Visibilité et rendu procédural

Mathieu Biston

Rapport de stage de Master II Recherche - Mention Informatique

Maître de stage : Jean-Claude Iehl

Visibilité et rendu procédural

Mathieu Biston

Résumé

Ce rapport de stage de deuxième année de master recherche informatique rend compte du travail effectué sur l'étude du couplage de méthodes de génération procédurale et de visibilité afin de générer en temps réel une ville selon le point de vue de l'utilisateur. Le plan des routes de la ville est généré à partir d'un algorithme original basé sur une grammaire de forme (*shape grammar*). Pour produire le moins possible de géométrie inutile, une méthode de *view frustum culling* est couplée avec la génération des routes puis la technique du Z-Buffer hiérarchique est appliquée lors de la génération des bâtiments.

Remerciements

Je souhaite remercier Jean-Claude Iehl, Éric Galin, Éric Guérin ainsi qu'Adrien Peytavie pour l'aide qu'ils m'ont apporté et les conseils qu'ils m'ont prodigués tout au long du stage.

Enfin un grand merci à toute l'équipe du LIRIS pour son accueil chaleureux et sa bonne humeur.

Table des matières

1 Introduction - Résumé

Contexte du stage

La sophistication récente des graphismes dans le domaine du jeu vidéo et du cinéma d'animation permet à l'utilisateur d'être de plus en plus immergé dans des mondes réalistes construits en 3D. La construction de ces mondes 3D nécessite le travail d'une équipe plus ou moins conséquente de graphistes selon le résultat escompté, en particulier lors de la création d'un environnement de type ville. Dans les loisirs numériques notamment, mais également dans les environnements de simulation, ces types d'espace ont tendance à devenir de plus en plus vastes et leur processus de réalisation, s'il est fait manuellement, devient une tâche longue et répétitive. La génération procédurale apporte une solution pour générer, grâce à des algorithmes, du simple modèle 3D à une ville entière en incluant les textures et la végétation. Le stage s'est déroulé au laboratoire du LIRIS à Lyon, dans l'équipe *R3AM* (*Rendu Réaliste pour la Réalité Augmentée Mobile*) et en collaboration avec l'équipe *GéoMod*. Il porte sur la génération procédurale de villes bien que ce ne soit ici qu'un cadre applicatif, les concepts développés pouvant se généraliser à d'autres types d'environnement. Il a pour but de réaliser des expérimentations sur les méthodes de génération procédurale et s'inscrit dans une démarche de recherche exploratoire.

Problématique

Actuellement dans les processus de création s'appuyant sur la génération procédurale, la méthode consiste à générer d'abord la ville en entier. Puis lors du rendu, une fonction de visibilité est chargée d'éliminer les parties qui ne seront pas visibles par la caméra, en particulier les polygones masqués par d'autres et ceux hors de la zone de vision (voir le schéma du pipeline figure 1). Cette approche soulève plusieurs problèmes :

- La génération en entier du modèle nécessite :
  - du temps processeur,
  - de l'espace disque pour stocker le modèle hors-exécution,
  - de la mémoire vive pour le stocker pendant l'exécution.
- lors de l'application de la fonction de visibilité, plus le modèle 3D est grand, plus la masse de données à traiter est grande et donc plus les calculs seront longs.

Ces points évoqués précedemment représentent un grand désavantage car dans la majorité des cas à un instant t, l'utilisateur a seulement besoin de voir une partie relativement faible du modèle, donc une grande partie de la génération et des divers traitements associés devraient être évités.

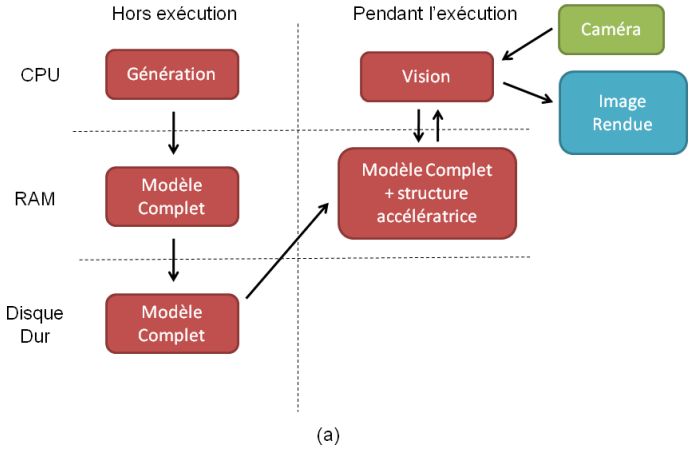

FIGURE 1 – Pipeline classiquement utilisé

Méthode utilisée

L'idée serait de coupler une méthode de génération procédurale avec une fonction de visibilité afin de limiter au maximum la génération de la géométrie *inutile* dans le sens non visible par la caméra à l'instant t (voir le diagramme du pipeline théorique figure 2). Le début du stage a d'abord été consacré à la recherche bibliographique concernant les approches de production automatique de villes et également sur l'accélération de rendu.

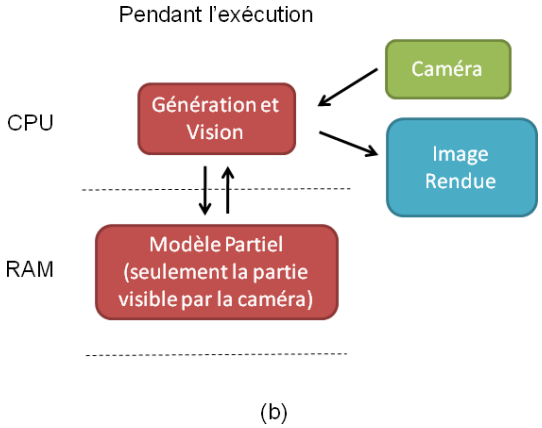

FIGURE 2 – Le pipeline de l'idée du couplage. Il n'y plus de stockage sur le disque dur, ni de chargement en mémoire.

Le choix de la méthode de génération procédurale s'est porté sur une idée originale. Elle utilise un algorithme basé les grammaires de formes (*shapes grammar* en anglais) pour subdiviser le plan des routes de la future ville. Ces grammaires sont simples et faciles à implémenter, et également efficaces et rapides pour le problème donné. Ceci a permis d'obtenir rapidement des

résultats graphiquement satisfaisants, afin de s'intéresser ensuite au problème de visibilité. L'algorithme de génération procédurale a ensuite été adapté afin d'être couplé avec les méthodes d'élimination des objets hors du volume de vision (*View Frustum Culling* en anglais) et du Z-Buffer hiérarchique. Il n'y a pas eu d'apport au domaine des fonctions de visibilité mais l'originalité de cette deuxième partie réside dans ce couplage, qui n'a jamais été mis en oeuvre auparavant.

