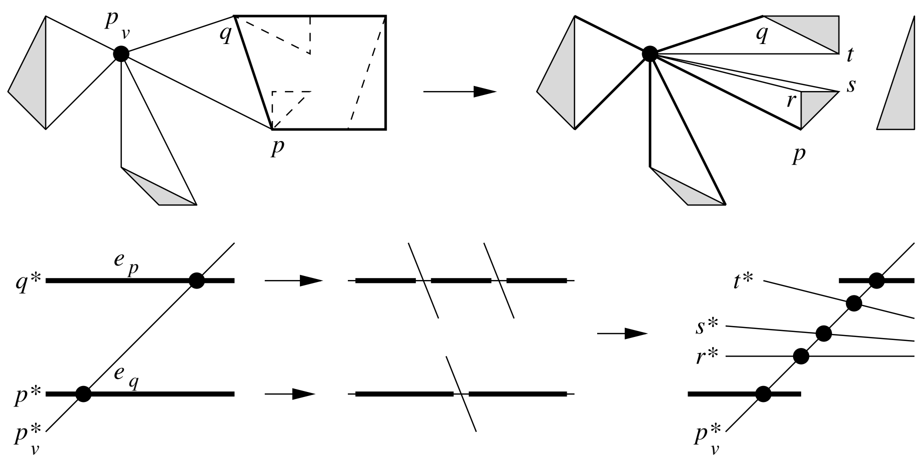


FIG. 6.19 – *Calcul de vue hiérarchique dans le complexe de visibilité hiérarchique.*

Nous envisageons la hiérarchisation du complexe de visibilité non seulement comme un moyen pour effectuer des calculs plus rapides selon le niveau de détails demandé, mais aussi comme un moyen permettant de rendre le complexe plus utilisable en pratique sur de grandes scènes.

En effet, nous avons déjà fait remarqué que la place mémoire occupée par le complexe pouvait être élevée. La hiérarchisation, telle que nous l'envisageons, permet de réduire cette place mémoire sans pénaliser lourdement les calculs de visibilité. Au lieu de calculer le complexe de tous les sous-polygones, nous calculons seulement le complexe des méta-polygones et les complexes des sous-ensembles de polygones de ces méta-polygones, ce qui prend au total une place mémoire nettement moins élevée. Nous n'avons pas alors dans le complexe hiérarchique les calculs de visibilité entre sous-polygones de méta-polygones différents. Si les méta-polygones sont relativement éloignés les uns des autres, alors ces calculs sont en général trop précis et la visibilité entre méta-polygones suffit comme approximation. Sinon, ces calculs peuvent être effectués rapidement grâce aux informations déjà présentes dans le complexe hiérarchique.

Il faudra aussi, pour tirer pleinement profit du complexe hiérarchique, développer des critères et des tests permettant de définir à chaque instant des calculs de visibilité à quel niveau de détails ces calculs doivent être effectués.

6.3.3 Maintien topologique du complexe de visibilité

Nous avons exposé un algorithme de balayage topologique du graphe de visibilité qui permet de calculer ce dernier en temps optimal. Le but d'un balayage topologique est de faire un balayage cohérent, sans être obligé de passer à chaque fois l'élément minimal global. Le balayage topologique est plus rapide : il permet d'éviter l'utilisation d'une queue de priorité et le coût logarithmique supplémentaire qui en découle. Il est aussi plus souple : plusieurs éléments peuvent être passés à chaque étape et même tous ces éléments peuvent être passés simultanément en parallèle.

Les algorithmes de maintien de visibilité dynamique – notamment le maintien dynamique du complexe de visibilité – que nous avons exposés utilisent toujours un balayage droit, c'est-à-dire que les événement sont traités dans l'ordre de leur date d'apparition. Or, quand nous cherchons à maintenir le complexe de visibilité lorsque les objets se déplacent, certains changements élémentaires de visibilité sont indépendants : l'ordre de mise à jour de deux changements de visibilité se produisant dans deux endroits mutuellement cachés de la scène est indifférent. Il serait alors intéressant d'avoir des structures de données qui permettent de déterminer quels sont tous les changements de visibilité qui peuvent être traités simultanément à un instant donné. Cela permettraient alors de ne plus utiliser de queue de priorité.

6.3.4 Le mot de la fin

Le complexe de visibilité n'est pas qu'une structure théorique : il existe vraiment, nous l'avons programmé :-) Bien que ce soit un outil puissant pour faire des calculs de visibilité, comme tout outil il a ses limites d'application et doit être employé à propos. Dans le cas de grandes scènes complexes, le complexe peut paradoxalement se révéler être trop puissant : contenant toutes les relations de visibilité entre les objets de la scène, il peut alors être trop grand et trop précis. L'utilisation du complexe hiérarchique, ou d'autres méthodes permettant de ne calculer complètement que les complexes de visibilité de parties de scènes et de faire rapidement des calculs de visibilité inter-scènes à un niveau de détails suffisant, peuvent alors concilier les avantages et les inconvénients du complexe de visibilité.

