Modélisation procédurale de mondes virtuels par pavage d'occultation

par
Dorian Gomez

Thèse de doctorat effectuée en cotutelle internationale

au

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences
Université de Montréal

et à

l'Institut de Recherche en Informatique de Toulouse (IRIT-CNRS UMR 5505)
Université de Toulouse III Paul Sabatier

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiæ Doctor (Ph.D.)
en informatique
et à
l'Université Toulouse III Paul Sabatier
en vue de l'obtention du grade de Docteur
en informatique

Avril, 2014

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

Modélisation procédurale de mondes virtuels par pavage d'occultation

présentée par:

Dorian Gomez

a été évaluée par un jury composé des personnes suivantes:

Victor Ostromoukhov,	président, représentant du doyen de la FES
Pierre Poulin,	directeur de recherche
Mathias Paulin,	codirecteur
Eric Galin,	rapporteur
Kadi Bouatouch,	rapporteur
Loïc Barthe,	membre du jury

Thèse acceptée le: 23 Avril 2014

RÉSUMÉ

Cette thèse porte sur la modélisation procédurale de mondes virtuels étendus dans le domaine de l'informatique graphique. Nous proposons d'exploiter les propriétés de visibilité entre régions élémentaires de la scène, que nous appelons tuiles, pour contrôler sa construction par pavage rectangulaire. Deux objectifs distincts sont visés par nos travaux : (1) fournir aux infographistes un moyen efficace pour générer du contenu 3D pour ces scènes virtuelles de très grande taille, et (2) garantir, dès la création du monde, des performances de rendu et de visualisation efficace.

Pour cela, nous proposons plusieurs méthodes de détermination de la visibilité en 2D et en 3D. Ces méthodes permettent l'évaluation d'ensembles potentiellement visibles (PVS) en temps interactif ou en temps réel. Elles sont basées sur les calculs de lignes séparatrices et de lignes de support des objets, mais aussi sur l'organisation hiérarchique des objets associés aux tuiles. La première technique (2D) garantit l'occultation complète du champ visuel à partir d'une distance fixe, spécifiée par le concepteur de la scène, depuis n'importe quel endroit sur le pavage. La seconde permet d'estimer et de localiser les tuiles où se propage la visibilité, et de construire le monde en conséquence. Afin de pouvoir générer des mondes variés, nous présentons ensuite l'extension de cette dernière méthode à la 3D. Enfin, nous proposons deux méthodes d'optimisation du placement des objets sur les tuiles permettant d'améliorer leurs propriétés d'occultation et leurs impacts sur les performances de rendu tout en conservant l'atmosphère créée par l'infographiste par ses choix de placement initiaux.

Mots clés : *Occlusion culling*, Visibilité, Modélisation procédurale, Pavage, *PVS*, *BVH*, *CHC++*

Procedural Modeling of Virtual Worlds using Occlusion Tiling

ABSTRACT

This thesis deals with procedural modeling applied to extended worlds for computer graphics. We study visibility applied to tiling patterns, aiming at two distinct objectives : (1) providing artists with efficient tools to generate 3D content for very extended virtual scenes, and (2) guaranteeing that this content improves performance of subsequent renderings, during its construction.

We propose several methods for 2D and 3D visibility determination, in order to achieve interactive or real-time evaluation of potentially visible sets (PVS). They are based on the concepts of separating and supporting lines/planes, as well as objects hierarchies over tiles. Our first 2D method guarantees full occlusion of the visual field (view frustum) beyond a fixed distance, regardless of the observer's location on a tiling. The second method enables fast estimation and localization of visible tiles, and builds up a virtual world accordingly. We also extend this method to 3D. Finally, we present two methods to optimize objects locations on tiles, and show how to improve rendering performance for scenes generated on the fly.

Keywords : Occlusion Culling, Visibility, Procedural Modeling, Tiling, PVS, BVH, CHC++

RÉSUMÉ VULGARISÉ

Cette thèse porte sur la génération automatique de mondes virtuels étendus dans le domaine de l'informatique graphique. Nous étudions les phénomènes de visibilité afin d'assurer de hautes performances d'affichage. Dans un premier temps, nous élaborons des techniques permettant de faire l'économie de l'affichage des objets non visibles depuis le point de vue, afin qu'ils ne surchargent pas le processeur graphique. Dans un second temps, nous mettons en place des outils de modélisation pour les infographistes afin de générer simplement des scènes virtuelles en fonction de ces propriétés de visibilité. Enfin, nous optimisons le placement des objets contenus dans ces scènes en fonction des performances d'affichage que l'on souhaite obtenir.

POPULARIZED ABSTRACT

This thesis deals with automatic creation of extended virtual worlds in computer graphics. We study visibility phenomena to ensure high display performances. First, we elaborate techniques to prevent invisible objects from being displayed uselessly, so that graphics processor unit overload is avoided. Second, we set up modeling tools for artists so that they can generate virtual worlds simply according to these visibility properties. Finally, we optimize our virtual scenes' objects locations according to the display performances we want to reach.

TABLE DES MATIÈRES

REMERCIEMENTS

Cette thèse est le fruit d'une recherche qui a commencé en 2010, lorsque Mathias m'a recommandé auprès de Pierre afin de réaliser mon stage de master à l'étranger. C'est donc de là que tous mes travaux découlent, à partir d'une idée, simple, basée sur des petits carrés dans des grilles, vaguement griffonnés sur un bout de papier, sur un coin de bureau du laboratoire d'informatique graphique de l'université de Montréal.

Il est donc naturel pour moi de remercier mes deux directeurs de thèse, à savoir Pierre Poulin, professeur au département d'informatique et recherche opérationnelle (DIRO) de l'université de Montréal, et Mathias Paulin, professeur à l'institut de recherche en informatique de Toulouse (IRIT) pour avoir cru en mes capacités tout au long de ces trois ans et sept mois de thèse.

Je souhaite aussi remercier les membres du jury pour avoir évalué mes travaux de thèse, à savoir messieurs Bouatouch et Galin, qui ont relu ce document avec attention ainsi que messieurs Ostromoukhov et Barthe pour leurs remarques avisées.

Je désire également remercier les membres de l'équipe VORTEX et du DIRO, Loïc Barthe, David Vanderhaeghe, Derek Nowrouzezahrai, Houari Sahraoui, pour leur bonne humeur et leurs conseils.

Je veux également remercier les étudiants et collègues que j'ai pu cotoyer au DIRO de Montréal et à l'IRIT de Toulouse : Marion, Anthony, Philippe, Omar, Nicolas, Gilles-Philippe, Jonathan, Guillaume, Olivier, Mathieu, Samir, Jocelyn, Luc, Frédéric, Eric, Marc-Antoine, Rodolphe, Aude, Monia, Andra, Florian, Charly, Thomas, Even, Jérémie, Adrien, les 2 Arnaud, Cynthia, David et tous ceux avec qui nous avons partagé les expériences, le vin, le fromage, la biâère, les McDo, le karting, les parties de Magic, les cigarettes, les beignes et le café.

Je remercie ma famille et mes amis pour m'avoir soutenu moralement même s'ils ne comprenaient pas très bien ce que je faisais.

Merci à GIT, pour ses voyages transatlantiques plus fréquents que les miens. Merci au café pour son pragmatisme. Et à Ted, le petit ours du DIRO, pour son oreille attentive.

Merci à Mr. Robert Dawson pour m'avoir permis de citer ses oeuvres comme exemples

de pavages. Merci à Mr. Frederic St-Arnaud pour m'avoir permis de citer une de ses oeuvres comme exemple d'art conceptuel dans ma présentation.

Les travaux de cette thèse ont été rendus possibles grâce au financement du projet PROMO du NCE GRAND (Canada).

NOTATION

Certains mots ou expressions sont restés non traduits de l'anglais dans le but d'éviter les confusions. Ces derniers sont en italique dans le texte.

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ALGORITHMES

INTRODUCTION

Aujourd'hui, les domaines du cinéma, de la télévision et du jeu vidéo sont en pleine expansion et se substituent souvent même aux autres médias. Nous pouvons constater que, dans les trente dernières années, les images de synthèse ont pu donner à ces trois domaines de plus en plus de crédibilité et ce, grâce au réalisme croissant des méthodes utilisées en informatique graphique.

Cette tendance peut s'expliquer par la demande croissante de performance et de nouveauté du public, friand de nouvelles technologies. Tandis que les cinéphiles souhaitent poser les yeux sur des paysages imaginaires gigantesques aux détails finement sculptés et colorés, les passionnés de jeux vidéo préfèrent voyager dans un monde riche et varié, et ce, afin de favoriser leur immersion au sein d'un univers virtuel.

La création de certains de ces mondes prend cependant beaucoup trop de temps aux infographistes, surtout en ce qui concerne leur variabilité à très grande échelle. La modélisation procédurale dans le domaine de l'informatique graphique vise à solutionner ce problème en développant des méthodes de génération automatique de géométries, de textures, de paysages, etc. Ces méthodes sont souvent spécialisées dans la génération d'un certain type d'entités et, bien que certaines d'entre elles puissent être généralisées pour en générer plusieurs, cela se fait trop souvent au prix d'une perte de précision d'expression et de contrôle lors de l'élaboration du modèle.

Un monde synthétique étendu [93].

Les résultats des méthodes procédurales ont parfois besoin d'être stockés en mémoire ou sur disque dur ; la compression des données est alors un point à ne pas négliger car ces méthodes peuvent en produire en de très grandes quantités.

Parallèlement, le principe de pavage, souvent vu comme un outil récréationnel – l'exemple le plus connu étant le célèbre jeu des "dominos" ou les "triominos" – était très utilisé par les mathématiciens du début du siècle dernier. Il est également très présent tout autour de nous, et cela sans que l'on s'en aperçoive pour autant, dans les carrelages, planchers, tapisseries, les pixels d'un écran ou tout autre motif régulier ou irrégulier. Le pavage possède des propriétés intéressantes, notamment au niveau de sa régularité et de la fréquence de ses motifs.

Le pavage est également utilisé dans le domaine du jeu vidéo. Il permet d'offrir aux joueurs des univers vastes car il possède une propriété intéressante d'instanciation. Nous pouvons citer des exemples de jeux vidéo tels que *"Heroes of Might and Magic"* [102], *"Civilization"* [67] et *"Sim City"* [25] qui, en se basant sur le principe d'agencement de tuiles, permettent d'aménager, conquérir et/ou occuper un territoire. D'ailleurs, un ensemble restreint de seulement quelques dizaines de tuiles peut permettre d'obtenir des pavages de l'espace donnant l'illusion de non répétition.

Une des contraintes les plus importantes à prendre en compte lors du rendu de grandes scènes est le principe de visibilité. Une manière intuitive d'appréhender ce concept est, à partir d'un point de vue donné (ou d'une région de l'espace où le point de vue peut être librement déplacé), de déterminer quels objets d'une scène tridimensionnelle sont potentiellement visibles (en totalité ou partiellement), ou, de manière duale, lesquels seront entièrement occultés.

La visibilité est sans doute le domaine le plus documenté de l'informatique graphique et c'est pourquoi, il est curieux que personne n'ait encore associé celle-ci aux pavages. C'est cette lacune que nous proposons de combler au cours de cette thèse.

Notre problématique est la suivante : "Elaborer des méthodes de pavage de l'espace intégrées à des outils de modélisation procédurale pour la création de mondes vastes, qui permettent de garantir une limite au nombre de primitives géométriques envoyées au processeur graphique." Le but de cette thèse est d'apporter un nouveau point de vue sur

la notion de modélisation procédurale. Il consiste à assurer que le contenu des scènes créées garantisse de bonnes performances de rendu dès leur création.

Dans notre étude, nous nous intéressons tout d'abord à la visibilité sur des tuiles rectangulaires en 2D et examinons les possibilités de structures hiérarchiques applicables à ce type de tuiles. Puis, nous généralisons le processus à différentes couches afin d'obtenir une visibilité précalculée pour des mondes 2D$\frac{1}{2}$. Par la suite, nous étudions la propagation de la visibilité en 3D sur un pavage avec un minimum de données précalculées et ceci, grâce à l'encodage de l'occultation partielle que peut créer chaque tuile. Ensuite, nous mettons en avant différentes pistes d'outils de modélisation procédurale capables d'inclure le travail de l'infographiste au sein même du système de pavage d'occultation, le tout s'insérant dans son pipeline traditionnel de travail. Enfin, nous montrons comment créer, afficher et optimiser le positionnement des objets sur un pavage afin d'accélérer le calcul de visibilité.

Au 1er chapitre, nous présentons les travaux antérieurs en modélisation procédurale, sur la visibilité, ainsi que dans le domaine des pavages. Nous évoquons les avantages et inconvénients de certaines de ces méthodes, puis nous soulignons certains aspects importants dont nous nous servons pour nos contributions.

Ensuite, au chapitre 2, nous présentons nos travaux effectués à ce jour sur les pavages d'occultation, permettant de calculer la visibilité 2D depuis une cellule de vue et de garantir une limite fixe (e.g., premier anneau de voisinage) au nombre de polygones à afficher sur un monde généré à la volée.

Le chapitre 3 explique comment lever les limitations du premier anneau de voisinage et offre deux possibilités différentes d'extension du calcul de la visibilité à la 3D, tout en respectant la contrainte de génération du monde à la volée.

Enfin, au chapitre 4, nous montrons plusieurs solutions permettant d'optimiser le placement des objets présents dans les tuiles afin d'augmenter leur taux d'occultation et ainsi accélérer le rendu de nos mondes virtuels.

CHAPITRE 1

TRAVAUX ANTÉRIEURS

Ce chapitre couvre trois domaines sur lesquels reposent les travaux de cette thèse : la modélisation procédurale, la visibilité et le pavage. Comme les deux premiers sont particulièrement vastes, nous nous contentons d'énumérer leurs contributions principales au vu de nos propres contributions, détaillées dans les chapitres suivants.

1.1 Modélisation procédurale

La modélisation procédurale est un terme général désignant l'ensemble des méthodes utilisées pour générer, de manière automatique, divers modèles géométriques possédant des propriétés similaires, le contenu d'une texture, l'animation de déformations, etc.

Dans cette thèse, il est important de mettre en avant les méthodes existantes de modélisation procédurale qui sont les plus significatives car elles peuvent permettre de créer les objets que nous utilisons pour "peupler" l'espace de notre monde virtuel.

La modélisation procédurale a été l'un des domaines émergeants en informatique graphique durant ces dernières décennies. Certains travaux indépendants se spécialisent dans la modélisation de coquillages [28], la génération de façades [26], la construction de modèles simples d'édifices à partir d'empreintes réelles [57], la production de villes à partir d'extrusions [41] ou par simulation [59], et de génération procédurale de routes [30] et de réseaux routiers [10] et hydrauliques [31]. On y retrouve également de nombreux travaux sur la génération de plantes [27, 29, 86].

Plus récemment, Talton et al. [99] unifient la plupart des méthodes de modélisation basées sur l'approche Monte Carlo des chaînes de Markov, appliquées à l'inférence de grammaire. Merrell et Manocha [68, 69] et Peyrat et al. [80] développent des méthodes de génération basées sur l'exemple. D'autres systèmes, tels que celui proposé par Leblanc [58], utilisent une approche basée langage afin de garantir une haute qualité de topologie sur la géométrie.

Par ailleurs, le pavage est beaucoup utilisé en modélisation procédurale afin d'assurer la variabilité de l'apparence d'objets similaires ainsi que sur des motifs de grande envergure à différentes échelles. Deux exemples explicites proviennent de Peytavie et al. [81], qui utilisent un ensemble apériodique de tuiles (cf. définition 6) pour générer des structures composées de rochers, telles que des huttes, des murs, etc., ainsi que de Kopf et al. [51] qui utilisent un ensemble de tuiles de Wang pour générer du bruit bleu à différentes échelles.

Outre ces travaux, nous pouvons distinguer cinq grandes catégories de génération procédurale : les modèles basés sur les grammaires (systèmes-L et grammaires de formes), les fractales, les langages, les modèles basés sur l'exemple et ceux basés sur les pavages.

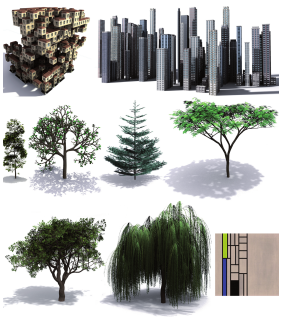
Nous allons maintenant passer en revue chacune de ces catégories.

1.1.1 Grammaires

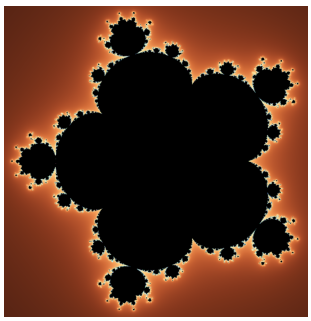
1.1.1.1 Systèmes-L

Les systèmes-L ont été introduits par Lindenmayer [60]. Un système-L est généralement composé d'un axiome de départ, de symboles et de règles de production (ou de réécriture). Ces systèmes sont similaires aux grammaires de Chomsky, à la différence que les règles sont appliquées en parallèle.

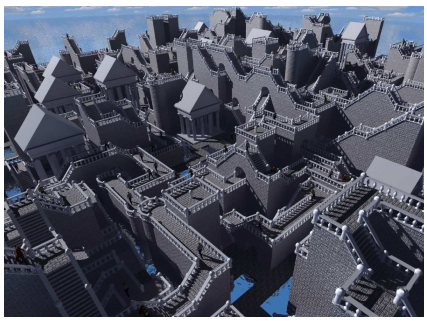
Les systèmes-L ont d'abord été utilisés dans la production d'arbres [5, 9, 83, 84] et la simulation de leur croissance [73], puis dans la modélisation de plantes et de fleurs [27,



(a) Modélisation procédurale par Metropolis [99]



(b) Fractale de Mandelbrot réalisée avec *Fraqtive* [70].



(c) Génération à partir d'exemples [68].

Figure 1.1 – Différents modèles procéduraux.

29, 86] et leur croissance [84], ou encore la génération de feuilles [42, 80]. Ils sont aussi utilisés dans l'art topiaire [85], mais également, à une échelle supérieure, dans la simulation d'écosystèmes [20, 82].

On peut aussi utiliser les systèmes-L pour concevoir des réseaux routiers [11, 78] et des édifices [65], même si ces derniers sont surtout créés à partir des grammaires de formes.

1.1.1.2 Grammaires de formes

Les grammaires de formes [34, 96, 97] sont similaires aux systèmes-L sauf que les règles de production ne sont plus appliquées en parallèle, mais en séquence. De plus les symboles utilisés par la grammaire sont des formes géométriques, caractérisées par un volume, une orientation, et possiblement d'autres attributs géométriques, texturaux, d'apparence, etc.

Wonka et al. [107] introduisent les grammaires de partitionnement pour modéliser des façades d'édifices. Müller et al. [72] intègrent plusieurs méthodes sur la génération de villes et de façades pour introduire la CGA (*computer graphics architecture*). Les temps de génération des villes par ces outils sont interactifs et permettent une bonne prise en main par l'utilisateur, même si le contrôle n'est pas aussi intuitif pour les infographistes qu'il serait souhaité.

Talton et al. [99] uniformisent l'ensemble des méthodes de modélisation procédurale par chaînes de Markov de Monte Carlo (MCMC) basée sur l'inférence de grammaire. À partir de simples croquis ou d'une voxélisation approximative d'un objet, ils réussissent à générer des modèles visuellement acceptables tels des modèles d'arbres et de villes (cf. figure 1.1(a)). Ils parviennent notamment à intégrer toutes les variantes des MCMC telles que le saut réversible (RJMCMC), le recuit simulé, le recuit simulé parallèle et des heuristiques sur le choix des symboles non terminaux et le rejet retardé de chaînes non valides. Ce genre de méthode reste très efficace et général si le critère du temps de génération des objets n'est pas important (de quelques minutes, pour la génération d'une peinture sur image peu complexe, jusqu'à plusieurs heures pour certains modèles d'arbres).

Globalement parlant, la modélisation procédurale par grammaire nécessite de concevoir des grammaires assez complexes et/ou peu intuitives pour parvenir à faire converger le modèle vers les souhaits de l'utilisateur. De plus, cette méthodologie n'est valable que pour la génération individuelle d'objets et ne convient pas à des méthodes d'optimisation de placement d'objets. En effet, ces dernières ne prennent pas en compte les interactions de visibilité entre les objets lors de leur création.

1.1.2 Fractales

Dans un modèle fractal, on retrouve le ou les mêmes motifs à chaque échelle. Ce genre de structure peut aisément s'obtenir par l'utilisation récursive des mêmes règles de grammaire. La fractale est un exemple concret d'objet autosimilaire (cf. figures 1.1(b) et 1.7(b)).

Définition 1. *Un objet **autosimilaire** est un objet qui conserve sa forme, quelle que soit l'échelle à laquelle on l'observe.*

Les fractales ont une structure aussi détaillée qu'on le souhaite à n'importe quelle échelle. Par ailleurs, elles sont trop irrégulières pour être facilement décrites dans le langage euclidien traditionnel. Toutefois, elles ont l'avantage d'avoir une définition simple et récursive, ce qui permet de modéliser facilement certains objets par une grammaire récursive, tels que les fougères, les arbres, les terrains, les réseaux de rivières, les nuages, les flocons de neige ou d'autres structures cristallines. Elles demeurent toutefois appliquées à des modèles spécialisés particuliers.

1.1.3 Génération basée sur l'exemple

Merrell et Manocha [68, 69] développent un système où l'utilisateur doit seulement fournir un modèle servant d'exemple. Ainsi, la synthèse de modèles peut être utilisée pour créer des objets symétriques, changeant au cours du temps, et des modèles soumis à des contraintes. Cette méthode se rapproche des algorithmes de synthèse de textures, mais diffère pourtant en deux points. Le premier point est que cette méthode effectue une recherche globale de conflits avant d'ajouter des polygones au modèle. Le second

point est qu'elle divise le problème de génération d'un modèle complexe en de plus petits problèmes plus faciles à résoudre. Les modèles de la figure 1.1(c) ont été générés entre quelques secondes et une demi-heure, dépendant du nombre de variations que l'on souhaite obtenir.

Maréchal et al. [80] présentent une méthode permettant de générer une grande variété d'objets de même nature à partir d'un modèle initial. Ce processus de génération de variétés consiste à découper l'objet initial en fragments. La géométrie et la texture sont ensuite modifiées avant d'être stockées dans un atlas. Les fragments contenus dans l'atlas sont alors recombinés par instanciation pour générer un très grand nombre de modèles différents pour un faible surcoût mémoire.

Une limitation importante de ce genre de méthode est qu'elle nécessite une décomposition manuelle de l'objet en plusieurs morceaux. De plus, si cette décomposition n'est pas réalisée de manière soigneuse, les nouveaux modèles générés peuvent être identiques au modèle en entrée. Finalement, tous les modèles ne se prêtent pas à de telles décompositions et réassemblages de façon générique.

1.1.4 Génération basée sur les langages

L'utilisation d'un langage est une approche de la modélisation procédurale moins populaire mais potentiellement très puissante. Ce langage est régi par des opérations spécifiques à un ou plusieurs types de modèles de base.

Cutler et al. [18] proposent une méthode de sculpture où un objet peut être modifié par l'application d'un ensemble d'opérations de simulation (e.g., par éléments finis). Le solide peut subir des déformations ou des bris causés par la réaction de forces externes. Un exemple montrant le résultat d'une simulation de sculpture est illustré à la figure 1.2.

Le GML (*generative modeling language*) [44, 45] est un langage de modélisation procédurale qui s'inspire du langage *PostScript*. Un maillage polygonal de surfaces de subdivision de type Catmull-Clark est utilisé comme primitive géométrique de base. Ce langage offre différentes opérations mathématiques (fonctions), mais aussi des opérations plus complexes telles des opérations booléennes entre primitives. Ce système se base sur les opérations d'Euler (assemblage de plusieurs morceaux pour former un ob-

jet) et d'extrusions. L'élaboration d'un modèle via cette méthode requiert donc de définir toute sa topologie, mais permet de générer des objets à tout niveau de complexité, par exemple, pour la création d'édifices [47]. Un exemple de modèle procédural créé par ce langage est montré à la figure 1.3(a).

Un dernier exemple de modélisation par langage est celui de Leblanc [58]. Il utilise un principe basé sur des blocs, assurant une spécification simple de la topologie, une définition volumique valide, un bon contrôle de la surface à l'aide des sommets des blocs et de fonctions (ou des textures) de déplacement (*displacement mapping*).

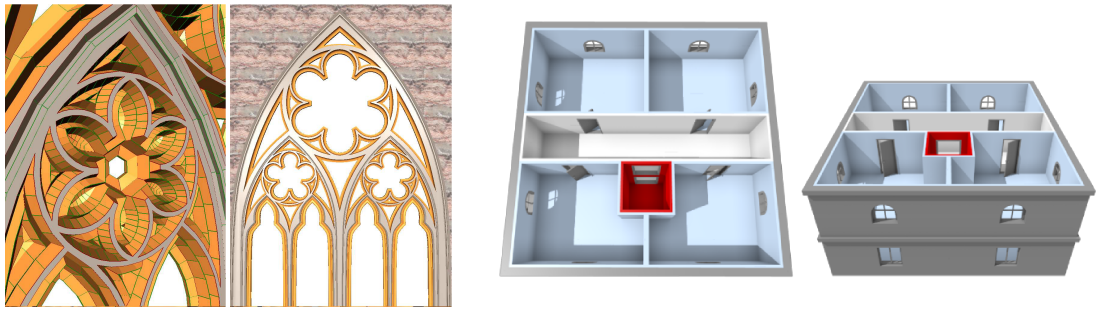
Cette modélisation par divers langages spécialisés demeure assez générale pour produire plusieurs types de modèles avec de bonnes propriétés géométriques, et même volumiques dans le cas de Leblanc. Elle demeure néanmoins encore peu adoptée par la communauté graphique, sans doute parce qu'elle requiert une bonne expertise en programmation, ce qui n'est pas souvent le point fort des infographistes.

1.1.5 Génération basée sur l'instanciation de tuiles

L'instanciation par tuiles – vue plus en détail en section 1.3 – est un outil très utilisé dans la synthèse de textures procédurales, mais l'est aussi dans la modélisation de géométries complexes.

Les textures procédurales ne sont pas définies par une image fournie au préalable mais par une procédure ou un algorithme. Contrairement aux textures habituelles, celles-

Figure 1.2 – Un modèle obtenu par simulation de sculpture [18].



(a) Modèle obtenu grâce au langage GML [45].

(b) Modèle par composants de Leblanc [58].

Figure 1.3 – Génération procédurale basé sur un langage.

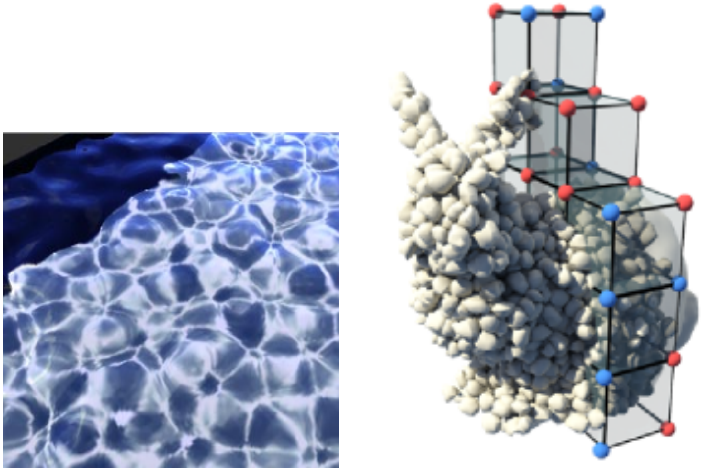
ci n'ont pas de taille, ni de résolution fixe et peuvent être facilement paramétrisées. De plus, elles sont capables de générer une grande variété de matériaux tels le bois, le marbre et la pierre. Ebert et al. [24] donnent une excellente vue d'ensemble de tous les travaux effectués dans ce domaine.

Stam [94] utilise des pavages afin de produire des caustiques sur une surface plane (cf. figure 1.4(a)). Il précalcule un ensemble fini de tuiles dont les instances sont distribuées de manière apériodique sur cette surface, permettant ainsi de diminuer le coût de stockage en mémoire mémoire par rapport à une procédure de génération de texture standard (convolution, texture de solide, etc.).

Par ailleurs, Peytavie et al. [81] utilisent une version modifiée des cubes de coins – présentés en section 1.3 – afin de construire des métastructures à partir de rochers (cf. figure 1.4(b)). Cette méthode s'effectue en deux étapes. D'abord, ils précalculent un ensemble apériodique (cf. définition 6) de tuiles composées de rochers en garantissant le contact entre les rochers voisins. Ensuite, ils génèrent des piles de rochers en pavant virtuellement l'espace avec ces tuiles et en n'instanciant que les rochers qui intersectent un objet de support.

1.2 Visibilité

La visibilité en informatique graphique regroupe l'ensemble des méthodes permettant de déterminer si un objet, un polygone, un fragment est visible depuis un point de



(a) Application de texture [94].

(b) Métastructures de roches par cubes de coins [81].

Figure 1.4 – Génération procédurale par pavage.

vue donné. Ces méthodes ont pour but de réaliser des économies d'affichage et gagner en performance. Par ailleur, la complexité algorithmique de chacune de ces méthodes représente un défi en soit. C'est pourquoi elles sont souvent décrites comme étant un des facteurs les plus limitants en terme de performance de rendu. Ainsi, la littérature sur le sujet, et particulièrement dans le domaine du rendu, est extrêmement vaste et s'étend bien au-delà de nos objectifs. Le lecteur intéressé peut se référer aux revues de littérature de Cohen-Or et al. [13] et Bittner and Wonka [8] pour une étude plus exhaustive des méthodes datant déjà de dix ans. Cependant, peu de travaux récents sont apparus, et nous les soulignons dans cette section lorsque nécessaire. Nous ne couvrons que les éléments nécessaires à l'élaboration de nos contributions, mais tout de même en énumérant les apports de différentes approches reliées.

1.2.1 Définitions

La plupart des notions évoquées dans cette section ont pour but de donner une vue d'ensemble des termes les plus usités en matière de visibilité, et qui reviendront régulièrement dans les chapitres qui suivent.

La suppression des surfaces non visibles ou *culling* permet notamment d'accélérer le

rendu en n'affichant que les polygones qui contribuent directement à la vue courante de la scène (cf. figure 1.5). Pour être efficace, il faut que le test de visibilité soit plus rapide que d'afficher l'objet caché lui-même. Il faut également que le nombre d'objets cachés dans une scène soit assez grand afin de rentabiliser le temps passé à déterminer s'ils sont effectivement visibles.

Le *culling* peut s'effectuer depuis un seul point de vue, tout comme depuis une région ou cellule regroupant un ensemble de points de vue.

Trois types de *culling* existent depuis un point de vue :

La suppression des faces arrières ou *back-face culling.* Implémentée aujourd'hui directement au niveau matériel, cette méthode teste tout simplement l'orientation des polygones (énumération des sommets dans le sens horaire ou anti-horaire) par rapport à la direction de vue. Elle permet de diminuer considérablement la charge du pipeline graphique, typiquement de moitié pour une scène constituée d'objets fermés. Toutefois, cette méthode n'est pas généralisable à une hiérarchie d'objets, sauf éventuellement par l'utilisation de cônes de normales (les normales des polygones sont regroupées en cônes qui englobent toutes les directions de ces normales, et ces cônes sont stockés dans un arbre).

La suppression des faces ou objets situées en-dehors de la pyramide de vue ou *frustum culling.* Elle teste la position d'un polygone (ou objet) par rapport aux plans de coupe (*clipping planes*) de la pyramide de vue pour ne préserver que les polygones qui se trouvent en totalité ou partiellement à l'intérieur de celle-ci. Un algorithme trivial consiste à tester si la boîte englobante d'un ou plusieurs polygones (ou objets) est à l'extérieur d'un des plans de coupe de la pyramide. Toutefois, même si cette boîte englobante intersecte la pyramide de vue, un polygone (ou objet) appartenant à celle-ci peut chevaucher deux plans de la pyramide de vue (donc non externe à aucun des deux) tout en résidant à l'extérieur de la pyramide de vue. Ainsi, un algorithme de *culling* sera considéré comme conservateur si l'on fait le choix de dessiner malgré tout un polygone (ou objet) potentiellement non visible, sans réaliser de test de visibilité supplémentaire. Cet algorithme se généralise aisément au traitement d'une hiérarchie de boîtes englo-

bantes.

La suppression des faces ou objets cachés ou *occlusion culling*. Elle consiste à supprimer les objets, respectivement polygones ou fragments occultés par d'autres objets, respectivement fragments, ou polygones. Les méthodes utilisables en temps réel sont implémentables directement sur le tampon de profondeur (*z-buffer*). Celles-ci permettent de déterminer si un fragment (élément de discrétisation d'un polygone en pixels) est occulté par d'autres fragments situés entre le point de vue et lui-même. Les cartes graphiques récentes bénéficient d'une implémentation matérielle de certaines fonctions telles que l'*early-z rejection*, qui consiste à rejeter l'écriture d'un fragment dans le tampon image (*framebuffer*) dès l'étape de rastérisation en comparant la profondeur du fragment courant avec celle déjà présente dans le tampon de profondeur. Les cartes graphiques implémentent aussi la requête d'occultation (*occlusion query*) qui permet à une application de vérifier si un objet (ou plus généralement sa boîte englobante) est visible ou non en effectuant une rastérisation de ce dernier et en retournant le nombre de pixels qui seraient écrits lors de son rendu (0 si invisible, > 0 si visible). La "fusion des bloqueurs" (*occluder fusion*) consiste à mettre à profit l'occultation générée par plusieurs objets (ou bloqueurs) pour cacher un ou plusieurs objets, polygones ou fragments.

Les deux premières méthodes de *culling* ont déja été optimisées pour être directement disponibles sur le matériel. La troisième, par contre, demande toujours un temps de traitement non négligeable. Ce temps dépend exclusivement de la taille, de la forme et de l'ordre de traitement des objets présents dans la pyramide de vue.

Il existe trois grandes heuristiques permettant d'effectuer la détection des faces cachées depuis une région ou cellule de vue – pouvant également servir pour le calcul depuis un seul point de vue.

La visibilité exacte. Elle se consacre à une détermination exacte de l'ensemble des polygones visibles et nécessite le calcul de ce que l'on nomme le complexe de visibilité, i.e., la segmentation de l'espace par les polygones en sous-régions de vue. Depuis cha-

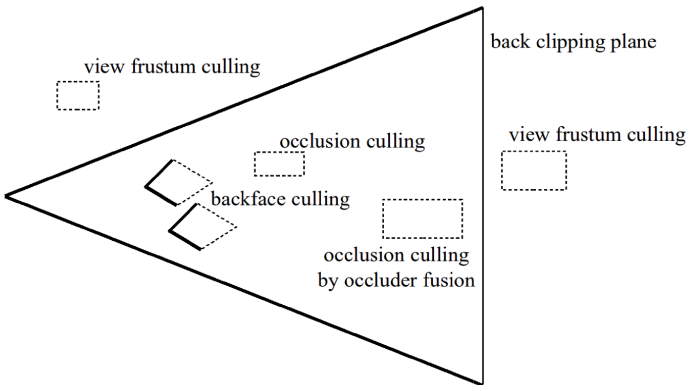


Figure 1.5 – Les trois types de *culling*.

cune de ces régions, on “voit” un ensemble de polygones (différents) qui sont conservés dans une liste *VS* (*Visible Set*). Il est donc naturel de constater que, comme ce sont les polygones eux-mêmes qui découpent la scène, la complexité d’un algorithme de calcul du complexe de visibilité dépend du nombre de polygones n composant la scène et peut être de l’ordre de $O(n^9)$ dans le pire cas. Le calcul est aussi extrêmement sensible aux erreurs d’imprécision numérique.

La visibilité conservatrice. Elle s’intéresse au calcul conservateur de régions, objets ou polygones occultés d’une scène en procédant à une simplification de la scène par une structure hiérarchique, généralement de type *octree* ou *bounding volume hierarchy* (BVH). Ce procédé permet une hiérarchisation des zones occultées et diminue drastiquement les temps de calcul. Toutefois elle néglige le calcul de l’occultation pour de trop petits objets et ne peut profiter éventuellement de l’occultation offerte par la combinaison des occultations de plusieurs petits objets (et des objets planaires). Ce type de visibilité surestime l’ensemble des polygones visibles ou *potentially visible set* (PVS) qui peut être précalculé et conservé en mémoire.

La visibilité par échantillonnage. Elle se distingue par un tracer de rayons permettant d’intersecter les polygones potentiellement visibles depuis un point de vue. Pour déterminer le PVS d’une région entière, les PVS de plusieurs points de vue peuvent être regroupés. Ce type de méthode, non optimisée, peut engendrer des artéfacts de *pop-*

ping si le nombre de rayons est insuffisant : des objets peuvent apparaître subitement dans la pyramide de vue lorsque l'observateur se déplace du fait qu'un polygone visible peut être détecté et affiché par intermittence. Par conséquent, ces méthodes ne sont pas conservatrices. Toutefois, certaines d'entre elles optimisent l'efficacité de l'échantillonnage en mutant les rayons ayant une intersection éloignée du point de vue afin d'affiner les résultats.

Lorsque l'on parle de visibilité depuis une région, il est nécessaire d'introduire certains concepts pour une meilleure compréhension de la littérature : ligne séparatrice, plan séparateur, ligne de support et plan de support. Ces derniers sont illustrés à la figure 1.6 entre deux objets.

Ligne séparatrice et plan séparateur [15] En 2D, une ligne séparatrice d de deux polygones A et B relie un point de A à un point de B, tel que A et B ne reposent pas dans le même demi-plan par rapport à d.

En 3D, un plan séparateur p de deux polyèdres E et F relie un point de E à une arête de F (ou l'inverse), tel que E et F ne reposent pas dans le même demi-espace par rapport à p.

Ligne de support et plan de support [15, 89] En 2D, une ligne de support d de deux polygones A et B relie un point de A à un point de B, tel que A et B reposent dans le même demi-plan par rapport à d.

En 3D, un plan de support p de deux polyèdres E et F relie un point de E à une arête de F (ou l'inverse), tel que E et F reposent dans le même demi-espace par rapport à p.

1.2.2 *Occlusion culling* et PVS

Nous portons un intérêt particulier aux méthodes de (pré)calcul de visibilité visant à limiter le nombre de polygones envoyés au GPU. Les deux grandes familles de méthodes utilisées pour afficher efficacement des scènes complexes sont l'*occlusion culling* et les PVS (*Potentially Visible Sets*) ou Ensembles Potentiellement Visibles. Par l'économie

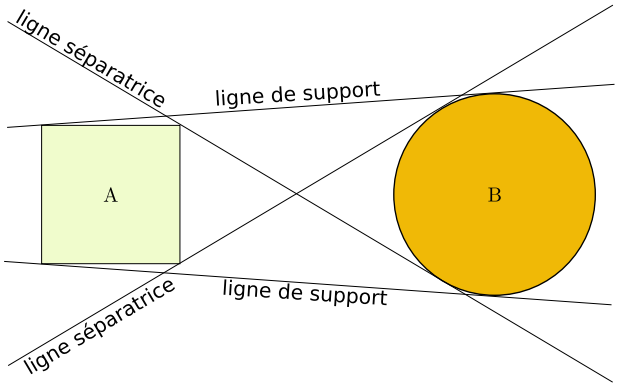


Figure 1.6 – Les lignes séparatrices et les lignes de support entre deux objets A et B. En 3D, elles deviennent plans séparateurs et plans de support.

de l'affichage des objets complètement occultés, ces algorithmes réduisent la charge du *CPU* et/ou du GPU.

Dans cette section nous cherchons à distinguer les différentes approches relatives au domaine de la détermination des polygones ou objets visibles. Nous soulignerons plusieurs critères, notamment le temps de calcul, le coût de stockage en mémoire et la possibilité de traiter la visibilité sur un monde statique ou dynamique. Nous nous intéressons tout particulièrement aux algorithmes conservateurs ou exacts car ils ont l'avantage de limiter les phénomènes de scintillement (*flickering* ou *popping*), i.e., les polygones apparaissent et disparaissent de manière inattendue. De plus, ce sont les algorithmes les plus utilisés dans le domaine du temps réel.

Les PVS possèdent leurs avantages et leurs inconvénients. Le moteur de rendu ne se limite qu'à consulter la liste de polygones de la cellule de vue où se situe le point de vue et il ne reste plus qu'à appliquer un algorithme de *frustum culling*. De plus, le temps de calcul ainsi économisé permet de libérer les ressources pour d'autres algorithmes (autant pour le moteur de rendu que pour les moteurs de physique ou d'intelligence artificielle). L'espace mémoire requis par ces PVS peut cependant être considérable. Le terme PVS est parfois utilisé pour faire référence à tout type d'*occlusion culling*, sachant que c'est toujours ce que ces algorithmes calculent. Cependant on l'emploie le plus souvent pour parler d'un *occlusion culling* basé sur des listes de polygones précalculées (i.e., hors

ligne). Afin de faire l'association entre une région de l'espace et son PVS, l'espace où évolue la caméra est généralement subdivisé en régions convexes. Ces régions sont plus communément appelées cellules de vue ou *view cells*. Les PVS tirent leur avantage du fait que le moteur de rendu doit seulement consulter la liste des polygones correspondant à la cellule de vue courante, et en faire le rendu possiblement en conjonction avec d'autres méthodes de *culling*, telles que le *frustum culling*. Toutefois, malgré de nombreuses améliorations [2, 14, 23, 50, 75, 89, 100, 106], l'espace mémoire requis pour stocker les PVS pour chaque cellule de vue peut être très grand pour des scènes complexes, et leur calcul demande également beaucoup de temps. Par ailleurs, il existe des méthodes prenant en compte la cohérence temporelle, qui permettent de limiter les transferts de PVS d'une cellule à une autre [90].

La notion de PVS est introduite par Airey et al. [3] où une partition de l'espace 3D est proposée afin de définir des cellules. On calcule pour chaque cellule, par tracer de rayons hors ligne, la liste des polygones potentiellement visibles depuis au moins un point de vue appartenant à la cellule. L'ensemble ainsi obtenu peut être efficacement calculé dès lors que l'on impose de fortes contraintes à la scène : les murs sont des plans, en général perpendiculaires par rapport au sol ; il n'y a pas de primitives complexes, sinon sous forme d'approximations planaires ; les mondes virtuels sont de facto plus simples. Ce genre de méthode a été utilisée dans le jeu "*Doom*" [48].

Teller [100] améliore la méthode de construction des PVS en introduisant des portails (*portals*) pour les scènes d'intérieur. La visibilité d'une cellule depuis une autre dépend de l'existence d'une ligne de vue entre elles, intersectant tous les portails se situant entre les deux cellules par connexité. Toutefois, l'inter-visibilité des cellules entre elles requiert un coût de stockage non négligeable et peut prendre énormément de temps de calcul si les portails sont nombreux et ont des formes complexes. De plus, les calculs peuvent souffrir d'instabilités numériques. Luebke et Georges [63] réduisent la complexité des portails de Teller en les simplifiant par leur boîte englobante.

Bern et al. [6] abordent la visibilité sur un autre plan, la multiplicité des points de vue. Ils posent le problème des évènements de discontinuité de la visibilité dans une scène tridimensionnelle lors du déplacement d'un observateur le long d'une ligne.

Ghali et al. [32, 33] traitent le problème des évènements de discontinuité de la visibilité de segments au travers de bloqueurs en utilisant les lignes de séparation et de support. Cohen-Or et al. [14] et Saona-Vasquez et al. [87] traitent le problème de manière plus générale en marquant un objet comme "occulté" s'il n'est visible depuis aucun point de la cellule de vue. Cela restreint à diminuer considérablement la taille de la cellule de vue et donc augmente le coût mémoire.

Afin d'améliorer l'efficacité des PVS, une étape importante consiste à réaliser la fusion des bloqueurs et l'aggrégation des régions dès lors que l'on considère des distances plus éloignées de la cellule de vue. Les scènes typiques d'extérieurs naturels peuvent être représentées par des champs d'altitudes (*height maps*). Plusieurs méthodes exploitent notamment les cohérences inhérentes à ces derniers.

Koltun et al. [50] introduisent cette notion de "fusion de bloqueurs" et d'aggrégation des volumes d'ombre grâce à une utilisation judicieuse de lignes séparatrices et lignes de support. De plus, la construction des PVS peut ne plus être précalculée hors ligne pour être utilisée pour la navigation interactive d'un observateur dans la scène, à condition que la vitesse de changement de cellule ne soit pas trop rapide. Toutefois, cette méthode ne se limite qu'à de la 2.5D par des champs d'altitudes. Schaufler et al. [89] adoptent une approche similaire en 3D, où les bloqueurs sont obtenus par une discrétisation volumique de l'espace au sein d'un *octree* (pour des bâtiments dans leur application). Certaines régions sont marquées comme occultées lorsqu'il est garanti qu'elles sont cachées. Ainsi, par projection conservatrice des régions occultées proches, ils détectent efficacement les régions occultées plus éloignées de l'espace pour une cellule de vue données. Le fait d'utiliser un *octree* pour la représentation des bloqueurs permet de simplifier la fusion des bloqueurs. Durand et al. [23] étendent notamment ce concept de fusion des bloqueurs en fusionnant les reprojections de groupes de bloqueurs sur des plans parallèles successifs.

Wonka et al. [106] proposent une approche de fusion bloqueurs en 2.5D par tracer de rayons. Cette méthode possède l'avantage sur les approches géométriques de calculer un PVS en quelques secondes seulement, mais peut provoquer des artéfacts de *popping*.

Lloyd et Egbert [61] ainsi que Downs et al. [21] proposent deux systèmes utilisant l'*horizon culling* en temps réel pour accélérer le rendu respectivement pour des *walk-throughs* de terrains et de villes. Toutefois, calculer l'*horizon culling* à la volée peut limiter les performances de tout le processus de rendu pour des scènes peu occultées et étendues puisque le calcul d'occultation doit être effectué à chaque image.

Les méthodes de détermination exacte de la visibiltié peuvent être intéressantes, mais seulement si l'on souhaite calculer un PVS de taille minimale (contrainte de mémoire) car ces dernières ne sont pas exploitables en temps réel dès lors que l'on souhaite utiliser des objets complexes. Par exemple, Nirenstein et al. [75] utilisent un changement de système de coordonnées (espace de Plücker) pour calculer une visibilité exacte. Cette dernière permet de calculer des PVS plus petits. De plus, bien que cette méthode demande des temps de calcul assez conséquents, elle se comporte de manière efficace lorsque le nombre de polygones de la scène augmente. Haumont et al. [43] présentent une autre méthode exacte qui simplifie le complexe de visibilité autant au niveau de l'implémentation que des calculs et qui permet d'effectuer des requêtes d'occultation entre deux cellules de manière très efficace en moyenne mais pouvant atteindre des dizaines de secondes dans le pire des cas. Cette méthode reste cependant adaptable pour des scènes 3D générales.