2 Etat de l'art

Le sujet de ce stage de fin d'étude se situe dans l'interface entre deux domaines, à savoir la modélisation automatique et le rendu efficace de modèles 3D. La première partie de l'état de l'art est donc consacrée à la génération procédurale appliquée à la production de cités. L'objectif final est d'obtenir la génération d'une ville en temps réel, génération qui se fait dans les approches actuelles en pré-calcul ou en intéractivité (non temps réel) avec l'utilisateur. Une étude des approches traditionnelles d'accélération de rendu est donc nécessaire et fait l'objet de la deuxième partie qui porte sur l'élimination des polygones non visibles selon le point de vue.

2.1 Génération de cités

La génération de cité est un sous domaine de la génération procédurale. Le terme de *génération procédurale* définit un processus qui, grâce à un algorithme, génère automatiquement à la volée du contenu comme par exemple du contenu sonore ou encore de la géométrie. Le but de la création automatique de géométrie est de gagner du temps lors de la production de vastes espaces virtuels. Le résultat doit cependant rester le plus crédible possible, c'est-à-dire par exemple ne pas introduire de redondances visuelles nuisant à l'immersion de l'utilisateur. Cette première partie de l'état de l'art s'intéresse aux principales approches de génération procédurale, en commençant par présenter dans une première sous-partie les concepts de base sur lesquels s'appuient la majorité des méthodes actuelles de génération de villes exposées dans la deuxième sous-partie.

2.1.1 Méthodes fondamentales de génération procédurales

Elles se basent dans leur majorité sur des systèmes en lien avec des grammaires. Le système le plus simple que l'on puisse considérer étant les *fractales*. Une fractale est une courbe, une surface, un volume de forme irrégulière ou morcelée qui se crée en suivant des règles déterministes ou stochastiques, impliquant une homothétie interne à différentes échelles. Elles représentent une méthode de génération procédurale dans le sens où elles peuvent générer de la géométrie de manière automatique. Les *L-Systems*[1] tiennent leur appellation de l'initiale du nom du biologiste Lindenmayer, ils servaient originellement à modéliser le développement des plantes de manière mathématique. Ils servent à générer des fractales. Les L-Systems sont des types de grammaires dont le processus de dérivation est contrôlé par l'utilisateur, qui peut fixer le nombre de réécriture des règles (exemple en figure 3). Une analogie peut-être établie entre ce contrôle du processus de réécriture et le contrôle du niveau de détail de l'objet généré. Les L-Systems donnent en sortie des chaînes de caractères terminaux qu'il est possible d'interpréter

graphiquement à la manière du langage de programmation LOGO[2], chaque symbole terminal représentant une action atomique du processus de dessin (exemple figure 4). Une extension des L-Systems, les *FL-Systems*[3] (comme F*unctionnal L-Systems*) ont été inventés afin d'être plus adaptés à la génération procédurale en remplaçant les symboles terminaux par des fonctions et offrant notamment la possibilité de manipuler des références sur des objets génériques (exemple figure 5).

$V = \{A, B\}$
$\omega = A$
$P = \{(A \rightarrow AB); (B \rightarrow A)\}$

On obtient la séquence suivante :

$\mathbf{n = 0} : A$
$\mathbf{n = 1} : AB$, le A a été remplacé par AB, la partie droite de la règle
$\mathbf{n = 2} : ABA$, le A a été remplacé par AB et le B par A
$\mathbf{n = 3} : ABAAB$, ainsi de suite à chaque itération
$\mathbf{n = 4} : ABAABABA$
$\mathbf{n = 5} : ABAABABAABAAB$
$\mathbf{n = 6} : ABAABABAABAABABAABABA$
$\mathbf{n = 7} : ABAABABAABAABABAABABAABAABABAABAAB$

FIGURE 3 – Exemple d'un L-Système et de son développement

ω : $A(100, w_0)$
p_1 : $A(s, w)$: $s >= min \rightarrow !(w)F(s)$
$[+(\alpha_1)/(\varphi_1)A(s * r_1, w * q \wedge e)]$
$[+(\alpha_2)/(\varphi_2)A(s * r_2, w * (1 - q) \wedge e)]$

a b c

FIGURE 4 – Exemple de code en langage Logo, et quelques figures qu'il est possible de générer en modifiant les paramètres α_1, α_2, φ_1, ... et en choisissant le nombre de dérivations approprié

Les shape grammars[4] (terme pouvant être traduit par grammaires de formes) sont un autre type de grammaires dont le formalisme repose sur la description de formes simples, faites de segments ou d'ensembles de segments. Comme dans une grammaire au sens de Chomsky, il s'agit de définir des règles de productions, un axiome, des états terminaux, etc. Elles permettent de transformer des segments ou des ensembles de segments en d'autres segments ou ensemble de segments et générer ainsi les formes désirées (exemple figure 6).

Ces méthodes fondamentales ne permettent pas en elles-mêmes de générer des cités, mais beaucoup de systèmes procéduraux s'appuient dessus

```
facade(width,m,w,H,a,d,h,smin...) =
    layout(m,n=floor((width-2*smin)/w),
           w,H,a,d,h,(width-n*w)/2,...)
    ...;

layout(m,n,w,H,a,d,h,s,...) =
    firstFloor(n,w,H,s,...)
    nextFloors(m-1,n,w,H,a,d,h,s,...)
    ...;

firstFloor(n,w,H,s,...) =
    wedging(0,H,s,...)
    for(i=0;i<n;i++) arcade(i,n,w,H,...)
    wedging(s+n*w,H,s,...);

nextFloors(m,n,w,H,a,d,h,s,...) =
    wedging(0,H,s+d,m*h,...)
    pilaster(s+d+a,H,2*d,m*h,...)
    for (i=0;i<m;i++)
        for(j=0;j<n;j++)
            window(s+d+j*w,H+i*h,...)
    for(j=2;j<n-1;j++)
        pier(s+j*w-d,H,2*d,m*h,...)
    pilaster(s+(n-1)*w-d,H,2*d,m*h,...)
    wedging(s+n*w-d,H,s+d,m*h,...);
```

FIGURE 5 – Exemple d'un code de FL-System et trois exemples de bâtiments qu'il est possible d'obtenir avec le même code en changeant les paramètres

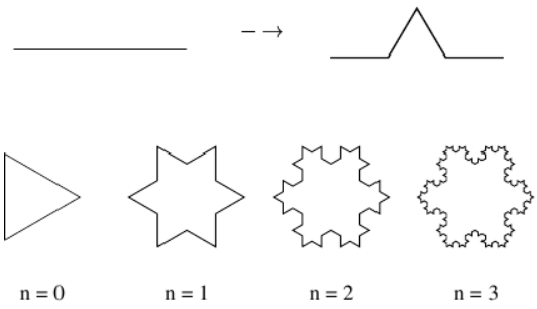

FIGURE 6 – Exemple de grammaire de forme avec seulement une seule règle, et un triangle en temps qu'axiome. Le développement a été stoppé après 3 dérivations.

pour produire la géométrie des éléments. Nous allons donc maintenant nous intéresser aux grandes méthodes de la littérature servant à créer automatiquement des villes.