Références bibliographiques

[AT81] Avis (D.) et Toussaint (G. T.). – An efficient algorithm for decomposing a polygon into star-shaped pieces. *Pattern Recognition*, vol. 13, 1981, pp. 295–298.

[BJMM93] Benouamer (M.), Jaillon (P.), Michelucci (D.) et Moreau (J.-M.). – A lazy solution to imprecision in computational geometry. *In : Proc. 5th Canad. Conf. Comput. Geom.*, pp. 73–78. – Waterloo, Canada, 1993.

[CG89] Chazelle (B.) et Guibas (L. J.). – Visibility and intersection problems in plane geometry. *Discrete Comput. Geom.*, vol. 4, 1989, pp. 551–581.

[CGL85] Chazelle (B.), Guibas (L. J.) et Lee (D. T.). – The power of geometric duality. *BIT*, vol. 25, 1985, pp. 76–90.

[Chv75] Chvátal (V.). – A combinatorial theorem in plane geometry. *J. Combin. Theory Ser.*, vol. B 18, 1975, pp. 39–41.

[CPT96] Chiang (Y.-J.), Preparata (F. P.) et Tamassia (R.). – A unified approach to dynamic point location, ray shooting, and shortest paths in planar maps. *SIAM J. Comput.*, vol. 25, 1996, pp. 207–233.

[DY93] Dobrindt (K.) et Yvinec (M.). – Remembering conflicts in history yields dynamic algorithms. *In : Proc. 4th Annu. Internat. Sympos. Algorithms Comput. (ISAAC 93)*. pp. 21–30. – Springer-Verlag.

[EG86] Edelsbrunner (H.) et Guibas (L. J.). – Topologically sweeping an arrangement. *In : Proc. 18th Annu. ACM Sympos. Theory Comput.*, pp. 389–403.

[EG89] Edelsbrunner (H.) et Guibas (L. J.). – Topologically sweeping an arrangement. *J. Comput. Syst. Sci.*, vol. 38, 1989, pp. 165–194. – Corrigendum in 42 (1991), 249–251.

[EM90] Edelsbrunner (H.) et Mücke (E. P.). – Simulation of simplicity: a technique to cope with degenerate cases in geometric algorithms. *ACM Trans. Graph.*, vol. 9, 1990, pp. 66–104.

[EOS86] Edelsbrunner (H.), O'Rourke (J.) et Seidel (R.). – Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, vol. 15, 1986, pp. 341–363.

[GHL+87] Guibas (L. J.), Hershberger (J.), Leven (D.), Sharir (M.) et Tarjan (R. E.). – Linear-time algorithms for visibility and shortest path problems inside triangulated simple polygons. *Algorithmica*, vol. 2, 1987, pp. 209–233.

[GM91] Ghosh (S. K.) et Mount (D. M.). – An output-sensitive algorithm for computing visibility graphs. *SIAM J. Comput.*, vol. 20, 1991, pp. 888–910.

[GSS89] Guibas (L. J.), Salesin (D.) et Stolfi (J.). – Epsilon geometry: building robust algorithms from imprecise computations. *In : Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pp. 208–217.

[HM90] Heffernan (P. J.) et Mitchell (J. S. B.). – Structured visibility profiles with applications to problems in simple polygons. *In : Proc. 6th Annu. ACM Sympos. Comput. Geom.*, pp. 53–62.

[LPW79] Lozano-Pérez (T.) et Wesley (M. A.). – An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, vol. 22, 1979, pp. 560–570.

[O'R87] O'Rourke (J.). – *Art Gallery Theorems and Algorithms.* – New York, NY, Oxford University Press, 1987.

[ORDP96] Orti (R.), Rivière (S.), Durand (F.) et Puech (C.). – Radiosity for dynamic scenes in flatland with the visibility complex. *Comput. Graph. Forum*, vol. 15, n° 3, 1996, pp. 237–248. – Proc. Eurographics '96.

[OvL81] Overmars (M. H.) et van Leeuwen (J.). – Maintenance of configurations in the plane. *J. Comput. Syst. Sci.*, vol. 23, 1981, pp. 166–204.

[OW88] Overmars (M. H.) et Welzl (E.). – New methods for computing visibility graphs. *In : Proc. 4th Annu. ACM Sympos. Comput. Geom.*, pp. 164–171.

[Poc90] Pocchiola (M.). – Graphics in Flatland revisited. *In : Proc. 2nd Scand. Workshop Algorithm Theory.* pp. 85–96. – Springer-Verlag.