Ensuite, les approches basées sur l'échantillonnage permettent de calculer rapidement le PVS depuis un point ou une cellule de vue. Toutefois elles génèrent beaucoup d'artéfacts de *flickering* et de *popping*. Ces appproches tendent tout de même à limiter ces artéfacts en se basant sur l'erreur.

Nirenstein et Blake [74] calculent une visibilité par échantillonnage en minimisant la distance entre deux échantillons dans la structure spatiale de la scène et accélérent les calculs sur GPU. Cette méthode prend tout de même quelques heures de calcul lorsque les scènes contiennent plusieurs millions de polygones.

Sayer et al. [88] apportent aussi une méthode non exacte mais créent des imposteurs hors ligne permettant de remplacer la géométrie de l'arrière-plan dès que celui-ci

n'occupe que très peu d'espace à l'écran.

Bittner et al. [7] utilisent la mutation de rayons, une idée similaire à ce qui est utilisé par le transport lumineux par échantillonnage de Metropolis [103], pour accélérer la construction de PVS. Cela reste une méthode approximative (donc non conservatrice et non exacte), mais ils montrent qu'on peut calculer le PVS d'un ensemble de cellules de vue afin d'obtenir des résultats précis en peu de temps.

Les méthodes *occlusion query* (requête d'occultation) et *early-z rejection* (rejet prématuré par test de profondeur) sont couramment utilisées pour l'*occlusion culling* en temps réel, pour des scènes dynamiques. Toutefois, ces tests d'occultation peuvent être trop conservateurs (i.e., ils ne cachent pas assez d'objets), trop lents (i.e., trop coûteux à calculer relativement au coût de rendu), ou inexacts (i.e., leurs approximations produisent des artéfacts de *popping*) selon les cas.

Greene et al. [40] traversent un *octree* en utilisant un tampon de profondeur (*z-buffer*) hiérarchique logiciel pour déterminer rapidement la visibilité d'un polygone en utilisant les cohérences spatiale et temporelle. Zhang et al. [109] introduisent une approche similaire mais utilisent des cartes hiérarchiques d'occultation (*occlusion maps)*, permettant la fusion des bloqueurs en espace image et donc une amélioration des performances lors de *walkthoughs* interactifs.

Staneker et al. [95] utilisent des cartes d'occupation (*occupancy maps*) afin de faciliter le rendu pour des systèmes utilisant des graphes de scène. Pour cela, ils organisent des requêtes d'occultation multiples par ordre de priorité, bien qu'une hiérarchie en graphe de scène ne soit pas aussi efficace qu'un BVH pour opérer un ordre d'affichage de l'avant vers l'arrière lors de la traversée des noeuds. Ils utilisent également le principe de cohérence temporelle pour prévoir un budget sur le nombre de tests à réaliser sur la carte d'occupation avant de réaliser des requêtes d'occultation multiples sur GPU. Cette méthode permet de bonnes cohérences spatiale et temporelle si le graphe de scène reste inchangé durant le rendu.

Mattausch et al. [66] présentent CHC++, utilisant une implémentation des requêtes d'occultation multiples sur une représentation de la scène par hiérarchie de volumes en-

globants (BVH). Cet algorithme peut prendre en charge des scènes de plusieurs millions de polygones et assurer de hautes performances de rendu. L'algorithme CHC++ traverse récursivement les noeuds du BVH et lance une requête d'occultation sur sa boîte englobante afin de déterminer s'il est visible depuis le point de vue. Si ce noeud est visible et que c'est une feuille ou que la profondeur limite est atteinte, sa géométrie est dessinée, sinon les noeuds fils sont testés à leur tour. Si ce noeud n'est pas visible, ce noeud et son sous-arbre sont occultés et ne sont donc pas affichés. Selon le nombre de polygones et l'étendue de la boîte englobante d'un noeud, les polygones d'un noeud et de son sous-arbre peuvent prendre moins de temps à afficher plutôt que de tester sa visibilité. Ainsi, le coût de traversée, et donc la profondeur maximum, doit être limitée afin que CHC++ puisse fonctionner sans surcoût. CHC++ utilise également le principe de cohérence temporelle pour qu'un noeud du BVH n'ait pas à être retesté à chaque rendu. Ces noeuds peuvent être marqués visibles pour un certain nombre de *frames* aléatoire afin d'étaler les requêtes d'occultation dans le temps et ainsi alléger la charge du GPU. Toutefois, comme dans les approches précédentes, la scène doit rester inchangée afin de bénéficier pleinement de la cohérence temporelle.

Pour finir, d'autres systèmes permettent de répondre à des problématiques communes aux trois manières de traiter la visibilité (exacte, conservatrice, échantillonnée). Laine [56] présente un algorithme de calcul de PVS indépendant du type de visibilité utilisée – exacte, conservatrice ou par échantillonnage. Il utilise la cohérence directionnelle entre les cellules de vue adjacentes. Le grand avantage de cet algorithme est que l'ensemble de la géométrie n'a pas à être considéré lors du précalcul de visibilité depuis une cellule. Les temps de calcul, pour des scènes statiques, prennent entre 1 et 10 minutes pour des scènes comprenant environ $350,000$ polygones.

Par ailleurs, Andújar et al. [4] permettent la réorganisation des PVS en triant les polygones plus où moins visibles depuis un point ou une cellule de vue en créant des "ensembles à peine visibles" (*hardly visible sets*). Il permettent également d'associer ces polygones à des niveaux de détails pour assurer de bonnes performances de rendu. La mesure déterminant le niveau de détails est réalisée dans l'espace écran.

Parallèlement, Aila et Miettinen [2] ont developpé un *framework* indépendant, *dPVS*, regroupant de nombreuses méthodes d'*occlusion culling* et de calculs de PVS dans un système fonctionnant efficacement avec des environnements dynamiques de grande envergure. Toutefois lorsqu'il s'agit de réaliser de l'*occlusion culling*, les données à envoyer et à mettre à jour sur le GPU peuvent réprésenter un volume de données trop important pour atteindre des performances temps réel pour le rendu de mondes dynamiques.

Toutes ces méthodes améliorent de manière significative la vitesse de rendu des scènes virtuelles. Toutefois, pour des mondes immenses, le précalcul d'une structure hiérarchique pour les méthodes d'*occlusion culling* en temps réel ou le stockage des PVS pour seulement quelques cellules de vue produisent des volumes de données trop coûteux en mémoire. De plus, pour des scènes potentiellement infinies générées à la volée, aucune de ces méthodes n'offre une solution satisfaisante car elles calculent des hiérarchies ou des structures visibilité dont le volume de données dépend directement de l'étendue de la scène et du nombre d'objets qui la composent.

1.3 Pavages

Le pavage est une des conceptions mathématiques les plus anciennes que l'on retrouve dans les motifs réguliers des textiles des Indes datant du II^e millénaire avant J.C., au pavage des routes romaines à partir du III^e siècle avant J.C. Le pavage est autant apprécié pour la facilité de ranger les tuiles qui le composent de manière compacte que pour la beauté des motifs que ces derniers peuvent engendrer telles que les oeuvres de Escher et Dawson (figures 1.7(b) et 1.7(c)). Il devient cependant plus difficile d'obtenir des motifs plus complexes si l'on souhaite que le pavage réponde à certaines propriétés telles que la régularité ou l'irrégularité d'apparition de certains motifs, des contraintes d'adjacence entre les tuiles, mais aussi la variété des types de tuiles composant le pavage.

Un pavage est une partition d'un espace (généralement un espace euclidien comme le plan ou l'espace tridimensionnel) par un ensemble fini d'éléments appelés tuiles (plus précisément, ce sont des compacts d'intérieurs non vides). Généralement, on considère

des pavages par translations, i.e., que deux mêmes tuiles du pavage sont toujours déductibles l'une de l'autre par une translation (à une rotation ou une symétrie près, cf. figure 1.7(a)). Il existe aussi des pavages d'espaces non euclidiens, les plus célèbres étant les nombreux pavages d'Escher (pavages d'espaces hyperboliques, cf. figure 1.7(b)).

Définition 2. *Un **pavage** est une famille dénombrable de sous-ensembles fermés $\mathscr{T}$ de $\mathbb{E}^d$ couvrant $\mathbb{E}^d$ sans trous ou recouvrements.*

Définition 3. *Un **ensemble de pavage** est l'ensemble de tuiles uniques dont l'instanciation permet le pavage du plan ou de l'espace.*

Des contraintes sur l'imbrication des tuiles peuvent forcer des contraintes globales, telles que l'apériodicité du pavage obtenu.

Définition 4. *Soit $\mathscr{T}$ un pavage de $\mathbb{E}^d$. On définit un groupe de symétrie de $\mathscr{T}$ par $S(\mathscr{T}) := \{\phi \mid \phi$ est une isométrie de $\mathbb{E}^d$ telle que $\phi(\mathscr{T}) = \mathscr{T}\}$.*

Définition 5. *Soit $\mathscr{T}$ un pavage de $\mathbb{E}^d$.*

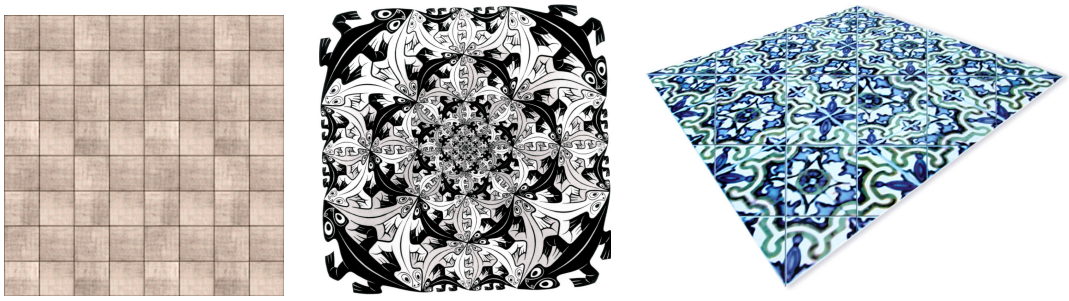
i *On dit que $\mathscr{T}$ est **périodique** si $S(\mathscr{T})$ contient d directions de translation indépendantes.*

ii *On dit que $\mathscr{T}$ est **sous-périodique** si $S(\mathscr{T})$ contient entre 1 et $d-1$ (inclus) directions de translation indépendantes. (Cet ensemble est parfois appelé non périodique dans la littérature française.)*

iii *On dit que $\mathscr{T}$ est **non périodique** si $S(\mathscr{T})$ ne contient aucune direction de translation autre que l'identité. (Cet ensemble est parfois appelé apériodique dans la littérature française.)*

Définition 6. *Un ensemble de tuiles est dit **apériodique** si et seulement si il n'admet que des pavages non périodiques.*

Définition 7. *Un pavage est dit **quasi-périodique** si pour tout motif m issu de ce pavage, il existe un entier i tel que toute fenêtre de taille i du pavage contienne m.*



(a) Carrelage classique.

(b) Pavage d'Escher dans l'espace hyperbolique, ici associé à un modèle par fractale.

(c) *Aesthetic Sabotage* [19] par Robert Dawson.

Figure 1.7 – Quelques pavages de l'espace 2D.

De manière intuitive, l'ensemble des motifs d'un pavage quasi-périodique apparaissent de manière uniformément distribuée sur ce dernier.

Propriété 1. *Tout pavage périodique est quasi-périodique.*

Pour plus de clarté, la figure 1.8 permet de visualiser comment se positionne chaque type de pavage l'un par rapport à l'autre.

Il existe trois méthodes pour construire des jeux de tuiles apériodiques :

- substitutions [39, 71] ;
- calculs sur les "couleurs" de tuiles [49] ;
- règles de correspondances faibles entre tuiles [92].

Lagae et al. [55] et Wei et al. [105] relatent la plupart des domaines où sont utilisés les pavages, e.g., modélisation de terrains, rendu non photoréaliste, distribution d'objets ou de points, application de textures, modélisation de surfaces et d'ornements, etc.

Dans nos contributions, le pavage est utilisé pour créer des mondes étendus et non répétitifs car ce genre de processus permet d'utiliser plusieurs fois les mêmes données par processus d'instanciation.

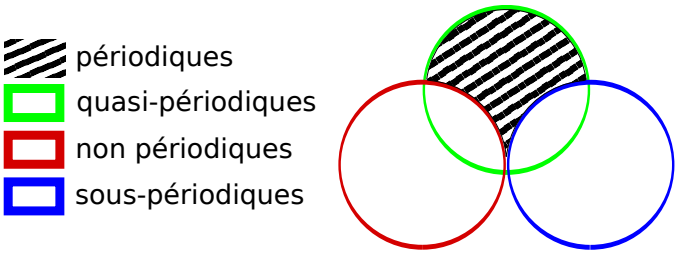
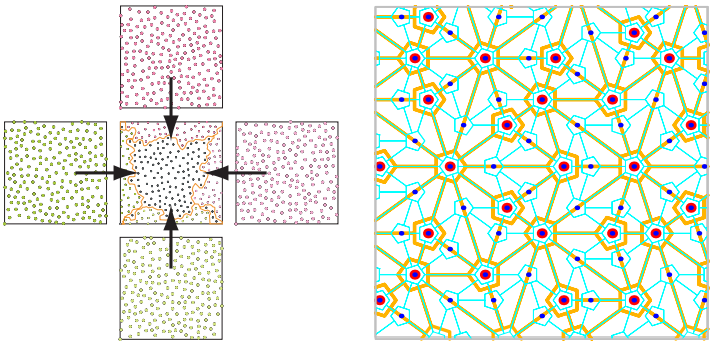


Figure 1.8 – La classification des différents types de pavage.



(a) Génération de bruit bleu [51].

(b) Échantillonnage d'importance [77].

Figure 1.9 – Différentes applications des pavages non périodiques.

1.3.1 Pavages de Wang

Wang [104] propose un système formel composé de tuiles carrées (*proto-tuiles*) de même taille. Chaque arête d'un carré se voit associer une couleur. Les règles de pavage de ce système sont simples : deux *proto-tuiles* adjacentes doivent avoir une même couleur sur leur arête commune pour que le pavage soit valide.

En tant que méthode de partition de l'espace, le pavage de Wang peut occuper l'espace de manière optimale. De plus, un ensemble de primitives peut potentiellement paver un plan infini de manière non répétitive. La figure 1.10 montre un exemple d'ensemble apériodique de tuiles et un exemple de pavage de Wang non périodique créé à partir de ces dernières.

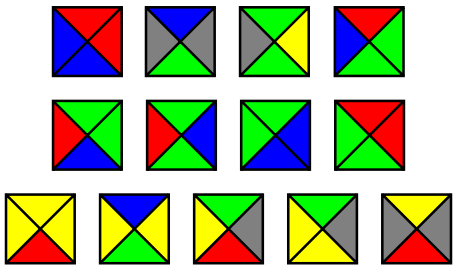
Le pavage de Wang est un outil efficace pour la génération de textures, l'application de texture basée sur des tuiles et la génération de distributions de disques de Poisson. Grâce à leurs arêtes colorées, les tuiles de Wang assurent la continuité avec leurs voisins directs. Elles mènent cependant à des problèmes de discontinuité au niveau des coins car elles ne contraignent pas les voisins des diagonales.

Afin de produire des textures de surfaces d'eau, Stam [94] utilise les pavages de Wang en précalculant un ensemble de tuiles (cf. figure 1.4(a)). Dans le même registre, Cohen et al. [12] introduisent une nouvelle manière de créer rapidement des pavages de Wang en s'assurant que le choix de la tuile à instancier soit toujours aléatoire, i.e., au moins deux choix possibles pour le placement d'un nouvelle tuile donnant un pavage valide.

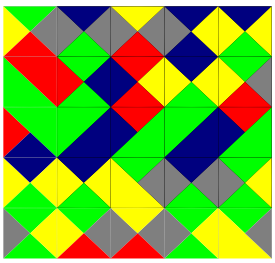
Kopf et al. [51] réalisent de l'échantillonnage de points à partir d'une fonction cible de densité de distribution (e.g., une photo) à différentes échelles en se servant d'un pavage récursif de Wang (cf. figure 1.9a).

1.3.2 Tuiles de coins

Lagae et Dutré [52] présentent une méthode de pavage à partir de tuiles carrées avec, pour chaque tuile, une coloration par coin au lieu d'une coloration par arête. A la figure 1.11, on peut voir que ce type de tuile impose des restrictions sur ses voisins dia-



(a) 13 tuiles de Wang apériodiques.



(b) Un pavage non périodique de Wang obtenu à partir de ces 13 tuiles.

Figure 1.10 – Pavage de Wang non périodique par Culik [17].

gonaux et n'est donc plus sujet au problème de discontinuité des coins. Ils montrent que les applications précédentes des tuiles de Wang sont aussi valables avec les tuiles de coins. De plus, il est plus aisé de paver grâce à ces tuiles et celles-ci augmentent l'échantillonnage possible pour la synthèse de textures, réduisent l'espace mémoire requis pour l'application de texture basée sur des tuiles et offrent de meilleures propriétés spectrales dans les distributions de disques de Poisson.

1.3.3 Cubes de Wang

Les cubes de Wang sont basés sur le même principe que les tuiles de Wang sauf que ce sont les faces des cubes qui sont colorées. Celles-ci permettent de fixer les contraintes d'adjacence et ainsi de paver un espace 3D.

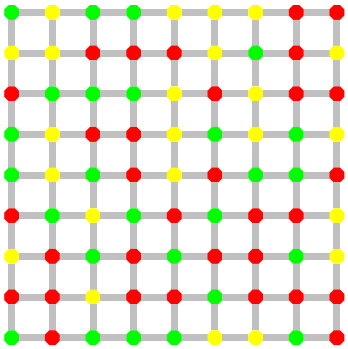


Figure 1.11 – Un pavage complet de 8×8 tuiles de coins basé sur trois couleurs par Lagae et al. [54].

Culik et Kari [16] introduisent les cubes de Wang avec des faces colorées comme étant une généralisation des tuiles de Wang avec des arêtes colorées (cf. figures 1.10(a) et 1.12). Ils montrent aussi qu'il existe un ensemble de 21 cubes de Wang apériodiques. Pour cela, ils utilisent les 13 tuiles de Wang de Culik [17] qu'ils convertissent en cubes, et complètent avec un ensemble de 8 cubes par l'utilisation d'un automate cellulaire.

Quelques travaux utilisent les cubes de Wang, notamment ceux de Sibley et al. [91], qui remplissent des cubes de Wang par des points distribués par disques de Poisson afin de réaliser une synthèse de vidéo en temps réel et du placement d'objets. Lu et Ebert [62] utilisent les cubes de Wang pour des illustrations de volumes basées sur l'exemple.

1.3.4 Cubes de coins

De manière similaire aux tuiles de Wang il est possible d'étendre les tuiles de coins à des cubes de coins. Un exemple de pavage par cubes de coins est montré à la figure 1.13.

Lagae et Dutré [53] utilisent les cubes de coins pour générer efficacement une distribution de sphères de Poisson. Ces dernières sont la généralisation à la 3D de l'échantillonnage de disques de Poisson en 2D. Leur méthode est capable de générer des distributions de sphères arbitrairement grandes en temps réel, permettant l'évaluation locale de ces dernières. Ils généralisent les algorithmes de pavage et les méthodes de construction des tuiles de coins aux cubes de coins. La distribution de sphères de Poisson possède plusieurs avantages tels que l'instanciation de géométrie et la distribution procédurale d'objets 3D.

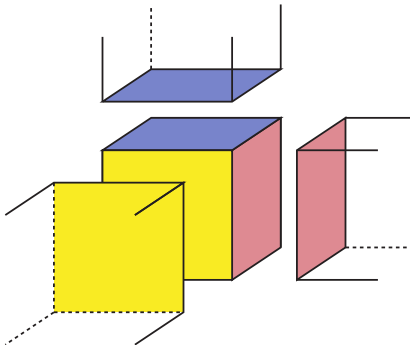


Figure 1.12 – Les cubes de Wang utilisés par Sibley et al. [91].

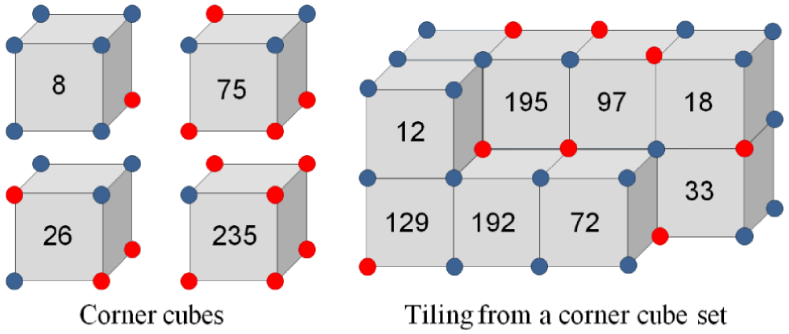
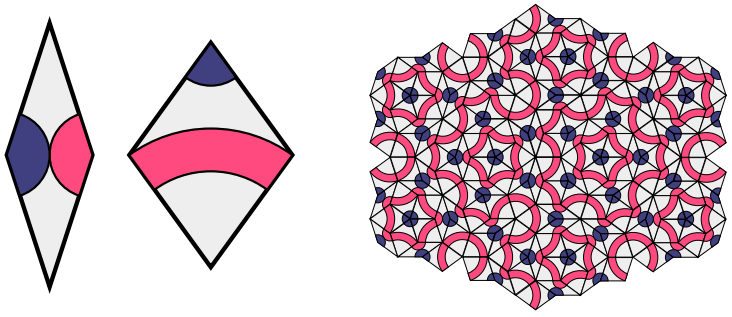


Figure 1.13 – Les cubes de coins utilisés par Peytavie et al. [81].

1.3.5 Pavages non périodiques de Penrose

Penrose trouve en 1973 un ensemble de six tuiles apériodiques qui forcent la quasi-périodicité et la non-périodicité du pavage associé. En 1974, il réduit ce nombre à deux [79] (cf. figure 1.14). Le principe des pavages de Penrose réside en ce qu'ils n'essayent pas d'agencer des tuiles de la même manière que l'on pourrait résoudre un puzzle, mais à l'inverse, à partir d'une forme de base (correspondant à l'oeuvre achevée). Il établit des règles de subdivision (e.g., un triangle divisé en d'autres triangles) apparentées à des règles de grammaire. Ainsi, il prouve qu'un pavage peut être infini et non périodique.

Ostromoukhov et al. [77] utilisent le pavage de Penrose afin de réaliser efficacement de l'échantillonnage d'importance (cf. figure 1.9(b)).



(a) Les tuiles de Penrose sont apériodiques : elles n'admettent que des pavages non périodiques.

(b) Un pavage non périodique de Penrose obtenu à partir de ces deux tuiles.

Figure 1.14 – Pavage de Penrose : un exemple de pavage non périodique.

CHAPITRE 2

LIMITATION DE L'ÉTENDUE D'UN MONDE VIRTUEL PAR PAVAGE D'OCCULTATION

Les contributions exposées dans ce chapitre ont donné lieu à une publication [38], présentée à la conférence *Graphics Interface* en mai 2011.

La création de mondes virtuels réalistes, complexes et diversifiés est d'une importance primordiale pour les jeux vidéo. Malheureusement, pour les infographistes, la création de contenu pour des scènes 3D peut être extrêmement longue et fastidieuse. Et même si les méthodes de modélisation procédurale peuvent faciliter cette tâche, la plupart ont été développées dans des contextes spécifiques au modèle que l'on souhaite générer. Lorsqu'un monde virtuel a besoin d'être immense, diversifié et complexe afin de favoriser l'immersion de l'utilisateur, la tâche particulièrement pénible de créer toute la variabilité du monde sur des surfaces étendues et à des différentes échelles est souvent laissée à l'infographiste. La modélisation procédurale peut être utilisée pour résoudre ce problème par le développement de méthodes de génération automatique, mais elles sont souvent limitées à des types de mondes spécifiques tels les forêts, les villes, les paysages, etc., et elles sont souvent difficiles à contrôler par les infographistes.

Le pavage est une solution couramment utilisée, où des portions prédéfinies de mondes sont réarrangées pour donner l'impression d'infini. Le pavage a par ailleurs beaucoup été utilisé dans les jeux vidéo, tels que *"Heroes of Might and Magic"* [102], *"Civilization"* [67] et *"Sim City"* [25].

Toutefois, lorsque l'on doit effectuer le rendu de mondes gigantesques depuis n'importe quel point de vue, un rendu par *z-buffer* traditionnel ou par lancer de rayons ne peut garantir de borner suffisamment le temps de traitement de toute la scène. Une solution consiste à précalculer le PVS pour des régions prédéfinies de la scène. De nombreux algorithmes de calcul de PVS existent mais aucun n'intègre le design de la scène comme processus prenant part au calcul, afin de réduire la taille des PVS. De plus, ces algo-

rithmes génèrent souvent des quantités de données trop grandes à stocker.

Nous proposons une méthode, le pavage d'occultation (*occlusion tiling*), où nous utilisons un pavage pour générer des mondes synthétiques. Ces mondes sont générés à la demande par détermination de visibilité des instances de tuiles depuis la cellule de vue. Cette méthode précalcule un ensemble de tuiles de base contenant des bloqueurs. Cet ensemble assure l'occultation complète de la pyramide de vue par les bloqueurs contenus dans les tuiles instanciés sur le plan 2D. Cette méthode garantit une distance limite des objets visibles depuis la cellule de vue et ainsi qu'un nombre limité d'instances de tuiles est nécessaire à la création du PVS. Ces tuiles sont ensuite utilisées en tant qu'extrusion en scène 3D, limitant par conséquent le nombre de polygones envoyés au GPU et garantissant de bonnes performances d'affichage.

2.1 Pavage d'occultation 2D / 2.5D

Nos scènes sont basées sur la tuile comme unité de composition. Chaque tuile regroupe un ensemble d'objets – e.g., des maisons, des arbres, ou toute autre géométrie. Nous souhaitons limiter le nombre de primitives envoyées au GPU en prenant en compte les parties visibles d'une scène depuis la cellule de vue, ainsi que minimiser l'espace mémoire occupé par les PVS par de l'instanciation de tuiles. Le temps de calcul des tuiles visibles est également une des contraintes majeures pour garantir un *walkthrough* en temps réel.

Nous souhaitons également assurer une visibilité conservatrice en supprimant les effets de *popping*, mais aussi fournir une génération de mondes riches, variés et non répétitifs. De plus il est nécessaire de garantir un faible coût de stockage et la persistance du monde généré.

Nous présentons dans ce chapitre une méthode de précalcul de tuiles occultantes, qui garantit une distance maximale à l'étendue de la visibilité en 2D. Dans ce contexte, notre méthode cible le 2D ou le 2.5D avec extrusion perpendiculaire au plan de support du monde. Contrairement aux méthodes qui suivent au chapitre 3, l'extension aux surfaces libres ou à la 3D pure n'est pas triviale, mais sera discutée en fin de chapitre.

Le pavage d'occultation (*occlusion tiling*) est une méthode qui optimise l'occultation du champ visuel tout en satisfaisant d'autres propriétés de la scène en disposant des objets, appelés bloqueurs, sur chaque tuile.

Chaque tuile est régulièrement subdivisée conformément à une grille, où les "pixels" occupés définissent des bloqueurs. Nous nommons *gridels* [98] les élements de cette grille. Une scène 3D correspond à une extrusion de ces *gridels* (e.g., immeubles, maisons, etc.), ou à n'importe quel objet 3D placé à cet endroit produisant une occultation au moins équivalente. Notre méthode de génération de tuiles occultantes garantit l'occultation de chaque tuile selon quatre directions canoniques (cf. section 2.5). Les tuiles générées peuvent ensuite être indépendamment assemblées en un pavage, potentiellement infini. Le pavage créé de cette manière limite le nombre de polygones envoyés au GPU puisqu'il assure que les objets situés au-delà d'une distance donnée (exprimée en nombre de tuiles) depuis la cellule de vue ne sont pas visibles. Le pavage d'occultation peut être utilisé comme un outil de génération automatique de mondes virtuels, ou pour assister un infographiste dans sa création, ou optimiser l'occultation d'un monde précédemment créé, garantissant au final de meilleures performances de rendu qu'avec l'utilisation seule d'algorithmes d'*occlusion culling*.

En section 2.3, nous expliquons brièvement une méthode simple permettant d'assurer la persistance de la scène lors de sa construction ainsi que son coût de stockage en mémoire faible et constant. En combinant cette méthode de pavage avec le calcul de visibilité dans des scènes 2D ou 3D, nous présentons notre méthode de pavage d'occultation. Nous décrivons ses caractéristiques (sections 2.4 et 2.5), du calcul d'occultation (section 2.6) à son optimisation (sections 2.7 et 2.8), ainsi que sa personnalisation (section 2.9). À la section 2.10, nous montrons comment cette méthode peut être utilisée pour effectuer une distribution d'immeubles et d'arbres, en considérant une hauteur limite fixée pour l'observateur en *walkthrough*. Finalement, nous concluons et proposons plusieurs directions de recherche pouvant être explorées à partir de notre proposition (section 2.11).

2.2 Discussion préliminaire

De par sa simplicité de représentation et de stockage, et de par sa généralité dans de multiples applications graphiques, nous considérons le carré comme la forme de base d'une tuile. Le plan définissant notre monde peut être entièrement pavé par des carrés 2D identiques. Une tuile est subdivisée en une grille régulière de $n \times n$ *gridels* (figure 2.2(a)). Chaque *gridel* peut être soit complètement vide, soit complètement occupé (formant alors un bloqueur), illustré respectivement par un pixel blanc et par un pixel foncé. Chaque tuile est également considérée comme une cellule de vue à part entière.

L'occultation d'une tuile est calculée en analysant l'occultation combinée de tous ses *gridels* occultants. Elle est dérivée pour les quatre directions canoniques dans le plan (figure 2.5(a)), et son calcul est décrit en section 2.6.

Pour de faibles résolutions de grille, un ensemble de tuiles occultantes peut être éventuellement généré de manière exhaustive en évaluant toutes les configurations possibles de bloqueurs sur une tuile. Pour n'importe quelle résolution, les tuiles peuvent aussi être générées en choisissant aléatoirement les *gridels* à activer (i.e., à rendre occultants) selon une probabilité de blocage (local par *gridel*, ou global par tuile).

Toutefois, le calcul étant $O(n^3)$ pour n *gridels* occultants, la recherche purement aléatoire peut s'avérer très inefficace. Nous proposons une approche multi-résolution qui améliore les calculs en section 2.8. Finalement, dans le but de donner plus de contrôle artistique sur la distribution des bloqueurs, nous décrivons des contraintes d'occupation de tuile en section 2.9.

A la section 2.10 quelques configurations de distributions de hauteurs et de pavages sur plusieurs niveaux sont expliquées dans le but de donner plus de variations sur la hauteur des bloqueurs.

2.3 Persistance du pavage

Dans un contexte de faible coût de stockage en mémoire du pavage, les instances de tuiles ne sont pas placées au préalable mais sont générées à la volée. Il est donc important que lorsque l'observateur quitte une région du pavage – cette dernière n'est

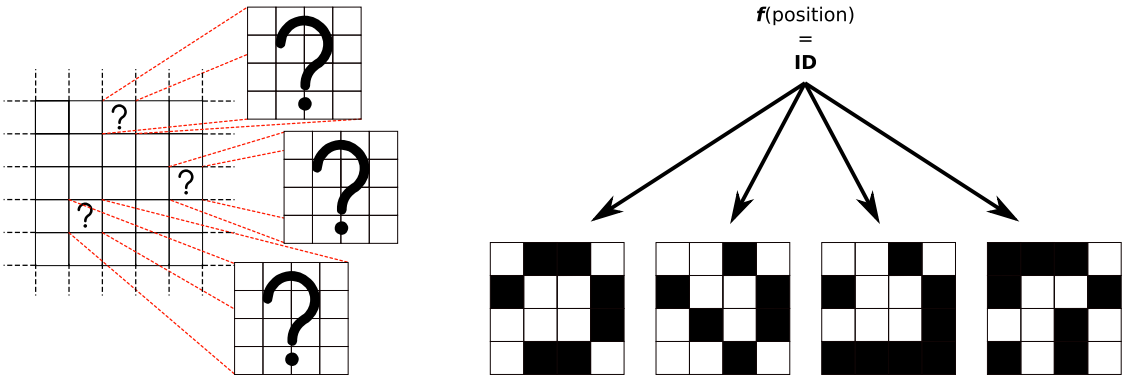


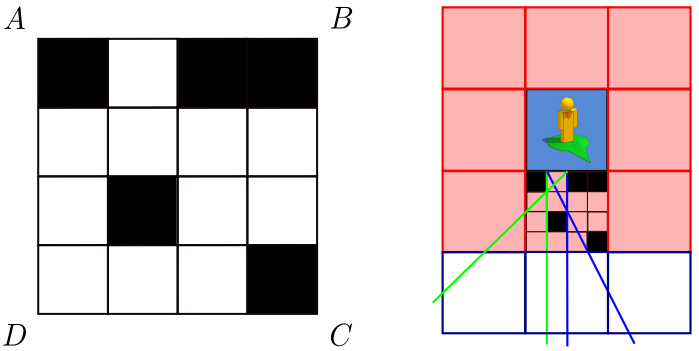
Figure 2.1 – Les tuiles de l'ensemble de pavage (à droite) sont choisies par une fonction de hachage et sont ensuite instanciées sur le plan de pavage (à gauche).

donc plus visible/instanciée – et que l'on ré-instancie cette région, celle-ci soit la même qu'auparavant.

Pour cela, lorsqu'une partie du monde (emplacement correspondant à une tuile) est déterminée comme étant visible depuis la cellule de vue, on vérifie que l'utilisateur n'a pas spécifié au préalable un type de tuile précis pour cette localisation (permettant de contrôler la personnalisation du pavage en certains endroits spécifiques du monde virtuel). Si ce n'est pas le cas, on détermine la tuile à choisir parmi l'ensemble des tuiles possibles grâce à une fonction de hachage $f : (x,y) \rightarrow i$. Cette fonction retourne, à partir des coordonnées x et y de la tuile à instancier, un entier i qui permet de choisir la tuile correspondante dans l'ensemble de pavage précalculé (cf. figure 2.1).

Il est tout de même préférable que la fonction f ne soit pas linéaire afin que l'on ne puisse pas voir apparaître de motifs dans la construction de la scène. Par exemple, la fonction $f(x,y) = a \times x \times y$ implique que toutes les tuiles des axes $x = 0$ et $y = 0$ seront identiques. Par ailleurs, si l'on souhaite que le monde virtuel généré ne se répète pas, f doit être non périodique.

Ainsi nous bénéficions d'un coût de stockage en mémoire du pavage instancié constant et quasi nul.



(a) Une tuile représentée par une grille de résolution 4×4, et ses cinq bloqueurs.

(b) Deux paires de lignes séparatrices issues de la cellule de vue.

Figure 2.2 – L'information de visibilité depuis le segment $[AB]$, représentant la bordure de la cellule de vue, vers les 3 autres segments. Les *gridels* occultants sont en noir, les *gridels* vides en blanc.

2.4 Propriétés d'occultation

La propriété d'occultation que nous cherchons à satisfaire est la limitation de l'étendue du pavage visible par un observateur situé sur n'importe quelle tuile du pavage (puisque chaque tuile instanciée sur le plan peut être considérée comme une cellule de vue) à partir d'une certaine distance comptée en nombre de tuiles. Pour cela, nous proposons de placer les bloqueurs dans les tuiles de manière à ce que chaque ligne de vue doit être interceptée par un bloqueur contenu par les tuiles du premier anneau de voisinage de la cellule de vue (i.e., les huit tuiles adjacentes). Le champ visuel sera alors complètement occulté par les bloqueurs situés sur le 8-voisinage depuis n'importe quel point de la cellule de vue .

Pour étudier maintenant les propriétés d'occultation que les tuiles du 8-voisinage doivent satisfaire, sans contraindre excessivement les positions de leurs *gridels* occultants, nous introduisons d'abord la notion de segments émetteurs (définition 8) et récepteurs (définition 9), mais aussi celles de paire occultante de segments (définition 10) et de schéma d'occultation (définition 11). Cette dernière définition représente les conditions nécessaires pour qu'une tuile soit validée et ajoutée à l'ensemble de pavage que nous cherchons.

Définition 8. *Un **segment émetteur** est un segment d'où partent des lignes de vue.*

Définition 9. *Un **segment récepteur** est un segment où parviennent des lignes de vue.*

Définition 10. *Soit (s_1, s_2) une paire de segments où s_1 est un segment émetteur et s_2 est un segment récepteur. (s_1, s_2) est une **paire occultante de segments** si et seulement si $\forall M \in s_1, \forall M' \in s_2$, la ligne de vue (MM') est bloquée, i.e., interceptée par un bloqueur.*

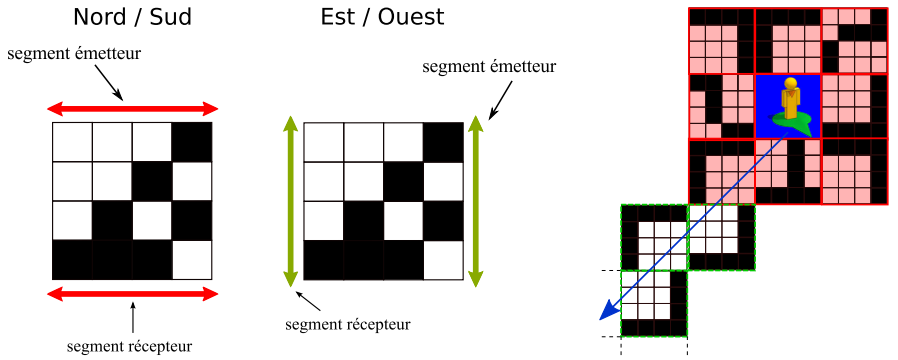
Définition 11. *Un **schema d'occultation** est défini par un ensemble S_{in} de segment émetteurs et un ensemble S_{out} de segment récepteurs où $\forall s \in S_{in}, \forall s' \in S_{out}$, (s, s') est une paire occultante de segments.*

Pour les sections qui suivent, chaque tuile est définie par quatre sommets, dans le sens horaire : A, B, C, D. Une tuile orientée comme dans la figure 2.2(a) a une arête "Nord" $[AB]$, une arête "Est" $[BC]$, une arête "Sud" $[CD]$, et une arête "Ouest" $[DA]$.

2.5 Schéma d'occultation $\bar{u}$

Un schéma d'occultation simpliste considèrerait qu'une tuile est complètement occultante si toutes les lignes de vue entre chaque paire de segments opposés d'une tuile (Nord/Sud et Est/Ouest dans la figure 2.3(a)) sont bloquées par les *gridels* occultants de la tuile. Malheureusement, ce schéma n'est pas suffisant pour satisfaire l'occultation complète sur un ensemble de tuiles répondant à ces conditions. En effet, une ligne de vue partant de la cellule de vue en diagonale ne serait ni bloquée par le premier anneau de voisinage, ni par les tuiles suivantes du pavage. Cela s'explique par le fait qu'une telle ligne n'est jamais bloquée par une paire occultante de segments d'une même tuile. La figure 2.3(b) illustre ce contre-exemple où la cellule de vue est en bleu, les tuiles du premier anneau de voisinage sont en rouge et les tuiles instanciées par une seule et même ligne de vue, même au delà du premier anneau de voisinage, sont en pointillés verts. Par complémentarité, étendre les segments opposés aux deux segments restants de la tuile (i.e., segment opposé, plus les deux segments latéraux en entier) donnerait sûrement des conditions suffisantes, mais ce schéma forcerait les *gridels* des coins de la tuile adja-

cents au segment émetteur à être occultants. Cela donnerait pour résultat des tuiles trop contraintes, avec des motifs trop réguliers.



(a) Un schéma d'occultation simple ayant seulement deux paires occultantes de segments, *Nord*/*Sud* en rouge et *Est*/*Ouest* en vert.

(b) Une ligne de vue peut traverser en diagonale ces tuiles, même si elle satisfait les conditions en (a).

Figure 2.3 – Conditions d'occultation insuffisantes.

Afin d'assurer l'occultation complète par la combinaison des occultations de deux tuiles adjacentes, en permettant plus de flexibilité dans le positionnement des bloqueurs, nous proposons le schéma *ū-shape* montré à la figure 2.4. La base du *ū-shape* est composée d'un segment émetteur et de trois segment récepteurs : le segment opposé au segment émetteur, augmenté des deux côtés ayant la hauteur de la moitié de la tuile. Si chaque tuile respecte individuellement le schéma d'occultation *ū-shape* pour chacune des quatre directions (cf. figure 2.5(a)), nous pouvons garantir que chacune des lignes de vue issue d'une cellule de vue sera bloquée par le premier anneau de voisinage de tuiles. Nous pouvons voir à la figure 2.5(b) que chaque ligne de vue issue de la cellule de vue passant au travers des tuiles X et Y est bloquée par le schéma d'occultation Nord/Sud de X et par le schéma Ouest/Est ou Sud/Nord de Y. Ces quatre contraintes sont nécessaires et suffisantes au pavage d'occultation (*occlusion tiling*) pour chaque tuile. Bien que d'autres schémas d'occultation puissent être utilisés pour assurer l'occultation à partir du premier anneau de voisinage, celui-ci se révèle être très satisfaisant selon toutes nos expériences.

Le grand avantage de ce schéma est qu'en respectant ces quatre conditions, chaque tuile peut être considérée indépendamment de son voisinage lors de la génération du

pavage.

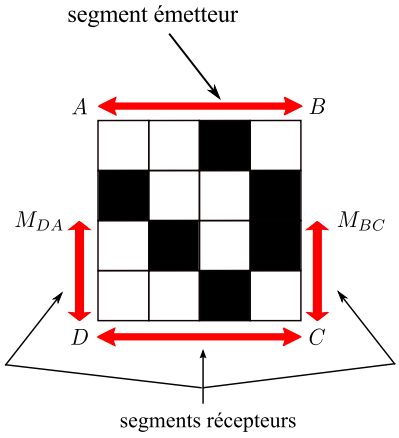
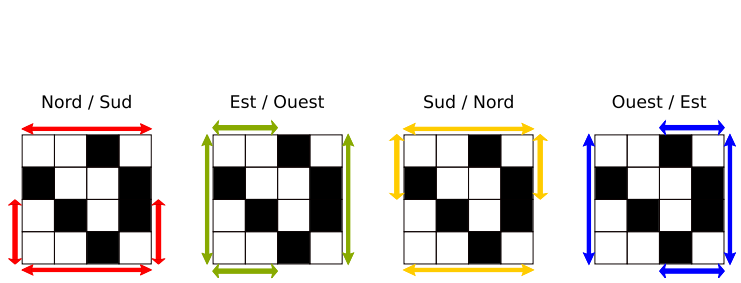
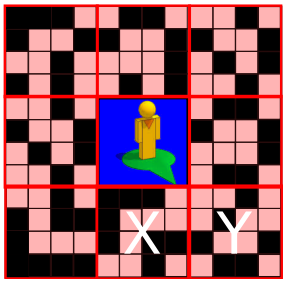


Figure 2.4 – Schéma d'occultation $\bar{u}$ pour la direction Nord/Sud.



(a) $\bar{u}$-*shape* : Les quatre contraintes d'occultation (De / Vers) qui doivent être vérifiées par chaque tuile de l'ensemble de pavage.



(b) Le blocage des lignes de vue au-delà du premier anneau de voisinage.

Figure 2.5 – Schéma d'occultation $\bar{u}$-*shape* pour chaque tuile du pavage.

2.6 Calcul de visibilité du schéma d'occultation $\bar{u}$-*shape*

Afin de tester si une tuile respecte les quatre contraintes d'occultation, nous utilisons la notion de *lignes séparatrices* [15] définies en section 1.2.1 et équivalentes aux plans séparateurs en 3D. Dès lors que deux objets sont disjoints, ils laissent passer des lignes de vue entre eux et une paire de lignes séparatrice peut être crée entre ces deux objets. Ensuite, si pour chaque combinaison de deux objets, les lignes séparatrices que l'on peut créer interterceptent un autre objet de la scène, alors l'ensemble des objets de la scène

n'admet aucune ligne séparatrice le séparant en deux sous espaces disjoint. De ce fait, si aucune ligne séparatrice ne peut être créée, aucune ligne de vue ne peut l'être.

Nous calculons alors l'ensemble des lignes séparatrices possibles et vérifions qu'aucune d'entre elles ne permette de séparer les *gridels* occultants sans intercepter un autre de ces *gridels*. Cette tâche est effectuée pour chaque direction par l'algorithme 1, instancié dans ce cas pour la direction Nord/Sud. La figure 2.2(b) montre les lignes séparatrices ainsi générées.

L'algorithme 1 précalcule toutes les lignes séparatrices entre chaque combinaison (o,o') de bloqueurs de la tuile à tester, et vérifie que chacune intersecte au moins un troisième bloqueur. Cette procédure est adaptée et exécutée pour chaque direction (Nord/-Sud, Est/Ouest, Sud/Nord, et Ouest/Est).

Pour n bloqueurs, l'algorithme naïf est de complexité $O(n^3)$, puisqu'il teste chaque combinaison ordonnée de bloqueurs (o, o', o''). Chaque triplet n'est testé qu'une seule fois. La figure 2.6 montre un cas particulier où les coins des bloqueurs o, o' et o'' sont alignés. Si o'' est du même côté de la ligne séparatrice que o', celui-ci doit être en-dessous de o pour avoir une ligne séparatrice valide. Si o'' est de l'autre côté par rapport à o', il doit être au-dessus de o pour avoir une ligne séparatrice valide.

Afin de préserver la simplicité de l'algorithme 1, certains détails et cas particuliers d'implémentation n'ont pas été retranscrits, tels que les tests de visibilité entre les points A et les coins inférieurs gauches des bloqueurs, B et les coins inférieurs droits, M_{DA} et les coins supérieurs gauches, M_{BC} et les coins supérieurs droits, A et M_{BC}, et B et M_{DA}. Toutefois, ces tests n'affectent en aucun cas la complexité de l'algorithme. Ces configurations peuvent être facilement traitée en considérant certains points comme de coins de bloqueurs supplémentaire : A comme un coin supérieur droit, B comme un coin supérieur gauche, M_{DA} comme un coin inférieur droit, et M_{BC} comme un coin inférieur gauche. D'autres tests de visibilité sont nécessaires entre A et M_{DA}, et entre B et M_{BC}. Ces derniers sont traités en testant s'il existe un bloqueur sur la bordure entre A et M_{DA} ou juste en-dessous de M_{DA}, et entre B et M_{BC} ou juste en-dessous de M_{BC}. La figure 2.2(b) montre une ligne séparatrice de bordure en bleu.

```
M_DA := milieu_de([DA]);
M_BC := milieu_de([BC]);
pour chaque bloqueur o ∈ ABCD faire
    pour chaque bloqueur o', o' ≠ o faire
        d1 = (coin_supérieur_gauche(o), coin_inférieur_droit(o'))
        d2 = (coin_supérieur_droit(o), coin_inférieur_gauche(o'))
        {// si d1 ∩ o | d1 ∩ o', d1 n'est pas traité}
        si d1 intersecte ū-shape (segment [AB] et polyligne [M_DA, D, C, M_BC]) alors
            d1_bloqué = faux ;
            pour chaque bloqueur o'', o'' ≠ o & o'' ≠ o' faire
                si d1 ∩ o'' alors
                    d1_bloqué = vrai
                    sortir de la boucle
                fin
            fin
            si ¬ d1_bloqué alors
                retourner faux
            fin
        fin
        {// tester d2 de la même manière que d1}
        ...
    fin
fin
retourner vrai ; {// la tuile est occultante}
```

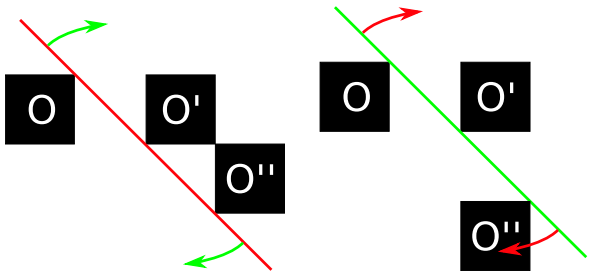
Notons que, pour chaque bloqueur, seuls leurs coins “libres” (i.e., non partagés avec un autre bloqueur) sont testés. Si tous ses coins sont partagés avec d’autres bloqueurs, un bloqueur n’est donc pas traité. Notez aussi que les paires de lignes séparatrices entre deux bloqueurs (cf. figure 2.2(b)) sont calculées séparément et ne résultent pas de la même combinaison (o,o').