2.1.2 Approches existantes de génération procédurale de villes

Les méthodes en rapport avec la génération automatique de villes sont nombreuses. Dans son état de l'art, Vanegas[5] explique que modéliser le développement d'une ville de manière réaliste est une tâche compliquée car les paramètres à prendre en compte sont nombreux : relief et coût du terrain, densité de population, densité du réseau routier, comportement du marché de l'immobilier, décisions politiques, ... Selon les approches, un ou plusieurs de ces paramètres sont pris en compte. Il est possible de les classifier selon qu'elles soient en génération temps réel ou non, en génération assistée (l'utilisateur contrôle des points clés de la génération), mais également se-

lon les types d'entrée qu'elles prennent (carte du terrain, vue aérienne d'une ville déjà existante afin de la regénérer informatiquement, carte de densité de population, ...). En général, la production est réalisée en trois phases : la génération du plan des routes, la génération des parcelles de bâtiments et la génération des bâtiments.

La génération du plan des routes est la première phase de la production automatique d'une ville. Elle peut tenir compte de plusieurs critères comme le relief, les obstacles (comme les plans d'eaux, les montagnes, ...), ... Parish et Müller dans leur *CityEngine*[6] s'appuient sur des L-Systems améliorés et suivent des objectifs globaux et des contraintes locales. Les objectifs globaux prennent en compte un patron global de ville (*réseau radial*, *réseau en branche* ou *réseau rectangulaire* comme illustré en figure 7) et imposent de relier les centres de densité de population (donnés par une carte de population en entrée). Les contraintes locales empêchent les routes d'entrer dans des *zones interdites* (plans d'eaux, montagnes et autres obstacles divers), et s'assurent que les routes se croisent de manière esthétique en formant des carrefours ou alors qu'elles ne sont pas trop courtes, etc. Chen et Esch[7] ainsi que Kelly et McCabe[8] proposent des approches avec génération semi-automatique assistée par l'utilisateur. Chen et Esch utilisent les tenseurs et leurs lignes de champs pour générer les routes : l'utilisateur place d'abord manuellement les tenseurs puis modifie les courbes de leurs lignes de champs (le processus est présenté graphiquement figure 8). Dans la méthode de Kelly et McCabe, l'utilisateur place plusieurs sommets correspondants aux carrefours du réseau puis l'algorithme génère la route la plus "esthétique" afin de les relier selon des critères prenant en compte l'élévation du sol (cf figure 9). D'autres approches originales comme celles de Lechner[9] se basent sur la simulation d'agents qui vont explorer le terrain et construire en conséquence les routes. Cette dernière méthode est cependant trop longue à exécuter pour l'appliquer à notre cas.

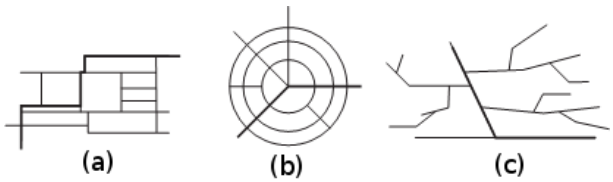

FIGURE 7 – Exemple de patrons de villes : réseau rectangulaire en (a), réseau radial en (b) et réseau en branche en (c)

Quand le plan des routes a été créé, il faut diviser les blocs de terrain (*blocks* en anglais) en parcelles (*lots* en anglais) afin de pouvoir implanter des bâtiments. Les empreintes produites doivent satisfaire plusieurs contraintes comme par exemple être connectées à la route (car un bâtiment doit être connecté à la route), ou avoir une forme réaliste, etc. Dans CityEngine[6], les polygones formés par les routes sont d'abord mis à l'échelle inférieure puis

FIGURE 8 – Pipeline de création de la ville avec la méthode de Chen et Esch [7]. Les champs de tenseurs dans (a) servent à générer les routes dans (b). Les bâtiments sont ensuite créés (c)

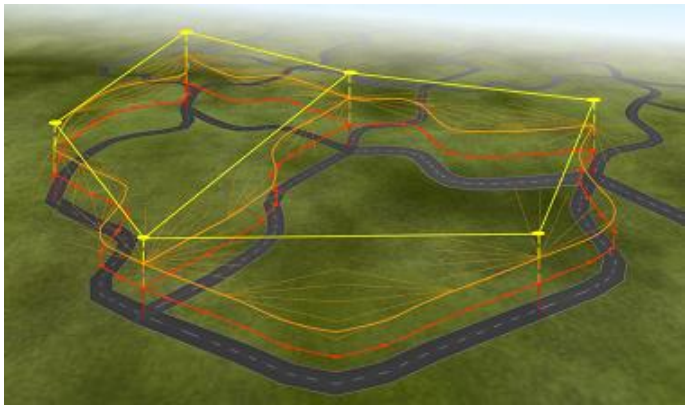

FIGURE 9 – Illustration de la méthode de Kelly et McCabe[8]

divisés selon un algorithme récursif qui divise les arêtes les plus longues et qui sont approximativement parallèles entre elles (cf figure 10). Les parcelles qui ne sont pas reliées à la route ou qui sont trop petites sont supprimées. Dans CityGen[8], le principe de subdivision est analogue, mais peut également traiter les blocs convexes et concaves. Un schéma et les explications de la subdivision sont donnés en figure 11.

FIGURE 10 – Exemple de division du plan de route en blocs puis en parcelles pour les bâtiments, dans l'article sur CityEngine de Müller et Parish[6].

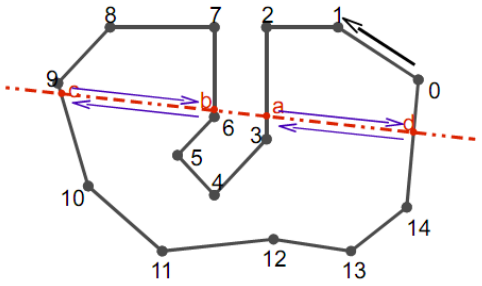

FIGURE 11 – Subdivision des blocs en parcelles dans l'article sur Citygen[8] : On identifie l'arête la plus longue du polygone (0-14). On trace une perpendiculaire à cette arête passant par un point éloigné du milieu du segment (droite en rouge). On stocke les sommets résultants des intersections entre la droite et les autres arêtes du polygone, créant ainsi de nouveaux segments [cb] et [ad]. L'algorithme continue récursivement sur les segments nouvellement créés, c'est-à-dire [cb] et [ad] pour la première étape. A la fin, les parcelles non reliées à la route ou trop petites sont supprimées.

La dernière étape de la production de la ville est la génération des bâtiments. Dans l'article de CityEngine[6], un L-System est encore utilisé ainsi qu'un module d'extrusion. Il ajoute des patrons géométriques pour les toits, les antennes et autres éléments impossibles à spécifier par des grammaires. La forme finale du bâtiment dépend de la forme initiale de la parcelle interprêtée par le L-System. Les itérations successives peuvent être considérées comme un raffinement de la géométrie du bâtiment et peuvent donc servir à contrôler le niveau de détail à donner. Les textures sont ensuite générées procéduralement (illustration du résultat final figure 12). Müller[10] et Wonka[11] ont également publié des approches spécifiquement consacrées à la génération de bâtiments (résultats figure 13). Celle de Müller est basée sur les grammaires de formes, et effectue des calculs de visibilité pour savoir si placer un élément n'entrainera pas d'auto-collision du bâtiment, ou si l'élement sera visible. L'approche de Wonka s'appuie sur les grammaires de divisions (*split grammar* en anglais), proches des grammaires de formes à la différence que le processus de dérivation est entièrement déterminé à l'avance et que le volume de l'espace occupé par les éléments reste le même tout au long des dérivations.