[PV93] Pocchiola (M.) et Vegter (G.). – Sweep algorithm for visibility graphs of curved obstacles. – manuscrit, 1993.

[PV94] Pocchiola (M.) et Vegter (G.). – *Order types and visibility types of configurations of disjoint convex plane sets.* – Rapport technique n° LIENS-94-4, Liens, Ecole Norm. Sup., Paris, janvier 1994.

[PV96a] Pocchiola (M.) et Vegter (G.). – Topologically sweeping visibility complexes via pseudo-triangulations. *Discrete Comput. Geom.*, décembre 1996. – To appear in the special issue devoted to ACM-SoCG'95.

[PV96b] Pocchiola (M.) et Vegter (G.). – The visibility complex. 1996. – To appear in the special issue devoted to ACM-SoCG'93.

[Sam84] Samet (H.). – The quadtree and related hierarchical data structures. *ACM Comput. Surv.*, vol. 16, n° 2, juin 1984.

[Sei94] Seidel (R.). – The nature and meaning of perturbations in geometric computing. *In : Proc. 11th Annu. Sympos. on Theorical Aspects of Computer Science (STACS 94).* pp. 3–17. – Springer-Verlag.

[ST86] Sarnak (N.) et Tarjan (R. E.). – Planar point location using persistent search trees. *Commun. ACM*, vol. 29, 1986, pp. 669–679.

[Veg90] Vegter (G.). – The visibility diagram: A data structure for visibility problems and motion planning. *In : Proc. 2nd Scand. Workshop Algorithm Theory.* pp. 97–110. – Springer-Verlag.

[Yapre] Yap (C.). – *Fundamental Problems in Algorithmic Algebra.* – Princeton University Press, à paraître.

[Yap90] Yap (C. K.). – A geometric consistency theorem for a symbolic perturbation scheme. *J. Comput. Syst. Sci.*, vol. 40, 1990, pp. 2–18.

[YD95] Yap (C.) et Dubé (T.). – *Computing in Euclidean Geometry.* – World Scientific, 1995, 2 édition, *Lecture Notes Series on Computing*, volume 1.

Résumé

Beaucoup de programmes de visualisation, de planification de trajectoire, etc., utilisent intensivement des calculs de visibilité. Si ces calculs de visibilité ne constituent qu'une petite partie de ces programmes, ils sont en revanche responsables d'une grande partie du temps d'exécution de ces programmes : leur efficacité est donc cruciale.

Les algorithmes traditionnels de calculs de visibilité ont deux défauts : ils effectuent – inutilement – des calculs sur des objets non visibles et refont tous ces calculs à chaque nouvelle requête, même si les changements avec la requête précédente sont minimes.

Pour remédier à ces inconvénients dans le cadre de scènes polygonales bidimensionnelles, nous nous servons d'une structure de données – le complexe de visibilité – qui code toutes les relations de visibilité entre objets d'une scène. Après avoir montré comment construire de façon optimale le complexe de visibilité, nous montrons comment il permet d'utiliser la cohérence spatiale de la scène dans les calculs de polygones de visibilité. Nous montrons aussi comment il permet d'utiliser la cohérence temporelle dans le maintien d'une vue autour d'un point se déplaçant dans la scène.

Nous étudions ces algorithmes non seulement d'un point de vue théorique mais aussi d'un point de vue pratique. Nous avons programmé ces algorithmes et effectué des comparaisons expérimentales entre algorithmes. Nous nous sommes aussi intéressés aux problèmes de dégénérescences et d'imprécisions des calculs numériques qui se posent dès que les programmes sont exécutés sur des données « réelles ».

Mots clés : Géométrie algorithmique, visibilité, analyse d'algorithmes, comparaison expérimentale, traitement des cas dégénérés, traitement des imprécisions numériques.

Abstract

The core part of computer programs such as visualization softwares, rendering engines or robotics path planners lies in visibly computations. Although the corresponding algorithms are only a small part of the whole application, they are responsible for most of the running time and their efficiency is therefore crucial.

Traditional visibility computation algorithms have two drawbacks: They perform useless computations on invisible objects, and recompute everything from scratch for every single query, even if the changes undergone by two successive solutions are minor.

We show how to remedy these problems in the context of polygonal scenes in the plane by using the *visibility complex* data structure, which encodes all the visibility relations between objects of the scene. First, we present an optimal algorithm for computing the visibility complex. Second, we show how to use this data structure to compute visibility polygons and to maintain views around a moving point in the scene.

Both the theoretical complexity and the practical performances of our algorithms are presented. Finally, we explain how to handle the degeneracies and numerical imprecisions caused by real-world data sets.

Keywords: Computational geometry, visibility, algorithmic complexity, experimental analysis degeneracies, numerical imprecisions.

Table des matières

Liste des figures