Algorithme 1: Procédure testant l’occultation d’une tuile dans la direction Nord/Sud.

2.7 Génération de tuile à résolution complète

Dans cette section, nous souhaitons identifier les configurations de blocage des *gridels* (bloquants ou non bloquant) qui satisfont le schéma d’occultation *ū-shape* et ainsi trouver un ensemble de pavage respectant la propriété d’occultation énoncée à la section 2.4.

Un calcul exhaustif de toutes les configurations de *gridels* devient rapidement inefficace avec une complexité d’ordre $O(n^3)$, puisque même une tuile de résolution 8×8 produit 2^{64} tuiles à tester. Le passage à échelle de l’approche exhaustive n’étant pas



(a) Ligne séparatrice créée : degré de liberté en rotation.

(b) Ligne séparatrice non créée : pas de degré de liberté en rotation.

Figure 2.6 – Un cas particulier de construction des lignes séparatrices : si une ligne séparatrice peut être créée, un trou existe et la tuile n'est pas occultante.

réalisable, nous reposons plutôt sur une approche stochastique de recherche de configuration de bloqueurs dans la grille, avec une probabilité de blocage assignée à chaque *gridel* de la tuile – menant à une densité de bloqueurs si l'on considère la tuile entière. À l'aide d'un générateur couvrant la séquence complète des valeurs binaires (vide ou occultant) associée à une configuration, nous pouvons rechercher différentes configurations non corrélées, résultant en un ensemble de tuiles ayant plus de variations qu'avec une recherche séquentielle.

Le tableau 2.1 donne le nombres de tuiles trouvées qui respectent le schéma d'occultation *ū-shape*. Certaines de ces tuiles sont montrées aux figures 2.7, 2.8 et 2.9. Chaque série de tests a été calculée pour une durée approximative de 90 minutes avec un CPU Intel® Core™ 2 Duo (P7450) 2.13 GHz et 4 Go de mémoire vive.

On peut voir qu'à une résolution donnée, pour le même temps de calcul, augmenter la densité des bloqueurs réduit le nombre de tuiles testées, mais augmente le nombre de tuiles occultantes trouvées. Aussi, pour une densité de bloqueurs donnée, le nombre de tuiles testées diminue rapidement lorsque la résolution augmente, mais plus de tuiles occultantes sont trouvées. Fixer la bonne densité de blocage pour une résolution donnée est alors la clef pour trouver des tuiles occultantes : une densité trop faible produira trop peu de tuiles et une densité trop élevée augmentera le temps de test par tuile.

Résolution	Densité de blocage	Tuiles occultantes / testées
16×16	1/4 (25%)	2,960 / 260,725
	1/5 (20%)	209 / 1,064,670
	1/6 (16,6%)	5 / 2,482,720
	1/7 (14,3%)	0 / 4,399,280
32×32	1/4 (25%)	329 / 801
	1/5 (20%)	112 / 3,710
	1/6 (16,6%)	9 / 19,608
	1/7 (14,3%)	0 / 62,224
64×64	1/4 (25%)	8 / 8
	1/5 (20%)	10 / 10
	1/6 (16,6%)	10 / 20
	1/7 (14,3%)	4 / 59
	1/8 (12,5%)	2 / 244
	1/9 (11,1%)	0 / 909

Tableau 2.1 – Densités d'occultation et nombre de tuiles occultantes trouvées.

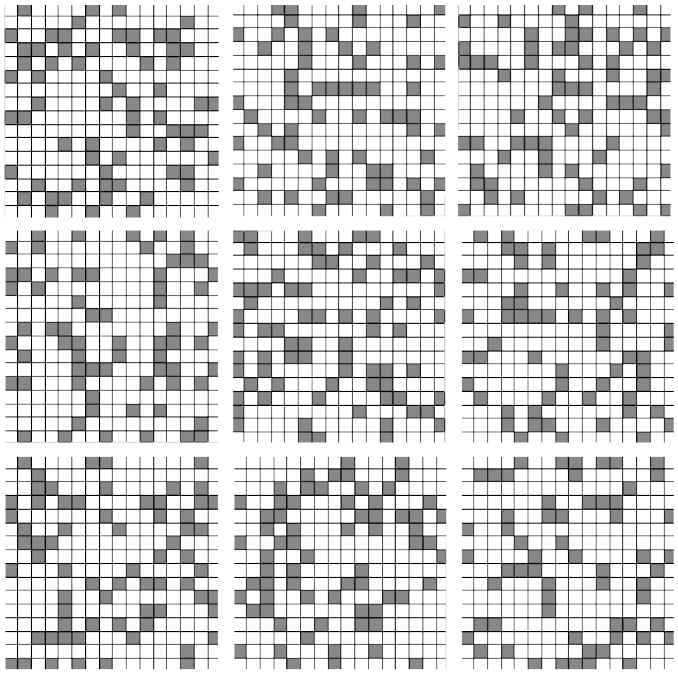


Figure 2.7 – 9 tuiles occultantes 16×16 de densité $1/5$ (20%).

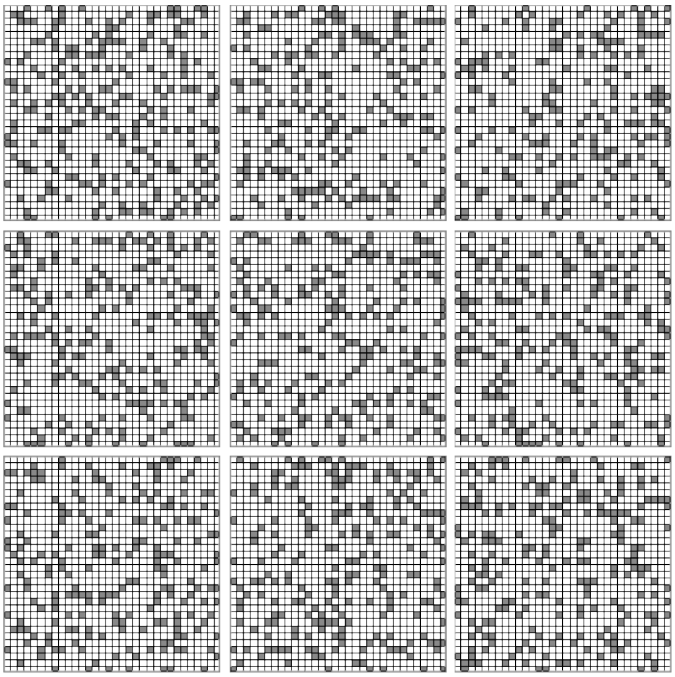


Figure 2.8 – 9 tuiles occultantes 32×32 de densité $1/5$ (20%).

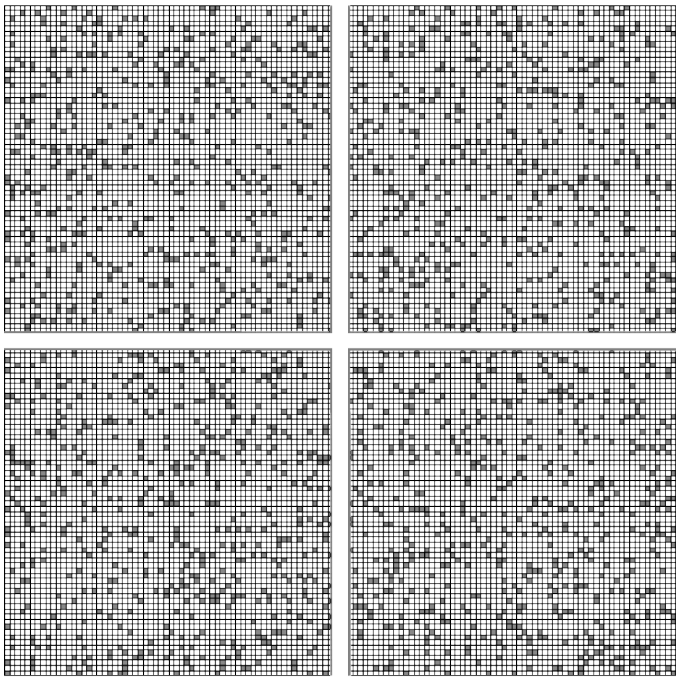


Figure 2.9 – 4 tuiles occultantes 64×64 de densité $1/7$ $(14,3\%)$.

2.8 Génération multi-résolution

Le calcul de visibilité *ū-shape* donne suffisamment de tuiles occultantes pour des résolutions inférieures à 32×32 et une densité de bloqueurs supérieure à 20%. Pour de plus grandes résolutions, tester une tuile devient trop coûteux par rapport au nombre de tuiles valides. Tandis qu'un traitement incrémental des bloqueurs ou une classification hiérarchique de ces derniers améliorerait sûrement la complexité de l'algorithme, cela n'augmenterait pas le potentiel d'une tuile d'être occultante.

Dans le but de réduire le temps de génération, nous proposons une méthode de dérivation de tuile permettant de guider la recherche stochastique de tuiles par une approche multi-résolution.

Les quatre étapes pour générer des tuiles dérivées sont :

1. **Générer** aléatoirement une tuile de résolution $n \times n$ satisfaisant les contraintes d'occultation dans les quatre directions.

2. **Doubler la résolution** de la tuile dans chaque dimension, i.e., $2n \times 2n$; un *gridel* devient un ensemble de 2×2 *gridels* ayant la même définition.

3. **Muter** les *gridels* de la tuile $2n \times 2n$ en déplaçant aléatoirement les *gridels* occultants selon une probabilité de mutation (fixée par l'utilisateur, localement ou globalement). Enfin, l'occultation de chaque configuration nouvellement obtenue est testée, de la même manière que dans la première étape.

4. **Répéter** les deux étapes précédentes avec les tuiles résultantes ayant passé avec succès les tests d'occultation jusqu'à ce que la résolution souhaitée soit atteinte.

La figure 2.10 montre quelques résultats issus de cette méthode multi-résolution. Ici, nous avons fixé à 100 le nombre maximum de mutations/tests (étape 4) à faire subir à une tuile avant d'abandonner la recherche de solution, limitant alors le nombre de solutions similaires trouvées pour les résolutions dérivées. Pour montrer l'efficacité de cette optimisation, une tuile occultante de résolution 8×8 est trouvée en approximativement une seconde, et 100 tuiles occultantes 16×16 dérivées de cette dernière sont trouvées en

moins de 36 secondes pour une densité de bloqueurs de 20%, tandis que la méthode originale trouve 100 tuiles occultantes en 45 minutes. Ces opérations guident effectivement la recherche de solutions et divisent approximativement par 100 le temps de recherche.

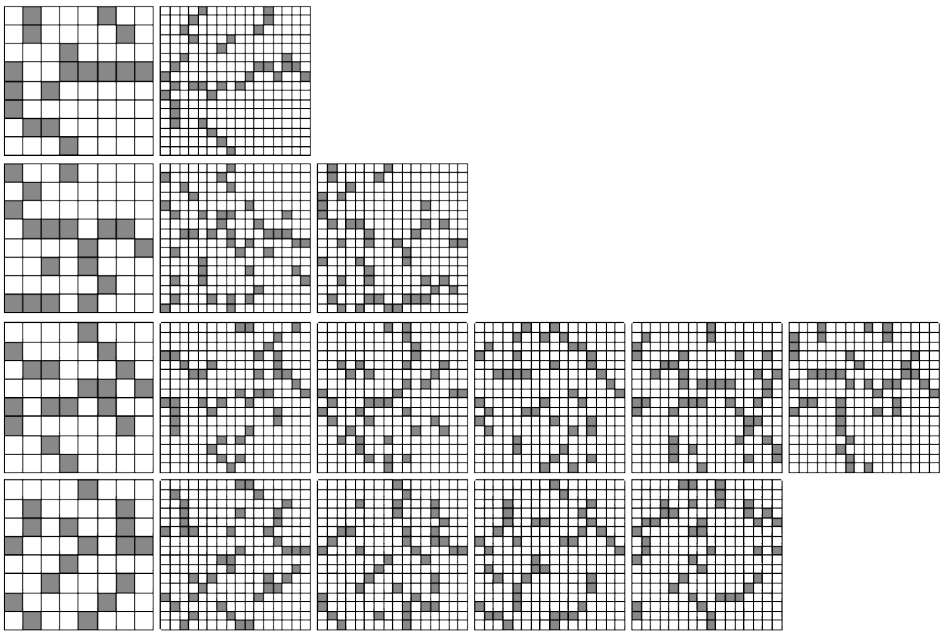
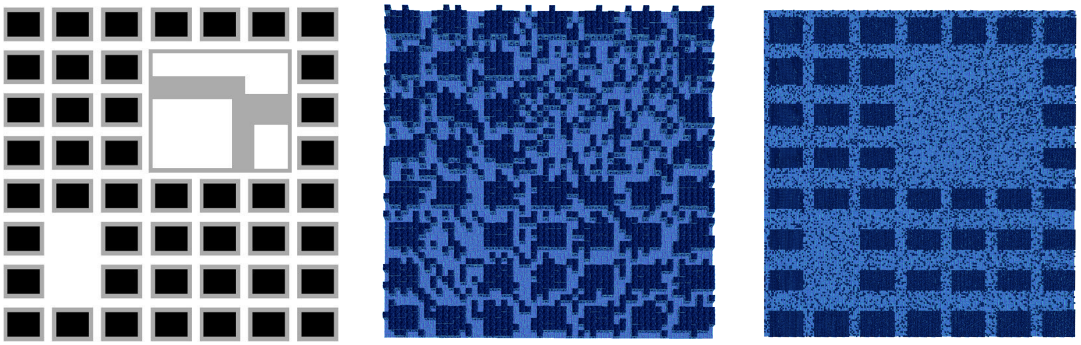


Figure 2.10 – Quatre tuiles occultantes de 8×8 et leurs tuiles occultantes dérivées à 16×16 respectives.

2.9 Contrôle de la méthode

Afin de fournir un meilleur contrôle aux infographistes sur la génération des tuiles occultantes, nous intégrons dans notre système un outil de dessin afin de spécifier les probabilités de blocage des *gridels* d'une ou plusieurs tuiles à la fois. L'image dessinée à la figure 2.11(a) est partagée en 7×7 régions. Chacune de ces régions est utilisée comme source de densité locale de bloqueurs dans la recherche de deux ensembles de 49 tuiles occultantes, respectivement pour des tuiles de résolution 8×8 et 32×32. Les pixels de l'image source, niveaux de gris allant du blanc au noir, sont transformés linéairement en une probabilité de blocage. Pour le premier ensemble de tuiles, les probabilités de blocage varient de 20% pour le blanc à 100% pour le noir. Pour le second ensemble, elles varient de $16,6\%$ pour le blanc à 100% pour le noir. Ces pourcentages ont été choisis

afin de rester dans les plages de densités de bloqueurs pour lesquelles nous savons que des tuiles occultantes seront trouvées (cf. tableau 2.1). Le premier ensemble de pavage (résolution 8×8) est trouvé en 14 secondes. Le second ensemble (résolution 32×32) est trouvé en 1765 secondes. Aux figures 2.11(b) et 2.11(c), ces tuiles sont juxtaposées et forment un pavage de 7×7.



(a) Image source dessinée, regroupant les 49 régions servant pour fixer la densité locale de bloqueurs.

(b) 7×7 tuiles occultantes juxtaposées, chacune de résolution 8×8.

(c) 7×7 tuiles occultantes juxtaposées, chacune de résolution 32×32.

Figure 2.11 – Personnalisation de la distribution des bloqueurs : chaque *gridel* est généré à partir d'une image source par la transformation en probabilité de blocage du niveau de gris du pixel correspondant.

2.10 Applications

Notre méthode a été utilisée pour créer une ville, montrée à la figure 2.12, dans laquelle les *gridels* occultants sont instanciés par des immeubles (i.e., des boîtes 3D texturées) et où la visibilité est limitée au premier anneau de voisinage, compte tenu du fait que l'observateur est située en-dessous du plus petit des immeubles. La cellule de vue en bleu est entourée par le premier anneau de voisinage de tuiles en rouge. Le second anneau de voisinage ainsi que les suivants, rendus en gris, ne sont pas visibles depuis la cellule de vue si l'observateur est situé entre le sol et le toit du plus petit immeuble. Pour obtenir ce pavage 10×10 tuiles de résolution 16×16 sont dérivées d'une seule tuile de résolution 8×8 d'une densité de bloqueurs de 20% avec un nombre de mutations de $16 \times 16 = 256$ par tuile dérivée. Les tuiles ont été calculées en 36 secondes et le pavage

en 1 seconde.

Grâce au précalcul de tuiles occultante, il n'y a plus besoin de calculer le PVS de la scène de manière explicite puisque, pour chaque point de vue sur le pavage, les polygones situés au-delà du premier anneau de pavage peuvent tout simplement être ignorés. Seuls les polygones présents dans la cellule de vue (en bleu) ou appartenant au premier anneau de voisinage (en rouge) ont besoin d'être rendus.

Afin de donner un aspect plus réaliste à nos villes, des contraintes additionnelles au schéma d'occultation *ū-shape* peuvent être prises en compte. Notamment, l'observateur devrait être capable de naviguer librement entre les tuiles, et les immeubles devraient avoir des hauteurs différentes les uns des autres.

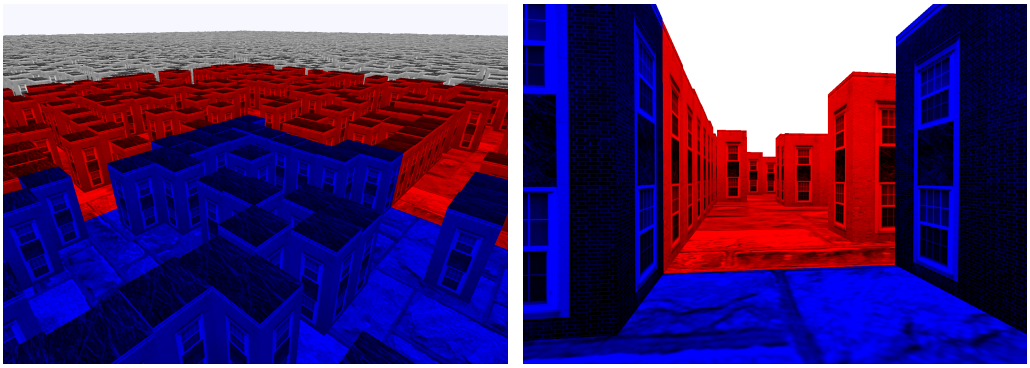
2.10.1 Connexité entre tuiles du pavage

Pour laisser la possibilité de circuler librement entre les tuiles, deux contraintes sont ajoutées. Premièrement, nous assurons qu'il existe un chemin entre les bords opposés d'une tuile i.e., du Nord au Sud et de l'Est à l'Ouest. Deuxièmement, nous forçons le fait qu'une tuile ne peut être ajoutée à l'ensemble des tuiles du pavage si un bord partagé par deux tuiles ne peut être traversé. Ceci est exprimé comme une forme de contrainte de pavage de Wang [104]. D'autres contraintes et propriétés pour certaines villes ou systèmes de routes peuvent être facilement ajoutées. Toutefois, si elles limitent trop sévèrement le nombre de tuiles satisfaisant le schéma d'occultation, ces contraintes devraient être intégrées dans les règles mêmes de génération des *gridels* occultants afin d'assurer l'efficacité du système entier de génération de tuiles occultantes.

Les villes extrudées en 3D des figures 2.12 et 2.13(gauche) respectent ces constraintes.

2.10.2 Hauteurs des immeubles

De par le fait que la visibilité est calculée en 2D, les hauteurs des immeubles ne sont pas supposées pouvoir varier. Toutefois, si l'observateur circule en-dessous d'une hauteur maximale limite par rapport au sol, il est possible de faire varier les hauteurs des immeubles dans un certain intervalle et de toujours assurer que les polygones situés



(a) Vue aérienne. (b) Vue du sol.

Figure 2.12 – Une ville pavée par des tuiles occultantes. La cellule de vue en bleu est entourée par le premier anneau de voisinage de tuiles en rouge. Le second anneau de voisinage ainsi que les suivants, rendus en gris, ne sont pas visibles depuis la cellule de vue si l'observateur est situé entre le sol et le toit du plus petit immeuble. Les deux contraintes de connexité sont prises en compte.

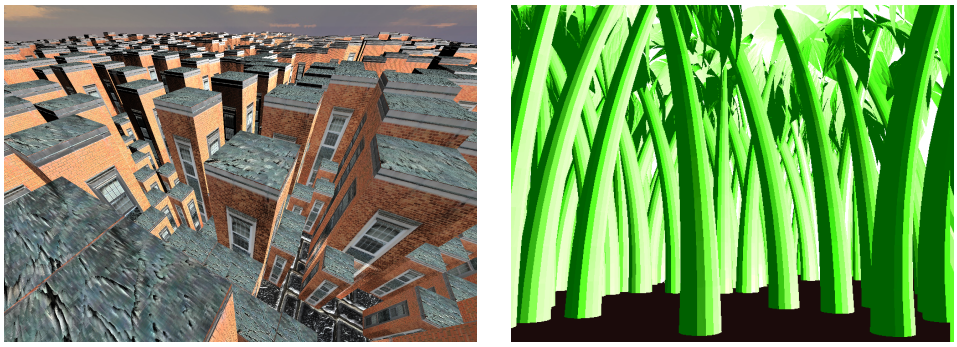


Figure 2.13 – Gauche : une ville extrudée de nos bloqueurs 2D ; le positionnement des immeubles garantit l'occultation complète du champ de vue à une distance donnée tant que l'observateur est situé en-dessous d'une hauteur spécifique. Droite : des palmiers positionnés par pavage d'occultation. L'occultation est basée sur un carré interne situé à la base de chaque arbre. La distribution garantit également que deux troncs d'arbre n'occupent pas deux *gridels* adjacents. Toutefois, étant donné que les troncs de palmiers sont courbes et rétrécissent, l'occultation n'est plus valide en augmentant la hauteur du palmier.

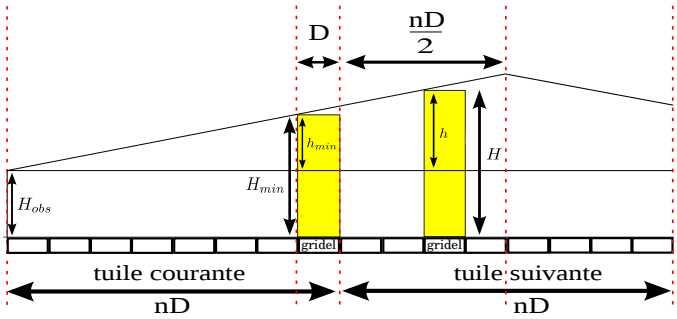


Figure 2.14 – Les variations des hauteurs des immeubles dépendent de la hauteur de l'observateur et de la taille des tuiles. Cette dernière est exprimée en nombre de *gridels*.

au-delà du premier anneau de pavage restent occultés.

La figure 2.14, illustre une tuile de résolution $n \times n$ *gridels*, avec chaque *gridel* ayant pour longueur de diagonale D (la tuile a donc pour longueur de diagonale nD). Supposons également qu'un immeuble occupe exactement un *gridel*. Un observateur à une hauteur maximale H_{obs} se déplace sur une tuile, i.e., la cellule de vue. Dans le pire des cas, l'observateur est situé dans le coin éloigné d'une tuile, l'immeuble de plus petite taille H_{min} est dans le coin opposé de la même tuile, et nous souhaitons placer un immeuble de taille H au *gridel* $i \in [0, \lceil n/2 \rceil - 1]$ sur la tuile suivante (appartenant au premier anneau de voisinage). On souhaite assurer qu'il soit caché par le plus petit immeuble. On notera que la limite supérieure de i à $\lceil n/2 \rceil - 1$ est due au fait que la hauteur maximale est permise seulement dans les *gridels* du centre de la tuile, car cette hauteur diminue symétriquement jusqu'à atteindre le bord opposé de la tuile.

Soit $h_{min} = H_{min} - H_{obs}$ et $h = H - H_{obs}$. Selon le théorème de Thales :

$$\frac{h_{min}}{(n-1)D} = \frac{h}{nD + iD}$$

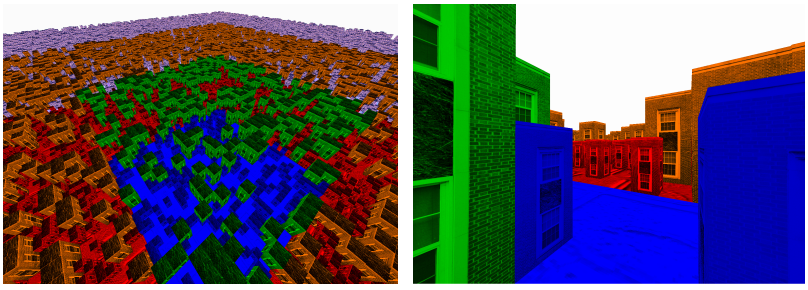
qui se simplifie en :

$$h = h_{min} \times \left(\frac{n+i}{n-1} \right). \tag{2.1}$$

Alors, n'importe quel immeuble de hauteur inférieure à $h + H_{obs}$ sera occulté par le plus petit immeuble.

2.10.3 Multi-pavage

Une manière de fournir un peu plus de flexibilité sur les variations permises dans les hauteurs des immeubles est de combiner des tuiles de différentes résolutions. Quatre tuiles de résolution $n \times n$ occupent exactement une tuile de $2n \times 2n$. Si on préserve le même rapport de longueur (hauteur immeuble / taille de *gridel*), les *gridels* des résolutions supérieures fournissent une occultation complète pour des immeubles plus fins et plus petits, tandis que les *gridels* issus des résolutions inférieures fournissent une occultation complète plus éloignée pour des immeubles plus grands et plus gros. Ce concept est illustré à la figure 2.15, pour une ville constituée de deux différents niveaux de pavage. Lors de l'instanciation d'une tuile, les immeubles occupant des *gridels* de niveaux inférieurs peuvent être ignorés s'ils sont situés dans des *gridels* de niveau supérieurs également occupés.



(a) Vue aérienne. (b) Vue du sol.

Figure 2.15 – Combiner le pavage à différentes échelles pour une ville. Le premier niveau a sa cellule de vue en bleu et son premier anneau de pavage en rouge. Le second niveau a sa cellule de vue en vert et son premier anneau de pavage en orange.

2.11 Conclusions

Nous avons présenté une méthode pour générer des tuiles satisfaisant des propriétés d'occultation afin de donner une limite au nombre maximal d'objets visibles depuis n'importe quel endroit sur un pavage créé à partir de ces tuiles.

Les tuiles sont de simples carrés subdivisés en grilles de bloqueurs, mais montrent une certaine efficacité lors du test d'occultation, de la génération à résolutions mul-

tiples, de leur stockage et de l'application d'autres contraintes de positionnement. Avec le schéma de test d'occultation $\bar{u}$ dans les quatre directions canoniques, seul l'affichage du premier anneau de voisinage suffit à bloquer toutes les lignes de vue. De plus, chaque tuile générée peut être utilisée indépendamment de son voisinage, c'est-à-dire que le positionnement d'une instance de tuile ne contraint pas celui des autres.

Bien que la visibilité soit calculée en 2D, elle reste conservatrice pour des objets extrudés à partir des *gridels* occultants. Nous avons démontré son potentiel pour la génération de villes, mais le pavage d'occultation peut être appliqué à d'autres types d'instances, telles que des troncs d'arbre pour une scène de forêt. Nous montrons à la figure 2.16, que l'étendue maximale des éléments visibles dans une scène de forêt peut être tout de même aisément calculée si on considère l'extrusion d'un *gridel* correspondant à l'enveloppe interne des troncs d'arbres. Toutefois, pour une scène constituée par exemple de palmiers, comme illustré à la figure 2.13(droite), les troncs sont courbes et rétrécissent, il n'y a alors plus d'occultation conservatrice générée en hauteur.

Notre algorithme peut être intégré dans un outil de modélisation pour être facilement contrôlé par un infographiste lors de la génération de tuiles occultantes. De simples probabilités dessinées peuvent servir de contrôle sur l'aspect général des tuiles. Notre méthode peut aussi être utilisée pour optimiser des positions d'objets ou pour suggérer des déplacements favorisant l'occultation. L'analyse de l'occultation des tuiles est notamment étudiée en section 4.1 et par cette approche, nous pensons seulement avoir entraperçu le potentiel d'outil de création de mondes complexes garantissant de bonnes performances de rendu.

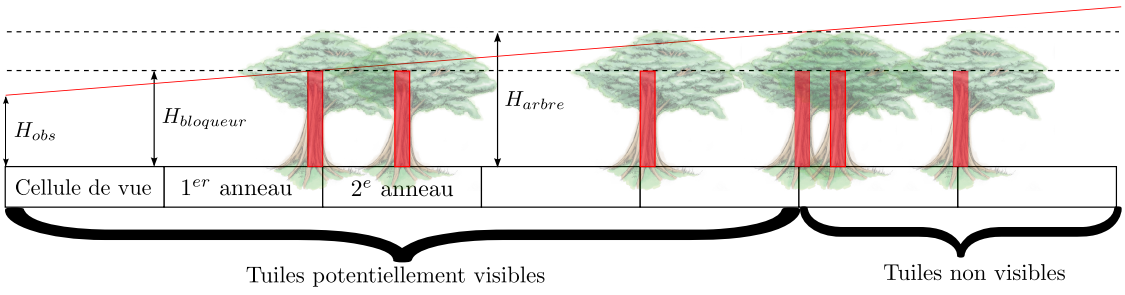


Figure 2.16 – Les occultations sont basées sur le carré interne à la base de l'arbre. On peut ensuite calculer l'extension maximale des éléments potentiellement visibles.

CHAPITRE 3

MINIMISATION DE L'ÉTENDUE D'UN MONDE VIRTUEL PAR PAVAGE D'OCCULTATION

Ce chapitre présente plusieurs méthodes permettant de relâcher les contraintes de construction de l'ensemble de pavage en permettant la propagation de la visibilité en 2D (section 3.1) ainsi que l'extension de ce principe à des tuiles 3D (section 3.2). Enfin, en section 3.3, nous présentons une méthode plus avancée, basée également sur des tuiles 3D, permettant de bénéficier de performances en temps réel.

3.1 Propagation de la visibilité en 2D

Les contributions exposées dans cette section ont donné lieu à deux posters [35, 36], présentés à la conférence *GRAND* en mai 2011 et mai 2012.

Bien qu'elle garantisse d'excellentes performances de rendu, la méthode présentée au chapitre précédent présente une limitation majeure due au fait que seul le premier anneau de voisinage (complètement occultant) est instancié. En effet, le design des tuiles est parfois beaucoup trop contraint dans le placement des bloqueurs. Il est donc judicieux de garder la possibilité d'instancier des tuiles à la demande, tout en permettant de ne pas seulement utiliser des occultations complètes. Un tel exemple serait une ville dont certaines rues droites couvrent plusieurs tuiles de suite, ou une clairière dans une forêt.

Le principe de visibilité s'exprime de la même manière que précédemment, à savoir qu'une tuile est considérée comme une cellule de vue, à la différence que la visibilité n'est plus bornée par le premier anneau de voisinage. On doit donc vérifier quelles sont les tuiles visibles depuis la cellule de vue. Pour cela, nous présentons deux méthodes utilisables indépendamment pour propager la visibilité au-delà du premier anneau de voisinage et construire le PVS : (1) les enveloppes convexes des polygones représentant les régions occupées (i.e., les parties occultantes) de chaque tuile, (2) la fusion de bloqueurs indiquant, pour un niveau de résolution donné, quelles sont les parties visibles

de la tuile depuis des segments de vue bordant la tuile. Ces deux méthodes permettent de propager la visibilité sur le pavage et d'instancier les tuiles nécessaires au blocage de toutes les lignes de vue.

En section 3.1.1, nous expliquons comment importer les objets créés soit par un infographiste, soit par une quelconque méthode de génération de géométrie. Nous présentons brièvement comment extraire l'information utile au calcul de l'occultation d'une tuile, i.e., la voxélisation conservatrice d'un ou plusieurs objets. Dans les sections 3.1.2 et 3.1.3, nous présentons deux méthodes différentes permettant de propager la visibilité à partir de la voxélisation conservatrice des tuiles ainsi que différentes optimisations.

3.1.1 Design de tuiles

Un infographiste construit un ensemble de tuiles grâce auxquelles il peuplera son monde. Chaque tuile est transformée en une version conservatrice voxélisée des objets qu'elle contient. Lorsque nous considérons une géométrie importée depuis n'importe quel logiciel de modélisation, nous réduisons les géométries à leur version 2D conservatrice lors du calcul de visibilité. C'est-à-dire qu'un *gridel* est considéré comme occultant s'il est entièrement situé à l'intérieur d'un objet de la tuile de la tuile. Un exemple de résultat de voxélisation sur polygone est présenté à la figure 3.1. On associe à chaque voxel un booléen représentant cette information de visibilité. Ainsi, nous pouvons procéder aux précalculs sur l'ensemble de pavage nécessaires au calcul du PVS.

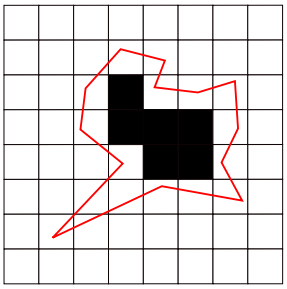


Figure 3.1 – Le résultat de la voxélisation conservatrice (en noir) d'un objet quelconque (contour en rouge).

3.1.2 Calcul de PVS par unions de polygones et tests d'intersections

3.1.2.1 Précalculs

Comme pour le schéma d'occultation *ū-shape*, nous calculons, pour chaque tuile, les lignes séparatrices intersectant les arêtes de la tuile. Cependant nous utilisons l'intégralité des trois autres segments. En effet, il faut pouvoir déterminer chacune des directions probables de propagation de la visibilité.

Lorsque toutes les droites séparatrices ont été construites et appariées, pour chaque paire de lignes séparatrices, on calcule la paire de polygones convexes qu'elles séparent avec l'enveloppe convexe de l'union des *gridels* occultants. Ils sont ensuite appariés et stockés pour chaque tuile et pour chaque combinaison de directions entrantes et sortantes.

3.1.2.2 Calcul de PVS par propagation de visibilité

Au départ l'observateur se positionne où il le souhaite dans la scène. Ensuite grâce à la fonction de choix de tuile f (définie en section 2.3), on récupère l'identificateur de la tuile locale et on l'ajoute au PVS. Ensuite, pour propager la visibilité depuis cette cellule de vue, si celle-ci possède des droites séparatrices interceptant la direction de propagation traitée, on génère l'identificateur pour la tuile adjacente et on l'ajoute au PVS si elle n'a pas encore été ajoutée à celui-ci pour cette position. Il en va de même pour chaque tuile que l'on traversera. Afin d'obtenir la visibilité "résiduelle" d'une combinaison de deux tuiles, pour une direction donnée, on teste chaque paire de polygones de la tuile courante avec chaque paire de la tuile suivante. Pour déterminer que l'intersection d'une paire de polygones (p_1, p_2) d'une tuile avec une paire (p'_1, p'_2) d'une autre tuile est occultante, on doit respecter les conditions suivantes :

- $(p_1 \cup p'_1) \cap (p_2 \cup p'_2) \neq \emptyset$
- $(p_1 \cup p'_2) \cap (p_2 \cup p'_1) \neq \emptyset$
- $(p_1 \cup p'_1 \cup p'_2) \cap p_2 \neq \emptyset$
- $(p_2 \cup p'_1 \cup p'_2) \cap p_1 \neq \emptyset$

Si elle ne l'est pas, on calcule les droites séparatrices des nouveaux polygones obtenus et on réitère la même opération pour chacune des combinaisons de paires non occultantes trouvées avec les polygones de la tuile suivante jusqu'à ce qu'il n'y ait plus de droites séparatrices. La librairie CGAL [1] est utilisée pour effectuer les opérations d'unions et d'intersections entre polygones.

Ainsi nous calculons les lignes séparatrices et testons les directions interceptées afin de propager la visibilité vers les directions appropriées. Ce concept est illustré à la figure 3.2.

3.1.2.3 Optimisations

Lorsque les tuiles sont peu densément peuplées, on peut avoir des problèmes de calculs trop coûteux. Cela ne permet pas le calcul du PVS en temps réel et s'explique par le fait que les polygones générés après chaque étape peuvent avoir de plus en plus de sommets et par le nombre excessif de paires de polygones à intersecter. Les unions de polygones, le calcul de leurs enveloppes convexes et les tests d'intersections deviennent alors très coûteux.

La simplification des polygones après chaque étape de propagation dès lors que le nombre de sommets est trop grand, peut réduire ce problème. Cette simplification peut se faire selon une métrique de minimisation de déformation (e.g., distance de Hausdorff). Cette optimisation permet de diminuer considérablement les temps d'exécution des opérations sur les polygones. Cependant elle implique que l'occultation générée par les bloqueurs devient encore plus conservatrice qu'avec l'occultation de base et crée donc plus de "trous" qu'il n'y en a. De ce fait, le temps d'exécution de l'algorithme de calcul du PVS par propagation ne diminue pas de façon assez significative.

Dans la section suivante, nous présentons une méthode de calcul de PVS par fusion de bloqueurs, qui propage la visibilité sur le pavage plus rapidement et permet donc de résoudre en grande partie les problèmes de performance.

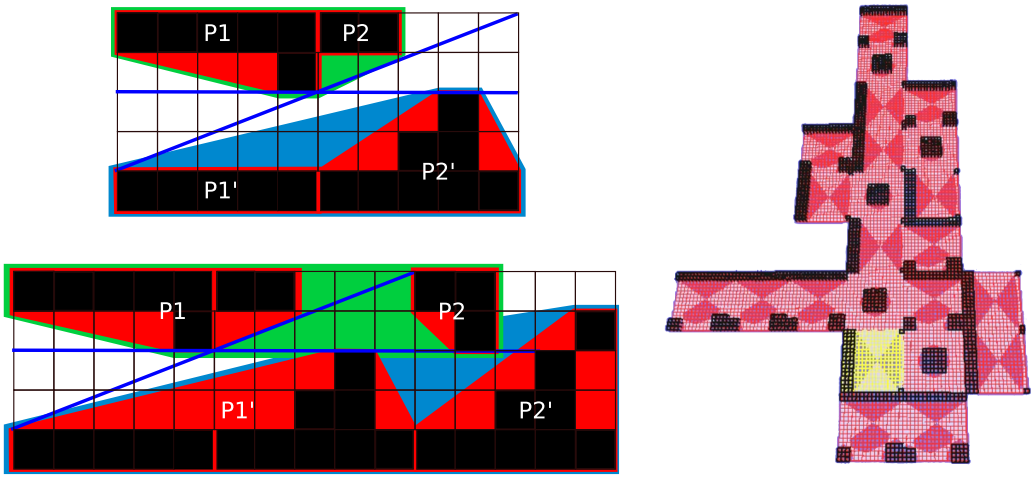


Figure 3.2 – Gauche : l'union des enveloppes convexes p_1 et p_2 (en vert) et p'_1 et p'_2 (en bleu). Lorsque ces unions ne s'intersectent pas, la visibilité est propagée (haut). Lorsqu'elles s'intersectent, aucune ligne séparatrice ne peut être créée et la propagation s'arrête (bas). Droite : l'utilisation de polygones convexes pour le calcul d'un PVS en temps réel. La cellule de vue est affichée en jaune.

3.1.3 Calcul de PVS en temps réel par fusion de bloqueurs

En section 3.1.2, nous nous heurtons à des problèmes de complexité dus au fait qu'une tuile assez densément occupée ne garantit pas forcément un taux suffisant d'occultation. Nous avons donc choisi d'adopter une approche du cas 2D qui est non seulement plus rapide en terme de complexité (gain de deux ordres de grandeur) mais aussi plus précise en ce qui concerne le taux et la localisation de l'occultation. De plus, la notion de visibilité ne contraint plus la création des tuiles. Cette approche s'inspire en partie de la méthode de l'occultation volumique de Schaufler et al. [89] décrite en section 1.2.2 et illustrée à la figure 3.3.

3.1.3.1 Précalculs

Vecteurs d'occultation par fusion de bloqueurs 2D Un vecteur d'occultation représente les parties visibles d'une bordure de tuile depuis un segment de vue. Par exemple, la figure 3.5 illustre les trois vecteurs d'occultation associés à un segment de vue bordant la tuile (représenté par un rectangle rouge). Les parties non visibles depuis le segment

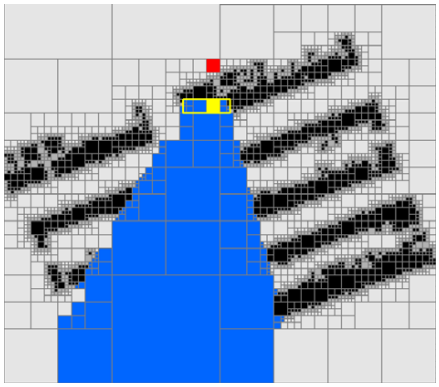


Figure 3.3 – La fusion des bloqueurs par occultation volumique conservatrice de Schaufler et al. [89].

de vue sont en bleu, les parties visibles (même partiellement) sont en jaune. Pour chaque segment de vue bordant la tuile, l'algorithme de fusion de bloqueurs permet en une seule passe de calculer ces trois vecteurs d'occultation.

La figure 3.4 montre différentes étapes de la construction des lignes de support et comment les "ombres" d'occultation (*shafts*) s'étendent en parcourant les bloqueurs à partir du segment de vue en rouge. La figure 3.5 montre comment ces ombres vont mettre à jour les vecteurs d'occultation représentant les parties visibles et non visibles depuis un segment de vue. D'autres exemples plus complexes sont donnés dans la figure 3.6. Les algorithmes 2, 3 et 4 décrivent le processus de calcul en une passe des ombres d'occultation entre les différentes cellules de vue et les bloqueurs de la tuile.

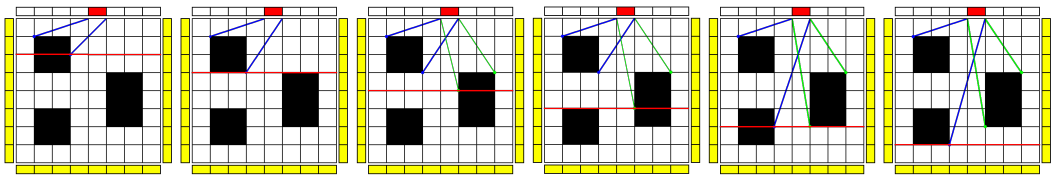


Figure 3.4 – De gauche à droite : la construction des lignes de support des bloqueurs, et leur extension, ligne de *gridels* par ligne de *gridels* horizontalement à partir d'un segment de vue, représenté par un rectangle rouge bordant la tuile en haut.

La figure 3.7 et le tableau 3.1 comparent les temps de calcul des vecteurs d'occultation d'une tuile avec ceux de la methode du $\bar{u}$-*shape* de la section 2.5 et ce, pour différentes résolutions et différentes densités d'occupation. Nous pouvons constater que les

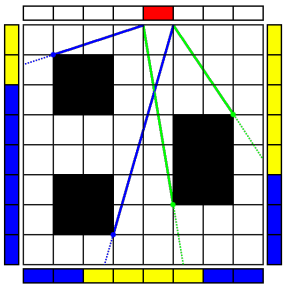


Figure 3.5 – Le marquage des vecteurs d'occultation par les ombres de contact. Ici, deux "ombres" d'occultation (*shafts*) sont créées : l'une est délimitée par deux lignes de support bleues ; l'autre est délimitée par deux lignes de support vertes.

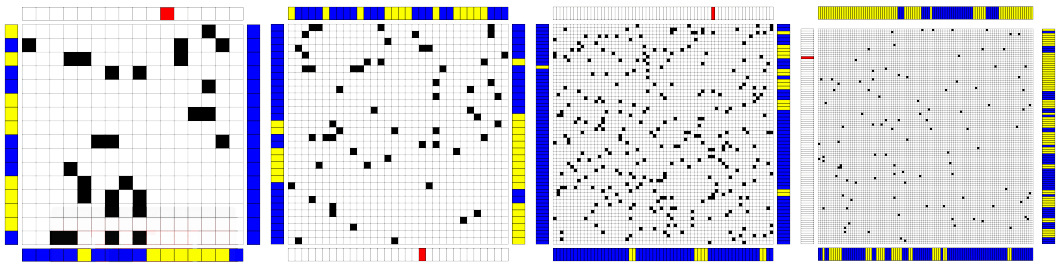


Figure 3.6 – Exemples de vecteurs d'occultation pour des tuiles de dimensions 16×16, 32×32, 64×64 et 100×100.

temps de calcul de la fusion de bloqueurs sont nettement moindres. Ceci s'explique par la diminution de la complexité par rapport à la méthode précédente qui calcule toutes les lignes séparatrices possibles existantes dans une tuile, alors que la fusion des bloqueurs ne traite que les lignes de support utiles.

Vecteurs de visibilité Dans cette section, nous présentons une structure précalculée pour l'adjacence de deux tuiles T_1 et T_2 qui permet l'accès rapide aux cellules potentiellement visibles d'une tuile depuis une cellule appartenant à la tuile adjacente. Nous calculons les lignes séparatrices entre chaque cellule de vue entrante T_1^{in} et chaque cellule de vue sortante T_1^{out} de T_1. La tuile T_2 contient des vecteurs de visibilité indiquant quelles sont les cellules T_2^{out} potentiellement visibles depuis T_1^{in} au travers de T_1^{out}. Ainsi, les lignes séparatrices agissent comme des portails (*portals*) sur la tuile T_2 et servent à mettre à jour ces vecteurs de visibilité en positionnant un booléen à "vrai" si la cellule est partiellement ou complètement visible. Ce calcul est illustré par la figure 3.8.