Pour augmenter la qualité visuelle du rendu, il est possible d'utiliser des méthodes spécifiques de génération de façades. Müller[12] en propose une qui à partir de photos de façades de bâtiments, analyse puis reconstruit une façade en géométrie 3D. La génération automatique du plan d'intérieur, comme le propose Merell[13] dans sa méthode basée sur un apprentissage par réseaux bayésiens, est également une façon de sophistiquer le rendu. La méthode de Merell n'est cependant pas temps-réel et sort donc du cadre de notre étude.

FIGURE 12 – Exemple de villes produites par CityEngine

FIGURE 13 – Exemples de bâtiments produits par les méthodes de Müller (à gauche) et Wonka (à droite)

2.1.3 Conclusion sur la génération de villes

Cette section de l'état de l'art a donné un aperçu global des méthodes de génération procédurale existantes appliquées à la production de villes. Les concepts généraux qu'il est possible de tirer de ces recherches bibliographiques sont l'utilisation des grammaires de formes ou de type L-System et la séparation de la génération en trois étapes : génération du plan des routes, génération des parcelles de bâtiments et génération des bâtiments. Le travail de Müller dans CityEngine[10] présente une méthode complète, l'utilisation des L-Systems y est une approche concluante. Les deux articles de Chen et Esch[7] et ainsi que Kelly et McCabe[8] présentent deux méthodes semi-automatiques nécessitant l'intervention de l'utilisateur, bien qu'il soit possible de s'en affranchir en définissant des règlages automatiques et esthétiques de ces paramètres. Cependant, toutes les méthodes trouvées dans la littérature posent le problème du temps de génération qui est en moyenne de quelques secondes, très éloigné de la création en temps réel du sujet de ce stage.

Après un examen de la littérature existante, il apparait que les grammaires de formes ont déja été utilisées pour générer les bâtiments, mais pas pour générer les routes. Une idée originale serait donc de les utiliser dès la première étape du processus de génération. Enfin, il n'est pas fait état d'une méthode combinant la génération procédurale de cités et la visibilité, ce couplage présente donc également une approche originale. La suite de cet état de l'art est ainsi consacrée à l'étude des approches d'accélération de rendu,

nécessaires pour atteindre l'objectif de temps réel visé par le stage.

2.2 Techniques traditionnelles d'élimination des polygones non visibles

Bien que les puissances des cartes graphiques ne cessent de croître, il en est de même pour les scènes qu'elles ont à traiter. Ceci est vrai en particulier dans le domaine du jeu vidéo où les concepteurs veulent plonger le joueur dans des univers toujours plus réalistes nécessitant des environnements 3D larges et complexes. Pour un jeu vidéo par exemple, une fréquence convenable pour l'affichage de ces univers est de 60 images par seconde. Ce qui est impossible à obtenir si l'univers en entier est dessiné à chaque image. Or, seule une partie relativement faible de la scène est visible par la caméra selon la position où se trouve l'utilisateur et la direction dans laquelle il regarde. Il est donc nécessaire d'utiliser des techniques permettant d'écarter les polygones qu'il est inutile d'envoyer à la carte graphique afin de ne pas les dessiner. L'objectif du stage est d'aboutir à un algorithme de génération dont le temps d'exécution est proportionnel à ce qui est visible afin d'économiser du temps qui pourra par exemple être utilisé pour augmenter le niveau de détail. Cette section dresse un bilan des grandes approches existantes dans la littérature scientifique à ce sujet. Elle commence par de brefs concepts généraux sur la visibilité pour continuer sur une présentation de la détection exacte des polygones visibles. Enfin, elle termine par deux parties consacrées aux deux grandes classes de méthodes de visibilité, à savoir celles travaillant à partir d'un point de vue *ponctuel*, et celles travaillant à partir d'une *zone de visibilité*.

2.2.1 Généralités

Bien que l'on puisse séparer les méthodes de visibilité en deux grandes classes, il n'en reste pas moins une multitude de critères permettant de les classifier et de comparer leurs performances. Les principaux critères de classification sont présentés ici :

- *visibilité à partir d'un point / visibilité à partir d'une zone* : Ces deux classes introduisent des contraintes différentes. La visibilité à partir d'un point (fig. 14.a.) n'est valable que pour une position et une direction dans l'environnement et doit donc être recalculée à chaque mouvement, elle est cependant plus rapide à calculer que la visibilité à partir d'une zone (fig. 14.b.). Cette dernière est valable pour une région de l'environnement déterminée et n'a besoin d'être recalculée qu'en cas de sortie de cette zone. Elle est plus coûteuse à calculer et est souvent réalisée à l'aide d'informations pré-calculées.
- *espace objet / espace image* : ce critère indique à quelle échelle se place l'algorithme pour éliminer les éléments qui ne seront pas visibles. Si

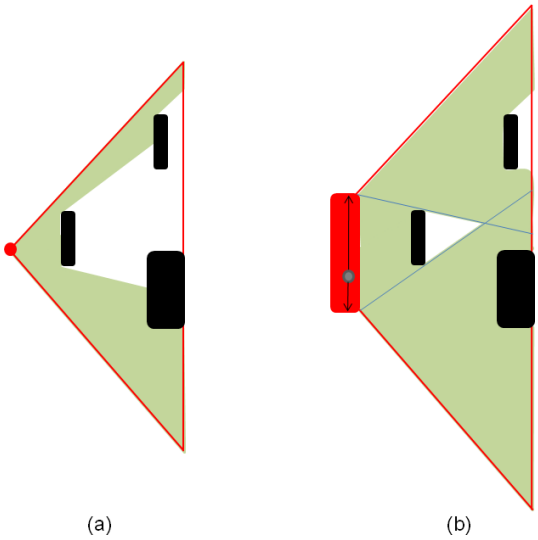

FIGURE 14 – Visibilité à partir d'un point en (a), et à partir d'une zone en (b)

l'algorithme travaille dans l'espace objet, alors il cherchera en général à déterminer quels polygones masquent quels autres polygones par des considérations géométriques. Dans l'espace image, l'algorithme se place au niveau du pixel et à l'aide de projections ainsi que de simulations de dessin, il détermine quels sont les pixels visibles.