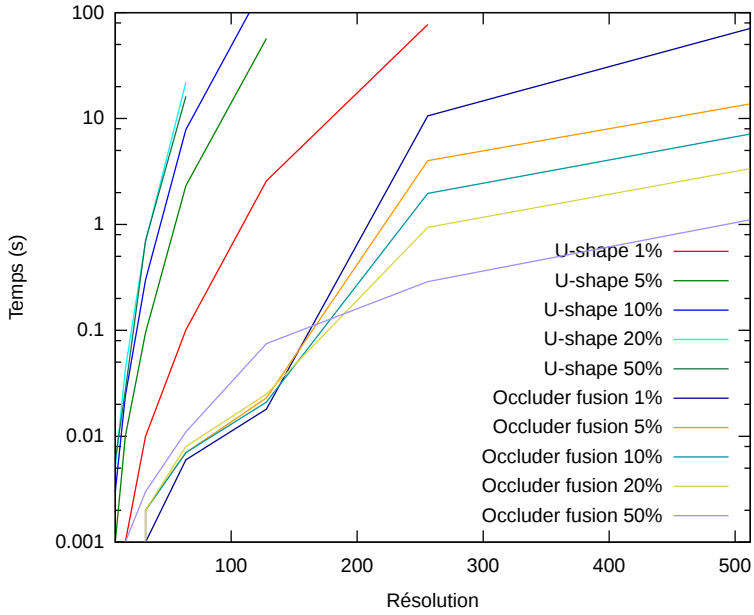


Figure 3.7 – Comparaison des temps de calcul (échelle logarithmique) de la méthode du *ū-shape* avec la fusion de bloqueurs en fonction de la résolution des tuiles pour certaines densités d'occupation par les bloqueurs.

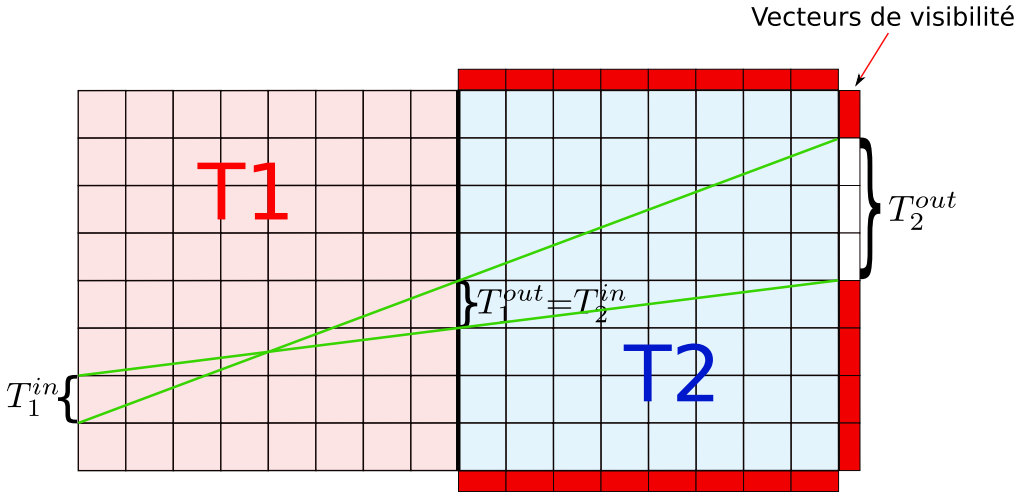


Figure 3.8 – Le précalcul d'un vecteur de visibilité par propagation des droites séparatrices à la tuile suivante.

Résolution	Densité (%)	Temps de calcul en secondes	
		Fusion de bloqueurs	*ū-shape*
8×8	1	< 0.000	0.000
	5		0.001
	10		0.003
	20		0.005
	50		0.006
16×16	1	≤ 0.001	0.001
	5		0.010
	10		0.024
	20		0.043
	50		0.027
32×32	1	0.001	0.010
	5	0.002	0.095
	10	0.002	0.299
	20	0.002	0.687
	50	0.003	0.704
64×64	1	0.006	0.101
	5	0.007	2.320
	10	0.007	7.868
	20	0.008	21.951
	50	0.011	16.224
128×128	1	0.018	2.575
	5	0.023	56.994
	10	0.021	198.407
	20	0.025	NA
	50	0.075	NA
256×256	1	10.581	77.224
	5	4.004	NA
	10	1.964	NA
	20	0.940	NA
	50	0.288	NA
512×512	1	70.993	NA
	5	13.745	NA
	10	7.152	NA
	20	3.366	NA
	50	1.108	NA

Tableau 3.1 – Tableau comparatif des temps de calcul pour obtenir des résultats avec la méthode *ū-shape* du chapitre 2, et celle de cette section.

```
pour chaque Cellule de vue faire
    Projections ← ∅
    si gridel voisin direct bloqué alors
        Marquer toutes les cellules visibles
        CalculProjs(Projections). [Algo. 3]
        Marquer les cellules occultées par les ombres projetées
    sinon
        Marquer toutes les cellules occultées
    fin
fin
```

Algorithme 2: Précalcul des vecteurs d'occultation.

```
pour chaque ligne ou colonne (ordonnées d'avant en arrière) faire
    Projections_Temporaires ← ∅
    pour chaque Bloqueurs (de gauche à droite) faire
        si
        CalculProj(Projections, Projections_Temporaires, Bloqueur). [Algo. 4]
        alors
            Projections ← Projections ∪ Projections_Temporaires
            retourner
        fin
    fin
    Projections ← Projections ∪ Projections_Temporaires
fin
```

Algorithme 3: CalculProjs(*Projections*).

Ce précalcul fait, nous pouvons le stocker et ce résultat est également valable pour toutes les tuiles du pavage au vu de sa généricité.

3.1.3.2 Calcul de PVS par propagation de visibilité en temps réel

Opérations binaires Nous avons mis en place un système de propagation de la visibilité en temps réel permettant de calculer le PVS d'une tuile de vue et de l'afficher. Nous considérons le périmètre de la cellule de vue comme un ensemble de segments de vue et nous propageons la visibilité à partir de chacun d'entre eux. Pour illustrer le principe, trois segments de vue parmi 32 sont mis en évidence sur une tuile de résolution 8×8 à la figure 3.9. Cette étape permet d'initialiser la propagation de visibilité.

Pour chaque tuile de l'ensemble de pavage, nous disposons de deux informations nécessaires, à savoir, la visibilité au sein d'une tuile ainsi que les cellules potentiellement visibles au travers de cette dernière. De ce fait, le processus de propagation en temps réel

```
bool intersection ← faux
bool partiel ← faux
Union_Projections ← ∅
pour chaque P ∈ Projections faire
    Inter ← intersection(Bloqueur, P)
    si Inter = ∅ alors
        si P.englobe(Bloqueur) alors
            intersection ← true
        fin
    sinon
        Projections ← Projections − {P}
        Union_Projections ← Union_Projections ∪ {P}
        intersection ← true
        partiel ← true
    fin
fin
si ¬intersection ∨ partiel alors
    si Union_Projections ≠ ∅ alors
        P ← Bloqueur.Update(Union_Projections)
    sinon
        P ← Projection(Bloqueur)
    fin
    Projections_Temporaires ← Projections_Temporaires ∪ {P}
    si P.englobe(lignes_ou_colomnes_restantes) alors
        retourner vrai
    fin
fin
retourner faux
```

Algorithme 4: CalculProj(*Projections, Projections_Temporaires, Bloqueur*).

ne contient aucun calcul d'intersection, ni d'union de polygones. Pour cela, il consulte l'information locale grâce aux vecteurs d'occultation et l'information de propagation grâce aux vecteurs de visibilité. Ensuite, le processus de propagation réalise une simple opération booléenne (opérateur ET) entre les éléments des vecteurs d'occultation et de visibilité de la tuile T_2. Ce processus est illustré dans la figure 3.10.

Mise à jour des droites séparatrices Toutefois, afin de devenir efficace, le système nécessite de valider une séquence de portails (segments de vue consécutifs) à la volée car le précalcul des vecteurs d'occultation et de visibilité permettent effectivement d'accéder plus rapidement aux tuiles potentiellement visibles depuis une tuile adjacente, mais ne testent en aucun cas si la tuile est visible depuis la cellule de vue. Il faut donc, pour chaque "chemin de visibilité", initialiser les deux lignes séparatrices entre les deux

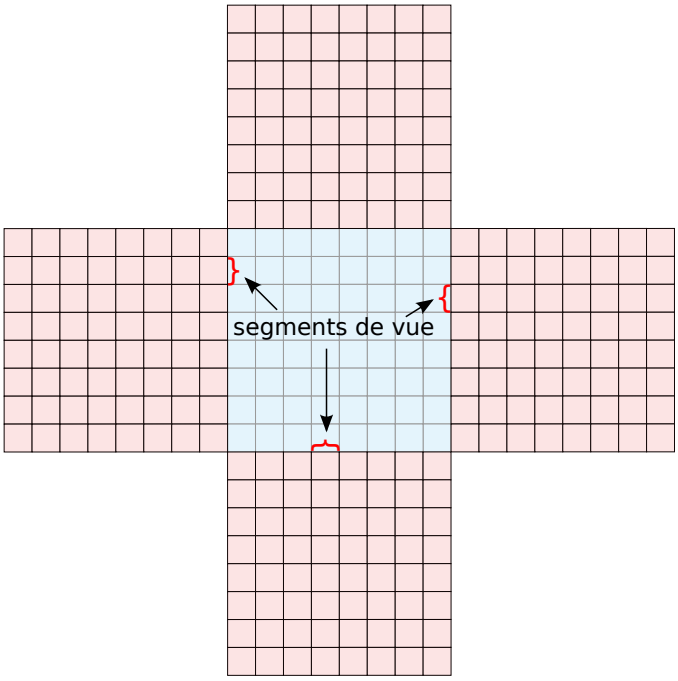


Figure 3.9 – Trois segments de vue parmi les 32 considérés depuis la tuile de vue (au centre). Ici on considère 8×4 (taille de la tuile $\times$ nombre de directions) segments de vue qui initialisent la propagation de visibilité.

premières tuiles. Puis, lors du test d'une prochaine tuile potentiellement visible, il faut tester si le nouveau portail est bien situé entre les lignes séparatrices et mettre à jour ces dernières en conséquence. Ce processus est illustré par la figure 3.11.

Résultats

Le tableau 3.2 montre pour 100 cellules de vue ainsi qu'une scène et une composition de tuiles aléatoires, le temps moyen de calcul d'un PVS et le nombre moyen de bloqueurs qu'il contient selon différentes résolutions et densités. Les calculs ont été réalisés sur un CPU Intel® Core™ i7-930 (QuadCore) 2.8 GHz et 16 Go de mémoire vive.

Nous pouvons voir que pour des densités d'occupation relativement faibles nous pouvons garantir un calcul de PVS en temps réel. Tout dépendant de la puissance de calcul de la machine utilisée, nous pouvons prévoir quel sera le temps moyen de calcul du PVS pour une scène générée aléatoirement à la volée. Il est également possible de

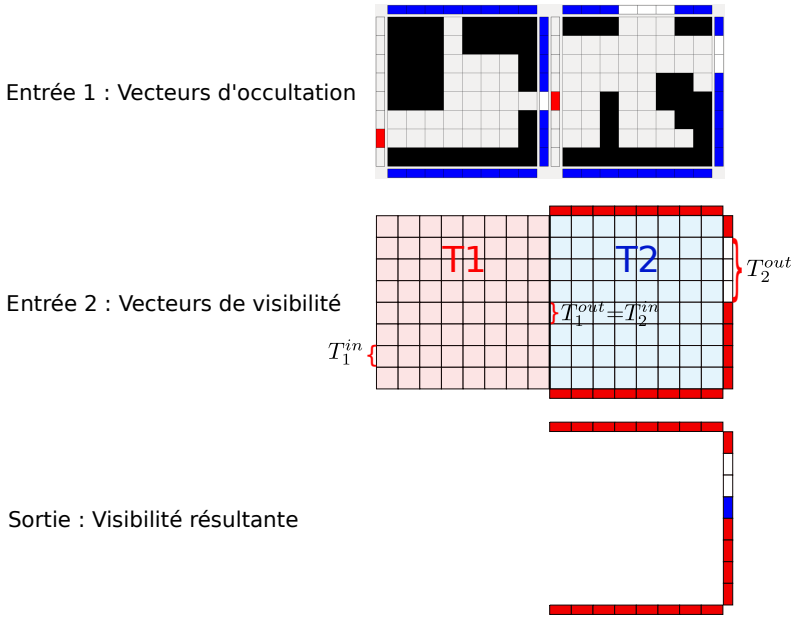


Figure 3.10 – Le processus de propagation utilisant un test binaire.

prévoir une limite au nombre moyen de polygones à afficher et par conséquent de garantir une performance moyenne minimale souhaitée pour le rendu de la scène.

3.1.4 Optimisations

À partir d'une certaine distance de propagation, l'algorithme peut devenir coûteux lorsque la densité de bloqueurs est trop faible. Cela implique que le précalcul sur chaque tuile génère trop de paires de polygones convexes ou de portails de visibilité. Nous proposons alors quelques optimisations qui peuvent réduire en partie ce problème.

3.1.4.1 Taille et résolution des tuiles

L'optimisation la plus triviale est d'augmenter la taille et la résolution des tuiles. Autrement dit, on peut remplacer quatre tuiles de résolution identique par un groupement de ces quatre tuiles. Ce groupement peut être obtenu par une permutation telle qu'illustrée dans la figure 3.12.

Cette optimisation permet de diminuer le nombre de "trous" dans les tuiles trop peu densément peuplées et peut être implémentée en groupant de manière systématique les

Résolution	Densité (%)	Temps moyen de calcul en secondes	Ecart type temps	Nombre moyen de bloqueurs	Ecart type nb bloqueurs
8×8	20	0.004	0.001	749	3.3
	17	0.009	0.003	987	13.4
	15	0.016	0.006	1261	7.7
	12	0.067	0.030	1944	6.9
	11	0.131	0.028	2258	33.3
	10	0.730	2.459	2905	36.3
16×16	15	0.006	0.001	2122	37.2
	10	0.037	0.008	4051	12.1
	7	0.301	0.009	7799	49.0
32×32	10	0.012	0.002	5618	38.7
	7	0.540	0.006	9828	19.1
	6	0.114	0.017	12960	50.4
	5	0.321	0.052	18727	71.0
	4	1.307	0.217	27858	96.0
64×64	7	0.021	0.002	12429	102.2
	5	0.071	0.006	19674	77.7
	4	0.170	0.013	28303	125.0
	3	0.724	0.062	46915	186.2
128×128	4	0.068	0.004	31616	90.0
	3	0.173	0.010	46154	382.4
	2	0.875	0.048	84742	74.7
256×256	4	0.076	0.002	50732	133.7
	3	0.120	0.004	56197	48.0
	2	0.296	0.013	84312	630.0
	1	4.552	0.188	237676	97.1
512×512	3	0.207	0.013	90167	322.8
	2	0.326	0.008	118363	1237.7
	1	1.301	0.001	209606	1151

Tableau 3.2 – Temps moyen de calcul par cellule de vue du PVS (effectuée sur 100 cellules de vue différentes).

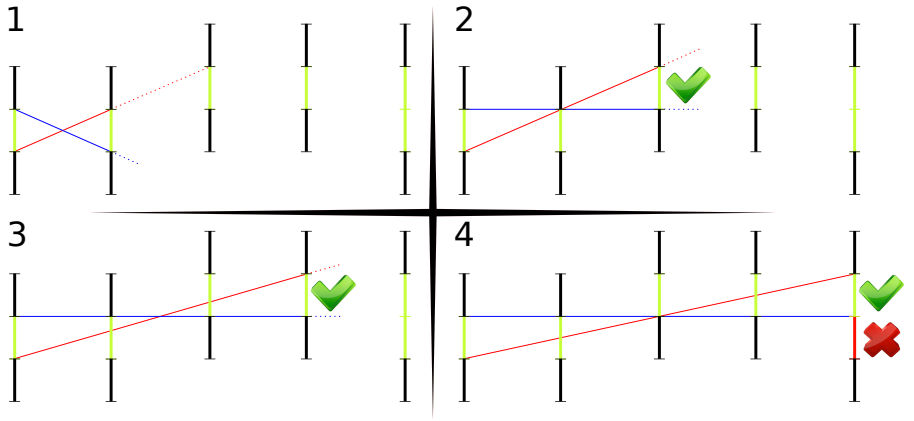


Figure 3.11 – La mise à jour des lignes séparatrices lors du parcours d'une séquence de portails, et le test de validité associé.

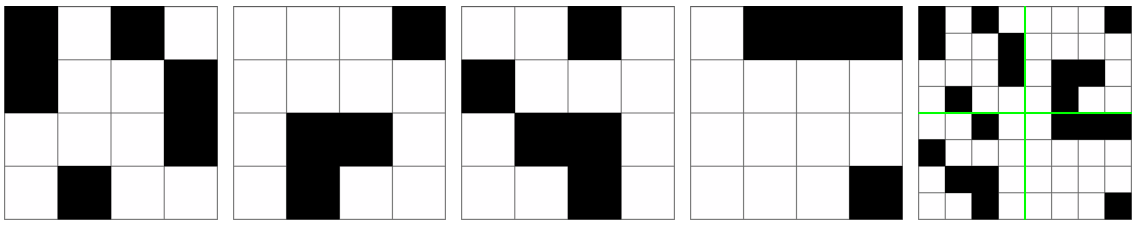


Figure 3.12 – Un exemple de groupement de quatre tuiles en une tuile de résolution supérieure, augmentant le taux d'occultation. Les tuiles (a), (b), (c) et (d) sont ici groupées dans la tuile (e).

tuiles qui possèdent un trop grand nombre de paires de polygones convexes ou des vecteurs d'occultation ayant trop de portails.

3.1.4.2 Intégration des niveaux de hauteur

Contrairement au pavage d'occultation qui limite la distance de l'observateur au premier anneau de pavage – où, par conséquent, il est possible de pouvoir calculer un intervalle de variation de hauteur des bloqueurs pour un niveau de propagation donné – le fait de propager, potentiellement indéfiniment, le pavage en utilisant cette méthode, ne permet pas de faire varier les hauteurs.

Il est toutefois possible de créer un monde ayant plusieurs niveaux de hauteurs prédéfinies en propageant le pavage, itérativement sur chacun de ces niveaux. Pour cela,

nous fixons un nombre τ de niveaux de hauteurs différents. A chaque niveau i est associé une hauteur H_i telle que $\forall i \in [1,\tau]$, $H_{i-1} < H_i$. Chacun des *gridels* d'une tuile doit appartenir à un de ces niveaux de hauteurs. Sa hauteur est donc fixée à l'avance. Lorsque l'on considère l'extrusion d'une hauteur H_i d'un *gridel* depuis la base de la tuile, cette extrusion est occultante pour les niveaux 1 à i mais ne l'est plus pour les niveaux supérieurs à i. Ensuite nous calculons l'ensemble des vecteurs d'occultation pour chaque niveau i en déterminant l'occultation de chaque *gridel* en fonction du niveau courant. Par exemple, lors de la première propagation de visibilité – niveau le plus bas – nous prenons en considération tous les *gridels*, i.e., nous pouvons paver l'espace avec toutes les instances de *gridels* déjà présentes dans chaque tuile visible. Pour l'itération suivante, nous considérons le niveau de hauteur suivant par ordre croissant et tous les *gridels* d'un niveau de hauteur inférieur sont considérés comme non occultants. Le processus de propagation est réitéré de la même manière pour les niveaux de hauteurs suivants, par ordre croissant.

Etendue maximale de pavage Au niveau i, on peut calculer D^i_{max}, la distance maximale, depuis la cellule de vue, au-delà de laquelle il n'est plus utile de propager un niveau de hauteur. Celle-ci est calculée par :

$$D^i_{max} = \frac{H_i \times D^{i-1}_{ext}}{H_{i-1}},\ i > 0$$

où H_i est la hauteur des bloqueurs au niveau i et D^{i-1}_{ext} est l'étendue maximale du pavage au niveau $i-1$. L'étendue d'un niveau dépend donc du niveau inférieur, et c'est pourquoi il est nécessaire de propager le pavage par ordre de niveau croissant. Le niveau 1, par définition, n'a pas de limite d'étendue ; on fixe alors $H_0 = +\infty$. Cette dernière hypothèse implique notamment que si le premier niveau n'a pas de limite, il est alors impossible de donner une valeur pour l'étendue maximale des niveaux supérieurs. Le premier niveau doit donc avoir une limite finie afin d'assurer que l'algorithme se termine.

Suppression des bloqueurs et des polygones trop bas Lors de l'exécution de l'algorithme de propagation sur les niveaux supérieurs, nous pouvons voir qu'il n'est pas né-

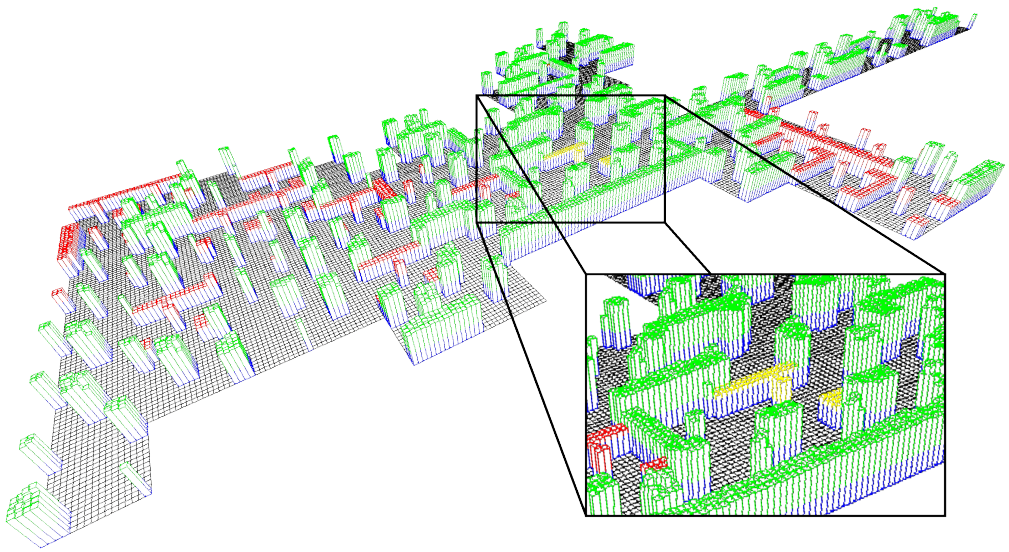
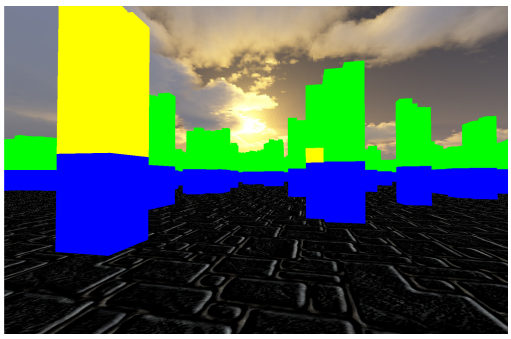


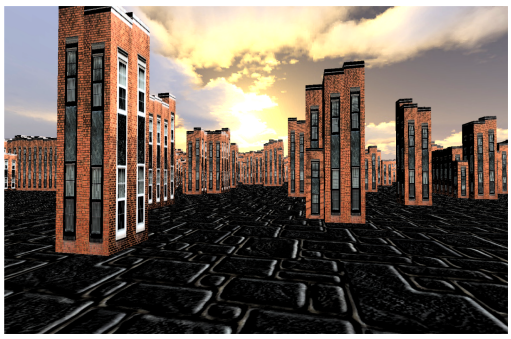
Figure 3.13 – Un exemple de PVS calculé par propagation de visibilité où les bloqueurs trop bas sont affichés en rouge et ne sont pas visibles depuis la cellule de vue (tuile en jaune).

cessaire d'afficher les objets situés plus bas que le niveau courant. On peut donc n'ajouter au PVS que les géométries du niveau de hauteur courant. La figure 3.13 illustre un exemple de PVS calculé ainsi. On peut aussi éviter d'ajouter au PVS les polygones de la géométrie associée aux bloqueurs si celle-ci est située en-dessous du niveau de hauteur courant.

La figure 3.14 montre un exemple de configuration, où la propagation de visibilité en 2D sur plusieurs niveaux de hauteurs permet de n'afficher uniquement que les bloqueurs potentiellement visibles depuis la cellule de vue. Cette méthode de propagation permet d'avoir une liberté totale sur le placement des bloqueurs dans chaque tuile.



(a) Vue simplifiée où les bloqueurs de la cellule de vue sont en jaune, et ceux des tuiles parcourues par l'algorithme de propagation de visibilité sont en vert. La partie bleue à la base des bloqueurs est la zone où l'observateur doit être situé pour garantir l'absence de *popping* lors du *walkthrough*.



(b) La même vue où les bloqueurs ont été simplement texturés pour leur donner une apparence d'immeuble. Par rapport au pavage d'occultation limité au premier anneau de voisinage, propager la visibilité permet de libérer plus d'espace pour mieux naviguer entre les bloqueurs composant le monde.

Figure 3.14 – La vue, depuis la cellule de vue, des immeubles instanciés par l'extrusion 3D des bloqueurs des tuiles 2D, après propagation de la visibilité en 2D.

3.1.5 Visibilité par échantillonnage

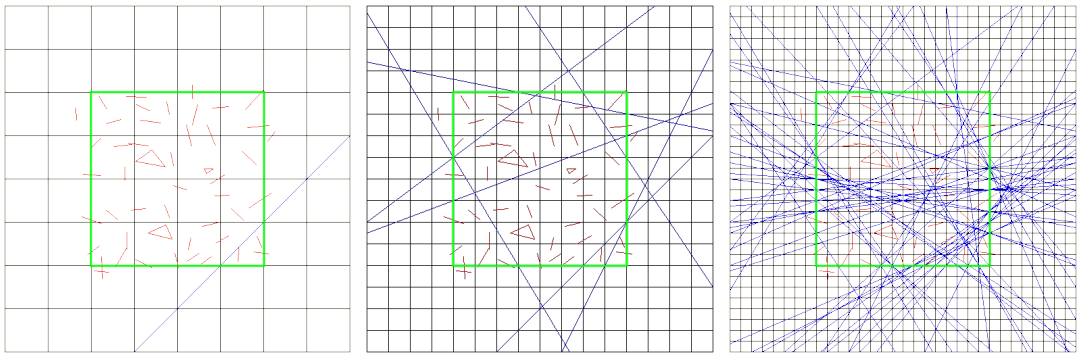
Lorsque les objets d'une scène sont trop petits et/ou trop fins, la voxélisation des objets résulte en un ensemble trop conservateur de voxels (trop peu de voxels occupés pour garantir une occultation efficace). Ce phénomène est lié directement au fait que la voxélisation est réalisée à une résolution trop faible (pour les objets trop petits) ou parce qu'un objet fin (un plan par exemple) ne pourra jamais occuper complètement un voxel. Calculer l'occultation combinée de ces objets serait potentiellement trop coûteux car nous ne pouvons pas bénéficier des mêmes simplifications utilisées en section 3.1.1. Pour ce genre de cas, il devient intéressant de réaliser le calcul de visibilité par échantillonnage, même si nous perdons l'aspect conservateur de la visibilité (notion définie en section 1.2) préservé jusqu'à présent.

Afin de maximiser le blocage d'une cellule composée de petits bloqueurs tels que présentés à la figure 3.15, une voxélisation conservatrice ne saurait être efficace. Pour cela, nous préférons discrétiser la cellule et tracer des rayons pour déterminer quelles sont les droites bloquées par les objets qui occupent celle-ci (par complémentarité, les droites visibles).

3.1.5.1 Stratégie de base

La première stratégie d'échantillonnage mise en place est toute simple. On échantillonne chacun des quatre segments de la cellule à tester (en vert sur la figure 3.15) par $n+1$ points. Ces points sont équidistants les uns des autres. Le premier et le dernier de ces points sont les extrémités du segment échantillonné. A partir de chacun des points générés de cette manière on essaie de relier chacun des autres points (échantillonnage $n \times n$). Si une telle droite intersecte la géométrie de la cellule, alors cette droite, qui correspond à une ligne de vue potentielle, est bloquée.

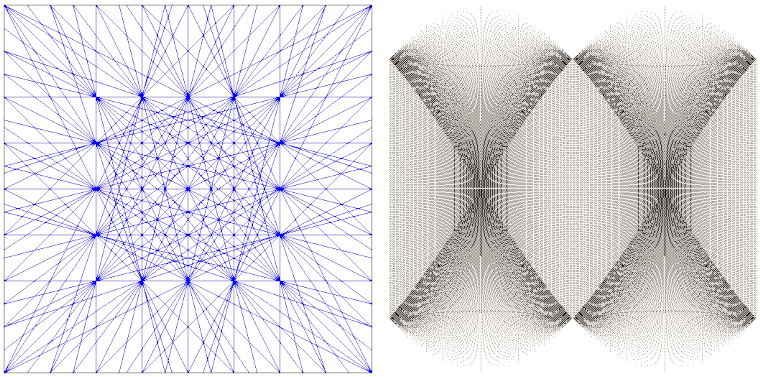
Evidemment, on remarque que selon la fréquence d'échantillonnage, on laisse la place à de grosses approximations sur le calcul de visibilité. Toutefois, nous ne nous intéressons pas ici à la fréquence d'échantillonnage à proprement parler, mais plutôt à sa qualité, à savoir si l'ensemble des lignes de l'espace est suffisamment et uniformément échantillonné. Pour visualiser le résultat de cette première stratégie d'échantillonnage, nous utilisons le même espace dual de lignes (θ, u) que Orti et Puech [76], où les droites sont supposées être équidistribuées. On considère alors la transformation duale qui à toute droite d'équation polaire : $y\cos\theta - x\sin\theta - u = 0$ associe le point de coordonnées (θ, u) dans l'espace dual tel que $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], u \in \left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$. La valeur u correspond à la distance algébrique de la droite à l'origine et θ son angle par rapport à l'axe (O, x).



(a) 4×4 (b) 8×8 (c) 16×16

Figure 3.15 – Un ensemble de petits bloqueurs dont la visibilité est échantillonnée à trois niveaux de résolution différents.

La figure 3.16 représente l'échantillonnage d'une cellule vide dans l'espace euclidien



(a) Espace euclidien (b) Espace dual

Figure 3.16 – Une droite dans l'espace euclidien est representée par un point dans l'espace dual (θ, u). $\theta \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right], u \in \left[-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$.

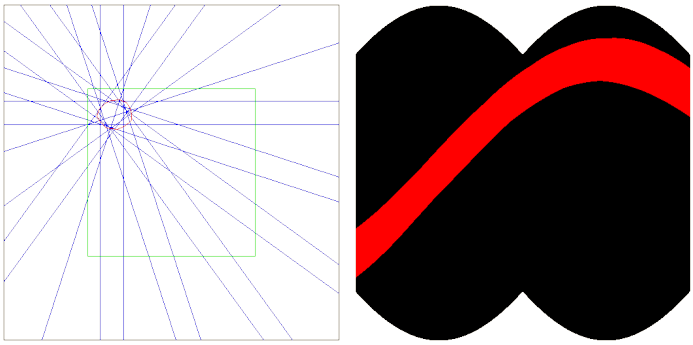
reporté dans cet espace dual. Nous avons dû augmenter suffisamment l'échantillonnage afin de discerner plus facilement ce qui se produit dans l'espace dual. La résolution de l'échantillonnage dans l'espace euclidien de la cellule est de 4×4 pour l'affichage dans ce même espace et de 32×32 pour son affichage dans l'espace dual. Dans l'affichage dans l'espace dual, certaines zones sont plus sombres (sur-échantillonnées) et d'autres plus claires (sous-échantillonnées). De plus, même lorsque l'on augmente la résolution à outrance, on ne parvient pas à échantillonner suffisamment l'espace des droites pour lesquelles θ est proche de $\pm\pi/4$ et u proche de $\pm\frac{1}{\sqrt{2}}$. La seconde stratégie sert à palier à ce problème.

3.1.5.2 Espace dual

Cette section illustre un phénomène visible dans l'espace dual des lignes de vue. Ce dernier est utilisé par Orti et Puech [76] afin de calculer le facteur de forme 2D entre deux objets.

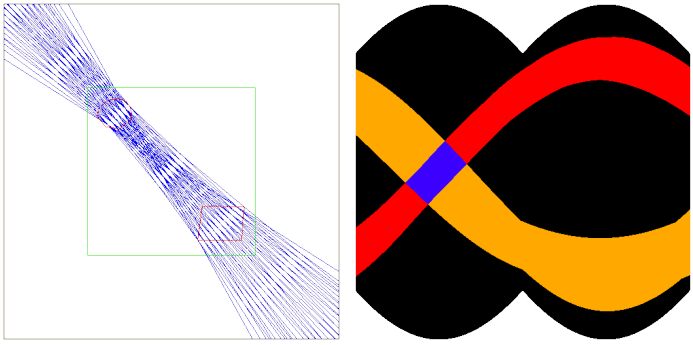
Toutes les lignes de vue qu'intercepte un bloqueur sont représentées dans l'espace dual par deux courbes (cf. figure 3.17). Celles-ci délimitent une bande couvrant l'ensemble des lignes intersectant ce bloqueur. Ici, le bloqueur est approximativement un disque, et c'est pourquoi les deux courbes ont une allure similaire l'une par rapport à

l'autre. Il est bon de remarquer que lorsque l'on parle de plusieurs bloqueurs, l'aire où s'intersectent les bandes (en bleu dans l'espace dual) représente les lignes intersectant à la fois les deux bloqueurs (cf. figure 3.18).



(a) Espace euclidien (b) Espace dual

Figure 3.17 – Les lignes interceptées par un bloqueur forment l'aire rouge dans l'espace dual.



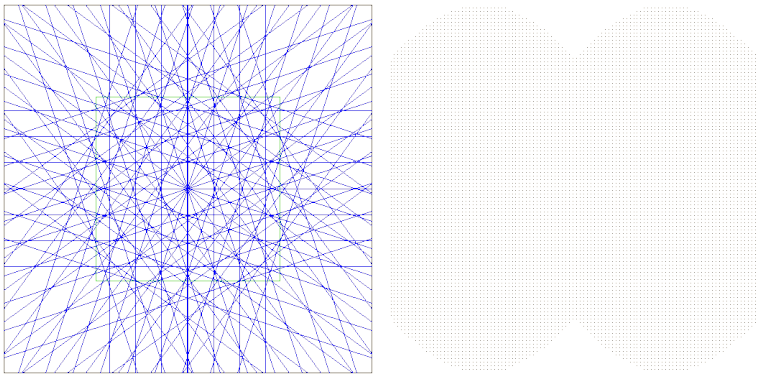
(a) Espace euclidien (b) Espace dual

Figure 3.18 – Les lignes interceptées par deux bloqueurs.

3.1.5.3 Stratégie duale

Le principe de cette stratégie réside dans le fait que tous les échantillons sont générés uniformément dans l'espace dual et sont ensuite ramenés dans l'espace euclidien 2D pour tester si la droite associée intersecte la cellule. Puis, afin de déterminer si c'est une droite bloquée ou non, on la teste finalement avec la géométrie. La figure 3.19 illustre

cette seconde stratégie. Ainsi, l'échantillonnage de l'espace des droites est plus équilibré que le précédent et permet de "converger" plus vite vers une meilleure approximation de la visibilité avec une quantité moindre d'échantillons.



(a) Espace euclidien (b) Espace dual

Figure 3.19 – Un échantillonnage régulier effectué directement dans l'espace dual, et les lignes correspondantes dans l'espace euclidien.

3.1.6 Discussion et mise en application

Dans cette section, nous discutons plusieurs pistes à explorer pouvant permettre l'extension et la mise en application des vecteurs d'occultation.

Un algorithme trop conservateur ? La voxélisation conservatrice de certaines scènes, telles qu'une forêt, résulterait en une structure quasi vide, voire entièrement vide. Il est toutefois possible de préserver une visibilité conservatrice, par projection, depuis la cellule de vue, de chaque feuille sur des plans de reprojection successifs (à la manière de Durand et al. [23]). Ceci permettrait un marquage effectif des vecteurs d'occultation.

Intégration de l'échantillonnage dans le *framework* de calcul de PVS L'intérêt d'utiliser l'échantillonnage dans un système tel que les vecteurs d'occultation est que cela permet, dès lors que la géométrie d'une tuile est trop petite et qu'elle ne parvient pas à occuper un *gridel*, d'avoir une bonne estimation de la "valeur" d'occultation d'un

ensemble de géométries. Ainsi, pour que le système de propagation de la visibilité fonctionne en temps réel il est nécessaire de combiner ces deux méthodes de manière à ce que les tests de visibilité par échantillonnage soient réalisés à l'échelle du *gridel*. Si l'on estime que l'ensemble de toutes les petites géométries d'un *gridel*, voire des *gridels* adjacents, bloque suffisamment de lignes de vue passant par celui-ci, alors on peut le substituer par un *gridel* occultant pour un calcul de visibilité par fusion de bloqueurs à plus grande échelle. Nous pensons donc que réaliser la fusion entre ces deux systèmes pourrait allier la robustesse d'une propagation conservatrice et alléger la tâche d'un calcul d'occultation local initialement très coûteux. Cette intégration est surtout valide lors d'une forte densité de petits bloqueurs.

Autres pavages Il est assez intuitif de remarquer qu'une tuile en forme de losange possède les mêmes propriétés de visibilité qu'un carré, à condition que la grille de voxélisation respecte le cisaillement du losange. Cette propriété est valable pour toute forme issue d'une quelconque similitude – une ligne reste une ligne après transformation affine. Pour cette raison, il est envisageable de faire évoluer les propriétés de relation de visibilité entre les bordures des tuiles de pavages rhombiques de Penrose (losanges), voire d'autres formes de tuiles. Le processus décrit à la figure 3.20 illustre le cas du losange dont la voxélisation conservatrice et les vecteurs d'occultation peuvent être calculés après transposition de la géométrie dans une grille régulière carrée. Puis ces dernières sont ramenées dans l'espace du losange sans pour autant altérer l'occultation générée par la tuile.

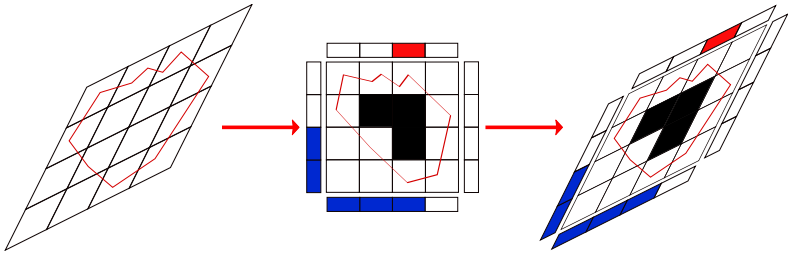


Figure 3.20 – Extension des pavages d'occultation aux pavages de Penrose.

3.1.7 Conclusions

Il est important que, lors de la création d'un monde 3D, un infographiste puisse bénéficier d'une liberté d'action sur le contenu d'une scène. C'est pourquoi nous soulignons l'importance des outils simples et efficaces dans le pipeline de création de contenu de scènes 3D.

Dans la première partie de ce chapitre, nous avons établi une méthodologie innovante alliant calcul de visibilité et modélisation procédurale en 2D appliqués à la génération rapide de scènes potentiellement infinies et riches en géométrie. Cette méthodologie donne beaucoup plus de flexibilité lors du design que le schéma d'occultation *ū-shape* du chapitre 2.

D'autre part, nous possédons un outil permettant d'estimer le taux d'occultation moyen d'une tuile, en fonction de la fréquence de la géométrie par unité d'espace, permettant d'estimer le temps de calcul et les ressources nécessaires. C'est pourquoi nous mettons l'accent sur l'importance du calcul de visibilité pour l'optimisation du rendu en temps réel de telles scènes.

3.2 Extension à la 3D

Au chapitre 2 et dans les sections précédentes, la visibilité est calculée en 2D et ne permet donc pas une liberté de design totale dans la troisième dimension. Pour cela notre thèse veut s'attaquer maintenant à l'extension complète à la 3D du calcul de visibilité. Afin d'intéger la visibilité dans la construction de mondes 3D, notre contribution est un processus de génération par pavage qui propage l'information de visibilité 3D au travers des tuiles.

Dans cette section et la suivante, nous présentons deux méthodes de propagation de la visibilité en 3D. Ces deux méthodes utilisent pour unité de pavage le cube, i.e., l'extension des carrés 2D utilisés précédemment à la 3D (voir la figure 3.21). Ces tuiles peuvent être arrangées de manière déterministe ou aléatoire pour donner une impression d'infinité.

Les sections suivantes sont structurées comme suit. En section 3.2.1, nous expliquons

comment procéder à un calcul, par échantillonnage et individuel pour chaque tuile, de ce que nous appelons les matrices de visibilité. Elles sont l'extension des vecteurs d'occultation à la 3D. Nous montrons aussi comment procéder au calcul rapide du PVS pour chaque cellule de vue. Nous mettons ensuite en évidence les performances d'affichage possibles selon la taille du PVS en utilisant un moteur de rendu standard. Nous montrons en section 3.2.2 des exemples de paysages de villes et de forêts générés par notre algorithme de propagation de visibilité. En section 3.2.3, nous présentons certaines limitations de notre méthode et montrons comment les réduire. Enfin, nous donnons une brève conclusion et discutons des possibilités d'extension de cette méthode en section 3.2.4.

3.2.1 Construction rapide de PVS par propagation de visibilité

La méthode présentée dans cette section utilise un précalcul local de visibilité par échantillonnage sur chaque tuile. Elle utilise ces informations pour propager la visibilité de tuile en tuile. Notre algorithme de propagation s'inspire des reprojections de Durand et al. [22], afin de calculer le PVS de la scène depuis la cellule de vue courante. La méthode proposée ici permet une très grande liberté dans la modélisation des tuiles et donne un *feedback* sur les performances minimales nécessaires à l'affichage du monde généré. Ce concept de modélisation par visibilité peut être utile dans la plupart des processus de production de jeux vidéo. Du fait que les performances d'affichage peuvent être estimées, les éditeurs de niveau d'un jeu vidéo nécessitent moins de retour d'expérience de la part des testeurs de niveau.

Le rendu de mondes virtuels de grande taille nécessite de limiter le plus possible le nombre de polygones envoyés au GPU. Dans l'idéal, il faudrait limiter ces derniers seulement à l'ensemble des polygones visibles depuis le point de vue. Au lieu d'utiliser un algorithme dynamique d'*occlusion culling* pour chaque image rendue, notre méthode est basée sur un précalcul d'occultation pour chaque tuile 3D. Ensuite, afin de limiter l'utilisation mémoire d'un PVS précalculé, le calcul dynamique du PVS repose sur l'instanciation de tuile et le stockage compact des matrices de visibilité pour chaque tuile. Enfin, nous souhaitons limiter les artéfacts de *popping* lors du rendu. Notre algorithme prend en entrée un ensemble de pavage quelconque. Les tuiles qui le composent peuvent

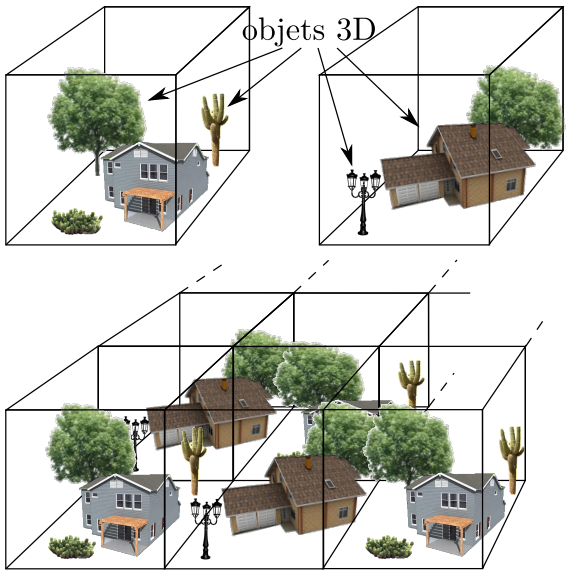


Figure 3.21 – Haut : Deux tuiles prédéfinies utilisées dans notre processus de pavage 3D. Les objets sont placés dans ces tuiles par un infographiste ou par une méthode procédurale de placement. Bas : Les tuiles de l'ensemble de pavage sont ensuite distribuées de manière déterministe ou aléatoire dans l'espace 3D afin de former la géométrie finale du monde.

être modélisées par un infographiste ou générées de manière procédurale. Afin de générer le monde 3D, ces tuiles sont distribuées dans l'espace. Nous considérons chacune de ces tuiles comme une cellule de vue à part entière, comme dans le chapitre précédent. Ensuite, depuis la cellule où se situe l'observateur, le PVS est calculé par propagation de visibilité en se basant sur les informations locales de visibilité précalculées pour chaque tuile. Cela permet la construction dynamique de l'ensemble minimal de tuiles du pavage à afficher nécessaire pour occulter le champ de vue, donnant un *feedback* lors de l'évaluation des performances d'affichage.

A la section suivante, nous présentons le calcul des matrices de visibilité, représentant les portions de "chemins de visibilité" qui traversent une tuile. Le PVS est construit par propagation de la visibilité, en considérant la première tuile visible comme cellule de vue, jusqu'à ce que chaque "chemin de visibilité" soit bloqué. Dans les deux sections qui suivent, nous expliquons la mise à jour des matrices de visibilité temporaires durant la construction du pavage et le processus principal de propagation de la visibilité.

3.2.1.1 Précalcul des matrices de visibilité par échantillonnage

Chaque face de la tuile est subdivisée en une grille de résolution $n \times n$. Nous nommons *gridels* [98] les élements de cette grille. Pour chaque *gridel* de chaque face de la tuile, sont calculées cinq matrices $n \times n$, représentant les *gridels* visibles de chacune de cinq autres faces. Une matrice de visibilité encode la relation de visibilité entre un *gridel* entrant g_{in} – appartenant à une face de la tuile – et tous les *gridels* sortants g_{out} sur une des autres faces de la tuile. Chaque *gridel* non occulté d'une matrice de visibilité représente en fait un portail au travers de la face correspondante. La résolution des *gridels* d'une face et leurs matrices de visibilité doivent être les mêmes afin que l'algorithme de propagation puisse fonctionner efficacement. De plus, chaque matrice de visibilité peut être stockée avec un seul bit par *gridel* (visible ou occulté).

Afin de calculer la matrice de visibilité entre un *gridel* entrant et une face de la tuile, nous effectuons le rendu dans un tampon de profondeur depuis plusieurs points de vue appartenant au *gridel*. La face pour laquelle on calcule la matrice de visibilité est utilisée comme plan distant (*far plane*) du volume de projection (voir figures 3.22(gauche)

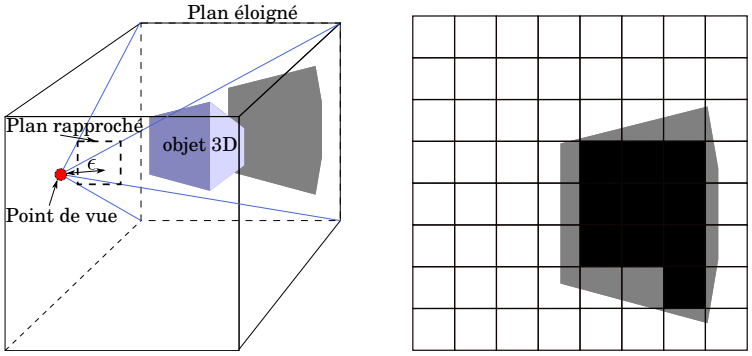


Figure 3.22 – Gauche : la projection perspective oblique standard. Droite : une matrice de visibilité depuis un point de vue unique. Un *gridel* blanc vaut 1, un *gridel* noir vaut 0, le gris est un facteur intermédiaire d'occultation.

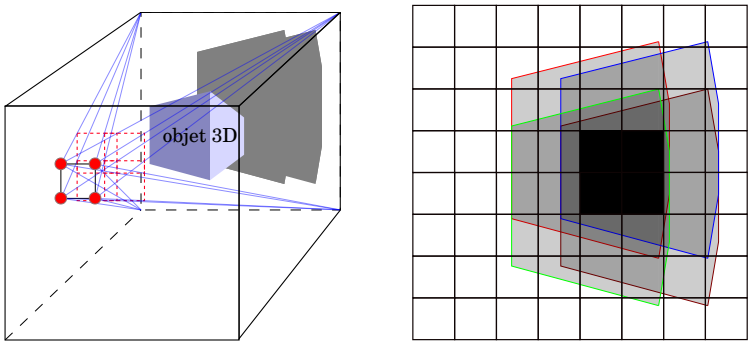


Figure 3.23 – Gauche : l'échantillonnage depuis plusieurs points de vue appartenant au même *gridel*. Droite : le résultat des projections depuis ces mêmes points de vue réunies en une seule matrice de visibilité en utilisant l'opérateur logique OU. Les *gridels* de couleur grise représentent l'information perdue par cette simplification conservatrice de l'information de visibilité.

et 3.23(gauche)). Le plan rapproché (*near plane*) de ce volume de vue est situé à une distance ε du point de vue, en direction du plan distant, et est parallèle à ce dernier. Cette configuration du volume de vue définit une projection perspective oblique standard. La projection des objets 3D contenus dans la tuile produit un tampon de profondeur, considéré comme une matrice de visibilité depuis un point où un *gridel* non occulté produit une valeur de 1. La figure 3.22 montre ce processus de projection pour un cas simple, ainsi que la matrice de visibilité résultante dans laquelle un *gridel* blanc vaut 1, un *gridel* noir vaut 0, le gris est un facteur intermédiaire d'occultation. En projection *z-buffer* d'un point unique, les centres de pixels en gris sont alors considérés comme occultés, et les autres comme visibles. Bien que les artéfacts de *popping* soient principalement causés par l'échantillonnage des points de vue, cette imprécision de rastérisation d'un pixel dans le *z-buffer* peut ajouter des artéfacts de *popping* supplémentaires. Ensuite, les matrices de visibilité qui ont été calculées pour l'ensemble des différents points de vue (appartenant à un même *gridel*) sont regroupées selon de la résolution choisie en utilisant l'opérateur logique OU entre les différentes matrices (factorisation des points de vue). La figure 3.23 montre le processus de projection pour plusieurs points et la matrice de visibilité finale résultante après factorisation.

En terme d'utilisation mémoire, une tuile de pavage possède six faces, chacune contenant $n \times n$ *gridels* entrants. Chaque *gridel* entrant possède cinq matrices de visibilité, chacune de résolution $n \times n$ correspondant aux cinq directions sortantes possibles (gauche, droite, haut, bas et en face). L'espace mémoire utilisé pour stocker les matrices de visibilité pour une tuile est donc de $6(n \times n) \times 5(n \times n) = 30n^4$ bits, où n est la résolution de grille.

3.2.1.2 Propagation de la visibilité

La visibilité est propagée depuis la cellule de vue par voisinage vers les tuiles adjacentes. Dès lors qu'une matrice de visibilité détermine que l'on peut voir au travers d'une tuile, la tuile voisine dans la direction de propagation est alors instanciée. Cette dernière est choisie dans l'ensemble des tuiles prédéfinies (ensemble de pavage), et est ensuite ajoutée au PVS. Une vue d'ensemble du processus de propagation est donnée à

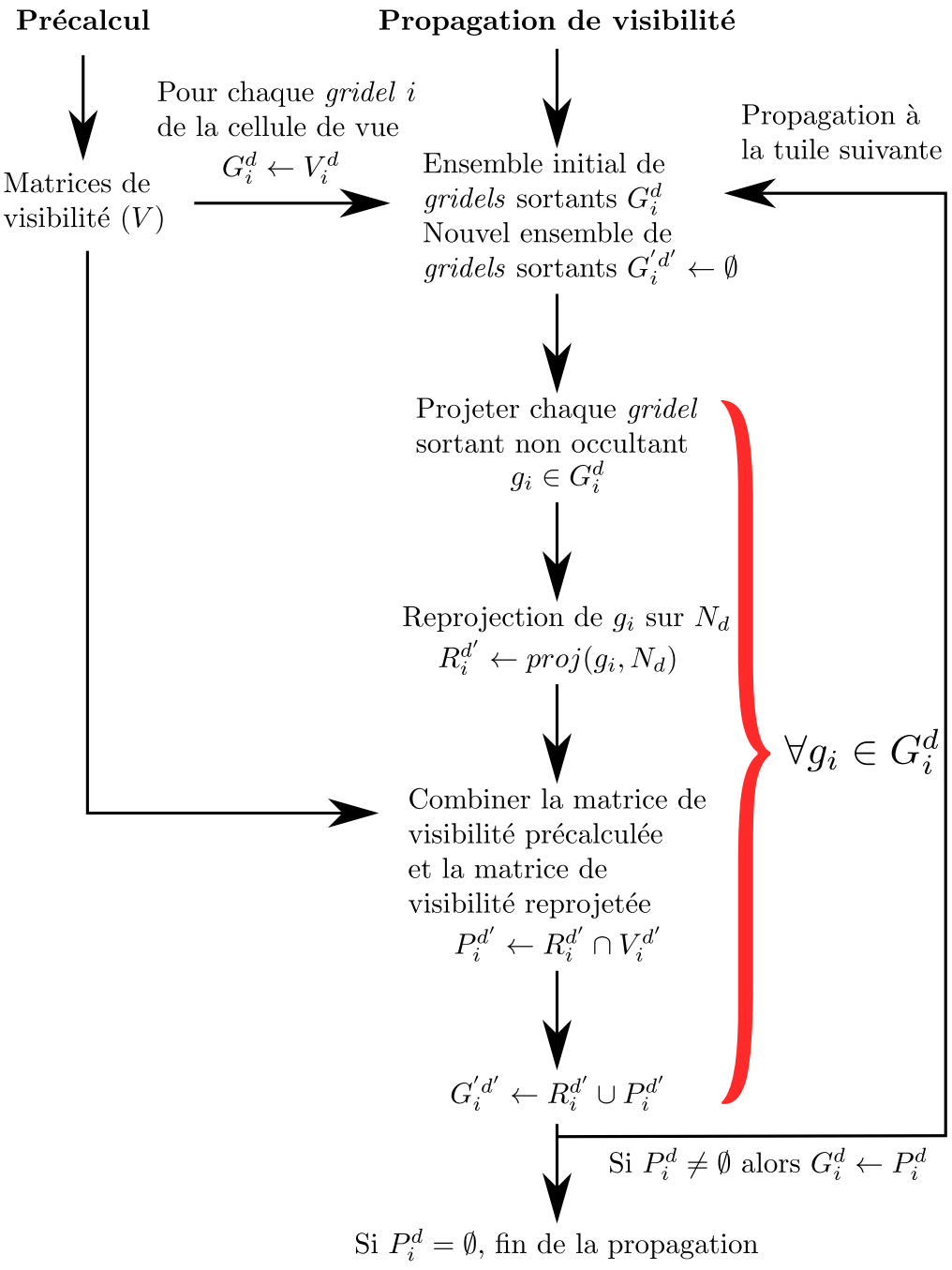


Figure 3.24 – Une vue d'ensemble du processus de propagation utilisant les reprojections.

la figure 3.24.

Pour chaque *gridel* sortant i de la cellule de vue, une matrice de visibilité globale G_i^d est créée et mise à jour par propagation sur les tuiles voisines pour chaque direction $d \in Directions = \{\text{gauche,droite,face,haut,bas}\}$. La propagation s'arrête dès lors que la matrice est nulle. G_i^d est tour à tour initialisée avec chacune des matrices de visibilité précalculées V_i^d de chacun des *gridels* sortant de la cellule de vue – correspondant au *gridel* i dans la direction de propagation d. G_i^d est ensuite mise à jour à chaque frontière de tuile où la visibilité est propagée.

Pour mettre à jour la matrice de visibilité globale, chaque *gridel* sortant non occulté $g_i \in G_i^d$ est projeté vers la tuile suivante en utilisant quatre matrices de projection perspective oblique. Ces dernières sont créées depuis chacun des quatre coins du *gridel* de départ – celui de la cellule de vue – et en utilisant la face sortante de la tuile suivante N_d. Ces projections utilisent les faces sortantes de la tuile suivante comme plans distants (*far planes*) et résultent en des matrices de visibilité reprojetées $R_i^{d'}$, où $d' \in Directions - \{ opposé(d)\}$. Comme précédemment, un *gridel* non occulté d'une matrice de visibilité reprojetée produit une valeur de 1.

Un *gridel* sortant non occulté g_i sur la tuile courante correspond à un *gridel* entrant g_i' sur la tuile suivante N_d. La matrice de visibilité précalculée $V_i^{d'}$ de g_i' est ensuite combinée à la matrice de visibilité reprojetée $R_i^{d'}$ en utilisant l'opérateur logique ET (intersection des *gridels* non occultés, union des *gridels* occultés). Le résultat de cette opération matricielle est la matrice partielle de propagation $P_i^{d'}$.

Ensuite, la nouvelle matrice de visibilité globale $G_i'^{d'}$ est calculée en réalisant l'opération logique OU entre chacune des matrices partielles de propagation $P_i^{d'}$ calculées pour chacun de *gridels* non occultés $g_i \in G_i^d$. Enfin, ce processus est réitéré jusqu'à ce que G_i^d n'ait plus aucun *gridel* sortant non occulté.

La figure 3.25(haut) montre la reprojection d'un *gridel* sortant (en orange) appartenant à la matrice de visibilité de la face opposée du *gridel* entrant (en vert foncé). Les *gridels* projetés résultants sont montrés à la figure 3.25(bas).

Cette approche est similaire à celle de Durand et al. [23], qui utilise des plans de reprojection successifs pour calculer un PVS. Le grand avantage de notre approche est

que le coût de construction du PVS ne dépend pas du nombre de polygones dans la scène, mais seulement de la résolution de grille choisie pour les tuiles.

3.2.1.3 Construction de PVS par propagation de visibilité

Dans cette méthode, la construction du PVS nécessite l'utilisation des matrices de visibilité et de reprojection : les matrices de visibilité permettent d'aller chercher directement les *gridels* non occultés d'une tuile. Les matrices de reprojection permettent de trouver les *gridels* potentiellement visibles au travers de l'un de ces *gridels* non occultés. L'algorithme de propagation de la visibilité utilise des projections et des opérations binaires simples entre ces matrices et se révèle suffisamment efficace pour une estimation du PVS induit. Par la même occasion, nous pouvons également estimer sa taille et en déduire la qualité de l'occultation procurée par l'ensemble de pavage.

Toutefois, pour un *walkthrough* dans un environnement 3D décrit par seulement une couche de tuiles, nous proposons de simplifier le problème en limitant le nombre de directions de propagation possibles à trois : face, gauche, droite. Nous pouvons également borner la hauteur de l'observateur. En effet, certains calculs s'avèrent inutiles dans la détermination du PVS si, par exemple, nous considérons que l'observateur reste situé entre deux hauteurs fixes dans la tuile, i.e., une tranche horizontale de la cellule de vue – déterminée par quelques rangées adjacentes horizontales de *gridels*. Si l'observateur se situe dans n'importe quel endroit d'une tranche unique, par exemple, celle de couleur bleue dans la figure 3.26, seul un sous-ensemble des *gridels* de la cellule de vue a besoin d'être utilisé dans le calcul du PVS. Ce sous-ensemble est en bleu et en vert clair tandis que ceux qui ne sont pas pris en compte pour le calcul du PVS sont en rouge. Les niveaux minimum et maximum des *gridels* nécessaires au calcul du PVS peuvent être calculés grâce aux équations suivantes :

$$\begin{aligned} L_{min} &= \lfloor (L_{obs}/2) \rfloor \\ L_{max} &= \min\left(L_{obs} + \lceil (resZ_{tile} - 1 - L_{obs})/2 \rceil, resZ_{tile}\right) \end{aligned}$$

où L_{obs} est la tranche dans laquelle se situe l'observateur, et $resZ_{tile}$ est la résolution en

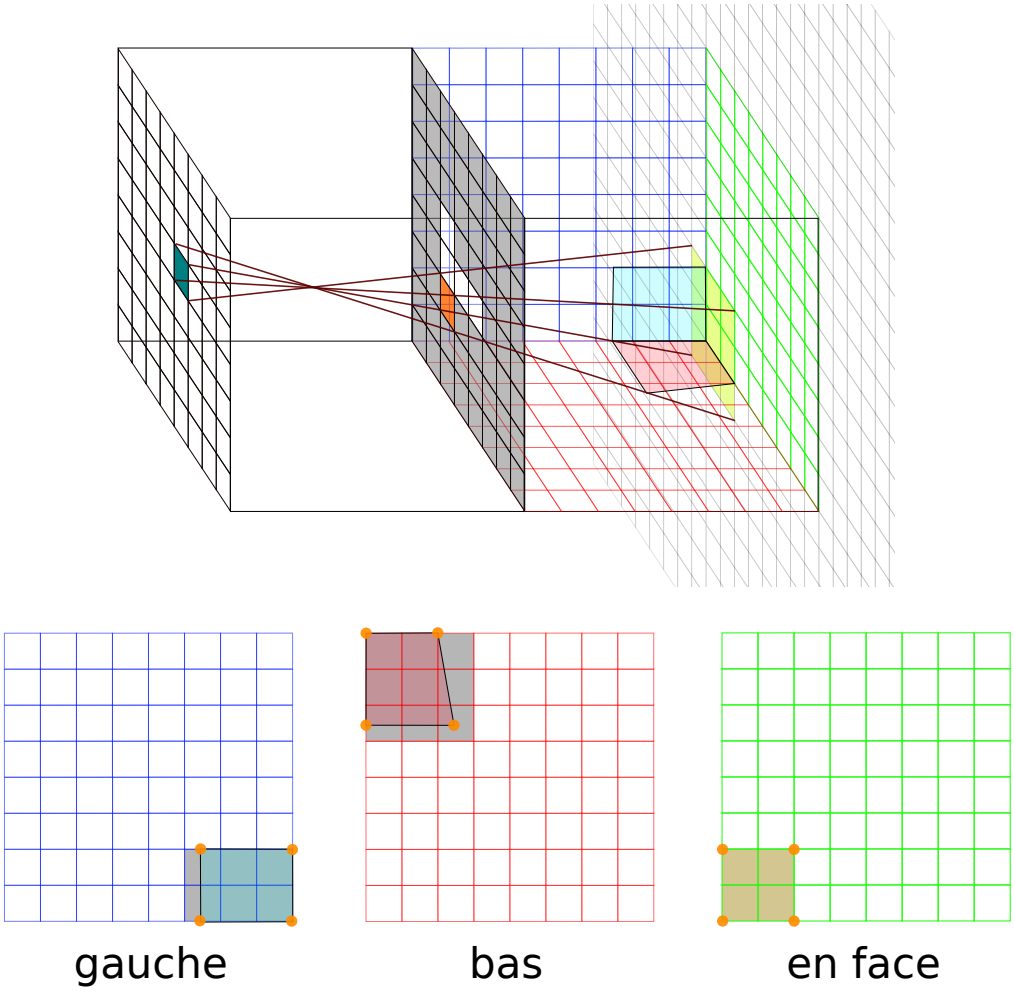


Figure 3.25 – Haut : La projection d'un *gridel* sortant (en orange) appartenant à la matrice de visibilité de la face opposée du *gridel* entrant (en vert foncé). Bas : Les matrices reprojetées résultantes $R_i^{d'}$ pour trois directions sortantes.

hauteur de la grille de la tuile.

Une autre simplification peut être ajoutée : nous pouvons considérer que les *gridels* situés plus haut (respectivement, plus bas) que la tranche ne propagent jamais la visibilité plus bas (respectivement, plus haut) que leur propre niveau. Les *gridels* potentiellement visibles dans la première tuile adjacente depuis la cellule de vue peuvent être trouvés plus rapidement grâce aux restrictions suivantes :

$$L_{prop+1} \in \begin{cases}) - L_{obs} + 2L_{prop} - 2, resZ_{tile}(& (L_{obs} < L_{prop}) \\ (0, -L_{obs} + 2L_{prop} - 2(& (L_{obs} > L_{prop}) \\ (0, resZ_{tile}(& (L_{obs} = L_{prop}) \end{cases} \tag{3.1}$$

où $L_{prop} \in (L_{min}\ , L_{max})$ est la tranche du *gridel* de la cellule de vue et où L_{prop+1} est une tranche valide pour les *gridels* sortants de la tuile adjacente.

3.2.2 Implémentation et résultats

Nous avons testé notre algorithme sur différents ensembles de tuiles composés de maisons et d'arbres, comme illustré à la figure 3.27. Pour l'expérimentation, nous avons considéré uniquement une tranche de *gridels* sortants depuis la cellule de vue. Chaque tuile s'est vue attribuer plusieurs objets 3D pour un total compris entre 16K et 2.5M polygones par tuile. Les calculs ont été effectués sur un CPU Intel® Core™ i7-930 (QuadCore) 2.8 GHz, 16 Go de mémoire vive et une carte graphique GeForce GTX 480/PCIe/SSE2. Les temps de précalcul des matrices de visibilité locales sont donnés au tableau 3.3. On peut voir que le temps de précalcul des matrices de visibilité est directement fonction du nombre de polygones traités et de la résolution d'échantillonnage des tuiles. Nous avons décomposé l'algorithme de propagation en quatre phases principales : sélection des *gridels*, projection des *gridels*, opérations basiques sur les images (lecture du *z-buffer* et allocation mémoire CPU, etc.) – et les opérations binaires entre matrices de visibilité et matrices reprojetées. Le tableau 3.4 fait état des temps de calcul de l'algorithme de propagation en terme d'écart type. Nous pouvons voir que pour de faibles résolutions, les temps de construction des PVS sont interactifs et que le nombre

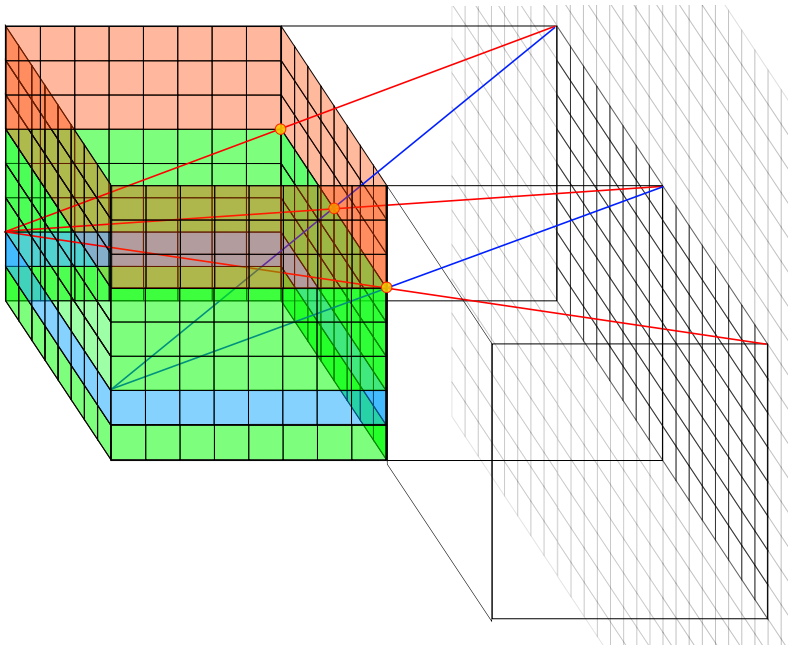
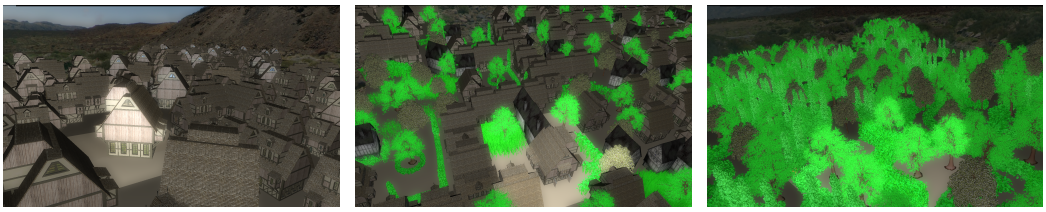


Figure 3.26 – Certains calculs peuvent être évités durant le calcul du PVS : si l'observateur se situe dans la tranche en bleu, la propagation depuis les *gridels* en rouge peut être évitée, car inutile.



(a) Ville (b) Scène hybride (c) Forêt

Figure 3.27 – Les PVS de trois types de scènes. Les objets de la cellule de vue sont plus clairs que ceux du reste du PVS.

de polygones à afficher peut être rapidement estimé pour une cellule de vue. Pour des résolutions plus grandes les temps de calcul deviennent plus prohibitifs tout dépendant de la configuration de la scène.

L'implémentation GPU de la projection des *gridels* sortants permet des calculs rapides. Lors de nos expérimentations, seuls quatre échantillons par *gridel* entrant ont été utilisés pour le précalcul des matrices de visibilité, ce qui implique un échantillonnage très grossier de la visibilité pour des résolutions basses et donc un taux d'erreur plus élevé. Toutefois, il est parfaitement possible d'augmenter cette densité d'échantillonnage puis de sous-échantillonner le résultat afin d'obtenir des matrices plus exactes sans pour autant affecter la validité de l'algorithme de propagation subséquent. Nous avons également observé que du fait de cet échantillonnage, ainsi que de l'utilisation d'une seule tranche de *gridels* de la cellule de vue pour initialiser la propagation, le calcul ne fournit qu'une approximation du nombre minimum de tuiles à instancier nécessaires et donc du nombre de polygones contenus dans le PVS. Cet outil permet, dans son utilisation interactive, d'estimer rapidement les performances de rendu nécessaires à l'affichage du pavage généré. La position des objets sur les tuiles peut également être modifiée de manière interactive afin de tester les nouvelles performances de rendu. Ainsi, dès lors que l'occultation induite par le placement des objets est suffisante et assure de bonnes performances d'affichage, il est possible de réajuster leur position, ajouter de plus petits objets sur les tuiles, et ensuite augmenter la densité d'échantillonnage et/ou la résolution des tuiles.

Le processus de reprojections et de comparaisons binaires (accolade rouge sur la figure 3.24) pourrait également être parallélisé dans les prochaines générations de GPU si l'on considère que chaque portion de "chemin de visibilité" est indépendante des autres.

3.2.3 Discussion

3.2.3.1 Réduire les artéfacts liés au *popping*

Durant le précalcul des matrices de visibilité, la discontinuité entre deux points de vue échantillonnés n'assure pas qu'un objet situé plus loin que la dernière tuile instan-

Résolution	8×8	16×16	32×32	8×8	16×16	32×32	8×8	16×16	32×32	8×8	16×16	32×32	8×8	16×16	32×32
ID-#polys	tuile 0 - 117.00K			tuile 1 - 16.71K			tuile 2 - 66.86K			tuile 3 - 50.14K			tuile 4 - 50.14K		
Temps (s)	0.87	3.42	13.03	0.38	1.41	5.95	0.71	2.68	10.19	0.60	2.24	9.60	0.61	2.42	9.65
ID-#polys	tuile 0 - 1.22M			tuile 1 - 1.08M			tuile 2 - 2.58M			tuile 3 - 0.99M			tuile 4 - 0.98M		
Temps (s)	1.82	6.87	26.73	1.60	6.35	24.25	2.81	11.08	44.40	1.77	6.72	26.97	1.94	7.00	27.35
ID-#polys	tuile 0 - 2.41M			tuile 1 - 2.31M			tuile 2 - 2.48M			tuile 3 - 1.98M					
Temps (s)	2.72	9.64	39.67	2.44	9.31	36.46	2.73	10.38	40.87	2.37	8.91	35.19			

Tableau 3.3 – Temps de précalcul des matrices de visibilité des tuiles pour trois types de scènes : (haut) ville, (milieu) scène hybride, (bas) forêt.

ciée est occulté complètement par les objets instanciés. Par conséquent, pour des scènes complexes qui pourraient mettre en exergue ce défaut en faisant apparaître un objet subitement, il est nécessaire d'augmenter le nombre d'échantillons par *gridel* afin de réduire l'erreur liée à cet échantillonnage, et donc la probabilité que ce type d'artéfact se produise.

3.2.3.2 Hautes performances de rendu

Parfois, l'occultation générée par les tuiles de pavage n'est pas suffisante pour permettre à l'algorithme de propagation de se terminer assez rapidement. Cela est dû au fait que la géométrie des objets ne couvre pas suffisamment les *gridels* sortants. Pour cette raison, il est nécessaire soit de diminuer la résolution des grilles des tuiles – sans pour autant diminuer le nombre d'échantillons pour ne pas perdre en précision –, soit d'agrandir la taille de la tuile afin que plus d'objets soient englobés par cette dernière en précalculant l'union de plusieurs tuiles, utile lorsque les objets sont de petite taille.

Pour réduire les temps de construction des PVS, il faut réduire la résolution des matrices de visibilité. Ces dernières sont alors sous-échantillonnées. Ainsi, il faut calculer, pour chaque tuile, pour chaque *gridel* et pour chaque direction, sa matrice à basse résolution. Pour cela, il faut réaliser la combinaison des matrices de chaque *gridel* de haute résolution en une seule matrice correspondant au *gridel* de basse résolution (ce *gridel* regroupe chaque *gridel* de haute résolution). Réduire la résolution implique que la visibilité sera plus conservatrice lors de la construction du PVS, mais permet d'accélérer son calcul. Cette approche est plus efficace lorsque les bloqueurs sont assez grands pour

Résolution 8×8. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.002	0.01	0.06	0.42	0.01	0.08
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.00007	0.0005	**0.07**	**0.52**	564 K	33
Résolution 16×16. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.008	0.06	0.29	2.06	0.08	0.59
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.0007	0.005	**0.38**	**2.72**	480 K	25
Résolution 32×32. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.96	2.29	17.14	11.75	5.71	0.48
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.10	0.21	**23.99**	**49.59**	1.43 M	362

(a) Ville

Résolution 8×8. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.0007	0.005	0.02	0.16	0.006	0.04
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.00002	0.0001	**0.03**	**0.21**	12.67 M	658
Résolution 16×16. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.001	0.008	0.04	0.30	0.009	0.07
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.00004	0.0003	**0.05**	**0.38**	9.35 M	325.562
Résolution 32×32. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.06	0.12	2.31	4.94	0.71	1.56
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.005	0.01	**3.09**	**6.66**	18.66 M	0.000002

(b) Scène hybride

Résolution 8×8. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
0.09	0.03	1.33	0.30	0.36	0.10
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.0008	0.0004	**1.79**	**0.41**	469.74 M	898
Résolution 16×16. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
1.72	2.39	4.39	0.96	1.23	0.28
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.005	0.001	**7.37**	**2.98**	219.55 M	1.78 M
Résolution 32×32. Temps moyen pour 100 cellules de vue.					
Sélection des *gridels*		Projection des *gridels*		Opération sur les images	
Temps (s)	σ	Temps (s)	σ	Temps (s)	σ
1.31	0.80	19.16	3.37	6.28	1.14
Opérations binaires		**Temps total**		Polygones par PVS	
Temps (s)	σ	**Temps (s)**	σ	Nombre	σ
0.04	0.010	**26.90**	**5.08**	198.21 M	5.07 M

(c) Forêt

Tableau 3.4 – Temps d'exécution de l'algorithme de propagation de la visibilité en terme d'écart type pour (a) la ville, (b) la scène hybride et (c) la forêt. Les tuiles et leur matrices de visibilité précalculées sont celles calculées au tableau 3.3. Aucun sous-échantillonnage n'a été réalisé pour générer ces résultats.

couvrir les *gridels* de la nouvelle résolution choisie.

Faire l'union de tuiles peut être une solution plus pratique pour accélérer la construction du PVS. En 3D, huit tuiles (quatre dans notre cas applicatif) peuvent être réunies pour créer une tuile plus grande. Lorsque le nombre d'objets dans une tuile augmente, celle-ci devient statistiquement plus occultante. De plus, l'algorithme de propagation parcourt plus rapidement de plus grandes distances et peut donc se terminer plus rapidement. Cette solution est une alternative satisfaisante lorsque les tuiles contiennent de nombreux polygones de petite taille.

3.2.4 Conclusions

Dans cette section, nous avons fourni un outil qui permet aux créateurs de niveaux de jeux vidéo de réaliser à faible coût et de manière automatique des mondes basés sur du pavage. Cet outil permet d'estimer le coût d'affichage d'un tel monde par une détermination rapide du PVS. De plus, l'utilisation de cette méthode permet d'alléger le coût de construction total du PVS en répartissant ce dernier entre le précalcul des matrices de visibilité d'une part et l'algorithme de propagation d'autre part. Dans notre prototype de démonstration, nous avons choisi d'utiliser une méthode d'échantillonnage pour le précalcul des matrices de visibilité. Mais il est tout à fait possible de la substituer par n'importe quel autre type de méthode, conservatrice ou exacte. Toutefois, même si la liberté de création est étendue à la 3D, les objets qui dépassent des bordures d'une tuile ne sont pas pris en compte et risquent de faire apparaître de nouveaux artéfacts de *popping*.

Notre méthode de base crée une instance entière de la tuile dès lors qu'elle est considérée comme visible depuis la cellule de vue. Cela implique que tous les objets de la tuile sont ajoutés au PVS quand celle-ci est parcourue par l'algorithme de propagation de la visibilité. Il est également possible d'éviter ce surcoût en stockant les références vers les objets intersectés par chaque volume *gridel* entrant / *gridel* sortant. Lors de l'ajout d'une tuile au PVS il suffit de consulter la liste des objets intersectés par le "chemin de visibilité" courant. Ainsi, les objets non visibles de chaque tuile ne sont pas inclus dans le PVS.

Il serait également intéressant de développer une version hiérarchique de cette méthode : la visibilité serait propagée très rapidement et grossièrement sur de longues distances au plus haut niveau de la hiérarchie tandis que les niveaux plus bas seraient traités en parallèle par propagation locale de manière plus précise.

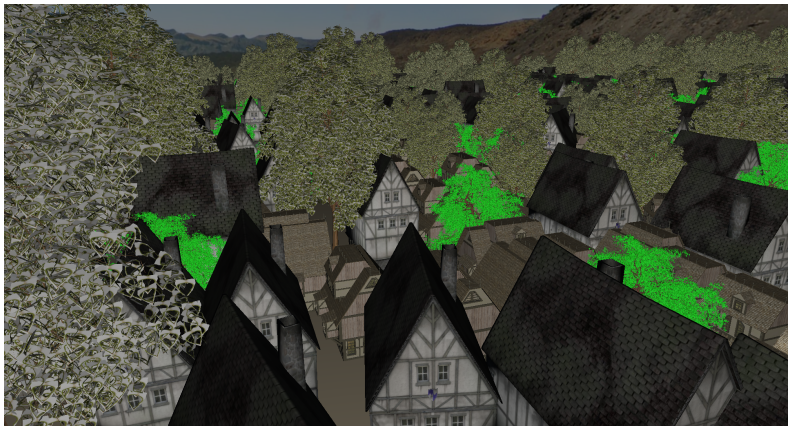
3.3 Technique 3D temps réel

Les contributions exposées dans cette section ont donné lieu à une publication [37], présentée à la conférence *Computer-Aided Design and Computer Graphics* en novembre 2013.

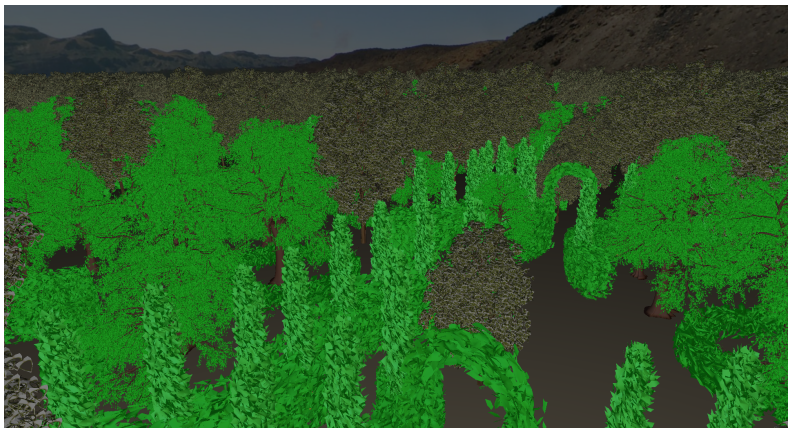
Dans la section précédente, le calcul de visibilité s'effectue par échantillonnage en construisant des PVS par cellule de vue. De plus, ce calcul monopolise la carte graphique durant tout le calcul du PVS, provoquant des temps de latence car le moteur de rendu n'utilise pas la carte pour afficher le monde pendant ce temps-là. Une solution performante d'implémentation sur un seul GPU est de créer deux contextes concurrents. Ainsi le premier contexte précalcule le PVS des cellules de vue voisines tandis que le second affiche le PVS de la cellule courante. Le temps de calcul réparti entre les deux contextes permet d'allouer au premier contexte un temps qui dépend de la fréquence d'affichage du PVS courant par le second contexte. Cela permet de lisser les temps de latence et de les rendre imperceptibles si l'observateur ne se déplace pas trop rapidement d'une cellule de vue à une autre. Toutefois les performances d'affichage sont réduites du fait du partage du temps de calcul entre les deux contextes.

Dans cette section, nous utilisons un algorithme d'*occlusion culling* en 3D par point de vue – au lieu de la cellule de vue entière – recalculé à la volée pour chaque image. Notre contribution se base sur un processus de pavage similaire à celui de la section précédente. Celui-ci utilise aussi le principe de propagation de visibilité, mais se distingue par l'utilisation d'un algorithme d'*occlusion culling* pour chaque tuile : l'algorithme CHC++ [66] – par la suite, nous nommons *"contexte CHC++"* l'ensemble des ressources (files de requêtes, BVH local à une tuile, etc.) utilisées par CHC++ pour déterminer la visibilité des objets au sein d'une même tuile – tandis que la visibilité au

(a) Ville



(b) Scène hybride



(c) Forêt

Figure 3.28 – Trois différentes scènes générées avec trois différents ensembles de tuiles. Notre processus de création par pavage intègre la propagation de la visibilité pour la construction des mondes 3D permettant un rendu beaucoup plus interactif.

niveau supérieur est propagée de tuile en tuile par voisinage grâce à des requêtes d'occultation sur les boîtes englobantes de tuile. La méthode proposée permet une liberté totale de création des tuiles, supprime complètement les artéfacts de *popping*, et donne dans de nombreuses situations des performances temps réel pour le rendu de mondes virtuels très vastes.

En section 3.3.1, notre processus de pavage précalcule l'information de visibilité locale (par tuile) par instanciation rapide de hiérarchies de volumes englobants (*bounding volume hierarchy* ou BVH) qui permet l'exécution locale de CHC++. La propagation de visibilité est réalisée par la conjonction d'un *view frustum culling* et des requêtes d'occultation GPU, permettant d'éliminer certaines limitations de CHC++ et d'ajouter de la cohérence temporelle pour accélérer le processus. Les artéfacts de *popping* issus de la variation de la taille des boîtes englobantes des tuiles sont évités par notre méthode. De manière similaire à Greuter et al, les tuiles situées hors de la pyramide de vue ne sont pas instanciées, tandis que notre algorithme intègre l'*occlusion culling* à la génération du monde.

Ensuite, nous analysons les performances sur un ensemble de scènes complexes générées automatiquement (section 3.3.2) et discutons des bénéfices et des limitations de notre approche (section 3.3.3).

3.3.1 Pavage par visibilité

Les méthodes simples de pavage peuvent générer efficacement des mondes virtuels par l'instanciation multiple des tuiles d'un ensemble de pavage réduit. Toutefois, construire entièrement le BVH d'une scène devient très coûteux en temps et en mémoire dès lors que sa taille croît. La figure 3.29 montre ces coûts pour une construction naïve du BVH d'un monde généré par pavage. Notre méthode améliore les performances de génération d'un tel monde indépendamment de la procédure de placement des tuiles. Ainsi, cette dernière ne requiert pas que la position de chaque tuile soit stockée de manière explicite, e.g., par une carte d'instanciation globale ou par le stockage des informations de voisinage.

Notre méthode de génération de monde virtuel prend en entrée un ensemble de pa-

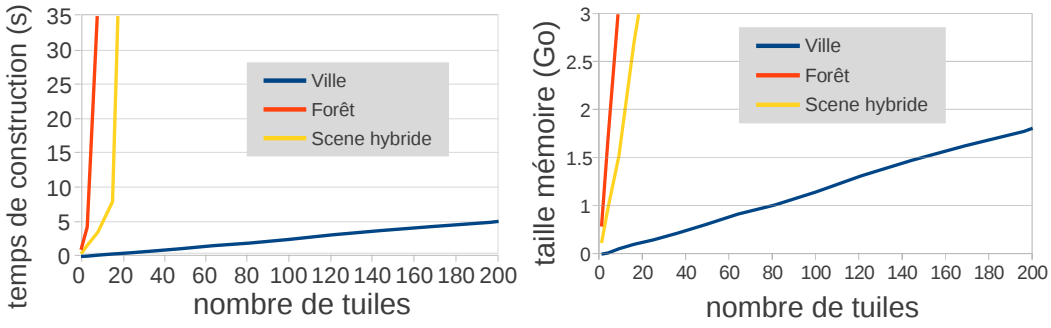


Figure 3.29 – Temps et taille mémoire mesurés pour la création naïve du BVH d'un monde généré par pavage. Notre méthode évite ces coûts par un court temps de construction d'un BVH pour chaque tuile.

vage ayant des motifs quelconques pour paver le plan (carrés, losanges, etc.). Dorénavant, ces motifs seront qualifiés de tuiles et considérés comme des boîtes englobantes alignées sur les axes pour plus de simplicité, mais sans perte de généralité. En réalité, ces tuiles n'ont aucune restriction sur leur forme. La géométrie associée à chaque tuile est définie par un graphe de scène d'objets 3D disposés librement dans la tuile. Ces objets peuvent dépasser des bordures de la tuile. Afin de générer un monde 3D, les tuiles visibles depuis le point de vue sont instanciées dans l'espace 3D, sur une surface 2D pour nos scènes test, grâce à un algorithme d'*occlusion culling* agissant à deux niveaux de hiérarchie. Le premier niveau réalise un *occlusion culling* par requêtes d'occultation matérielles (GPU) sur une boîte englobante commune à l'ensemble de pavage, i.e., l'union des boîtes englobantes de toutes les tuiles de l'ensemble de pavage (sections 3.3.1.1 et 3.3.1.3) pour déterminer la visibilité sur de longues distances. Le second niveau utilise l'algorithme CHC++ de Mattausch et al. [66] pour déterminer l'occultation locale à l'intérieur de chaque tuile.

La prochaine section décrit comment le BVH de chaque tuile est construit à partir de son graphe de scène. Les boîtes englobantes de chaque tuile – définies par les noeuds racines de leur BVH – sont combinées pour obtenir la boîte englobante commune à l'ensemble de pavage.

La section suivante explique comment cette boîte est utilisée pour éviter les artéfacts de *popping*.

Enfin, la dernière section montre comment, à chaque image, le pavage est construit depuis le point de vue jusqu'à ce que le champ visuel soit complètement occulté par les objets visibles par notre algorithme d'*occlusion culling* sur deux niveaux.

3.3.1.1 Représentation de l'ensemble des tuiles

Afin de limiter l'utilisation de la mémoire pour un monde 3D de grande taille, notre méthode utilise un processus d'instanciation d'objets, représentés de manière efficace par un graphe de scène. Chaque tuile de l'ensemble de pavage possède son propre graphe de scène et peut avoir une instance d'objet partagée avec d'autres tuiles du même ensemble. A partir de ce graphe de scène (figure 3.30, gauche), nous construisons un BVH par tuile. Chaque noeud de BVH contient une référence vers les noeuds instances du graphe de scène qu'il contient, et non la géométrie elle-même. Chaque BVH est construit en utilisant l'heuristique *binned SAH* [46, 64]. La construction récursive du BVH s'arrête lorsqu'une des conditions suivantes est remplie : une seule instance est référencée par le noeud, la profondeur maximale est atteinte ou le nombre minimum de polygones par noeud est atteint (figure 3.30, droite). Précalculer le BVH pour chaque tuile de l'ensemble de pavage permet de répartir les coûts de construction de la hiérarchie sur chacune des tuiles de l'ensemble de pavage, mais aussi d'accélérer l'instanciation à la volée pendant le rendu. De plus, si les tuiles contiennent des objets animés, seul le BVH local sera recalculé, ce qui peut être réalisé rapidement.

La boîte englobante située à la racine du BVH est considérée comme la boîte englobante de la tuile. Même si ses objets dépassent la bordure de la tuile (unité alignée sur la grille de pavage), la boîte englobe tous ses objets, sans exception. Ensuite, la boîte englobante commune, unique pour chaque ensemble de pavage, est calculée comme étant l'union de chacune des boîtes englobantes des tuiles de l'ensemble de pavage (figure 3.31). Cette boîte commune permet une détermination conservatrice de la visibilité, indépendante des objets présents dans la tuile : on évite ainsi les artéfacts de *popping* et l'instanciation inutile d'une tuile. De plus, grâce à l'utilisation de cette boîte englobante, le processus de pavage reste indépendant du contenu des tuiles.

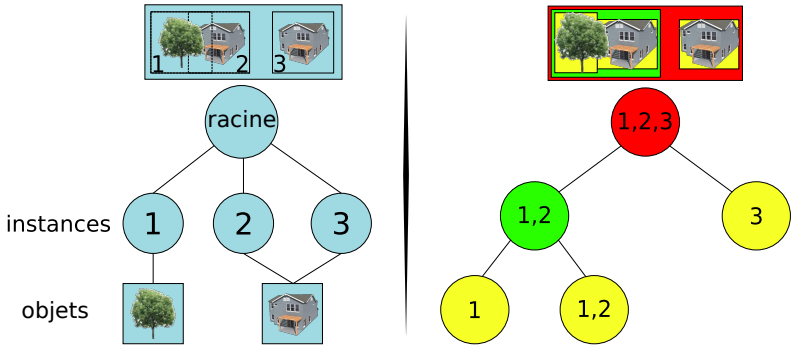


Figure 3.30 – Gauche : la structure d'un graphe de scène utilisée pour l'instanciation d'objets. Les noeuds numérotés correspondent à des instances d'objets. Droite : la structure d'un BVH construite pour une tuile (en rouge). Chaque noeud conserve les références vers les noeuds correspondants du graphe de scène auxquels il appartient. La construction récursive du BVH s'arrête lorsqu'une des conditions suivantes est remplie : une seule instance est référencée par le noeud, la profondeur maximale est atteinte ou le nombre minimum de polygones par noeud est atteint.

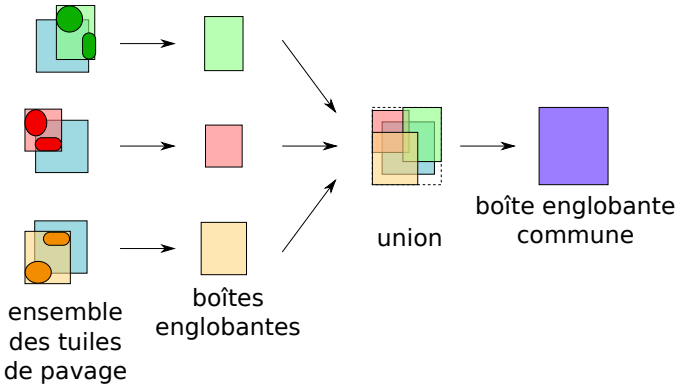


Figure 3.31 – Un ensemble de pavage 2D et ses boîtes englobantes illustrant notre méthode. Les boîtes englobantes (en vert, rouge et orange) sont calculées en fonction des objets sur chaque tuile, même ceux qui dépassent les bordures de la tuile. Chacune de ces boîtes englobantes correspond au noeud racine du BVH précalculé de la tuile. La boîte englobante commune, en violet, est calculée en réalisant l'union de chacune de ces boîtes englobantes. C'est cette boît englobante commune qui est utilisée pour les requêtes d'occultation servant à la propagation de la visibilité sur le pavage.

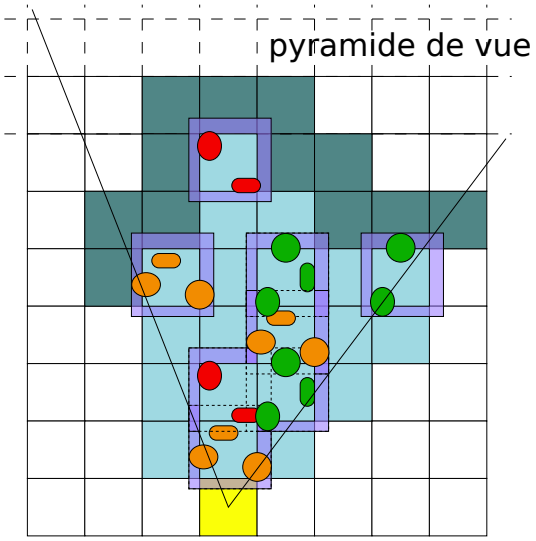


Figure 3.32 – Le processus de pavage dans la pyramide de vue (vue 2D). La visibilité est propagée depuis la tuile visible la plus proche (en jaune). Les tuiles en bleu clair sont celles pour lesquelles les requêtes d'occultation donnent le résultat "visible", celles en bleu foncé sont celles pour lesquelles les requêtes donnent le résultat "non visible".