- *détection exacte / conservative / approximative* : Certaines approches permettent d'obtenir l'ensemble exact des polygones visibles, mais elles sont très coûteuses en temps de calcul et donc inutilisable en temps réel. Il est possible d'avoir recours à des algorithmes dits *conservatifs* ou alors *approximatifs*. Les méthodes conservatives déterminent un sur-ensemble des polygones réellement visibles, elles assurent donc d'afficher tous les polygones visibles par la caméra, voire en incluant même d'autres polygones qui seront dessinés de manière superflue. Les méthodes approximatives (également appelées méthodes *agressives*) sont plus rapides mais calculent un PVS approximatif, c'est-à-dire un sous-ensemble des polygones réellement visibles, ce qui signifie que certains polygones normalement visibles peuvent être déclarés non visibles et donc des "trous" peuvent apparaître lors du rendu.
- *environnement intérieur / extérieur* : Le type d'environnement joue un rôle très important dans l'organisation de l'algorithme. En effet, si la caméra est placée dans un appartement contenant des meubles et des portes ouvertes alors il faudra potentiellement effectuer le rendu d'une partie de la pièce d'à côté (fig. 15.a.). Les méthodes d'intérieur utilisent la notion de *cellules* et *portails*[14]. Tandis que si l'environnement est de type *extérieur* alors tous les objets occultés et occultants se situent dans un même espace ouvert (fig. 15.b.).

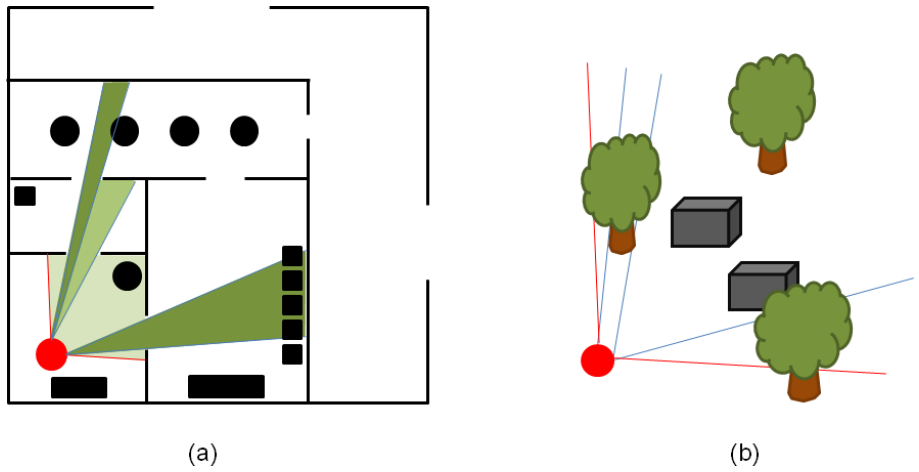

FIGURE 15 – (a) Illustration de la problématique du rendu en intérieur
(b) Le rendu en extérieur pose une problématique moins complexe

- *objets nécessairement convexes ou non* : Certains algorithmes de calcul d'occultation ne fonctionnent que lorsque que tous les objets sont convexes, ce qui limite la géométrie de la scène ou impose des découpages des objets non convexes en sous-objets convexes, ou encore l'utilisation de volumes englobants convexes.
- *environnement statique / dynamique* : Certaines méthodes s'appuient sur un partitionnement de l'espace, par exemple de type *octree*. La mise à jour de ce genre de structure peut s'avérer très coûteuse mais leur utilisation permet d'accélérer grandement le calcul de l'occultation. D'où la nécessité de savoir si la scène est statique ou non.
- *méthode avec pré-calculs ou sans pré-calculs* : Selon les applications, il est possible ou non d'utiliser une méthode avec pré-calculs. A savoir que les résultats des pré-calculs devront être stockés en mémoire lors de l'exécution, entraînant un coût mémoire supplémentaire.

Nous remarquons dans la figure 14 qu'il est possible d'établir une analogie entre la recherche d'élements occultants (en travaillant dans l'espace objet) et le calcul des ombres. En effet, si un objet B se trouve dans l'ombre d'un autre objet A à partir du point de vue (ou zone de vue) de la caméra, alors il sera effectivement caché par cet objet A. Ce concept est utilisé dans de nombreuses approches basées objets.

Après une partie consacrée à quelques généralités introductives sur la visibilité, nous allons nous intéresser aux méthodes existantes pour atteindre la visibilité exacte.

2.2.2 La visibilité exacte

Même si elle peut apparaître comme un résultat idéal, la visibilité exacte dans les méthodes développées jusqu'à maintenant ne s'obtient qu'au prix de calculs très coûteux, hors de portée du temps réel et difficilement utilisable en réalisant des pré-calculs étant donné la capacité mémoire nécessaire pour stocker les informations pré-calculées. Nous allons nous intéresser aux deux méthodes principales existantes : le *graphe d'aspect* puis le *complexe de visibilité 3D*.

Le graphe d'aspect[15] est un graphe permettant de représenter les changements de visibilité d'une scène 3D. Le principe étant de subdiviser l'environnement en cellules dans lesquelles la visibilité est qualitativement invariante. Pour cela, on construit le *graphe de structure d'image* (*ISG* pour *Image Structure Graph* en anglais) pour chaque vue d'un objet et il est dit que deux vues différentes ont le même *aspect* si leurs ISG sont isomorphiques. Deux graphes sont isomorphiques si leurs structures sont identiques, c'est-à-dire si leurs représentations graphiques sont visuellement différentes mais que le nombre de sommets et que leurs voisins sont identiques (cf figure 16). On regroupe toutes les vues dont les ISG sont isomorphiques dans une même région de la *partition de l'espace de visibilité* (*VSP* pour *Visibility Space Partition*), chaque région représentant un noeud du graphe d'aspect. Les noeuds du graphe d'aspect sont séparés entre eux par des *évènement visuels*, c'est-à-dire un changement qualitatif dans la visibilité. Chaque évènement visuel représente une arête du graphe d'aspect.

Une fois le graphe d'aspect construit, il peut être utilisé pour modifier uniquement ce qui est nécessaire dans la visibilité selon le changement de région de la caméra au sein du VSP. Le problème est que la complexité de ce graphe est grande et sa taille peut atteindre $O(n^9)$, n étant le nombre de segment de la scène, ce qui devient impossible à mettre en oeuvre à partir d'une dizaine de miliers de segments.

Le complexe de visibilité 3D[17] (*3D visibility complex* en anglais) est une autre façon de décrire la visibilité de l'espace 3D utilisant le principe du lancer de rayon. Il se base sur un concept de *segment libre maximum* (*maximal free segment* en anglais) au lieu des ISG (cf partie 2.2.2) pour partitionner l'espace. Un segment libre maximum est un segment qui part de la surface d'un objet et qui arrive à la surface d'un autre objet sans obstacle, traduisant le fait qu'un objet puisse "voir" un autre objet. Ce graphe a une complexité moindre que le graphe d'aspect mais s'élevant tout de même à $O(n^4)$.