3.3.1.2 Propagation de visibilité par requête d'occultation sur les tuiles

A chaque image, les objets visibles sont déterminés par propagation de visibilité depuis la tuile visible la plus proche du point de vue. Cette tuile initiale est déterminée par une recherche dans le domaine de pavage. La propagation de visibilité s'effectue par voisinage depuis la tuile initiale, de l'avant vers l'arrière. Le processus de pavage utilise la boîte englobante commune afin de déterminer si une tuile doit être instanciée et affichée, ou si la propagation doit s'arrêter (figure 3.32). Cette étape est nommée "requête de tuile". Si la tuile doit être instanciée, le processus de pavage choisit une tuile dans l'ensemble des tuiles de pavage en fontion de sa position (cf. section 2.3), évitant ainsi le stockage explicite des positions de chaque tuile instanciée. Ce point est particulièrement important pour permettre le passage à échelle de la construction de mondes de grande taille.

Les étapes principales du rendu sont détaillées dans l'algorithme 5. En commençant par la cellule de vue, le BVH correspondant est instancié (s'il ne l'a pas déjà été). Cette instance est créée de manière à ce que chaque noeud possède son propre statut de visi-

```
1  𝒬 ← ∅ // file des requêtes en cours
2  𝒫 ← ∅ // file des requêtes retardées
   // première tuile à afficher
3  𝒯 ← tuileLaPlusProcheDansLaPyramideDeVue(caméra)
4  tant que 𝒯 ≠ ∅ faire // propagation de la visibilité
5
6      rendreAvecCHC++(instanceBVH(𝒯))
7      requêtesVoisins(𝒯,𝒬)
       // prochaine tuile à afficher
8      𝒯 ← requêtePositiveSuivante(𝒬,𝒫)
9  fin
10 miseAJourRequêtesRetardées(𝒫)
```

Algorithme 5: Algorithme principal des requêtes d'occultation.

Entrées:
- la tuile courante $\mathscr{T}$
- file des requêtes en cours $\mathscr{Q}$

Sorties:
- file des requêtes en cours $\mathscr{Q}$

```
   // requêtes des tuiles voisines de 𝒯
1  pour chaque voisin n ∈ 𝒯 faire
2      si ¬ n.demandéDurantCetteImage alors
3          si intersectePyramideDeVue(n) alors
               // lancement asynchrone de la requête
4              q ← requête(boîteCommuneTranslatée(n))
5              mettreEnFinDeQueue(𝒬,(q,n))
6          fin
7          n.demandéDurantCetteImage = vrai
8      fin
9  fin
```

Algorithme 6: requêtesVoisins($\mathscr{T}$,$\mathscr{Q}$).

bilité (de manière similaire à CHC++) lors du déroulement de l'algorithme d'*occlusion culling*. Lorsqu'un noeud de l'instance est rendu, les noeuds instances associés du graphe de scène sont rendus, s'ils ne l'ont pas déjà été pendant l'image courante (figure 3.30). Dès lors que la tuile courante a été rendue par *occlusion culling* local avec CHC++, une requête d'occultation est lancée pour chacune des tuiles voisines (algorithme 6) en utilisant la boîte englobante commune décalée à sa position dans le pavage. Afin de bénéficier d'une génération du monde de l'avant vers l'arrière et ainsi améliorer l'efficacité de notre méthode d'*occlusion culling*, les requêtes d'occultation des tuiles sont stockées dans une file.

Dans l'algorithme 7, les tuiles dont la requête d'occultation n'est pas directement disponible et dont le statut a été déterminé comme étant visible à l'image précédente

Entrées:
- file des requêtes en cours $\mathscr{Q}$
- file des requêtes différées $\mathscr{P}$

Sorties:
- file de requêtes en cours $\mathscr{Q}$
- file des requêtes différées $\mathscr{P}$
- la tuile voisine n à afficher

```
// première requête disponible dont le résultat est positif
1  tant que (Q ≠ ∅) faire
2      (q,n) ← pop(Q)
3      si ¬ résultatDisponible(q) ∧ étaitVisibleALImagePrécédente(n) alors
           // retardement de la récupération du résultat de la
              requête
4          mettreEnFinDeQueue(P,(q,n))
5          retourner n // afficher la tuile n
6      sinon
7          attendreResultatDeLaRequête(q)
8          // prédiction de visibilité pour l'image suivante
9          si résultatPositif(q) alors
10             n.visible = vrai
11             retourner n // afficher la tuile n
12         sinon
13             n.visible = faux
14         fin
15     fin
16 fin
17 return ∅ // plus de tuile à afficher
```

Algorithme 7: requêtePositiveSuivante($\mathscr{Q}$,$\mathscr{P}$).

sont rendues (principe de cohérence temporelle, algorithme 7, ligne 3) : dans ce cas-là, la récupération du résultat de la requête est différée jusqu'à la fin de l'image courante (file des requêtes différées $\mathscr{P}$) et la tuile est rendue. Etant donné que la plupart des résultats de requêtes sont disponibles, la plupart des tuiles possèdent un statut de visibilité correct avant le début de la prochaine image.

Le pavage est généré d'avant en arrière, ce qui implique : (1) que les premières requêtes d'occultation sur les tuiles situées dans la file $\mathscr{Q}$ sont celles des tuiles proches de l'observateur, donc plus enclines à être visibles ; et (2) que leur projection en espace écran est plus grande et leur requête d'occultation prend plus de temps à résoudre. Ces requêtes sont donc les plus susceptibles d'être différées car leur résultat sera demandé très tôt dans le processus de rendu. Ces requêtes ont ensuite jusqu'à la fin de l'image pour fournir leur résultat.

```
   Entrées:      • file des requêtes différées 𝒫
1  // retour des résultats de l'image courante
2  tant que 𝒫¬∅ faire
3  |  (q,n) ← pop(𝒫);
   |  // prédiction de visibilité pour l'image suivante
4  |  si résultatDisponible(q) ∧ résultatPositif(q) alors
5  |  |  n.visible = vrai;
6  |  sinon
7  |  |  n.visible = faux;
8  |  fin
9  fin
```

Algorithme 8: miseAJourRequêtesRetardées($\mathscr{P}$).

A l'opposé, les tuiles les plus éloignées sont les plus enclines à être occultées : c'est leur résultat qui en général peut provoquer de la latence lors de l'attente du résultat d'une requête d'occultation (algorithme 7, ligne 7). Toutefois, ces tuiles situées plus loin voient leur projection en espace écran être plus petite : elles prennent donc moins de temps à dessiner et le résultat de la requête asssociée prend aussi moins de temps pour être retourné. De plus, ces requêtes de tuiles éloignées sont plus nombreuses dans la file $\mathscr{Q}$ car la pyramide de vue englobe plus de tuiles lorsque la distance à l'observateur augmente. Alors, les requêtes des tuiles éloignées possèdent en général plus de temps pour fournir leur résultat. Du fait qu'il n'y ait que très peu de latence dans le processus de requêtes d'occultation, la prédiction de visibilité s'effectue de manière assez souple pour que le rendu soit efficace.

Toutefois, certaines configurations peuvent affecter la visibilité de manière à ce que la latence soit trop grande. Dans ce cas-là le temps passé à attendre peut être mis à profit en réalisant la préinstanciation des tuiles correspondantes.

A la fin de l'algorithme, la disponibilité du résultat de chaque requête est vérifiée et le statut de visibilité de la tuile correpondante est mis à jour en conséquence (algorithme 8). Ainsi, lors de la prochaine image, le résultat de l'image courante pourra être utilisé pour prédire le statut de visibilité des tuiles du pavage. A la ligne 4 de l'algorithme 8, les résultats de requêtes qui ne sont pas encore disponibles correspondent à des tuiles de bordure de pavage, i.e., les plus éloignées du point de vue. Prédire ces tuiles comme non visibles force l'algorithme 7 en attente active à la ligne 7, mais du fait que la projection

en espace écran est petite, ce coût est léger. Nous remarquons également que le nombre de tuiles concernées par ce cas de figure est faible : dans nos scènes test, le nombre de requêtes dont le résultat n'est toujours pas disponible à la fin de l'image est estimé à seulement 0.02% (Ville), 0.04% (Scène hybride), et 0.14% (Forêt) du nombre total de requêtes.

L'adjonction de la composante de cohérence temporelle permet une accélération substancielle du rendu de nos scènes test. Puisque les tuiles sont rendues d'avant en arrière, la plupart des résultats de requêtes qui ne sont pas encore disponibles sont pour des tuiles de bordure du pavage actuellement instanciées. De ce fait la plupart des tuiles étaient déjà visibles dans les images précédentes et sont donc déjà prédites comme visibles pour l'image courante.

Lorsqu'une instance de tuile est marquée visible pour la première fois, une instance du BVH correspondant est créée et gardée dans un cache afin de pouvoir stocker son propre contexte CHC++. Ainsi, la cohérence temporelle des requêtes de tuiles est indépendante de celle de CHC++ et évite des surcoûts d'instanciation d'une image à l'autre. Dès lors qu'une tuile n'est pas visible depuis un certain temps, l'instance de BVH correspondante est retirée du cache, réduisant ainsi l'utilisation mémoire.

3.3.1.3 Eviter les artéfacts de *popping*

L'utilisation seule de la boîte englobante de chaque tuile – celle correspondant au noeud racine du BVH de chaque tuile – adjointe à une relation de voisinage ne suffit pas à garantir qu'une tuile proche de l'observateur recouvre complètement une tuile plus éloignée. La figure 3.33 montre un exemple dans lequel l'utilisation de la boîte englobante de la tuile échoue. En effet la tuile 1 est instanciée et ses objets masquent complètement la boîte englobante de la tuile 2. De ce fait la propagation par voisinage n'ajoute pas la tuile 3 au pavage alors que l'objet le plus haut de celle-ci est visible et donc censé être affiché. Par contre, l'utilisation de la boîte englobante commune pour générer les requêtes d'occultation des tuiles permet de solutionner ce problème car chaque requête d'occultation est effectuée avec une boîte de même étendue. De ce fait, dès lors que le résultat de l'une d'entre elles est négatif, cela signifie que les boîtes des tuiles plus

éloignées sont masquées entièrement par les objets contenus dans les tuiles du pavage instancié. Les tuiles situées au-delà de l'étendue courante du pavage ne sont donc pas visibles. Par conséquent, lancer des requêtes d'occultation pour de telles tuiles n'est pas nécessaire et l'algorithme de propagation de visibilité peut s'interrompre.

3.3.2 Résultats et analyse de performance

Nous avons testé notre algorithme sur différents ensembles de pavages, composés de maisons et d'arbres, comme illustré à la figure 3.28. Les objets ont été importés depuis le site internet *turbosquid* [101]. Chaque tuile contient entre 20K et 2.5M polygones. Aucun niveau de détails (*LOD*) n'a été utilisé, et ainsi chaque objet est rendu avec sa forme la plus détaillée dès lors qu'il est déterminé visible. Les tests de performance ont été réalisés sur un Intel® Core™ i7-930 avec une GeForce GTX 680. Les BVH des tuiles sont par ailleurs construits grâce à l'implémentation parallèle de construction de BVH *embree* [108], utilisant l'heuristique de découpe spatiale. Les temps de construction, profondeur et taille mémoire résultant de la construction des BVH de chaque tuile sont donnés au tableau 3.5.

Nous avons mis en place trois chemins de caméra[1] au travers de trois paysages différents générés par chacun des trois différents ensembles de pavage. Nous comparons quatre algorithmes d'*occlusion culling* différents : requête d'occultation sur les tuiles & CHC++ par tuile (*tile query* + CHC++), requête d'occultation sur les tuiles seulement (*tile query*), CHC++ par tuile (*per-tile* CHC++), ainsi que *view frustum culling* seul. Une comparaison directe entre notre méthode (*tile query* + CHC++) et l'algorithme CHC++ seul ne peut être équitable envers CHC++ puisque ce dernier nécessiterait la construction entière des BVH de nos scènes très étendues. Afin de montrer les bénéfices de notre méthode, nous considérons, pour l'algorithme CHC++ par tuile seul (*per-tile* CHC++), pour chaque image, les BVH de chaque tuile comprise dans la boîte englobante du pavage instancié par l'algorithme des requêtes d'occultation sur les tuiles (*tile query*) – donc un pa-

[1] Ces chemins caméra sont montrés dans une vidéo de démonstration : l'adresse `fr.linkedin.com/in/doriangomez/` contient un lien vers mon site personnel ou vous pourrez retrouver toutes mes vidéos de démonstration.

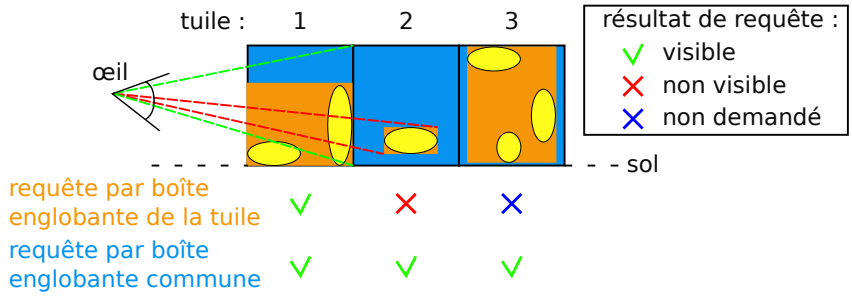


Figure 3.33 – La boîte englobante commune, contrairement à la boîte englobante de chaque tuile, assure la validité de l'heuristique de voisinage lors de la propagation de la visibilité. Dans l'exemple illustré ci-dessus, la requête d'occultation de la tuile 2 retourne un résultat valide, et la tuile 3 est également testée. Ainsi, les objets de cette dernière n'apparaissent pas soudainement lorsque l'observateur traverse la tuile 1 ou lorsque sa hauteur croît.

vage dont l'étendue reste conservatrice. L'algorithme des requêtes d'occultation sur les tuiles (*tile query*), lorsqu'il est utilisé seul, rend tous les objets d'une tuile quand elle est déterminée visible au lieu de la rendre avec CHC++. Les figures 3.34 et 3.35 montrent le nombre d'images par seconde et le nombre de triangles rendus pour ces quatre chemins de caméra.

Comme mentionné précédemment, il est parfois préférable de rendre une tuile entièrement plutôt que de déterminer son statut de visibilité. Ainsi, nous pouvons voir qu'utiliser l'algorithme seul des requêtes d'occultation sur les tuiles est parfois plus performant que de l'utiliser en conjontion avec CHC++ dans les scènes de ville et hybride, où les objets sont moins complexes et produisent plus d'occultation (figures 3.34(a) et 3.34(b)). Toutefois, cet algorithme seul induit beaucoup plus de variance dans la fréquence de rendu car le nombre de triangles rendus diffère beaucoup plus d'une image à l'autre.

Nous pouvons constater que, pour la plupart des points de vue, notre algorithme utilisé en conjonction avec CHC++ est meilleur que CHC++ seul sur chaque tuile (figures 3.34(a), 3.34(b) et 3.34(c)). Cela s'explique simplement par le fait que CHC++ ne possède pas l'information de visibilité ou de cohérence temporelle sur les tuiles voisines et donc plus de tuiles sont instanciées sur le pavage et plus de temps est alloué à la résolution des requêtes d'occultation des tuiles. Le chemin de caméra de la forêt est celui

pour lequel notre algorithme montre toute son efficacité dans l'aggrégation de l'occultation de plusieurs tuiles (figure 3.34(c)). Dans chacun des trois chemins de caméra, la fréquence d'affichage décroît de manière drastique lorque l'observateur prend de l'altitude et regarde le pavage à angle rasant. En effet, beaucoup de tuiles sont alors visibles et donc instanciées et rendues.

En ce qui concerne le nombre de triangles rendus (figures 3.35(a), 3.35(b) et 3.35(c)), notre algorithme est tout aussi conservateur en terme d'occultation que CHC++ par tuile.

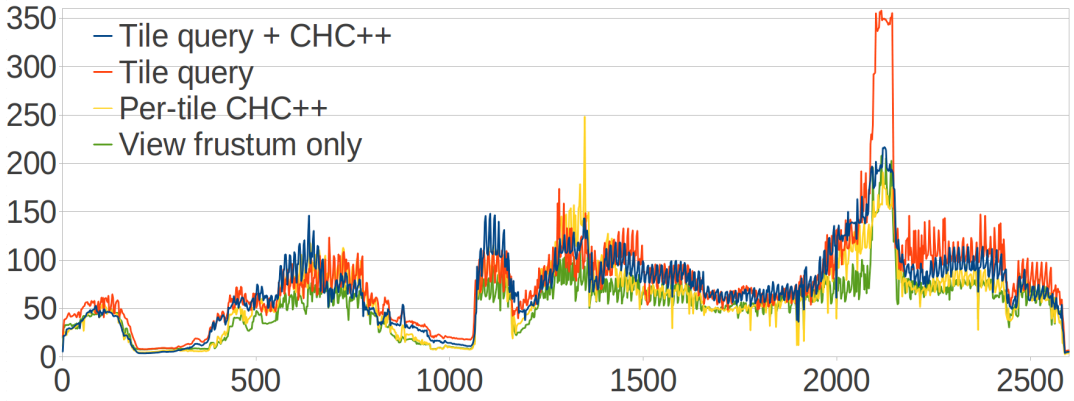
3.3.3 Discussion

Comme expliqué en section 3.3.1, l'utilisation de la boîte englobante commune assure l'absence d'artéfacts de *popping*. Toutefois, cela peut affecter la fréquence d'affichage dans les cas où la visibilité est trop surestimée (algorithme trop conservateur). Si la boîte englobante d'une tuile est trop petite par rapport à la boîte englobante commune, la tuile sera très souvent déterminée visible, même si ce n'est pas le cas. La tuile sera donc instanciée inutilement.

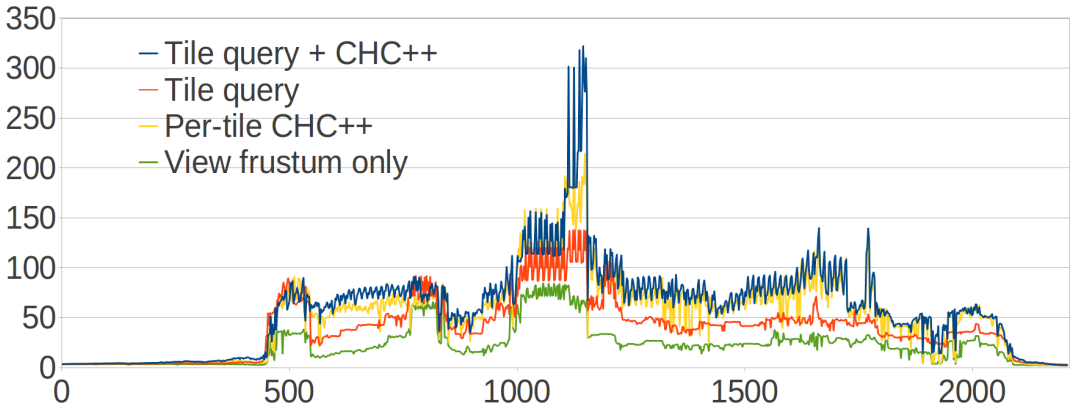
Comme pour chaque méthode d'*occlusion culling*, l'efficacité de notre algorithme dépend grandement de l'occultation générée par la scène. Dans notre cas, comme l'*occlusion culling* est réalisé indépendemment sur chaque tuile, l'efficacité du blocage repose sur l'occultation locale d'une tuile, mais aussi sur le positionnement des tuiles les unes par rapport aux autres. Si les tuiles ne contiennent pas suffisamment d'objets ou que les tuiles choisies par la fonction de hachage laissent propager la visibilité dans des directions similaires, le pavage visible peut alors s'étendre sur de très longues distances.

Comme pour CHC++, la cohérence temporelle peut être augmentée de telle manière qu'une tuile soit marquée visible pour un nombre aléatoire d'images successives, résultant en moins de requêtes d'occultation réparties au cours du temps. Il est également possible d'utiliser l'ordre d'avant en arrière pour diminuer la fréquence des requêtes des tuiles plus proches de l'observateur. Dans nos scènes test, la modification de la fréquence des requêtes n'apporte pas d'amélioration significative à la vitesse de rendu car nos tuiles génèrent suffisamment d'occultation.

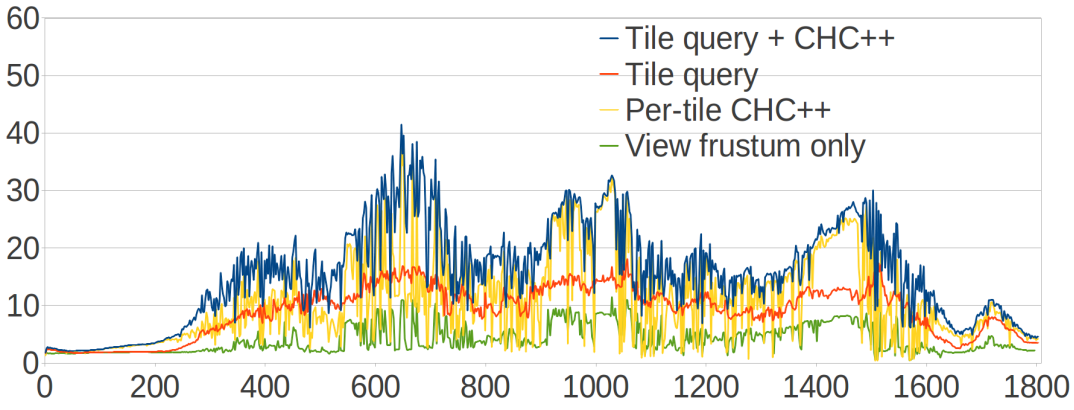
Dans notre algorithme, tous les polygones d'un objet déterminé visible sont rendus,



(a) Ville

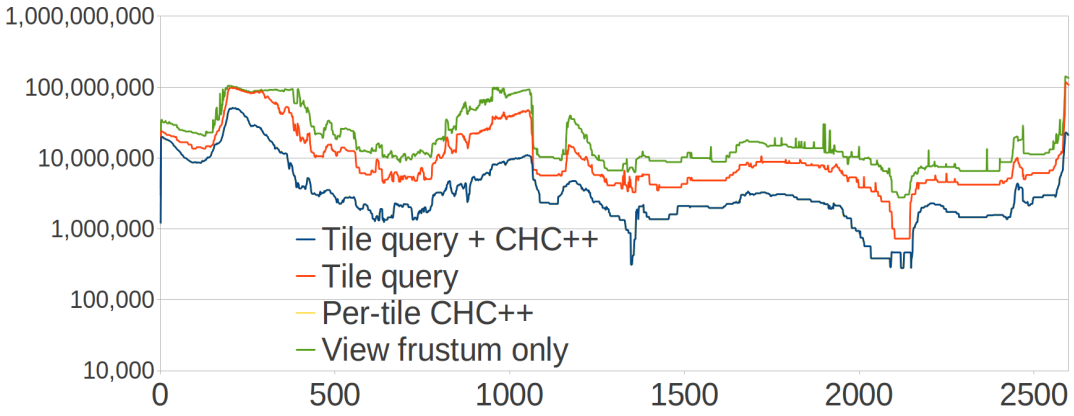


(b) Scène hybride

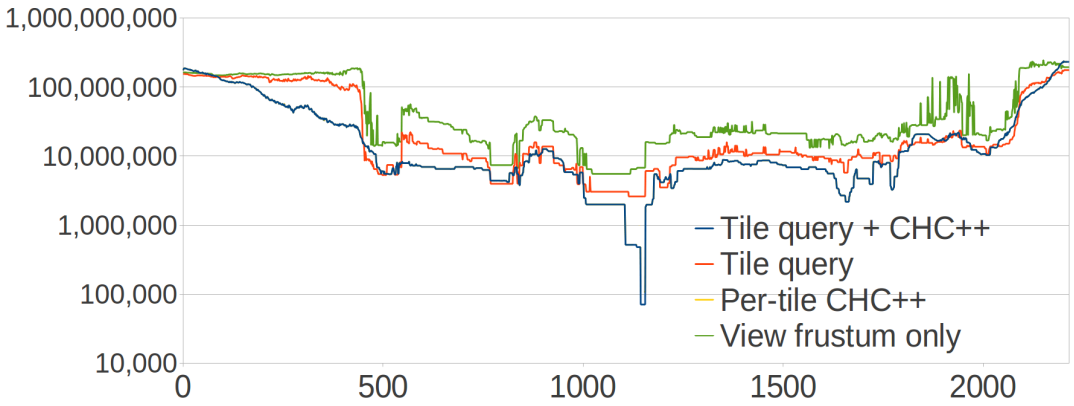


(c) Forêt

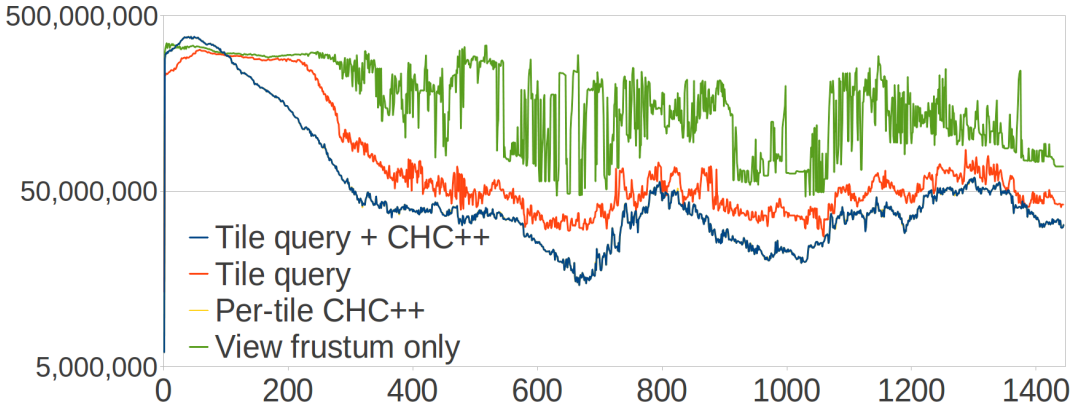
Figure 3.34 – La fréquence d'affichage en *Frames Per Second* (images par seconde) pour les trois chemins de caméra différents dans trois paysages virtuels différents, utilisant chacun un ensemble de pavage différent (la courbe la plus haute est la meilleure).



(a) Ville



(b) Scène hybride



(c) Forêt

Figure 3.35 – Le nombre de triangles rendus pour les chemins de caméra dans chacune des scènes générées (la courbe la plus basse est la meilleure, les courbes bleues et jaunes se superposent).

ID tuile	Ville				Scène hybride				Forêt			
	temps (ms)	profondeur	taille (Mo)	nb. polygones	temps (ms)	profondeur	taille (Mo)	nb. polygones	temps (ms)	profondeur	taille (Mo)	nb. polygones
tuile 0	184	16	24.6	0.768M	251	16	46.6	1.100M	453	16	61.2	2.41M
tuile 1	132	16	27.1	0.728M	181	16	35.9	0.934M	484	16	67.8	2.31M
tuile 2	103	16	20.9	0.580M	391	16	54.9	2.129M	472	16	66.6	2.48M
tuile 3	85	16	16.7	0.474M	184	16	34.0	0.967M	431	16	59.8	1.98M
tuile 4	85	16	16.1	0.674M	114	16	26.8	0.543M	215	16	47.1	1.03M
tuile 5	117	16	21.2	0.363M	125	16	30.3	0.607M	217	16	44.2	1.06M
tuile 6	67	16	15.8	0.276M	45	16	11.2	0.174M	278	16	55.5	1.42M
tuile 7	61	16	14.9	0.580M	96	16	23.8	0.445M	283	16	54.5	1.41M
tuile 8	104	16	19.9	0.050M	192	16	42.2	0.908M	121	16	30.3	0.47M
tuile 9	81	16	15.6	0.459M	111	16	27.6	0.528M	99	16	25.3	0.42M
tuile 10	106	16	27.8	0.562M	10	13	1.3	0.024M				
tuile 11	72	16	18.3	0.365M	164	16	32.8	0.838M				
tuile 12	58	16	12.0	0.313M	74	16	19.9	0.316M				
tuile 13	90	16	18.0	0.494M	184	16	37.7	0.935M				
tuile 14	70	16	14.8	0.369M	94	16	22.5	0.462M				
tuile 15	84	16	18.6	0.472M								
tuile 16	102	16	22.4	0.565M								
tuile 17	57	16	14.9	0.275M								
tuile 18	86	16	19.2	0.472M								
tuile 19	58	16	17.0	0.275M								

Tableau 3.5 – Les statistiques de construction pour les tuiles de nos trois scènes test.

impliquant une baisse drastique des performances lors d'un survol aérien de nos scènes lorsque les objets distants du point de vue sont complexes. Cette baisse de performance pourrait être considérablement limitée par l'intégration effective de niveaux de détails sur les objets, utilisant des maillages simplifiés pour les objets distants du point de vue.

3.3.4 Conclusion

Dans cette section, nous avons associé pavage et visibilité en trois dimensions. Dans ce contexte, nous proposons une solution pour réaliser de l'*occlusion culling* sur des mondes virtuels générés à la volée. Grâce à un faible coût d'instanciation de BVH, notre méthode permet d'afficher plusieurs centaines de millions de polygones en temps réel.

Dans nos exemples, les tuiles sont instanciées à la volée sur une surface 2D mais pourraient l'être dans l'espace 3D. Seule la relation de voisinage doit être étendue à la 3D. Il est toutefois important de savoir si une scène nécessite une telle mise en œuvre, à savoir une forte densité d'objets dans un volume très étendu, dans chacune des trois dimensions.

La construction d'un BVH peut être réalisée en quelques millisecondes pour un nombre de polygones par tuile et une profondeur du BVH raisonnables. Dans une implémentation parallélisée, le BVH d'une tuile voisine peut être calculé à la volée dans un processus indépendant, dès lors qu'une requête est lancée. Ainsi, le processus de pa-

vage pourrait lui-même créer les objets de la tuile à la volée lorsqu'elle est instanciée, à condition de respecter la contrainte de la boîte englobante commune. De cette manière l'ensemble de pavage ne nécessiterait même pas de contenir au préalable des tuiles prédéfinies. Ce genre de performance serait sûrement très appréciable dans un système de génération procédurale, avec la contrainte de connaître au préalable la taille maximale d'une tuile.

De plus, l'utilisation de cette méthode allège le coût total de construction du PVS induit par le nombre de polygones présents dans la scène, par plusieurs étapes locales de calculs de BVH. Enfin, nous avons choisi d'utiliser l'algorithme CHC++ pour les calculs de visibilité locale, car CHC++ est l'algorithme de référence dans le domaine de l'*occlusion culling* en temps réel. Toutefois il reste totalement interchangeable avec n'importe quelle autre méthode de détermination de la visibilité, qu'elle soit conservatrice, exacte ou échantillonnée (voir section 1.2.2).

Afin d'améliorer la détermination des objets cachés de nos scènes, et donc accélérer le rendu de ces dernières, notre méthode peut également être associée avec un *z-buffer* hiérarchique [40] ou une carte d'occupation [95]. La dernière de ces deux méthodes utiliserait directement les noeuds du BVH au lieu des noeuds du graphe de scène afin de tester et mettre à jour la carte d'occupation.

3.4 Conclusion

Dans ce chapitre, nous avons présenté ce qui compose le coeur de cette thèse. Nous avons tout d'abord relâché la contrainte d'étendue du pavage, puis celle de la troisième dimension. Puis, nous avons supprimé les artéfacts de *popping* liés à l'échantillonnage en introduisant une méthode conservatrice temps réel basée sur l'algorithme CHC++.

Par ailleurs nous avons pu mettre en place plusieurs outils d'analyse de l'occultation locale d'une tuile par échantillonnage, en 2D, par les vecteurs d'occultation et l'espace dual des lignes de vue, et en 3D, par les matrices de visibilité. Ces dernières sont notamment utilisées dans le chapitre suivant afin d'augmenter l'occultation d'une tuile, par optimisation du placement des objets qu'elle contient.

On pourrait également choisir de propager la visibilité jusqu'à une distance fixée d'avance, puis insérer des tuiles complètement occultantes ou des imposteurs pour assurer une limite finie à cette propagation – nous ne traitons toutefois pas ce cas dans cette thèse.

CHAPITRE 4

ACCÉLÉRATION DU RENDU DES TUILES VISIBLES

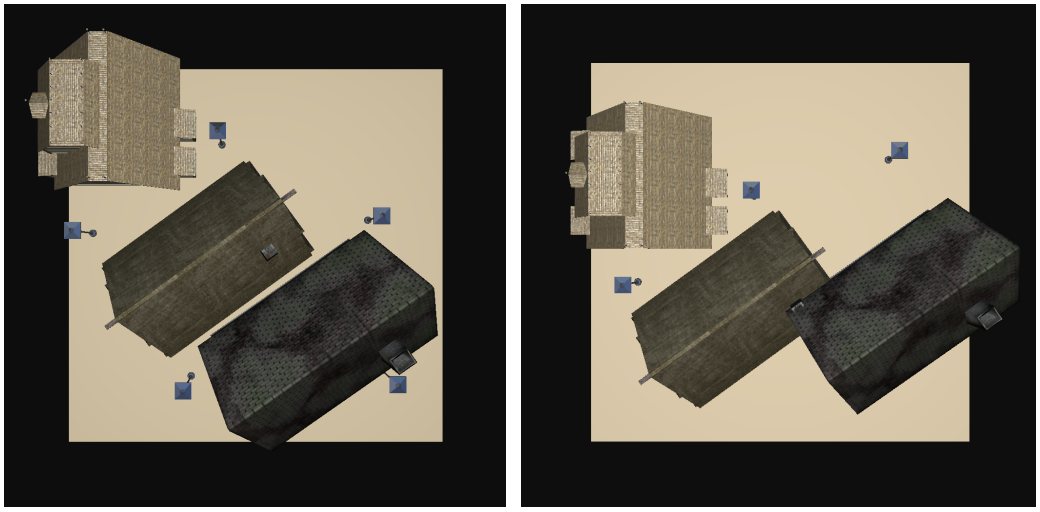
Nos précédentes contributions s'orientent vers l'affichage efficace de géometries instanciées à la volée sur un plan 2D, selon leur visibilité depuis le point de vue. Notre moteur de rendu utilise des tuiles, auxquelles peuvent être attachés des objets 3D quelconques. Ce moteur, grâce à l'utilisation des requêtes d'occultation, se comporte de manière suffisamment stable lorsque le placement des objets est occultant. Toutefois il n'est présentement pas possible de garantir de bonnes performances dans la plupart des scènes où les occultations ne sont pas suffisamment prédominantes.

Les performances d'affichage sont moins bonnes lorsque les objets sont complexes et bloquent peu la visibilité. Dans de tels cas, les objets visibles deviennent de plus en plus nombreux dans la pyramide de vue. Ce chapitre regroupe plusieurs optimisations qui permettent d'accélérer leur rendu. Les optimisations de la prochaine section modifient la scène en repositionnant les objets sur les tuiles (figure 4.1). Les méthodes qui suivent tablent sur des heuristiques de détermination des tuiles visibles.

4.1 Optimisation du placement des bloqueurs

Nous souhaitons exposer dans cette section, deux approches, l'une déterministe et l'autre stochastique, qui permettent d'optimiser la position des objets dans les tuiles de base de l'ensemble de pavage. Ces optimisations ont pour but d'accélérer le rendu de nos scènes. Pour cela, nous utilisons les matrices de visibilité présentées en section 3.2.1.1 à partir des positions préexistantes des objets sur les tuiles de base. Ensuite, nous extrayons les informations de visibilité et d'occultation.

Pour la méthode déterministe (section 4.1.1), nous calculons, à partir des matrices de visibilité, quel est le point le plus occulté et le point le moins occulté dans le voisinage d'un objet. Nous déduisons ensuite les déplacements possibles qui peuvent affecter l'occultation des tuiles.



(a) Tuile de base.

(b) Un exemple d'optimisation pour le blocage dans la direction Nord/Sud appliqué à la même tuile. Ici, les collisions potentielles entre les objets de la tuile ne sont pas prises en compte.

Figure 4.1 – L'optimisation de l'occultation par déplacement des objets.

Pour la méthode stochastique (section 4.1.2), nous calculons, à partir des matrices de visibilité, un score de visibilité, basé sur le nombre de "trous" de visibilité dans la tuile. Ce score permet d'évaluer le potentiel occultant d'une tuile. Ainsi nous procédons à une recherche aléatoire des positions optimales pour les objets de la tuile et obtenons plus de variations possibles dans la disposition de objets.

Autant pour la première que pour la seconde méthode, nous assurons que les objets ne soient pas trop éloignés de leur position d'origine, ceci afin que les tuiles de base restent le plus possible conformes au design initial prévu par l'infographiste.

Enfin, dans une troisième section, nous analysons et comparons les résultats obtenus par chacune des deux méthodes.

4.1.1 Approche déterministe : déplacement par points d'intérêts

4.1.1.1 Descripteur de visibilité

Dans cette section, la visibilité et l'occultation d'une tuile sont évalués par estimation de la densité d'occultation. Afin de mesurer la densité d'occultation en tout point de la tuile, nous nous servons des matrices de visibilité, dont le calcul est expliqué en section 3.2.1.1. En utilisant ces matrices, nous lançons un rayon depuis le centre de chacun des *gridels* entrants (éléments de grille) vers le centre de chacun des *gridels* sortants. Chacun des voxels de position v_i, où i est l'indice du voxel de la tuile, possède deux compteurs : un compteur d'occultation cpt_occ_i et un compteur de visibilité cpt_vis_i. Le rayon lancé incrémente soit le compteur d'occultation, soit le compteur de visibilité de chacun des voxels qu'il traverse, selon que la relation *gridel* entrant / *gridel* sortant est occultante ou visible. Cette opération est répétée pour chacun des couples *gridel* entrant / *gridel* sortant possibles.

$$cpt\_occ_i = \sum_{g_{in} \in G} \sum_{g_{out} \in G} non\_visible(g_{in}, g_{out}) \times intersect(v_i, g_{in}, g_{out}) \tag{4.1}$$

$$cpt\_vis_i = \sum_{g_{in} \in G} \sum_{g_{out} \in G} visible(g_{in}, g_{out}) \times intersect(v_i, g_{in}, g_{out}) \tag{4.2}$$

où G est l'ensemble des *gridels* de la tuile t, $non\_visible(g, g')$ la fonction retournant 1 si le *gridel* g n'est pas visible depuis g', 0 sinon, $visible(g, g')$ la fonction retournant 1 si le *gridel* g est visible depuis g', 0 sinon, et $intersect(v_i, g, g')$ la fonction retournant 1 si le rayon reliant les centres de g et g' intersecte le voxel v_i, 0 sinon.

Pour finir, chacun des compteurs est normalisé pour obtenir un coefficient compris entre 0 et 1. Les coefficients d'occultation $\bar{w}_i$ et de visibilité w_i d'un voxel v_i sont donnés par les équations suivantes :

$$\bar{w}_i = \frac{cpt\_occ_i}{cpt\_occ_i + cpt\_vis_i} \tag{4.3}$$

$$w_i = 1 - \bar{w}_i \tag{4.4}$$

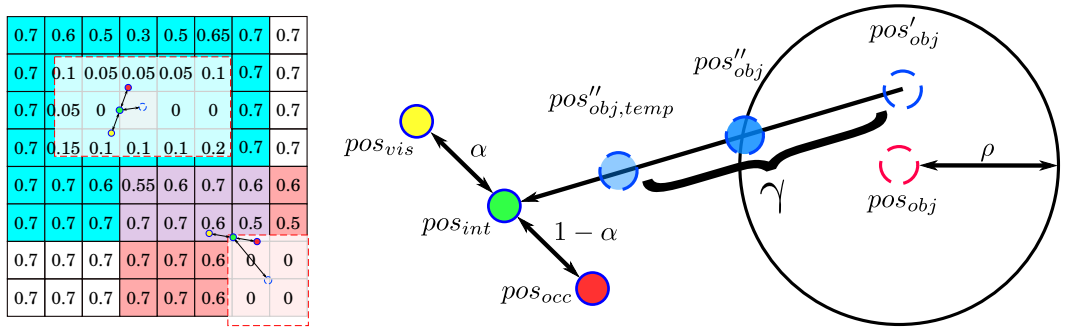
où i est l'indice du voxel dont le compteur est calculé.

Ainsi, la grille de voxels possède deux champs de densité 3D, complémentaires l'un de l'autre, permettant de déterminer, pour chacun des points de l'espace d'une tuile, son coefficient d'occultation, i.e., s'il laisse passer la visibilité au travers de la tuile. Un exemple de grille de densité de visibilité obtenue pour le voisinage de deux objets est montré à la figure 4.2(a). Dans notre exemple, les couches horizontales de champ de densité 3D ont été fusionnées (moyenne) afin de pouvoir calculer des déplacements d'objets sur une seule couche (en 2D).

4.1.1.2 Points d'intérêts

Une fois les champs de densité d'occultation et de visibilité calculés pour une tuile, il s'agit dès lors de maximiser l'occultation globale de cette tuile en déplaçant les objets qu'elle contient. Pour cela, plusieurs contraintes doivent être respectées : un objet ne doit pas être déplacé trop loin de son point d'origine afin de respecter le modèle initial de l'infographiste ; si un objet génère beaucoup d'occultation localement, il doit être d'autant plus difficile de le déplacer ; et un objet a tout intérêt à se déplacer vers une zone où la densité de visibilité est élevée afin de générer plus de blocage après déplacement.

Ainsi, la première contrainte implique l'utilisation d'un paramètre de distance maximale ρ par rapport à la position d'origine de l'objet. ρ peut être fixé par l'utilisateur et sera exprimé dans ce chapitre comme une fraction de la taille unitaire d'une tuile. Les deux autres contraintes suggèrent l'utilisation de points d'intérêts. Lorsqu'un objet doit être déplacé, nous utilisons la position courante du centre de sa boîte englobante pos'_{obj} pour obtenir la position v_{obj} du voxel auquel elle appartient. Lors de la première passe d'optimisation pos'_{obj} est égale à la position d'origine pos_{obj} de l'objet. Ensuite, nous calculons deux positions moyennes à partir de la position du centre du voxel central v_{obj} et des positions des centres des voxels voisins v_j. Ces deux positions servent de points d'intérêts : la première, pos_{occ}, est calculée en pondérant les positions v_{obj} et v_j par chacun des coefficients d'occultation associés ; la seconde, pos_{vis}, est calculée en pondérant



(a) (b)

Figure 4.2 – (a) Un exemple de densité de visibilité obtenu dans les voisinages de deux objets. Les voxels bleus représentent le voisinage du premier objet. Les voxels rouges représentent le voisinage du second objet. Les voxels mauves représentent le voisinage des deux objets. Les points rouge et jaune désignent respectivement les points d'intérêts d'occultation et de visibilité de ce voisinage. (b) Schéma de la méthode utilisée pour déplacer un objet en fonction des points d'intérêts pos_{vis}, pos_{occ} et de la distance maximale de déplacement ρ.

ces positions par chacun des coefficients de visibilité :

$$pos_{occ} = \frac{\bar{w}_{obj} \times v_{obj} + \sum_j (\bar{w}_j \times v_j)}{\bar{w}_{obj} + \sum_j \bar{w}_j} \tag{4.5}$$

$$pos_{vis} = \frac{w_{obj} \times v_{obj} + \sum_j (w_j \times v_j)}{w_{obj} + \sum_j w_j} \tag{4.6}$$

Ici, chacun des poids $\bar{w}_j$ et w_j est utilisé tel quel mais il est également possible d'utiliser un filtre gaussien appliqué au rayon de voisinage afin de donner un poids plus grand aux voxels centraux (ceux qui sont proches de la position courante pos'_{obj}) et un poids moindre aux voxels de périphérie. Lorsque le voisinage d'un voxel dépasse de la grille de voxels de la tuile, les valeurs de bordure sont répétées pour éviter que les positions moyennes pos_{occ} et pos_{vis} ne soient trop attirées au centre de la tuile lorsque l'on choisit un voisinage trop grand

Dans notre cas, nous utilisons simplement une combinaison linéaire entre les points

pos_{occ} et pos_{vis} afin de trouver le point d'intérêt pos_{int} :

$$pos_{int} = \alpha \times pos_{occ} + (1-\alpha) \times pos_{vis} \ , \qquad 0 \leq \alpha \leq 1 \tag{4.7}$$

où α est un coefficient permettant de privilégier l'occultation courante ($\alpha = 1$) ou de d'occuper les endroits les plus visibles ($\alpha = 0$).

4.1.1.3 Déplacement

Cette position d'intérêt permet ensuite de déplacer l'objet plus ou moins loin d'un facteur γ dans la direction $\overrightarrow{pos'_{obj}\ pos_{int}}$ où pos'_{obj} est la position courante de l'objet. Pour nos tests, le facteur de déplacement γ est une valeur constante. Toutefois, ce facteur peut être calculé par une fonction qui dépend par exemple de l'aire de la surface 2D de la boîte englobante ou du nombre de polygones de l'objet à déplacer, ou toute autre fonction définie par l'utilisateur selon le résultat qu'il souhaite obtenir (fonction proportionnelle, d'ordre n, etc.). La nouvelle position temporaire de l'objet est ainsi obtenue :

$$pos''_{obj,temp} = pos'_{obj} + \gamma \times \overrightarrow{pos'_{obj}\ pos_{int}} \tag{4.8}$$

où γ permet d'accentuer ou de diminuer l'amplitude des déplacements.

Lorsque la nouvelle position $pos''_{obj,temp}$ est obtenue, il faut limiter le déplacement de l'objet par rapport à sa position d'origine. Pour cela nous utilisons le calcul suivant pour obtenir la position finale pos''_{obj} de l'objet :

$$pos''_{obj} = pos_{obj} + \frac{\min(\rho, \|\overrightarrow{pos_{obj}\ pos''_{obj,temp}}\|)}{\|\overrightarrow{pos_{obj}\ pos''_{obj,temp}}\|} \times \overrightarrow{pos_{obj}\ pos''_{obj,temp}} \tag{4.9}$$

L'intégralité de cette démarche est illustrée par la figure 4.2(b) et elle peut être réitérée plusieurs fois en recalculant les matrices de visibilité à chaque passe afin de pouvoir évaluer l'évolution de l'occultation au niveau global de la tuile.

Nous pouvons remarquer qu'un objet peut potentiellement se retrouver coincé dans une position. En effet, si la position courante pos'_{obj} a été limitée (par l'équation 4.9)

au cercle $\mathscr{C}$ de centre pos_{obj} et de rayon ρ ($pos'_{obj} \in \mathscr{C}$) lors d'un précédent déplacement et qu'une nouvelle position $pos''_{obj,temp}$ calculée est située dans le demi-espace extérieur et tangent au cercle de rayon ρ en pos'_{obj}, le déplacement sera nul. Ce cas n'est tout de même pas récurrent du fait que tous les objets d'une même tuile sont déplacés en même temps et l'occultation locale d'un objet potentiellement bloqué varie énormément. Les positions pos_{vis} et pos_{occ} varient donc beaucoup et elles font également varier $pos''_{obj,temp}$.

4.1.2 Approche stochastique : déplacement d'objets par Metropolis

Afin de trouver les déplacements optimaux des objets de la tuile, nous avons choisi de mettre en place un algorithme de recherche stochastique utilisant des chaînes de Markov. Un processus de Markov est une séquence $X_0, ..., X_n$ de variables aléatoires déterminant chacune un état du système. La valeur X_e est l'état du processus à l'instant e et possède une probabilité $\mathscr{P}(X_{e+1} = j \mid X_e = i)$ de transition de l'état i à l'état j. On note ce nombre $\mathscr{P}_{i,j}$. Dans notre cas, chaque état X_e de la chaîne de Markov représente les positions des objets d'une tuile : $X_e = \{pos'_{obj_1}, pos'_{obj_2}, pos'_{obj_3}, ..., pos'_{obj_p}\}$.

A chaque itération, une chaîne est choisie aléatoirement et un changement d'état est effectué par une translation aléatoire de chacun des objets de la tuile. Chaque translation est réalisée dans la limite de déplacement autorisée par le paramètre de distance maximale ρ par rapport à sa position d'origine. Après chaque étape de déplacement, le score de visibilité est recalculé afin d'évaluer l'amélioration possible de l'occultation. Ce processus est illustré à la figure 4.3 pour n chaînes de Markov. Les chaînes sont développées indépendamment les unes des autres.

Lorsque l'on doit changer de chaîne de Markov, chacune d'entre elles se voit affecter une probabilité d'être choisie en fonction de son meilleur score d'occultation. Ainsi, le choix de la chaîne de Markov courante à développer est fonction d'une densité de probabilité (PDF), de manière similaire à l'approche Metropolis. Pour mettre en oeuvre cette dernière, la probabilité de choisir une chaîne de Markov doit être inversement proportionnelle à son plus bas score de visibilité (ou proportionnelle au meilleur taux d'occultation). Cette approche permet de favoriser les meilleurs candidats, sans toutefois négliger

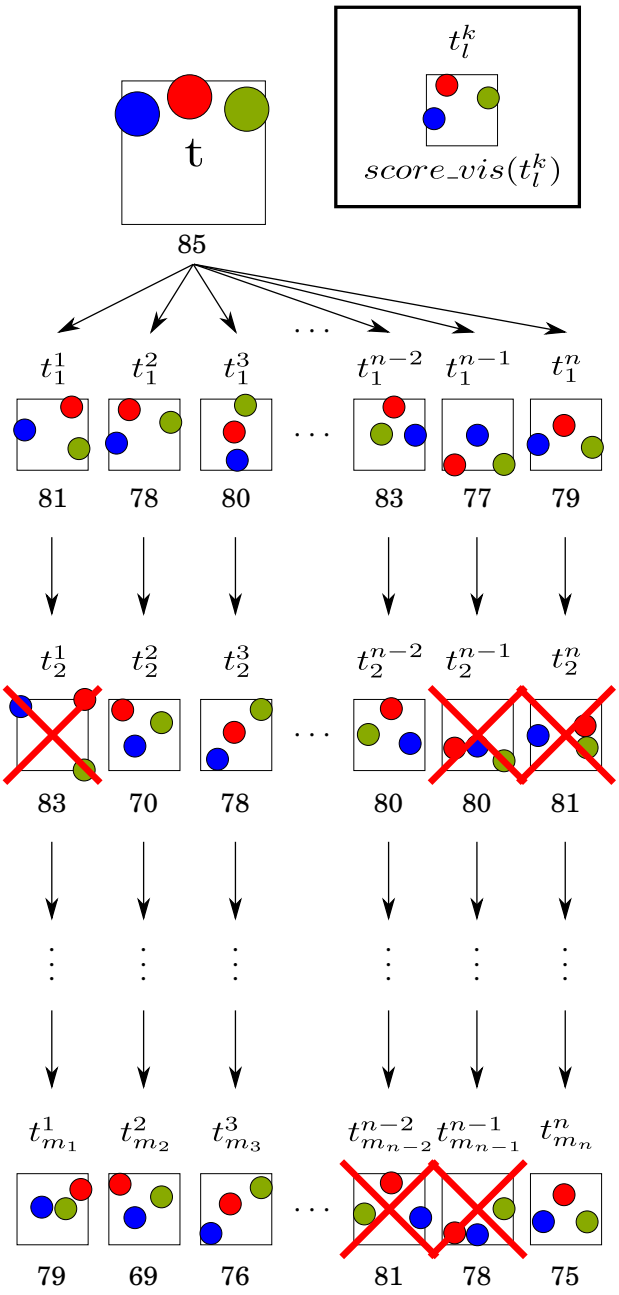


Figure 4.3 – Le développement de n chaînes de Markov pour une tuile après $\sum_{i=1}^{n} m_i$ itérations. Chaque chaîne peut avoir une longueur différente selon le nombre de fois où elle aura été choisie et développée.

les moins prometteurs.

Lorsque le score de visibilité est plus petit que celui du dernier état validé dans la chaîne de Markov courante (i.e., plus occultant), alors le nouvel état est validé. Dans ce cas-là, si le gain d'occultation est supérieur à un certain seuil η (fixé par l'utilisateur) alors on continue à développer la même chaîne à l'itération suivante. Si le gain est inférieur au seuil, alors on change de chaîne pour l'itération suivante selon la PDF.

Lorsque le score de visibilité est supérieur (moins occultant), alors l'état est invalidé, la chaîne courante retourne à l'état précédent, et on change de chaîne pour l'itération suivante selon la PDF.

Le nombre d'itérations peut être fixé à l'avance, ceci permettant d'estimer le temps d'exécution global de l'algorithme d'optimisation. Dans ce cas-là, on ne peut garantir que l'algorithme ait trouvé un extremum (même local). Pour résoudre ce problème, on peut établir un contrôle de la terminaison de l'algorithme par un mécanisme de roulette russe paramétré par un seuil évolutif en fonction du meilleur taux d'occultation plus l'occultation est grande, plus l'algorithme aura de probabilité de se terminer. Cette heuristique ne permet toutefois pas d'estimer le temps d'exécution de l'algorithme, ni d'assurer un extremum global, mais assure que chacune des n chaînes de Markov retournera un extremum local.

4.1.3 Evaluation du gain d'occultation

Afin d'évaluer le gain d'occultation de la tuile nous utilisons le score de visibilité suivant, qui est un score global ou directionnel évalué à partir des matrices de visibilité de la tuile (section 3.2.1.1). Pour cela, nous comptons le nombre de *gridels* non occultés

$$score\_vis(t) = \sum_{g_{in} \in G} \sum_{g_{out} \in G} visible(g_{in}, g_{out}) \tag{4.10}$$

où G est l'ensemble des *gridels* de la tuile t et $visible(g, g')$ la fonction retournant 1 si le *gridel* g est visible depuis g', 0 sinon.

4.1.3.1 Directionnalité de l'occultation et facteur de forme

Lorsque le déplacement de l'observateur dans la scène est connu d'avance, il est préférable de n'optimiser les positions des bloqueurs que depuis cet ensemble de points de vue donnés. En effet, notre algorithme de base échantillonne équitablement chacune des faces du cube unitaire représentant la tuile. Or, certains points de vue ne seront sans doute jamais utilisés dans le rendu final de l'application. Pour ainsi dire, ils ne contribuent pas au rendu final. Pire, il peuvent même surcontraindre le système d'optimisation du positionnement des objets et défavoriser le rendu depuis d'autres points de vue qui, eux, sont utilisés.

Il est donc primordial, dès lors que l'on sait où seront situés la plupart des points de vue, d'établir une directionnalité lors de l'évaluation des descripteurs de visibilité. Cette étape peut être aisément réalisée en ne sommant aux compteurs de visibilité et d'occultation (respectivement au score de visibilité) que les voxels traversés (respectivement *gridels* visibles) depuis les points de vue connus.

Cette étape permet de préparer l'ensemble de pavage à une application particulière. Par exemple, pour le rendu d'un circuit automobile en environnement urbain, les zones de passage des voitures sont restreintes. Cela permet d'optimiser la position des bâtiments en fonction des différents points de vue possibles car ces derniers sont bornés.

Afin que l'estimation de l'occultation soit plus précise, il est approprié d'affiner la mesure de la visibilité d'un *gridel*. En effet, la surface visible d'un *gridel* n'est pas toujours la même, selon son orientation. D'autre part, plus un *gridel* est éloigné, plus il restreint la pyramide de visibilité qu'il délimite. Pour cela, le score des *gridels* visibles doit aussi être pondéré par le "facteur de forme".

Ainsi, le score de visibilité permettant d'estimer le gain d'occultation devient :

$$score\_vis(t) = \sum_{g_{in} \in G} \sum_{g_{out} \in G} visible(g_{in}, g_{out}) \times D(g_{in}, g_{out}) \times F(g_{in}, g_{out}) \qquad (4.11)$$

où G est l'ensemble des *gridels* de la tuile t, $visible(g, g')$ la fonction retournant 1 si le *gridel* g est visible depuis g', 0 sinon, $F(g, g')$ est la fonction de facteur de forme entre les

gridels g et g'. $D(g,g')$ est la fonction de directionnalité. Celle-ci donne une pondération à la direction courante testée $\overrightarrow{gg'}$ en fonction de la direction d'optimisation. Pour nos tests, nous utilisons une fonction retournant la valeur absolue du produit scalaire entre $\overrightarrow{gg'}$ et la direction d'optimisation, mais il est également possible d'utiliser une fonction gaussienne, $cos^n(x)$, ou tout autre expression de la direction dans laquelle on souhaite maximiser l'occultation.

4.1.3.2 Recuit simulé

Afin que la probabilité de trouver les meilleures positions des objets sur une tuile soit plus grande, l'utilisation du recuit simulé permet d'affiner la recherche de positions intermédiaires. Dans notre cas, le coefficient σ de recuit simulé est calculé en fonction du nombre d'itérations valides appliquées au modèle initial pour arriver à l'état courant.

On a donc, pour la méthode déterministe :

$$\sigma = \frac{1}{j} \tag{4.12}$$

où j est le nombre courant d'itérations ayant apporté un gain d'occultation.

Et pour la méthode stochastique :

$$\sigma = \frac{1}{m_j} \tag{4.13}$$

où m_j est le nombre d'états transitoires valides dans la chaîne de Markov courante.

4.1.4 Résultats et comparaison des deux méthodes

Nous avons réalisé des tests de performance de rendu de la scène de ville après optimisation sur un CPU Intel® Core™ i7-930 (QuadCore) 2.8 GHz, 16 Go de mémoire vive, et une carte graphique GeForce GTX 680/PCIe/SSE2. Ces tests comparatifs ont été réalisés pour chacune des deux méthodes, déterministe et stochastique, et sont détaillés dans les deux sections suivantes. Pour mettre en évidence le gain de performance lié au déplacement des objets, les tuiles instanciées "sur" le trajet de la caméra ne contiennent

pas d'objets afin qu'un objet déplacé qui intersecterait ce trajet n'affecte pas les mesures de fréquence d'affichage. Pour la scène non optimisée, servant de référence, la fréquence d'affichage moyenne est de 67.36 FPS avec un écart type de 14.83.

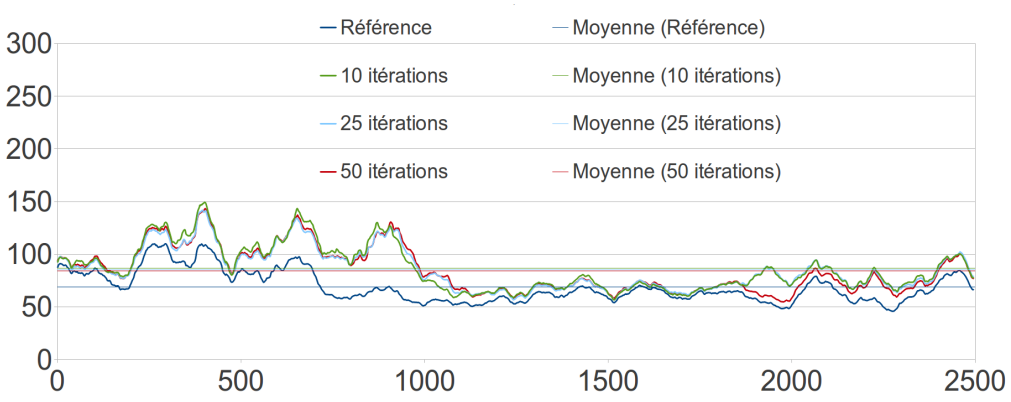
4.1.4.1 Méthode déterministe

Ces tests ont été réalisés pour des valeurs de $\alpha = 0.5$, $\gamma = 1$, $\rho = 0.3$ et un nombre d'itérations variable. La figure 4.4 montre l'évolution, en FPS, des scènes modifiées par optimisation du placement d'objets dont les champs de densité de visibilité ont été calculés à partir de matrices de visibilité de résolution 8×8. Les tableaux 4.1 et 4.2 montrent les pourcentages de gains d'occultation par tuile et moyens, les fréquences d'affichage moyennnes et l'écart type de la scène modifiée. La première constatation est que chaque optimisation permet d'améliorer les performances d'environ 10 FPS par rapport aux performances de la scène non optimisée. Nous pouvons constater qu'il est difficile de noter une augmentation significative des performances au-delà de 10 itérations d'optimisation, et ce, même en relâchant la contrainte de distance maximale de déplacement ρ et nous pouvons donc en déduire que plus le nombre d'itérations devient grand, moins le gain moyen d'occultation s'améliore, ce qui permet de supposer la convergence de l'algorihtme vers un optimum local.

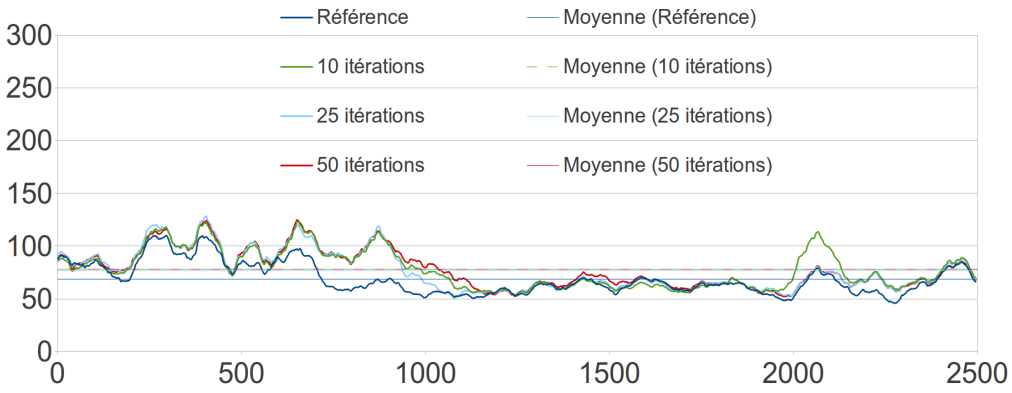
4.1.4.2 Méthode stochastique

Nous avons réalisé des tests de performance de rendu comparatifs pour des valeurs de ρ, n (nombre de chaînes de Markov) et un nombre d'itérations variable. Les figures 4.5 et 4.6, ainsi que le tableau 4.3 montrent les gains, en fréquence d'affichage, des scènes modifiées par optimisation du placement d'objets dont les scores de visibilité ont été calculés à partir de matrices de visibilité de résolution 8×8.

Nous pouvons constater qu'il y a une nette amélioration des performances dès lors que les objets ont été déplacés par rapport à la méthode déterministe. Cette amélioration est d'autant plus notable lorsque l'on contraint la directionnalité. On peut aussi remarquer une amélioration notable des performances lorsque le nombre de chaînes de Mar-



(a) $\rho = 0.3$



(b) $\rho = 0.5$

Figure 4.4 – Incidence de la variation du nombre d'itérations d'optimisation déterministe de l'occultation sur la fréquence d'affichage (en FPS) pour la scène de ville. La courbe la plus haute est la meilleure.

Score de visibilité initial	10 itérations	Gain %	25 itérations	Gain %	50 itérations	Gain %
4881.96	4881.96	0	4881.96	0	4881.96	0
6423.62	6404.22	0.30	6401.3	0.34	6397.16	0.41
4241.38	4241.38	0	4241.38	0	4241.38	0
6069.41	6069.41	0	6069.41	0	6069.41	0
5912.12	5912.12	0	5912.12	0	5912.12	0
5540.13	5540.13	0	5540.13	0	5540.13	0
5847.06	5815.49	0.53	5804.03	0.73	5798.74	0.82
6745.49	6637.88	1.59	6590.5	2.29	6587.31	2.34
5979.21	5979.21	0	5979.21	0	5979.21	0
6296.76	6296.76	0	6296.76	0	6296.76	0
5931.39	5931.39	0	5931.39	0	5931.39	0
6228.02	6106.08	1.95	6106.08	1.95	6106.08	1.95
5213.09	5213.09	0	5213.09	0	5213.09	0
5281.9	5281.9	0	5281.9	0	5281.9	0
6458.65	6458.65	0	6458.65	0	6458.65	0
6057.14	6057.14	0	6057.14	0	6057.14	0
6729.63	6729.63	0	6729.63	0	6729.63	0
6346.98	6346.98	0	6346.98	0	6346.98	0
6466.94	6174.12	4.52	6174.12	4.52	6174.12	4.52
6455.74	6455.74	0	6455.74	0	6455.74	0
Gain moyen		0.44		0.49		0.50
FPS moyen / Ecart type	84.26	21.32	83.32	19.47	81.95	20.95

Tableau 4.1 – Les statistiques de gain d'occultation et de performance pour les 20 tuiles de la scène de ville après optimisation déterministe ($\rho = 0.3$).

Score de visibilité initial	10 itérations	Gain %	25 itérations	Gain %	50 itérations	Gain %
4881.96	4881.96	0	4881.96	0	4881.96	0
6423.62	6404.06	0.30	6398.32	0.39	6397.54	0.40
4241.38	4241.38	0	4241.38	0	4241.38	0
6069.41	6069.41	0	6069.41	0	6069.41	0
5912.12	5912.12	0	5912.12	0	5912.12	0
5540.13	5540.13	0	5540.13	0	5540.13	0
5847.06	5814.41	0.55	5802.67	0.75	5800.75	0.79
6745.49	6575.3	2.52	6575.3	2.52	6575.3	2.52
5979.21	5979.21	0	5979.21	0	5979.21	0
6296.76	6296.76	0	6296.76	0	6296.76	0
5931.39	5931.39	0	5931.39	0	5931.39	0
6228.02	6033.83	3.11	6033.83	3.11	6033.83	3.11
5213.09	5213.09	0	5213.09	0	5213.09	0
5281.9	5281.9	0	5281.9	0	5281.9	0
6458.65	6458.65	0	6458.65	0	6458.65	0
6057.14	6057.14	0	6057.14	0	6057.14	0
6729.63	6445.16	4.22	6445.16	4.22	6445.16	4.22
6346.98	6346.98	0	6346.98	0	6346.98	0
6466.94	6241.76	3.484	6241.76	3.48	6241.76	3.48
6455.74	6455.74	0	6455.74	0	6455.74	0
Gain moyen		0.71		0.72		0.72
FPS moyen / Ecart type	76.01	18.16	74.63	18.70	76.15	17.71

Tableau 4.2 – Les statistiques de gain d'occultation et de performance pour les 20 tuiles de la scène de ville après optimisation déterministe ($\rho = 0.5$).

kov devient grand, mais pas nécessairement lorsque le nombre d'itérations augmente. Il est donc plus important d'accorder une variabilité importante aux premières dérivations d'une tuile, plutôt que de chercher à optimiser plus finement une tuile.

La figure 4.7 et le tableau 4.4 montrent comment l'augmentation de la résolution d'échantillonnage, la prise en compte de la direction principale de vue, ainsi que l'augmentation de la distance maximale de déplacement permettent d'améliorer les performances de rendu.

Nous pouvons remarquer une augmentation significative des performances de rendu dès lors que la résolution d'échantillonnage dépasse 32×32. En effet, lorsque la résolution augmente, le score de visibilité est plus précis et devient un indicateur beaucoup plus intéressant afin de guider la recherche stochastique de positions optimales d'objets. Ce score augmente avec la résolution.

Lorsque la direction principale de vue est prise en compte, nous pouvons également observer un gain significatif des performances de rendu. Pour ce cas-là, il devient délicat de déduire une corrélation directe avec le gain moyen d'occultation par rapport au score de visibilité initial car la prise en compte de la direction principale de vue prend part au calcul de ce score.

Lorsque la distance maximale de déplacement augmente, nous pouvons observer une augmentation potentielle du gain d'occultation. Toutefois, au-delà d'un certain seuil, ce gain n'est pas significatif et peut être trompeur. En effet, la recherche aléatoire peut rendre incertain le taux d'occultation.

4.1.4.3 Comparaison

La figure 4.8 compare les méthodes déterministe et stochastique. Lorsque le nombre d'itérations est petit, il est préférable d'utiliser la méthode déterministe, mais dès lors que le nombre d'itérations augmente, la méthode stochastique devient beaucoup plus intéressante en termes de gain moyen de taux d'occultation (2.01% contre 0.50% de gain d'occultation) et de performances de rendu (82.67% contre 81.95 FPS). Cela est dû au fait que la probabilité que la méthode stochastique reste bloquée dans un optimum local est moindre. De plus, la méthode déterministe converge beaucoup plus rapidement

et c'est cela qui explique de meilleures performances lorsque le nombre d'itérations est réduit. Une approche combinant les deux méthodes améliorera alors la vitesse de convergence.

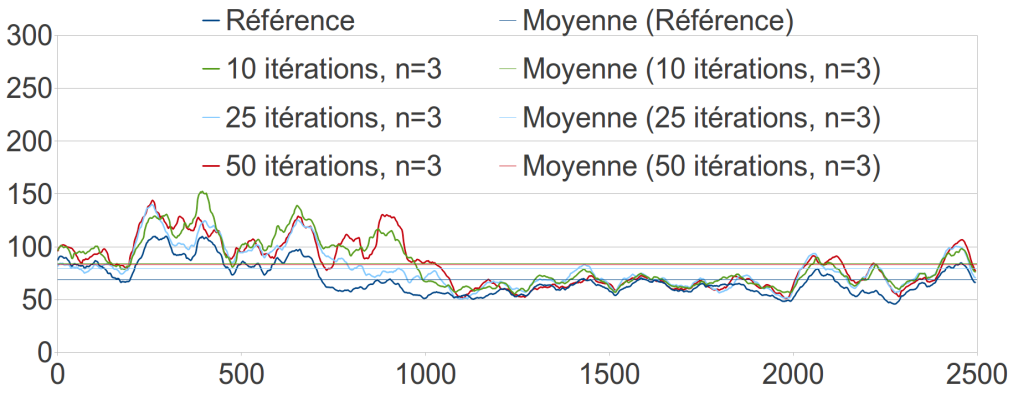
4.1.5 Autres scènes

Nous avons réalisé des tests d'optimisation pour l'ensemble des tuiles de forêt. La figure 4.9 et le tableau 4.5 montrent les performances de rendu d'un *walkthrough* similaire à celui de la ville dans la scène de forêt. La fréquence d'affichage de cette scène non optimisée est de 14.31 FPS avec un écart type de 14.61. Nous pouvons constater que les performances ne sont pas améliorées après optimisation. Cette observation s'explique par le fait que les polygones de cette scène sont beaucoup plus petits que ceux de la scène de ville. Afin de pouvoir assurer de manière significative de meilleures performances, il est nécessaire d'augmenter la résolution d'échantillonnage, ce qui implique des temps de calcul beaucoup plus longs. Ces méthodes d'optimisation de placement sont donc peu adaptées pour des scènes ne contenant que des polygones de petite taille.

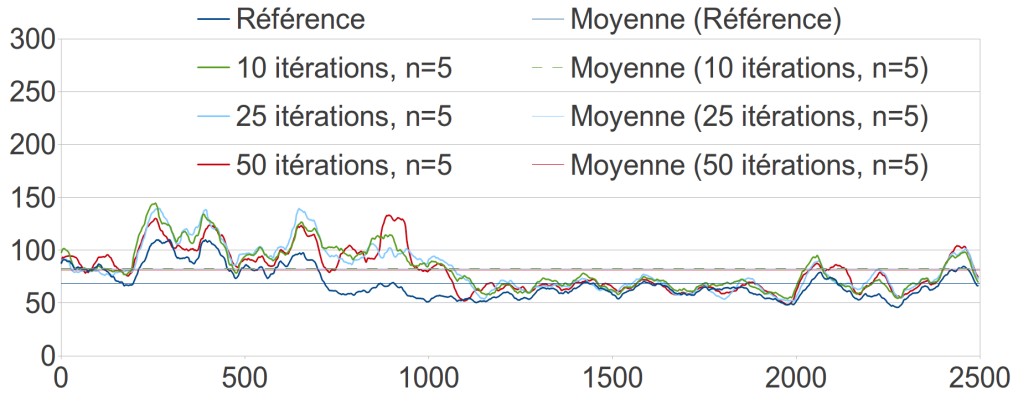
4.1.6 Conclusion

Dans cette section, nous avons introduit deux méthodes permettant d'accélérer le rendu par optimisation du placement des objets sur les tuiles. Ces deux méthodes permettent chacune d'améliorer les performances de rendu d'une scène générée à la volée à partir de tuiles dont le placement des objets sur chacune d'elles a été optimisé. Nous montrons également comment paramétrer cette optimisation de manière à tenir compte de la direction principale de vue de l'observateur, mais également pour respecter le plus possible les positions initiales des instances d'objets sur les tuiles de base.

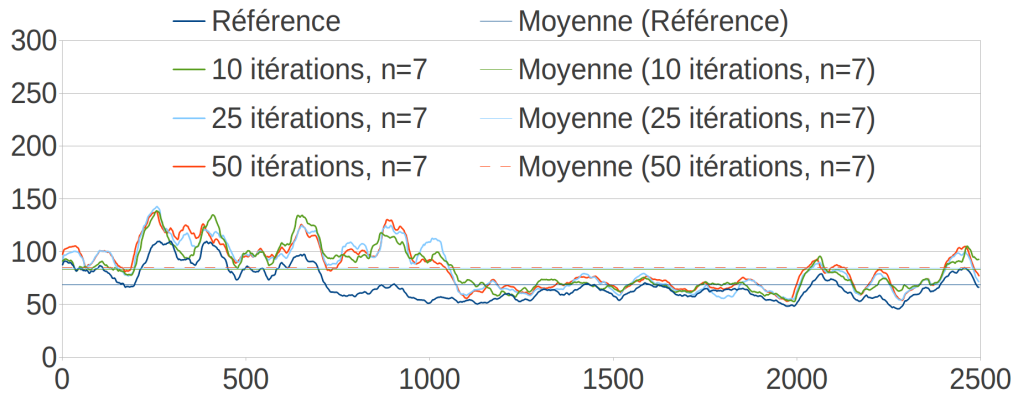
Il serait envisageable d'étudier l'occultation individuelle de chaque objet de manière indépendante et d'en déduire l'agencement optimal d'un sous-groupe d'objets sur la tuile. Il serait toutefois difficile de paramétrer la distance maximale de chaque objet par rapport à sa position initale.



(a) $\rho = 0.3, n = 3$

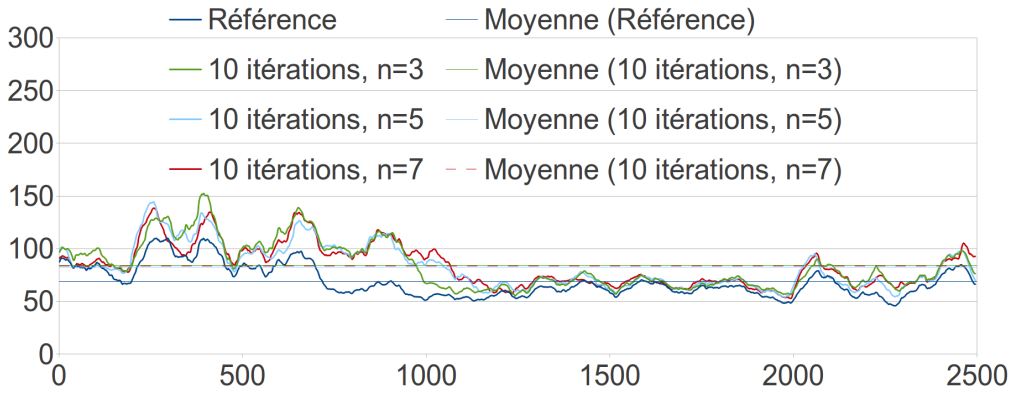


(b) $\rho = 0.3, n = 5$

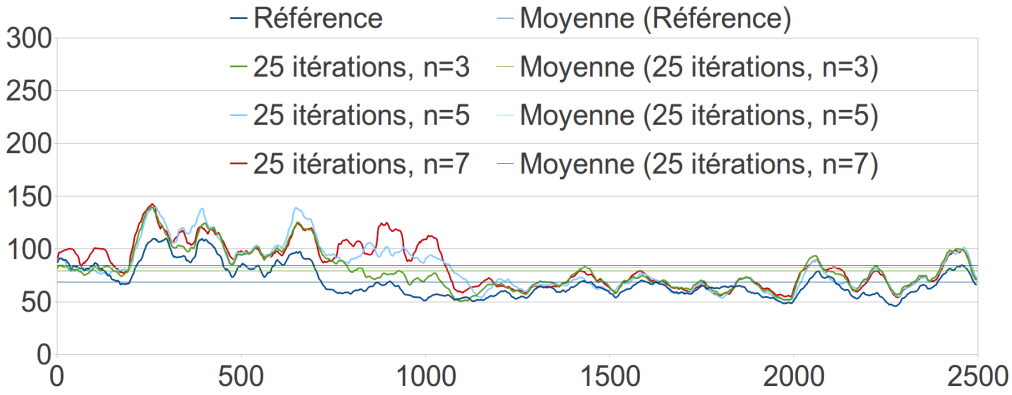


(c) $\rho = 0.3, n = 7$

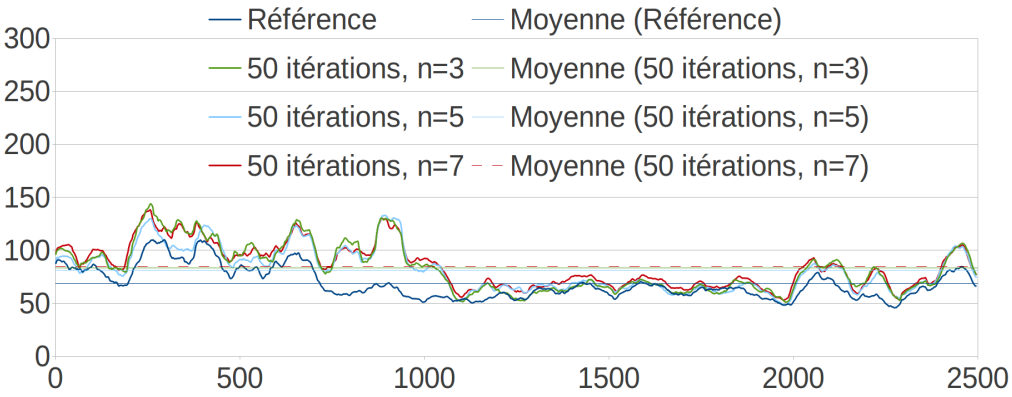
Figure 4.5 – Incidence de la variation du nombre d'itérations d'optimisation stochastique de l'occultation sur la fréquence d'affichage (en FPS) pour la scène de ville.



(a) 10 itérations, $\rho = 0.3$

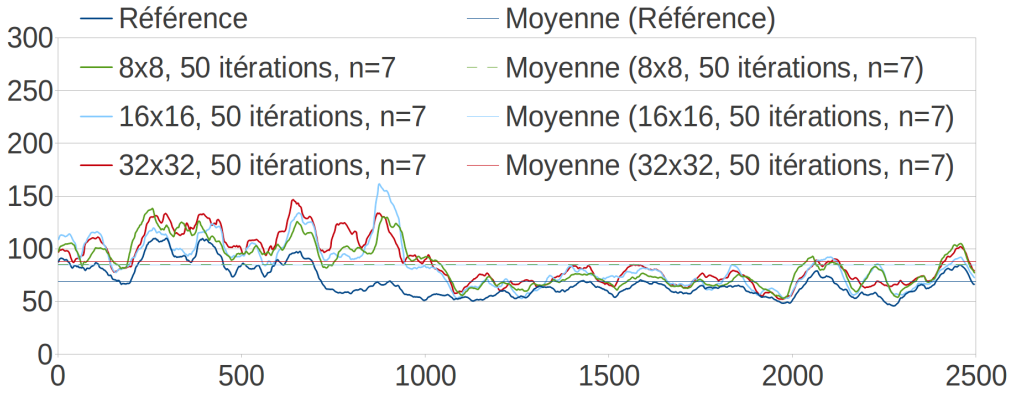


(b) 25 itérations, $\rho = 0.3$

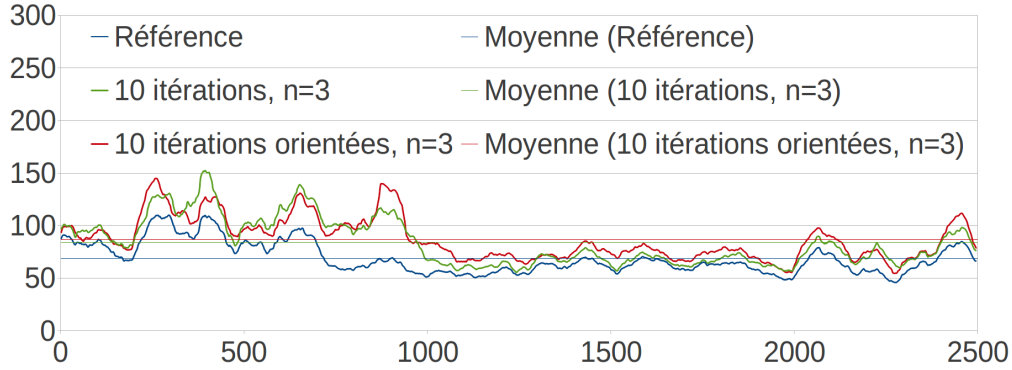


(c) 50 itérations, $\rho = 0.3$

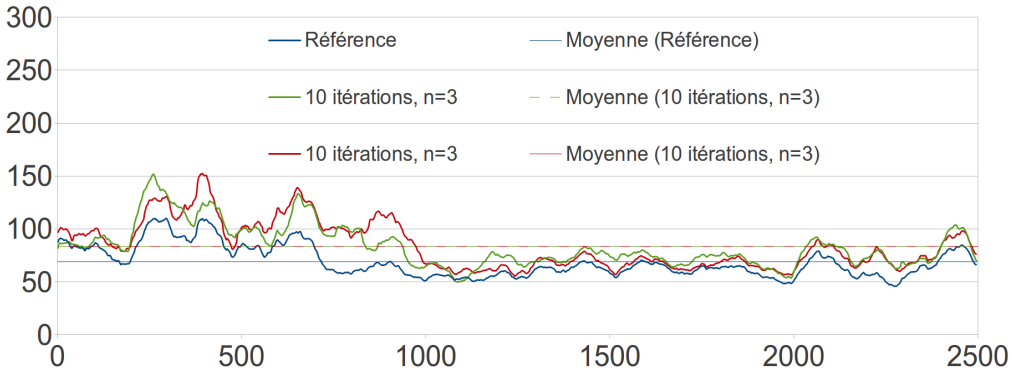
Figure 4.6 – Incidence de la variation du nombre de chaînes de Markov sur l'optimisation stochastique de l'occultation sur la fréquence d'affichage (en FPS) pour la scène de ville.



(a) Comparaison de différentes résolutions d'échantillonnage.



(b) Amélioration des performances par l'orientation de la direction principale de vue.



(c) Augmentation de la distance maximale de déplacement. En vert $\rho = 0.3$, en rouge $\rho = 0.5$.

Figure 4.7 – Incidence de la variation de la densité d'échantillonnage, de la prise en compte de la direction de vue et de la distance maximale de déplacement des objets dans l'optimisation stochastique de l'occultation sur la fréquence d'affichage (en FPS) pour la scène de ville.

Score de visibilité initial	10 itérations, $n = 3$	Gain %	10 itérations, $n = 5$	Gain %	10 itérations, $n = 7$	Gain %
4881.96	4881.96	0	4881.96	0	4881.96	0
6423.62	6423.62	0	6423.62	0	6423.62	0
4241.38	4241.38	0	4241.38	0	4241.38	0
6069.41	5908.68	2.64	5908.68	2.64	5881.22	3.10
5912.12	5687.13	3.80	5687.13	3.80	5687.13	3.80
5540.13	5540.13	0	5540.13	0	5540.13	0
5847.06	5718.58	2.19	5736.48	1.89	5736.48	1.89
6745.49	6514.04	3.43	6558.85	2.76	6542.79	3.00
5979.21	5872.2	1.78	5852.9	2.11	5872.2	1.78
6296.76	6296.76	0	6296.76	0	6296.76	0
5931.39	5874.87	0.95	5864.78	1.12 2	5864.78	1.12
6228.02	5690	8.63	5687.89	8.67	5685.19	8.71
5213.09	5213.09	0	5213.09	0	5213.09	0
5281.9	5281.9	0	5281.9	0	5281.9	0
6458.65	6372.92	1.32	6372.92	1.32	6372.92	1.32
6057.14	6057.14	0	6057.14	0	6057.14	0
6729.63	6729.63	0	6729.63	0	6729.63	0
6346.98	6313.24	0.53	6313.24	0.53	6313.24	0.53
6466.94	6077.01	6.02	6104.43	5.60	6034.67	6.68
6455.74	6455.74	0	6455.74	0	6455.74	0
Gain moyen		1.56		1.52		1.59
FPS moyen / Ecart type	81.26	22.10	80.47	20.28	81.40	19.71
Score de visibilité initial	25 itérations, $n = 3$	Gain %	25 itérations, $n = 5$	Gain %	25 itérations, $n = 7$	Gain %
4881.96	4881.96	0	4881.96	0	4881.96	0
6423.62	6421.4	0.03	6421.4	0.03	6421.4	0.03
4241.38	4241.38	0	4241.38	0	4241.38	0
6069.41	5947.91	2.00	5896.7	2.84	5896.7	2.84
5912.12	5678.04	3.95	5652.88	4.38	5672.83	4.04
5540.13	5540.13	0	5540.13	0	5540.13	0
5847.06	5705.75	2.41	5680.99	2.84	5698.21	2.54
6745.49	6486.38	3.84	6512.2	3.45	6513.76	3.43
5979.21	5808.58	2.85	5808.58	2.85	5808.58	2.85
6296.76	6296.76	0	6296.76	0	6296.76	0
5931.39	5854.5	1.29	5854.5	1.29	5838.31	1.56
6228.02	5862.85	5.86	5705.95	8.38	5727.21	8.04
5213.09	5213.09	0	5213.09	0	5213.09	0
5281.9	5281.9	0	5281.9	0	5281.9	0
6458.65	6373.79	1.31	6373.79	1.31	6368.23	1.39
6057.14	6057.14	0	6057.14	0	6057.14	0
6729.63	6729.63	0	6729.63	0	6729.63	0
6346.98	6294.15	0.83	6294.15	0.83	6294.15	0.83
6466.94	6025.65	6.82	6004.07	7.15	6010.63	7.05
6455.74	6455.74	0	6455.74	0	6455.74	0
Gain moyen		1.56		1.77		1.73
FPS moyen / Ecart type	77.34	18.54	79.81	21.00	81.75	20.82
Score de visibilité initial	50 itérations, $n = 3$	Gain %	50 itérations, $n = 5$	Gain %	50 itérations, $n = 7$	Gain %
4881.96	4881.96	0	4881.96	0	4881.96	0
6423.62	6423.62	0	6423.62	0	6423.62	0
4241.38	4241.38	0	4241.38	0	4241.38	0
6069.41	5881.67	3.09	5870.96	3.26	5881.67	3.09
5912.12	5803.75	1.83	5638.11	4.63	5638.11	4.63
5540.13	5540.13	0	5540.13	0	5540.13	0
5847.06	5716.05	2.24	5504.79	5.85	5678.04	2.89
6745.49	6475.93	3.99	6492.06	3.75	6476.7	3.98
5979.21	5784.48	3.25	5749.95	3.83	5701.66	4.64
6296.76	6296.76	0	6296.76	0	6296.76	0
5931.39	5826.69	1.76	5825.66	1.78	5826.69	1.76
6228.02	5681.4	8.77	5711.57	8.29	5709.87	8.31
5213.09	5210.25	0.05	5210.25	0.05	5210.25	0.05
5281.9	5281.9	0	5281.9	0	5281.9	0
6458.65	6383.18	1.16	6391.53	1.03	6371.51	1.34
6057.14	6057.14	0	6057.14	0	6057.14	0
6729.63	6647.28	1.223	6638.76	1.35	6638.76	1.35
6346.98	6130.02	3.41	6239.14	1.69	6239.14	1.69
6466.94	6110.66	5.50	6067.28	6.18	6041.18	6.58
6455.74	6455.74	0	6455.74	0	6455.74	0
Gain moyen		1.81		2.08		2.01
FPS moyen / Ecart type	80.42	22.19	78.81	19.85	82.69	20.01

Tableau 4.3 – Les statistiques de gain d'occultation et de performance pour les 20 tuiles de la scène de ville après 10, 25 et 50 itérations d'optimisation stochastique ($\rho = 0.3$).

8×8			16×16			32×32		
Score initial	Optimisé	Gain %	Score initial	Optimisé	Gain %	Score initial	Optimisé	Gain %
4881.96	4881.96	0	72341.4	72341.4	0	1.11×10^6	1.11×10^6	0
6423.62	6423.62	0	98031.8	98031.8	0	1.53×10^6	1.51×10^6	0.92
4241.38	4241.38	0	64319.8	64319.8	0	0.98×10^6	0.98×10^6	0
6069.41	5881.67	3.09	93825.1	88080.4	6.12	1.44×10^6	1.35×10^6	6.17
5912.12	5638.11	4.63	88559.3	84308	4.8	1.37×10^6	1.32×10^6	4.05
5540.13	5540.13	0	82101.4	82101.4	0	1.25×10^6	1.25×10^6	0
5847.06	5678.04	2.89	89214.2	82650.7	7.35	1.37×10^6	1.27×10^6	6.87
6745.49	6476.7	3.98	102560	99427.4	3.05	1.59×10^6	1.54×10^6	2.63
5979.21	5701.66	4.64	88871.4	85704.6	3.56	1.36×10^6	1.29×10^6	5.54
6296.76	6296.76	0	97191.4	97191.4	0	1.52×10^6	1.52×10^6	0.19
5931.39	5826.69	1.76	90231.2	84941.7	5.86	1.38×10^6	1.32×10^6	4.73
6228.02	5709.87	8.31	92534.2	86560.1	6.45	1.42×10^6	1.33×10^6	6.27
5213.09	5210.25	0.05	77901.7	77901.7	0	1.20×10^6	1.20×10^6	0
5281.9	5281.9	0	80264.1	80264.1	0	1.24×10^6	1.24×10^6	0
6458.65	6371.51	1.34	98671.4	96647	2.05	1.52×10^6	1.49×10^6	1.84
6057.14	6057.14	0	90910.1	90910.1	0	1.39×10^6	1.39×10^6	0
6729.63	6638.76	1.35	102554	100316	2.18	1.59×10^6	1.56×10^6	2.13
6346.98	6239.14	1.69	95181.8	91844.9	3.50	1.47×10^6	1.41×10^6	4.14
6466.94	6041.18	6.58	97289.9	90835.3	6.63	1.50×10^6	1.37×10^6	8.86
6455.74	6455.74	0	95721.3	95721.3	0	1.48×10^6	1.48×10^6	0
Gain moyen		2.01			2.57			2.72
FPS moyen / Ecart type	80.26	20.84		81.99	21.49		85.34	22.12

non orienté			orienté		
Score de visibilité initial	10 itérations, $n=3$	Gain %	Score de visibilité initial	10 itérations, $n=3$	Gain %
4881.96	4881.96	0	2961.93	2961.93	0
6423.62	6423.62	0	3894.57	3894.57	0
4241.38	4241.38	0	2547.46	2547.46	0
6069.41	5908.68	2.64	3658.72	3541.91	3.19
5912.12	5687.13	3.80	3581.2	3415.12	4.63
5540.13	5540.13	0	3312.11	3312.11	0
5847.06	5718.58	2.19	3579.66	3381.9	5.52
6745.49	6514.04	3.43	4063.04	3925.33	3.38
5979.21	5872.2	1.78	3624.77	3574.39	1.38
6296.76	6296.76	0	3774.4	3774.4	0
5931.39	5874.87	0.95	3633.36	3629.87	0.09
6228.02	5690	8.63	3820.68	3711.1	2.86
5213.09	5213.09	0	3141.1	3141.1	0
5281.9	5281.9	0	3185.58	3185.58	0
6458.65	6372.92	1.32	3930.89	3895.62	0.89
6057.14	6057.14	0	3677.08	3677.08	0
6729.63	6729.63	0	4105.62	4105.62	0
6346.98	6313.24	0.53	3910.34	3910.34	0
6466.94	6077.01	6.02	3907.63	3738.65	4.32
6455.74	6455.74	0	3855.2	3855.2	0
Gain moyen		1.56	Gain moyen		1.31
FPS moyen / Ecart type	81.26	22.10	FPS moyen / Ecart type	91.04	20.53

Score de visibilité initial	$n=3$, $\rho=0.3$	Gain %	$n=3$, $\rho=0.5$	Gain %
4881.96	4881.96	0	4881.96	0
6423.62	6423.62	0	6423.62	0
4241.38	4241.38	0	4241.38	0
6069.41	5908.68	2.64	6069.41	0
5912.12	5687.13	3.80	5767.76	2.44
5540.13	5540.13	0	5540.13	0
5847.06	5718.58	2.19	5590.71	4.38
6745.49	6514.04	3.43	6509.24	3.50
5979.21	5872.2	1.78	5607.26	6.22
6296.76	6296.76	0	6296.76	0
5931.39	5874.87	0.95	5931.39	0
6228.02	5690	8.63	5974.51	4.07
5213.09	5213.09	0	5213.09	0
5281.9	5281.9	0	5281.9	0
6458.65	6372.92	1.32	6413.16	0.70
6057.14	6057.14	0	6057.14	0
6729.63	6729.63	0	6457.8	4.03
6346.98	6313.24	0.53	6312.63	0.54
6466.94	6077.01	6.02	6178.87	4.45
6455.74	6455.74	0	6455.74	0
Gain moyen		1.56		1.51
FPS moyen / Ecart type	81.26	22.10	81.19	20.47

Tableau 4.4 – Les statistiques de gain d'occultation et de performance pour les 20 tuiles de la scène de ville après 50 itérations d'optimisation stochastique ($n=7$) pour différentes résolutions d'échantillonnage, après 10 itérations d'optimisation stochastique ($n=3$) avec prise en compte de la direction principale de vue, et après 10 itérations d'optimisation stochastique pour différentes valeurs de déplacement maximal.