Pour l'accélération du rendu, les calculs servant à l'élimination des polygones non visibles n'ont de sens que s'ils demandent moins de temps qu'afficher la totalité de la géométrie, sinon envoyer tous les polygones à la carte graphique et les dessiner serait alors temporellement plus rentable. La visibilité exacte est trop lourde à calculer et a été présentée ici uniquement par

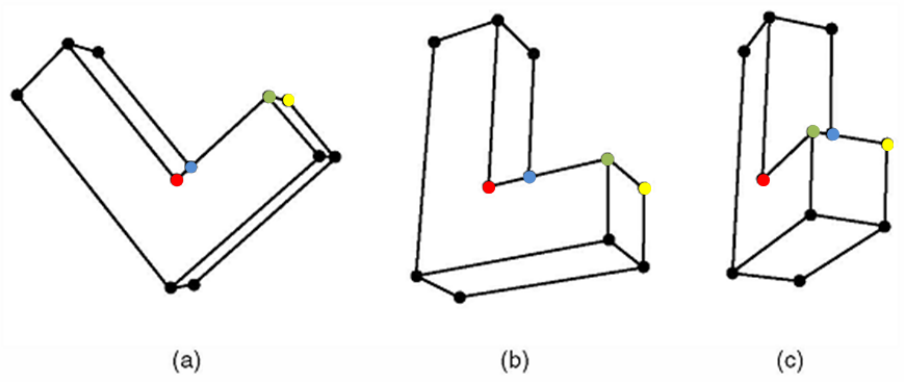

FIGURE 16 – Après projection planaire des vues de objets, on construit leurs ISG indiquant comment les sommets du polyhèdre sont reliés entre eux sur la projection. On voit que (a) et (b) ont le même *aspect* car leurs structures sont identiques, tandis que (c) a un aspect différent puisqu'en considérant les sommets *rouge*, *bleu*, *vert* et *jaune*, on remarque que le sommet *bleu* est relié au *rouge* et au *vert* dans (a) et (b), mais il est relié au *vert* et au *jaune* dans (c). (image issue de l'article [16])

souci d'élargir la vision de cet état de l'art. Nous allons maintenant étudier les méthodes approximatives et conservatives, qui elles, sont beaucoup plus rapides.

2.2.3 La visibilité en fonction d'un point de vue dans l'espace objet

La visibilité en fonction d'un point de vue (comme illustrée en fig. 14) peut être calculée plus rapidement que la visibilité en fonction d'une zone. Cependant, il faut la recalculer à chaque fois que la caméra change de direction ou de position. Dans cette classe de méthode, il existe des approches basées objet présentées dans la première section et des approches basées image présentées dans la deuxième.

Les approches basées objets utilisent la cohérence qu'il existe entre les objets ou entre les sous-objets, par exemple au niveau des polygones. Généralement par des considérations géométriques et des projections, elles cherchent à déterminer si un élément (un polygone par exemple) est masqué partiellement ou totalement par un ou plusieurs autre éléments.

Dans ce domaine, deux fonctions sont couramment utilisées dans tout processus de rendu graphique : l'*élimination des faces retournées* et l'*élimination des objets hors du volume de vision*. La méthode d'élimination des faces retournées (*backface culling* en anglais) se base sur une considération très simple : les polygones dont la face avant ne pointe pas vers la caméra ne sont pas visibles. Le calcul réalisé est simplement un produit scalaire de la normale de la caméra avec la normale du polygone testé (cf fig. 17). Si le résultat

est négatif, alors le polygone est affiché, car sa normale est opposée à la normale de la caméra, il pointe donc vers la caméra. Si le résultat est positif ou nul alors le polygone n'est pas affiché. Cette méthode est implémentée matériellement dans les cartes graphiques d'aujourd'hui. L'élimination des objets hors du volume de vision (*VFC* pour *View Frustum Culling* en anglais) est une méthode couramment utilisée qui consiste à détecter les objets qui ne se situent pas dans la zone appelée *View Frustum* en anglais. Le view frustum est en fait un cube déformé par une projection, et il est constitué de six plans (bas, haut, gauche, droite, lointain et près). Tout polygone ne se trouvant pas totalement ou partiellement entre ces six plans sera éliminé (cf fig. 18). Pour accélérer cet algorithme, un article[18] propose quatres optimisations basées sur des considérations géométriques et temporelles.

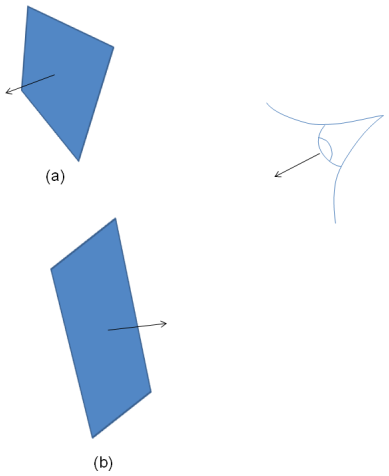

FIGURE 17 – Le polygone (a) sera éliminé par le *backface culling* car sa normale ne pointe pas vers la caméra, tandis que le polygone (b) le sera

Dans l'espace objet, il est également courant d'utiliser une structure de partitionnement[19] de l'espace à base d'octrees, de voxels ou autre. Elles permettent de diviser l'espace en une hiérarchie de régions avec une information globale attachée à chacune d'entre-elles dépendant des objets qui y sont contenus, par exemple l'englobant des objets présents dans chaque zone. Ensuite, cette information servira à décider si les objets à l'intérieur d'une région donnée doivent être testés ou rejetés, éliminant ainsi très rapidement de grande zones. Utilisé conjointement avec une structure de partitionnement de l'espace, le lancer de rayon (*ray-casting* en anglais) est une technique visant à simuler l'envoi d'un rayon lumineux pour chaque pixel de l'image en cours de dessin et tester la surface qu'il frappe en premier. Cette surface est alors visible et tous les autres objets derrières ne le sont pas. Dans le cas où il est utilisé avec un octree, si le rayon frappe une zone, alors on descend dans la hiérarchie (et donc dans les sous-zones) de l'octree afin de déterminer plus précisément quels sont le ou les objets atteints par le rayon.

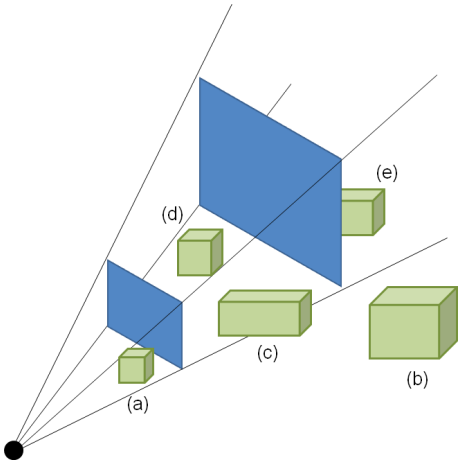

FIGURE 18 – Par la méthode du VFC : (a) sera éliminé car il est trop près, (b) sera éliminé car il est trop à droite, (c) sera partiellement visible, (d) sera totalement visible et (e) sera éliminé car trop loin

Une autre grande classe de méthodes travaillant dans l'espace objet utilise le concept d'ombres pour calculer quels objets sont masqués par d'autres. Une idée intuivive est proposée par Hudson et Manocha[20]. Elle considère le fait que l'ombre d'un objet occultant peut être modélisée par un cône généralisé (voir figure 19), équivalent au volume de vue (le *View Frustum Culling* abordé au début de cette section). La méthode de l'article découpe tout d'abord l'espace en zones, puis détermine les potentiels bons occultants de chaque zone selon certains critères pendant une phase de pré-calcul. Pendant l'exécution, on sélectionne les bons occultants en fonction de la direction de la caméra en choisissant parmi ceux déjà pré-calculés. Enfin, des tests d'intersection entre les cônes des occultants et les autres objets sont réalisés. Wonka et Schmalstieg[21], dans une méthode hybride exploitant la cohérence image et objet, s'appuyent sur le même principe de pré-calcul de bons occultants et de sélection finale en temps réel. Ils proposent ensuite de dessiner les cônes d'ombre des occultants dans un Z-Buffer (voir figure 20). Dans ce cas, la composante Z représente la hauteur minimale des occultants. Les boîtes englobantes des objets sont d'abord testées avec ce buffer pour un rejet rapide des objets non occultés.

Cette partie a donné un aperçu des méthodes de visibilité en fonction d'un point de vue, travaillant dans l'espace objet. La méthode du volume de vision sera impérativement utilisée au vu du nombre d'objets qu'elle permet d'éliminer pour des calculs relativement peu coûteux. Enfin une autre méthode plus fine devra être choisie pour éliminer les objets masqués se trouvant dans le volume de vision. Bien que les méthodes d'occultation par les ombres nécessitent des pré-calculs, elles peuvent être utilisées à condition de pouvoir générer les informations requises pendant la phase de production au-

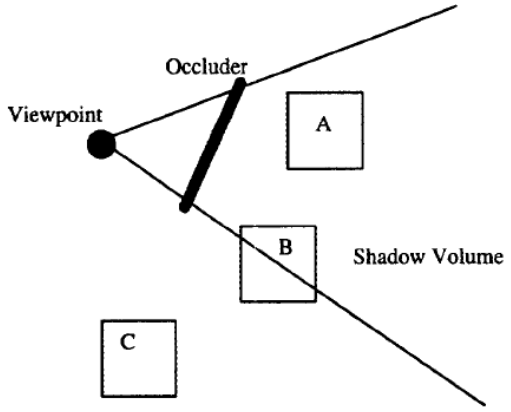

FIGURE 19 – Relation entre ombre d'un objet occultant et le volume de vision

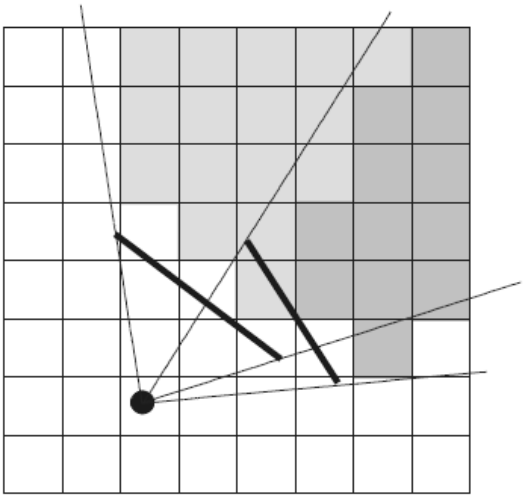

FIGURE 20 – L'ombre des occultants est rendue dans un Z-Buffer, la composante Z représentant ici la hauteur minimum des occultants de la case. Avant d'être dessinés, les occultants sont réduits (par une mise à l'échelle) pour rendre la méthode conservative.

tomatique de la géométrie. La phase de génération peut également permettre de créer un partitionnement de l'espace à la volée, permettant d'optimiser le choix ou les tests des objets occultants ou occultés.

2.2.4 La visibilité en fonction d'un point de vue dans l'espace image

Les approches basées images utilisent la cohérence qu'il existe entre les pixels ou entre les groupes de pixels. Elles cherchent quels pixels sont masqués par quels autres pixels afin de n'afficher que les pixels visibles. Ce sont les méthodes les plus utilisées. Une méthode classique, appelée Z-Buffer,

consiste à parcourir tous les pixels de chaque objet de la scène. Pour cela, on maintient un tableau de pixel représentant l'écran et un tableau de la même taille contenant les profondeurs (les coordonnées Z, d'où le nom de Z-Buffer) des pixels déjà dessinés à l'écran. Lorsqu'un pixel doit être dessiné, il faut vérifer dans le tableau de profondeur quelle est la profondeur du pixel déjà dessiné à cette position. Si le pixel que l'on veut dessiner se trouve derrière le pixel déjà dessiné (son Z est plus grand) alors on passe au pixel suivant, sinon on dessine le pixel et on met à jour le tableau avec la profondeur du pixel que l'on vient de dessiner. Cette méthode est simple à mettre en place, mais très coûteuse si le nombre d'objets de la scène est grand, car il faudra tous les parcourir et les dessiner pour identifier sur quels pixels du Z-Buffer ils se projettent. Cette méthode est intégrée dans les cartes graphiques. La littérature propose une extension de cette méthode basique, apppelée le Z-Buffer hiérarchique (*HZB* pour *Hierarchical Z-Buffer* en anglais). Bien que cette méthode soit une méthode hybride exploitant également la cohérence de l'espace objet et la cohérence temporelle, la majeure partie de son traitement s'effectue dans l'espace image. L'approche de Greene, Kass et Miller appelée *Z-Buffer hiérarchique*[22] propose qu'au lieu de tester la profondeur de chaque pixel, on ait une hiérarchie de pixels, et qu'au lieu de tester chaque objets, on ait une hiérarchie d'objets. On réalise les tests avec les englobants des objets à dessiner pour savoir rapidement quelles sont les parties visibles de la hiérarchie d'objets. Il existe une méthode dont les concepts sont analogues, appélée *carte d'occultation hiérarchique*[23] (*HOM* pour *Hierarchical Occlusion Map* en anglais). C'est une technique conservative basée image. Elle est découpée en plusieurs étapes. La première consiste à éliminer les objets qui ne sont pas dans le volume de vision (cf partie 2.2.3). Puis elle sélectionne des objets occultants dans la scène pour construire la carte. Pour la sélection, une base de donnée est construite pendant une phase de pré-calcul qui s'appuie sur plusieurs critères pour pré-sélectionner les bons occultants. Les objets occultants sélectionnés sont ensuite dessinés dans une texture (avec la carte graphique pour utiliser l'accélération matérielle). Cette texture forme le premier niveau d'une pyramide que l'on appelle hiérarchie de cartes d'occultation, qui repose sur le même principe que le Z-Buffer hiérarchique, à la différence près qu'entre deux niveaux, ce n'est pas la valeur la plus grande sur les quatre qui est conservée, mais la moyenne de l'opacité. L'algorithme maintient en parallèle une carte des profondeurs. Cette pyramide est ensuite utilisée pour déterminer quels objets sont occultés totalement ou partiellement. L'avantage est que le seuil d'opacité peut être ajusté pour que l'algorithme soit plus ou moins conservatif, voire devienne même approximatif.