Score initial	Optimisé	Gain %
1.40×10^6	1.40×10^6	0
1.56×10^6	1.56×10^6	0
1.46×10^6	1.46×10^6	0
1.45×10^6	1.45×10^6	0
1.46×10^6	1.46×10^6	0
1.44×10^6	1.39×10^6	2.90
1.18×10^6	1.13×10^6	4.50
1.40×10^6	1.40×10^6	0.10
1.14×10^6	1.04×10^6	7.98
0.96×10^6	0.96×10^6	0
Gain moyen		1.55
FPS moyen / Ecart type	12.13	1.77

Tableau 4.5 – Statistiques de gain d'occultation et de performance pour les 10 tuiles de la scène de forêt après 50 itérations d'optimisation stochastique pour une valeur de déplacement maximal de 0.3, 7 chaînes de Markov et une résolution d'échantillonnage de 32×32.

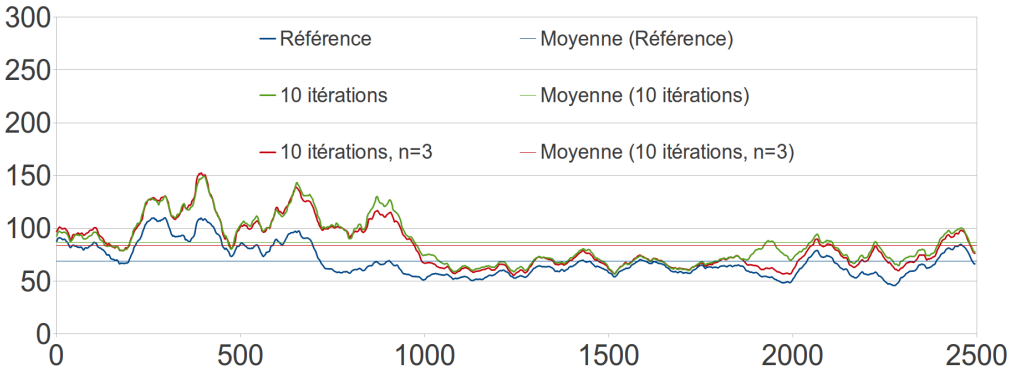
Ces méthodes permettent d'optimiser les positions de base des objets, mais il est aussi possible de faire varier les rotations des objets. Il est également envisageable de mettre en place des méthodes de génération procédurale d'objets prenant comme paramètres une élévation (étages d'édifices, montagnes), densité d'occultation (feuillage d'arbres), etc.

4.2 Discussion

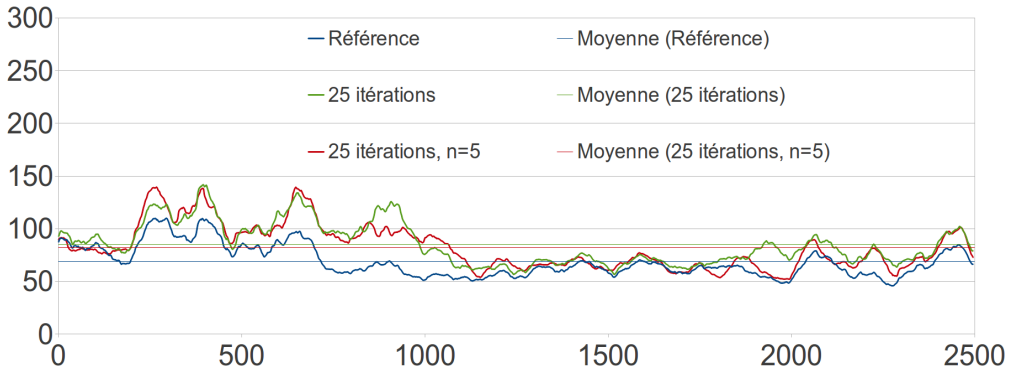
Dans cette section, nous présentons d'autres possibilités pour accélérer le rendu et discutons de leur mise en oeuvre.

4.2.1 Permutation occultante de tuiles

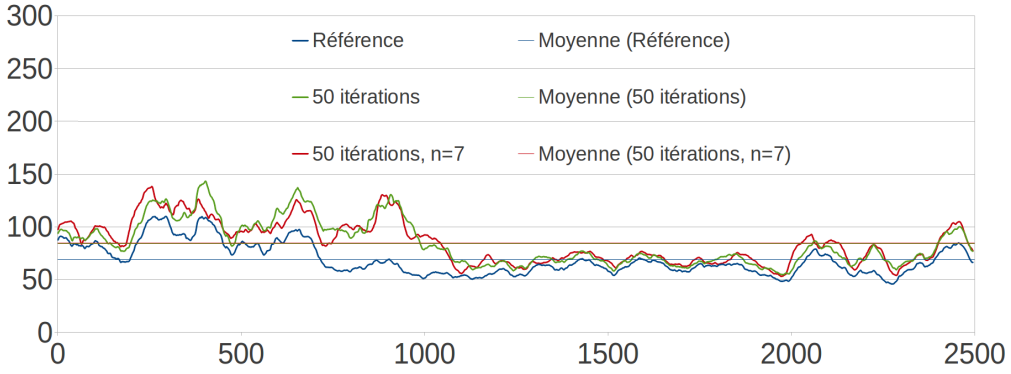
De manière similaire à nos premiers travaux, nous pourrions précalculer la visibilité au travers d'une séquence de tuiles afin de trouver plus rapidement la limite du PVS. Pour cela, une possibilité est d'étudier l'évolution du taux d'occultation au travers de différentes permutations de tuiles. Toutefois, nous constatons que le passage à l'échelle d'une telle opération devient coûteuse, même pour des ensembles de pavage de taille réduite. Pour donner un ordre d'idée, calculer l'ensemble des reprojections possibles pour



(a)



(b)



(c)

Figure 4.8 – Comparaison des deux méthodes d'optimisation de l'occultation en terme de fréquence d'affichage pour la scène de ville pour $\rho = 0.3$. En vert, la méthode déterministe. En rouge, la méthode stochastique.

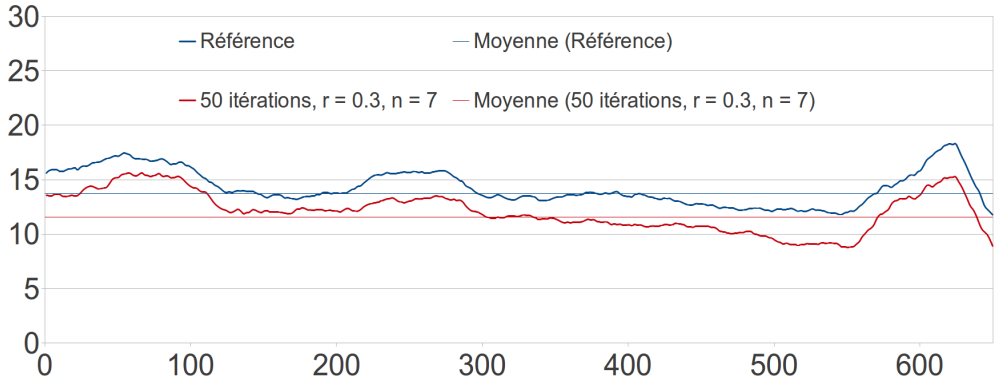


Figure 4.9 – Comparaison des performances de rendu de la scène de forêt avant et après optimisation.

l'ensemble des 20 tuiles de la scène de ville nécessiterait de calculer 400 reprojections dans chaque direction de pavage pour pouvoir bénéficier du précalcul d'une telle méthode, pour une propagation de visibilité de longueur 2, 8000 pour une propagation de longueur 3, etc. Cette méthode ne serait donc pas exploitable à des fins d'optimisation de placement de bloqueurs.

4.2.2 Ajout de bloqueurs

Dès lors qu'un ensemble de pavage de base est suffisamment occultant pour permettre des performances de rendu adéquates, il est toujours possible d'ajouter des bloqueurs lors de la construction de ces tuiles. En effet, de manière similaire à la scène hybride, il est toujours possible d'ajouter des objets typiques dans des scènes de villes (mobilier urbain, voitures, végétation, etc.). Ainsi, l'occultation générée par chaque tuile peut se voir accrue de manière considérable.

De même, ajouter des bloqueurs de manière interactive pourrait aider à arrêter la propagation de visibilité et limiter le plus possible le nombre de tuiles visibles. Pour cela, il faudrait assurer le stockage supplémentaire des objets sur la tuile, ainsi que le recalcul du BVH associé. Cela pose donc le problème de persistance versus mémoire car un objet supplémentaire sera spécifique à une instance particulière de tuile ou à un type de tuile.

4.2.3 Accélération par omission et groupement de requêtes d'occultation

Lorsque les objets présents sur les tuiles ne génèrent pas assez d'occultation, le nombre de tuiles instanciées devient grand, de même que le nombre de requêtes d'occultation envoyées à la carte graphique. Le but des optimisations de cette section est de diminuer ce nombre sans modifier pour autant la position des objets.

Le problème se situe au niveau de la détermination des tuiles visibles. Etant donné que notre algorithme se base sur une propagation de visibilité linéaire (de tuile en tuile), il est difficile de prédire l'étendue du pavage, et beaucoup de requêtes d'occultation intermédiaires sont générées inutilement.

Pour se faire une idée plus claire du problème, nous montrons à la figure 4.10 un pavage instancié par requêtes d'occultation. Ce pavage possède un grand nombre de tuiles instanciées, car les objets contenus dans chacune des tuiles sont petits, et leur positionnement ne procure pas suffisamment de blocage pour assurer une propagation minimaliste de la visibilité. On peut remarquer deux choses : l'étendue du pavage peut changer énormément d'une image à l'autre. De plus, seules les requêtes d'occultation des tuiles de bordure du pavage sont utiles pour déterminer l'étendue du pavage. D'autre part, plus le pavage est étendu, ce qui devient le cas lorsque les tuiles sont peu occultantes, plus le rapport entre le nombre de requêtes des tuiles de bordure et le nombre de tuiles instanciées devient petit. De manière duale, les requêtes d'occultation générées pour les tuiles à l'intérieur du pavage sont moins utiles car elles ne permettent de déterminer la visibilité des tuiles intermédiaires que lors d'un grand changement de visibilité.

De ces observations, nous pouvons tirer une idée clef qui permet l'accélération de la détermination des tuiles visibles en allégeant la charge du GPU par la diminution du nombre global de requêtes d'occultation : essayer le plus possible de générer des requêtes d'occultation utiles en déterminant la bordure du pavage le plus rapidement possible.

D'une image à la suivante, toutes les tuiles intermédiaires, i.e., comprises entre le point de vue et les tuiles de bordure déterminées à l'image précédente, sont rendues sans générer une seule requête d'occultation, tandis que les bordures du pavage visible sont mises à jour par requêtes d'occultation. Pour chaque tuile de bordure, une requête

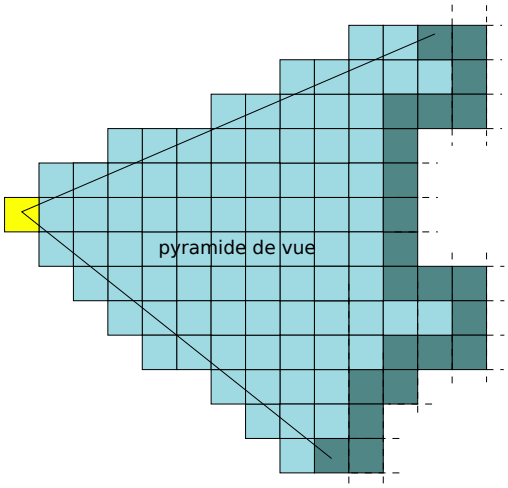


Figure 4.10 – Plus le pavage est étendu, plus il est probable que les tuiles situées entre le point de vue et les tuiles de bordure soient visibles dans les images suivantes. Les requêtes d'occultation de ces tuiles intermédiaires peuvent être économisées.

d'occultation est générée. Si le résultat de cette requête est positif, une seconde requête est générée pour les tuiles voisines suivantes, supposées non visibles (celles qui sont derrière la bordure). Si ces tuiles ne sont pas visibles, le contour n'évolue pas, sinon le contour est agrandi en conséquence, les nouvelles tuiles de bordure sont dessinées et le processus de construction de pavage suit son cours comme précédemment par propagation de visibilité. Si la requête de la tuile de bordure échoue, on réduit l'étendue du pavage et bien que cette tuile ait été affichée, le rendu de ces tuiles n'est pas superflu car il se comporte de manière similaire à l'effet de cohérence temporelle de la méthode de base. A partir de cette tuile dont la requête échoue, il s'agit maintenant de trouver la nouvelle tuile de bordure, plus proche du point de vue : nous procédons donc à une série de requêtes par propagation inverse jusqu'à ce que chacune des requêtes d'occultation des tuiles traversées réussisse ou rencontre une autre tuile de bordure.

Nous pouvons également remarquer qu'il y a beaucoup de requêtes dont la réponse est négative pour les tuiles non visibles situées juste derrière la bordure. La seconde idée est de grouper les requêtes prédites non visibles (celles situées au-delà de la bordure) par des requêtes combinées, et ainsi d'économiser le GPU par des *clusters* de requêtes. Ces requêtes peuvent être groupées par similarité : taille dans l'espace écran de la boîte

englobante commune, nombre de fois où cette tuile a été déterminée visible dans les images précédentes, nombre de pixels dessinés pour chaque image précédente, proximité d'une autre tuile prédite non visible, basée sur un échantillonnage de points de vue autour d'une permutation spécifique de tuiles (score d'occultation), etc.

Dans le cas où la visibilité change brusquement, e.g., un objet proche cache subitement une grande partie de la scène, le nombre de tuiles instanciées diminue fortement sur une partie du plan image. Il est envisageable de réaliser un traitement dichotomique des requêtes d'occultation afin de déterminer plus rapidement la nouvelle bordure.

CONCLUSION

Dans cette thèse, nous avons étudié plusieurs solutions pouvant offrir aux infographistes des outils de modélisation procédurale de mondes virtuels très étendus. En offrant de nouvelles solutions à la problématique de visibilité sur un pavage, nous permettons d'assurer de meilleures performances pour le rendu de telles scènes. Pour cela, nous mettons en place des outils de création semi-automatique basés sur un ensemble de tuiles prédéfinies. Nos algorithmes d'*occlusion culling* permettent, à partir de cet ensemble et d'une structure de données légère en mémoire, de réaliser à faible coût un traitement de la visibilité sur des mondes construits par instanciation massive d'objets.

Dans un premier temps, nous avons limité le problème au cas 2D, plus simple à résoudre. Ainsi, nous avons pu imposer une limite à la taille du pavage visible et borner le *framerate*. La mise en place d'une telle méthode a également permis de mettre en évidence un lien entre taux d'occultation et densité d'occupation d'une tuile mais aussi qu'en 3D notamment, la densité d'occupation doit être grandement augmentée pour garantir une occultation complète de manière identique à la 2D.

Dans un second temps, nous avons supprimé l'hypothèse de l'occultation complète au-delà du premier anneau de voisinage, permettant de relâcher les contraintes de modélisation des tuiles de la méthode précédente. Cette méthode met en place un mécanisme de propagation directionnelle de la visibilité et d'instanciation des tuiles visibles nécessaires à l'occultation du champ visuel.

Ensuite, nous avons étendu notre méthode de pavage à deux méthodes 3D, permettant le libre placement des objets dans les tuiles (même chevauchant les bordures). La première, précalculée, est basée sur l'échantillonnage et le calcul des reprojections de portails sur les bordures des tuiles. La seconde, en temps réel, est basée sur des requêtes d'occultation, l'instanciation de BVH à faible coût mémoire et l'utilisation découplée d'un algorithme d'*occlusion culling* par tuile.

Enfin, nos solutions de génération permettent d'assurer, dès sa création, que le monde virtuel sera affiché de manière efficace grâce à l'utilisation de plusieurs méthodes d'optimisation de l'occultation sur des tuiles préexistantes. Ainsi, nous avons introduit deux

méthodes de déplacement des objets, l'une déterministe, l'autre stochastique. Toutes deux utilisent un indicateur de mesure d'amélioration de l'occultation basé sur des matrices de visibilité. Ce déplacement peut être paramétré par une limite de déplacement, imposée par le créateur, sur les tuiles pour qu'elles soient plus occultantes. Pour chacune de nos méthodes, nous avons validé l'intérêt par des scènes représentatives et nous avons analysé les résultats et discuté de leurs implications.

La modélisation procédurale offre un panel d'outils permettant de créer des mondes complexes, variés et étendus. Le fait que la puissance des ordinateurs croît d'année en année laisse présager de l'utilisation prépondérante de ces méthodes pour la modélisation de scènes complexes et réalistes pour le jeu vidéo. Cependant, le traitement de la visibilité de telles scènes potentiellement infinies exige des solutions adaptées. Dans cette optique, nous avons introduit une nouvelle dimension à la génération de modèles procéduraux, impliquée directement dans l'optimisation d'un rendu subséquent et nous pensons que les futurs développements de telles méthodes nécessitent la prise en compte de cet aspect.

De nombreux travaux sont envisageables pour les développements futurs :

Changement de la forme des tuiles de base Les tuiles que nous utilisons pour paver le plan sont rectangulaires, car c'est un cas simple et général souvent utilisé dans les pavages actuels. De plus, des rectangles permettent une étude approfondie et un prototypage plus aisé. Toutefois, beaucoup d'autres types de pavages sont utilisables par nos méthodes à condition d'adapter les règles de visibilité : si les tuiles prennent une forme de parallélogramme, une simple transformation affine permet de ramener celui-ci à un rectangle et le schéma *ū-shape* peut être réutilisé (section 3.1.6) ; le schéma d'occultation *ū-shape* peut changer de forme selon la forme des tuiles utilisées ; les directions des vecteurs d'occultation et des matrices de visibilité peuvent être plus ou moins nombreuses, plus ou moins variées ; la boîte englobante commune peut toujours être calculée bien qu'elle devient plus conservatrice.

Extension à un pavage de l'espace 3D Il serait intéressant d'étendre notre méthode de pavage du plan au pavage d'une surface 2D non plane ou le pavage intégral de l'espace 3D. Nous pouvons citer par exemple la modélisation de cavernes, de villes ayant beaucoup de relief, de montagnes ou de champs d'astéroïdes, qui requerraient des règles de pavage additionnelles dans la troisième dimension et une adaptation de la recherche de voisinage. Il serait également possible de générer des milieux aquatiques et recréer l'occultation naturelle induite par l'opacité d'un volume d'eau contenant des particules.

Implémentation matérielle Bien qu'étant conservatrice, notre méthode d'instanciation de BVH à faible coût est robuste et très efficace lorsqu'il s'agit d'instancier de nombreux objets, c'est pourquoi une implémentation matérielle de cette dernière améliorerait grandement les performances de nos algorithmes d'*occlusion culling*.

Utilisation sous-jacente d'autres méthodes de génération procédurale Nous évoquons, au 1er chapitre, l'intégration potentielle de méthodes procédurales déjà existantes à notre système. En effet, le développement d'un logiciel de modélisation doit pouvoir offrir un panel complet d'outils procéduraux. Ainsi, un tel logiciel peut autant aider à la création massive d'objets qu'à leur positionnement.

Outils d'optimisation de l'occultation Le dernier chapitre de cette thèse ne fait que proposer des solutions simples à mettre en place pour tenter d'améliorer l'occultation procurée par l'instanciation d'un ensemble de pavage, mais il est envisageable de mettre en place des outils d'analyse de l'occultation beaucoup plus fins et intuitifs. Par exemple, il serait intéressant d'investiguer les espaces duaux de lignes, car les bloqueurs décrivent certaines courbes dans ces derniers et elles pourraient apporter d'autres indices sur la "qualité" du blocage d'une tuile (section 3.1.5.2), à savoir une information directionnelle sur l'occultation au niveau d'une cellule, permettant aux infographistes de modéliser de manière efficace des scènes riches et variées, tout en assurant de bonnes performances d'affichage.

Plusieurs échelles de représentation La plupart de nos méthodes ne sont valides qu'à un seul niveau de hiérarchie. Toutefois, du fait de l'utilisation d'un pavage rectangulaire, ces dernières ont un excellent potentiel d'extension à un traitement multi-échelles. Par exemple, un ensemble de tuiles peuvent être regroupées comme expliqué en section 4.2.3 afin de déterminer plus rapidement qu'elles sont occultées.

Scènes animées Basiquement, nous pouvons négliger l'occultation procurée par un objet qui se déplace – en effectuant tout de même le test de visibilité sur celui-ci. Toutefois, il peut être intéressant, dès lors qu'ils sont nombreux et volumineux, de prendre en compte l'occultation que procure un groupe d'objets mobiles.

Imposteurs et semi-transparence Pour des scènes très peu occultantes et/ou semi-transparentes, il est possible d'investiguer sur l'accumulation de l'occultation par l'utilisation de coefficients d'opacité, mais aussi l'utilisation d'imposteurs, pour n'afficher que les objets qui contribuent le plus à l'image finale, et "tricher" sur ceux qui contribuent le moins.

Persistance du pavage, tuiles de Wang, tuiles de coins et autres contraintes Un infographiste ou une méthode de génération procédurale doit être capable de "peupler" une tuile et ce, même sur les bords, afin de limiter la propagation de visibilité sur le pavage. La figure 4.11 montre une intersection entre deux polygones appartenant à deux tuiles différentes. Ce problème devient alors similaire à de l'échantillonnage de disques

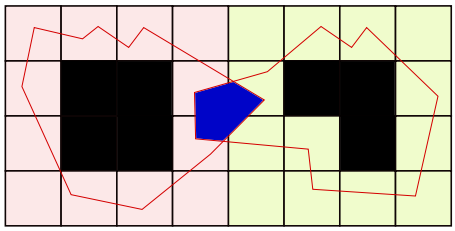


Figure 4.11 – La discontinuité entre les tuiles peut provoquer la collision des géométries sur la frontière commune entre deux tuiles. Ici, deux tuiles de résolution 4×4 voient leurs géométries s'intersecter dans la zone bleue.

de Poisson sur un pavage car il provoque des discontinuités au niveau des bordures et des coins des tuiles. Ce problème possède des solutions dans la littérature existante [52]. Par ailleurs, comme décrit en section 2.3, nous souhaitons utiliser une fonction de hashage pour le choix de la tuile à instancier en fonction de sa position dans la scène et ce, afin d'éviter de stocker les positions de chacune des instances de tuiles. Nous souhaitons pour cela que la première tuile instanciée (cellule de vue) puisse se situer à n'importe quel endroit de l'espace, pour ne pas avoir à revenir à l'origine et il serait donc envisageable d'utiliser les pavages récursifs de Wang [51]. Cependant, cette méthode semble difficile à appliquer si l'on souhaite préserver le potentiel d'un monde à pouvoir s'étendre à l'infini.

BIBLIOGRAPHIE

[1] Cgal, computational geometry algorithms library. 56

[2] Timo Aila et Ville Miettinen. dPVS : An Occlusion Culling System for Massive Dynamic Environments. *IEEE Computer Graphics and Applications*, 24(2):86–97, mars 2004. ISSN 0272-1716. 18, 23

[3] John M. Airey, John H. Rohlf et Frederick P. Jr. Brook. Towards Image Realism with Interactive Update Rates in Complex Virtual Building Environments. Dans *ACM Symposium on Interactive 3D Graphics*, pages 41–50, 258, 1990. 18

[4] Carlos Andújar, Isabel Navazo, Pere Brunet et Carlos Saona-Vasquez. Integrating Occlusion Culling and Levels of Detail through Hardly-Visible Sets. *EUROGRAPHICS*, 19(3), 2000. 22

[5] Masaki Aono et Tosiyasu L. Kunii. Botanical Tree Image Generation. *Computer Graphics and Applications, IEEE*, 4(5):10 –34, may 1984. ISSN 0272-1716. 6

[6] Marshall Bern, David Dobkin, David Eppstein et Robert Grossman. Visibility with a Moving Point of View. *Algorithmica*, 11(4):1–17, 1994. 18

[7] Jiri Bittner, Oliver Mattausch, Peter Wonka, Vlastimil Havran et Michael Wimmer. Adaptive Global Visibility Sampling. *ACM Transactions on Graphics*, 28 (3), 2009. 21

[8] Jiri Bittner et Peter Wonka. Visibility in Computer Graphics. *Journal of Environmental Planning*, 30:729–756, 2003. 12

[9] Algorithmic Botany. TreePad. URL `algorithmicbotany.org/TreePadForums`. 6

[10] Guoning Chen, Gregory Esch, Peter Wonka, Pascal Müller et Eugene Zhang. Interactive Procedural Street Modeling. Dans *ACM SIGGRAPH 2008 papers*, SIG-

GRAPH '08, pages 103 :1–103 :10. ACM, 2008. ISBN 978-1-4503-0112-1. URL http://doi.acm.org/10.1145/1399504.1360702. 5

[11] António Fernando Coelho, António Augusto de Sousa et Fernando Nunes Ferreira. Modelling urban scenes for lbms. Dans *Proceedings of the tenth international conference on 3D Web technology*, Web3D '05, pages 37–46, New York, NY, USA, 2005. ACM. ISBN 1-59593-012-4. URL http://doi.acm.org/10.1145/1050491.1050497. 7

[12] Michael F. Cohen, Jonathan Shade, Stefan Hiller et Oliver Deussen. Wang Tiles for Image and Texture Generation. *ACM Trans. Graph.*, 22(3):287–294, juillet 2003. ISSN 0730-0301. URL http://doi.acm.org/10.1145/882262.882265. 27

[13] Daniel Cohen-Or, Yiorgos L. Chrysanthou, Claudio T. Silva et Frédo Durand. A Survey of Visibility for Walkthrough Applications. *IEEE Trans. Visualization and Computer Graphics*, 9(3):412–431, 2003. 12

[14] Daniel Cohen-Or, Gadi Fibich, Dan Halperin et Eyal Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. *Computer Graphics Forum*, 17(3):243–253, august 1998. ISSN 0167-7055. 18, 19

[15] Satyan Coorg et Seth Teller. Temporally Coherent Conservative Visibility. *Comput. Geom. Theory Appl.*, 12:105–124, février 1999. ISSN 0925-7721. 16, 39

[16] Karel Culik et Jarkko Kari. An Aperiodic Set of Wang Cubes. Dans Claude Puech et Rüdiger Reischuk, éditeurs, *STACS 96*, volume 1046 de *Lecture Notes in Computer Science*, pages 137–146. Springer Berlin / Heidelberg, 1996. ISBN 978-3-540-60922-3. URL http://dx.doi.org/10.1007/3-540-60922-9_12. 10.1007/3-540-60922-9_12. 29

[17] Karel Culik, II. An Aperiodic Set of 13 Wang Tiles. *Discrete Mathematics*, 160:

245–251, November 1996. ISSN 0012-365X. URL http://dl.acm.org/citation.cfm?id=245761.245814. xxi, 28, 29

[18] Barbara Cutler, Julie Dorsey, Leonard McMillan, Matthias Müller et Robert Jagnow. A Procedural Approach to Authoring Solid Models. Dans *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '02, pages 302–311, New York, NY, USA, 2002. ACM. ISBN 1-58113-521-1. URL http://doi.acm.org/10.1145/566570.566581. xxi, 9, 10

[19] Robert Dawson. Aesthetic Sabotage, "Surface" tiling, janvier 2012. URL http://www.aestheticsabotage.com. 25

[20] Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomír Měch, Matt Pharr et Przemyslaw Prusinkiewicz. Realistic Modeling and Rendering of Plant Ecosystems. Dans *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '98, pages 275–286, New York, NY, USA, 1998. ACM. ISBN 0-89791-999-8. URL http://doi.acm.org/10.1145/280814.280898. 7

[21] Laura Downs, Tomas Möller et Carlo H. Séquin. Occlusion Horizons for Driving Through Urban Scenery. Dans *Proc. Symposium on Interactive 3D Graphics*, I3D '01, pages 121–124, 2001. ISBN 1-58113-292-1. 20

[22] Frédo Durand. Some Thoughts on Visibility. september 2000. 77

[23] Frédo Durand, George Drettakis, Joëlle Thollot et Claude Puech. Conservative Visibility Preprocessing Using Extended Projections. Dans *Proc. SIGGRAPH '00*, pages 239–248, 2000. 18, 19, 74, 83

[24] David. S. Ebert, F. Kenton Musgrave, Darwin Peachey, Ken Perlin et Steven Worley. *Texturing & Modeling : A Procedural Approach*. Morgan Kaufmann Publishers, Inc., 2002. 11

[25] Electronic Arts. Sim City Saga - 1989-2011. [Video Game]. 2, 31

[26] Dieter Finkenzeller, Jan Bender et Alfred Schmitt. Feature-based decomposition of façades. *Proceedings of Virtual Concept*, 2005. 5

[27] Deborah R. Fowler, James Hanan et Przemyslaw Prusinkiewicz. Modelling Spiral Phyllotaxis. *Computers Graphics*, 13(3):291 – 296, 1989. ISSN 0097-8493. 5, 6

[28] Deborah R. Fowler, Hans Meinhardt et Przemyslaw Prusinkiewicz. Modeling Seashells. *SIGGRAPH Comput. Graph.*, 26(2):379–387, juillet 1992. URL http://dx.doi.org/10.1145/142920.134096. 5

[29] Martin Fuhrer, Henrik Wann Jensen et Przemyslaw Prusinkiewicz. Modeling Hairy Plants. Dans *Proceedings of 12th Pacific Conference on Computer Graphics and Applications*, pages 217–226, 2004. 5, 7

[30] Eric Galin, Adrien Peytavie, Eric Guérin et Nicolas Marechal. Procedural Generation of Roads. *Computer Graphics Forum*, 29(2):429–438, 2010. 5

[31] Jean-David Genevaux, Eric Galin, Eric Guerin, Adrien Peytavie et Bedrich Benes. Terrain Generation using Procedural Models based on Hydrology. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 4, 2013. 5

[32] Sherif Ghali. Computation and Maintenance of Visibility and Shadows in the Plane. *In Sixth Int. Conf. in Central Europe on Computer Graphics and Visualization, WSCG '98*, pages 117–124, 1998. 19

[33] Sherif Ghali et A. James Stewart. A Complete Treatment of D1 Discontinuities in a Discontinuity Mesh. *Graphics Interface 96'*, 1996. 19

[34] James Gips. *Shape Grammars and their Uses : Artificial Perception, Shape Generation, and Computer Aesthetics*. Birkhauser, Verlag, 1975. 7

[35] Dorian Gomez, Mathias Paulin et Pierre Poulin. Occlusion Tiling and Visibility Propagation over on-the-fly Generated Cities. Dans *GRAND 2011*, 2011. 53

[36] Dorian Gomez, Mathias Paulin et Pierre Poulin. Occlusion Vectors for Visibility Computations. Dans *GRAND 2012*, 2012. 53

[37] Dorian Gomez, Mathias Paulin, David Vanderhaeghe et Pierre Poulin. Time and Space Coherent Occlusion Culling for Tileable Extended 3D scenes. Dans *CAD Graphics 2013*, novembre 2013. 92

[38] Dorian Gomez, Pierre Poulin et Mathias Paulin. Occlusion Tiling. Dans *Graphics Interface 2011*, pages 71–77, mai 2011. 31

[39] Chaim Goodman-Strauss. Matching Rules and Substitution Tilings. *Annals of Mathematics*, 147(1):pp. 181–223, 1998. ISSN 0003486X. URL http://www.jstor.org/stable/120988. 25

[40] Ned Greene, Michael Kass et Gary Miller. Hierarchical Z-Buffer Visibility. *Proceedings of SIGGRAPH*, 93:231–240, 1993. 21, 109

[41] Stefan Greuter, Jeremy Parker, Nigel Stewart et Geoff Leach. Real-time Procedural Generation of 'Pseudo Infinite' Cities. Graphite '03, pages 87–94. ACM, 2003. 5

[42] Mark S. Hammel, Przemyslaw Prusinkiewicz, Brian Wyvill et Mark Hammel Przemyslaw. Modelling Compound Leaves Using Implicit Contours. Dans *In Proceedings of CG International '92*, pages 119–131, 1992. 7

[43] Denis Haumont, Otso Mäkinen et Shaun Nirenstein. A Low Dimensional Framework for Exact Polygon-to-polygon Occlusion Queries. Dans *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques*, EGSR'05, pages 211–222, Aire-la-Ville, Switzerland, Switzerland, 2005. Eurographics Association. ISBN 3-905673-23-1. URL http://dx.doi.org/10.2312/EGWR/EGSR05/211-222. 20

[44] Sven Haveman. Generative Mesh Modeling. URL http://deposit.ddb.de/cgi-bin/dokserv?idn=977813207. 9

[45] Sven Havemann et Dieter W. Fellner. Generative Parametric Design of Gothic Window Tracery. Dans *Shape Modeling Applications, 2004. Proceedings*, pages 350 – 353, june 2004. 9, 11

[46] Vlastimil Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000. 96

[47] Bernhard Hohmann, Sven Havemann, Ulrich Krispel et Dieter Fellner. A GML Shape Grammar for Semantically Enriched 3D Building Models". *Computers Graphics*, 34(4):322–334, 2010. ISSN 0097-8493. URL http://www.sciencedirect.com/science/article/pii/S0097849310000749. 10

[48] Id Software. Doom. [Video Game], 1993. 18

[49] Jarkko Kari. A Small Aperiodic Set of Wang Tiles. *Discrete Mathematics*, 160(1-3):259–264, 1996. ISSN 0012-365X. URL http://www.sciencedirect.com/science/article/pii/0012365X9500120L. 25

[50] Vladlen Koltun, Yiorgos Chrysanthou et Daniel Cohen-Or. Virtual Occluders : An Efficient Intermediate PVS Representation. Dans *Proc. Eurographics Workshop on Rendering*, pages 59–70, 2000. 18, 19

[51] Johannes Kopf, Daniel Cohen-or, Oliver Deussen et Dani Lischinski. Recursive Wang Tiles for Real-Time Blue Noise. *ACM Trans. Graph.*, 25(3):509–518, 2006. 6, 26, 27, 143

[52] Ares Lagae et Philip Dutré. An Alternative for Wang tiles : Colored Edges Versus Colored Corners. *ACM Trans. Graph.*, 25:1442–1459, October 2006. ISSN 0730-0301. URL http://doi.acm.org/10.1145/1183287.1183296. 27, 143

[53] Ares Lagae et Philip Dutré. Poisson Sphere Distributions. Dans L. Kobbelt, T. Kuhlen, T. Aach et R. Westermann, éditeurs, *Vision, Modeling, and Visualization 2006*, pages 373–379, Berlin, November 2006. Akademische Verlagsgesellschaft Aka GmbH. ISBN 3-89838-081-5, 1-58603-688-2. URL http://www.vmv2006.rwth-aachen.de/. 29

[54] Ares Lagae, Craig S. Kaplan, Chi-Wing Fu, Victor Ostromoukhov, Johannes Kopf et Oliver Deussen. Tile-Based Methods for Interactive Applications. SIGGRAPH 2008 Course, Los Angeles, USA, August 2008. URL http://www.siggraph.org/s2008/attendees/program/item/?type=class&id=50. xxi, 28

[55] Ares Lagae, Craig S. Kaplan, Chi-Wing Fu, Victor Ostromoukhov, Johannes Kopf et Olivier Deussen. Tile-Based Methods for Interactive Applications. Dans *ACM SIGGRAPH 2008 Courses*, 2008. 25

[56] Samuli Laine. A General Algorithm for Output-Sensitive Visibility Preprocessing. Dans *Proceedings of ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games*, pages 31–39. ACM Press, 2005. 22

[57] Robert G. Laycock et Andy M. Day. Automatically Generating Large Urban Environments Based on the Footprint Data of Buildings. Dans *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, pages 346–351, New York, NY, USA, 2003. ACM. ISBN 1-58113-706-0. URL http://doi.acm.org/10.1145/781606.781663. 5

[58] Luc Leblanc. *Modélisation Procédurale par Composants*. Ph.d. thesis, Département d'Informatique et Recherche Opérationnelle, Université de Montréal, août 2011. 5, 10, 11

[59] Thomas Lechner, Ben Watson, Uri Wilensky et Martin Felsen. Procedural City Modeling. 2003. URL http://cs.northwestern.edu/~watsonb/projects/cities/. 5

[60] Aristid Lindenmayer. Mathematical Models for Cellular Interactions in Development I & II. *Journal of Theoretical Biology*, 18(3):280 – 315, 1968. ISSN 0022-5193. URL http://www.sciencedirect.com/science/article/pii/0022519368900799. 6

[61] Brandon Lloyd et Parris Egbert. Horizon Occlusion Culling for Real-time Rendering of Hierarchical Terrains. Dans *IEEE Visualization*, 2002. 20

[62] Aidong Lu et David S. Ebert. Example-based Volume Illustrations. Dans *Visualization, 2005. VIS 05. IEEE*, pages 655 – 662, oct. 2005. 29

[63] David Luebke et Chris Georges. Portals and Mirrors : Simple, Fast Evaluation of Potentially Visible Sets. *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, 6(1):105–107, 1995. 18

[64] David J. MacDonald et Kellogg S. Booth. Heuristics for Ray Tracing Using Space Subdivision. *Vis. Comput.*, 6(3):153–166, mai 1990. ISSN 0178-2789. 96

[65] Jean-Eudes Marvie, Julien Perret et Kadi Bouatouch. The FL-system : a Functional L-system for Procedural Geometric Modeling. *The Visual Computer*, 21 (5):329–339, jun 2005. URL http://www-roc.inria.fr/mirages/Publications/2005/MPB05. 7

[66] Oliver Mattausch, Jiří Bittner et Michael Wimmer. CHC++ : Coherent Hierarchical Culling Revisited. *Computer Graphics Forum (Proceedings Eurographics 2008)*, 27(2):221–230, avril 2008. ISSN 0167-7055. 21, 92, 95

[67] Sid Meier et Bruce Shelley. Civilization Saga- 1991-2011. [Video Game]. 2, 31

[68] Paul Merrell. Example-based Model Synthesis. Dans *In I3D 07 : Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 105–112. ACM Press, 2007. 5, 6, 8

[69] Paul Merrell et Dinesh Manocha. Constraint-based Model Synthesis. Dans *2009 SIAM/ACM Joint Conference on Geometric and Physical Modeling*, SPM '09,

pages 101–111, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-711-0. 5, 8

[70] Michal Mecinski. Fraqtive. [Software], 2012. 6

[71] Shahar Mozes. Tilings, Substitution Systems and Dynamical Systems Generated by them. *Journal d'Analyse Mathématique*, 53:139–186, 1989. ISSN 0021-7670. URL http://dx.doi.org/10.1007/BF02793412. 10.1007/BF02793412. 25

[72] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer et Luc Van Gool. Procedural Modeling of Buildings. Dans *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 614–623, New York, NY, USA, 2006. ACM. ISBN 1-59593-364-6. URL http://doi.acm.org/10.1145/1179352.1141931. 7

[73] Radomír Měch et Przemyslaw Prusinkiewicz. Visual Models of Plants Interacting with their Environment. Dans *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, pages 397–410, New York, NY, USA, 1996. ACM. ISBN 0-89791-746-4. URL http://doi.acm.org/10.1145/237170.237279. 6

[74] Shaun Nirenstein et Edwin Blake. Hardware Accelerated Visibility Preprocessing using Adaptive Sampling. *In Rendering Techniques 2004*, pages 207–216, 2004. 20

[75] Shaun Nirenstein, Edwin Blake et James Gain. Exact From-Region Visibility Culling. Dans *Proc. Eurographics Workshop on Rendering*, pages 191–202, 2002. 18, 20

[76] Rachel Orti et Claude Puech. Calcul efficace de facteurs de forme 2d applicable aux environnements dynamiques. Dans *Troisièmes journées de l'Association Française d'Informatique Graphique (AFIG '95)*, pages 15–22, Marseille, France, 1995. 71, 72

[77] Victor Ostromoukhov, Charles Donohue et Pierre-Marc Jodoin. Fast Hierarchical Importance Sampling with Blue Noise Properties. *ACM Trans. Graph.*, 23(3): 488–495, 2004. 26, 30

[78] Yoav I. H. Parish et Pascal Müller. Procedural Modeling of Cities. Dans *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, pages 301–308, New York, NY, USA, 2001. ACM. ISBN 1-58113-374-X. URL http://doi.acm.org/10.1145/383259.383292. 7

[79] Robert Penrose. The Role of Aesthetics in Pure and Applied Mathematical Research. *Bull. Inst. Math. & its Applications*, pages 266–271, 1974. 30

[80] Alexandre Peyrat, Olivier Terraz, Stephane Merillou et Eric Galin. Generating Vast Varieties of Realistic Leaves with Parametric 2Gmaps L-Systems. *The visual Computer*, 28(7-9):807–816, 2008. 5, 7, 9

[81] Adrien Peytavie, Eric Galin, Stephane Merillou et Jerome Grosjean. Procedural Generation of Rock Piles Using Aperiodic Tiling. *Computer Graphics Forum (Proceedings of Pacific Graphics)*, 28(7):1801–1810, 2009. xxi, 6, 11, 12, 30

[82] Przemyslaw Prusinkiewicz. Simulation Modeling of Plants and Plant Ecosystems. *Commun. ACM*, 43:84–93, July 2000. ISSN 0001-0782. URL http://doi.acm.org/10.1145/341852.341867. 7

[83] Przemyslaw Prusinkiewicz, Mark Hammel, Jim Hanan et Radomir Mech. L-systems : From the Theory to Visual Models of Plants. *Proceedings of the 2nd CSIRO Symposium on Computational Challenges in Life Sciences*, 3:1–32, 1996. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.6081. 6

[84] Przemyslaw Prusinkiewicz, Mark S. Hammel et Eric Mjolsness. Animation of Plant Development. Dans *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pages 351–360, New

York, NY, USA, 1993. ACM. ISBN 0-89791-601-8. URL http://doi.acm.org/10.1145/166117.166161. 6, 7

[85] Przemyslaw Prusinkiewicz, Mark James et Radomír Měch. Synthetic Topiary. Dans *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, pages 351–358, New York, NY, USA, 1994. ACM. ISBN 0-89791-667-0. URL http://doi.acm.org/10.1145/192161.192254. 7

[86] Przemyslaw Prusinkiewicz et Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990. ISBN 0-387-97297-8. 5, 7

[87] Carlos Saona-Vasquez, Isabel Navazo et Pere Brunet. The Visibility Octree : a Data Structure for 3D Navigation. *Computers & Graphics*, 23(5):635–644, 1999. 19

[88] Erez Sayer, Alon Lerner, Daniel Cohen-Or, Yiorgos Chrysanthou, Oliver Deussen et Universitt Konstanz. Aggressive Visibility for Rendering Extremely Complex Foliage Scenes. *Korea-Israel Bi-National Conference on Geometric Modeling and Computer Graphics*, 2004. 20

[89] Gernot Schaufler, Julie Dorsey, Xavier Decoret et François X. Sillion. Conservative Volumetric Visibility with Occluder Fusion. Dans *Proc. SIGGRAPH '00*, pages 229–238, 2000. xxiii, 16, 18, 19, 57, 58

[90] Daniel Scherzer, Lei Yang, Oliver Mattausch, Diego Nehab, Pedro V. Sander, Michael Wimmer et Elmar Eisemann. A Survey on Temporal Coherence Methods in Real-Time Rendering. Dans *EUROGRAPHICS 2011 State of the Art Reports*, pages 101–126. Eurographics Association, 2011. URL http://www.cg.tuwien.ac.at/research/publications/2011/scherzer2011c/. 18

[91] Peter G. Sibley, Philip Montgomery et G. Elisabeta Marai. Wang Cubes for Video Synthesis and Geometry Placement. Dans *ACM SIGGRAPH 2004 Posters*, SIGGRAPH '04, pages 20–, New York, NY, USA, 2004. ACM. ISBN 1-58113-896-2. URL http://doi.acm.org/10.1145/1186415.1186439. xxi, 29

[92] Joshua E. S. Socolar. Weak Matching Rules for Quasicrystals. *Communications in Mathematical Physics*, 129:599–619, 1990. ISSN 0010-3616. URL http://dx.doi.org/10.1007/BF02097107. 10.1007/BF02097107. 25

[93] Steven Spielberg. The Adventures of Tintin. [Cinema], 2011. 1

[94] Jos Stam. Aperiodic Texture Mapping. Rapport technique R046, European Research Consortium for Informatics and Mathematics (ERCIM), 1997. 11, 12, 27

[95] Dirk Staneker, Dirk Bartz et Wolfgang Straßer. Efficient Multiple Occlusion Queries for Scene Graph Systems. Rapport technique, Universitätsbibliothek Tübingen, 2004. 21, 109

[96] G. Stiny et J. Gips. Shape Grammars and the Generative Specification of Painting and Sculpture. Dans C. V. Friedman, éditeur, *Information Processing '71*, pages 1460–1465, Amsterdam, 1972. 7

[97] George Stiny. *Pictorial and Formal Aspects of Shape and Shape Grammars*. Birkhauser, Basel, 1975. 7

[98] W. Stürzlinger et Robert F. Tobler. Two optimization methods for raytracing. Dans *Proceedings of Spring School on Computer Graphics*, pages 104–107. Spring School on Computer Graphics, juin 1994. 33, 79

[99] Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch et Vladlen Koltun. Metropolis Procedural Modeling. *ACM Trans. Graph.*, 30:11 :1–11 :14, avril 2011. ISSN 0730-0301. 5, 6, 7

[100] Seth J. Teller. Visibility Computations in Densely Occluded Polyhedral Environments. Rapport technique CSD-92-708, EECS Department, University of California, Berkeley, 1992. 18

[101] Turbo Squid Inc. 3D models, textures and plugins at turbosquid, 2013. URL http://www.turbosquid.com. 103

[102] Ubisoft. Heroes of Might and Magic Saga- 1995-2011. [Video Game]. 2, 31

[103] Eric Veach et Leonidas J Guibas. Metropolis Light Transport. *SIGGRAPH 97' : Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 65–76, 1997. 21

[104] Hao Wang. Proving Theorems by Pattern Recognition I. *Commununications of the ACM*, 3:220–234, avril 1960. ISSN 0001-0782. 27, 48

[105] Li-Yi Wei, Sylvain Lefebvre, Vivek Kwatra et Greg Turk. State of the Art in Example-based Texture Synthesis. Dans *Eurographics '09, State of the Art Report (EG-STAR)*, 2009. URL http://www-sop.inria.fr/reves/Basilic/2009/WLKT09. 25

[106] Peter Wonka, Michael Wimmer et Dieter Schmalstieg. Visibility Preprocessing with Occluder Fusion for Urban Walkthroughs. Dans *Proc. Eurographics Workshop on Rendering*, pages 71–82, 2000. 18, 19

[107] Peter Wonka, Michael Wimmer, François Sillion et William Ribarsky. Instant Architecture. *ACM Transaction on Graphics*, 22(3):669–677, juillet 2003. ISSN 0730-0301. URL http://www.cg.tuwien.ac.at/research/publications/2003/Wonka-2003-Ins/. Proceedings ACM SIGGRAPH 2003. 7

[108] Sven (Intel) Woop. Embree - photo-realistic ray tracing kernels, 2012. URL http://software.intel.com. 103

[109] Hansong Zhang, Dinesh Manocha, Tom Hudson et Kenneth E. Hoff, III. Visibility Culling Using Hierarchical Occlusion Maps. SIGGRAPH '97, pages 77–88, 1997. 21

INDEX