Dans cette partie, nous venons de voir que, outre la méthode du Z-Buffer aujourd'hui intégrée dans toutes les cartes graphiques et ne nécessitant pas d'approfondissement, les méthodes basées images offrent un potentiel intéressant notamment quand elles se basent sur des pyramides pour le rejet rapide

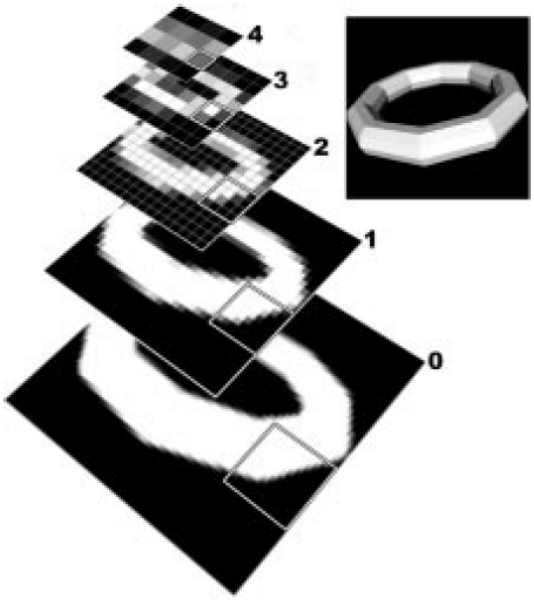

FIGURE 21 – Schéma illustrant la construction de la carte d'occlusion hiérarchique. Chaque niveau voit ses dimensions selon x et y divisées par deux lors du passage au niveau supérieur. Et pour un carré de 4 pixels d'un niveau donné, le niveau supérieur contient un pixel contenant la moyenne des 4 pixels du carré (pour le Z-Buffer hiérarchique, le principe est le même sauf que le pixel contient la valeur maximum des 4 pixels du carrés).

des objets occultés. Le Z-Buffer Hiérarchique[22] lorsqu'il est programmé sur GPU est relativement performant. Les méthodes du HZB[22] et du HOM[23] sont les plus performantes lorsque la scène est donnée du plus proche au plus éloigné, ce qui sera partiellement possible en contrôlant notre méthode de génération.

2.2.5 La visibilité en fonction d'une zone de visibilité

Contrairement à la partie précedente où l'on s'est intéressé à la visibilité à partir d'un point, cette section présente les principales méthodes de visibilité à partir d'une région de l'espace. Elles sont généralement plus rentables en considérant les calculs nécessaires sur plusieurs images consécutives puisque les résultats sont valables sur une zone de l'environnement virtuel et par conséquent sur plusieurs rendus en supposant que la caméra ne bouge pas trop rapidement. Wonka[24] et Koltun[25] proposent deux méthodes basées sur le concept d'objet occultant. Comme nous l'avons vu dans la partie 2.2.3, les objets occultants sont ceux qui contribuent le plus au masquage des autres objets de la scène, de par leur taille et/ou leur position. Koltun propose de considérer les objets occultants pour une zone de visibilité don-

née, puis de construire un objet occultant global en les fusionnant comme illustré figure 22. Une fois cet occultant global construit, on est certain que tout objet derrière cet occultant est masqué, ce qui réduit drastiquement le nombre de tests à effectuer quand la configuration de la scène est appropriée. On construit cet occultant global en agglomérant les objets occultants de la scène comme schématisé dans la figure 23. On considère d'abord un ensemble pré-calculé de bons objets occultants. Puis en considérant la portée de l'ombre de chaque occultant vus par les points extrêmes de la zone de visibilité, on peut déterminer une ombre globale, qui sera celle de l'occultant global. Cette méthode est efficace quand on considère seulement 2 dimensions, ou 2,5 dimensions (c'est-à-dire 2 dimensions avec discrétisation de la troisième). Wonka propose une autre méthode : il discrétise la frontière de la zone de visibilité en différents points et calcule les occultations en chacun de ces points (voir figure 24. Chaque calcul d'occultation donne un ensemble d'objets potentiellement visible (*PVS* pour *Potentially Visible Set* en anglais). Ces PVS sont fusionnés à la fin pour obtenir l'information d'occultation de la zone entière de visibilité.

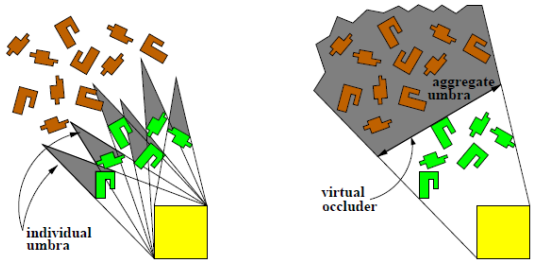

FIGURE 22 – Illustration du concept de l'*occultant global*. Cet occultant global n'existant pas physiquement dans la scène, c'est un occultant virtuel. Une fois cet occultant global construit, on est certain que tout objet derrière cet occultant est masqué, ce qui réduit drastiquement le nombre de tests à effectuer quand la scène est appropriée

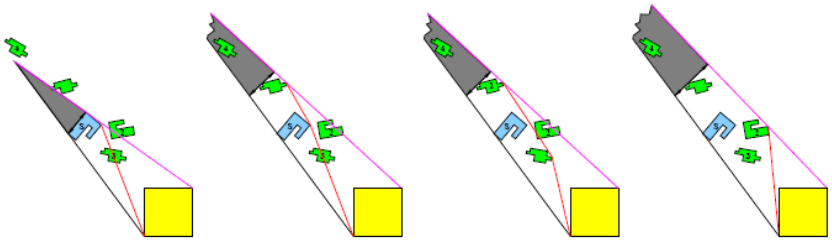

FIGURE 23 – Illustration de la construction pas-à-pas de l'occultant global

Les méthodes de visibilité à partir d'une zone ne sont pas applicables directement dans notre cas, car elles nécessitent des informations issues de pré-calculs, qui semblent difficile à générer en même temps que la géométrie

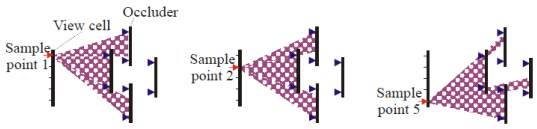

FIGURE 24 – Wonka[24] propose de discrétiser la zone de visibilité et de calculer les occultations en chaque point, puis de les fusionner. A cause de la discrétisation, la méthode doit, pour rester conservative, d'abord réduire (par une mise à l'échelle) les occultants par un ϵ à déterminer en fonction du contexte de discrétisation.

dans notre cas. Mais leurs concepts semblent intéressants, notamment celui d'occultant global de Koltun. En effet, il pourrait par exemple permettre de stopper la génération et les tests pour l'image courante si l'occultant global en cours est assez large et couvre la totalité ou une majeure partie de ce qui est visible à partir du point de vue.

2.2.6 Conclusion sur la visibilité

Entre les classes de méthodes à partir d'un point de vue ou à partir d'une zone, il semble préférable pour notre cas d'utiliser une méthode déterminant la visibilité à partir d'un point de vue, pour éviter les pré-calculs. En cas d'utilisation d'une approche basée objet, il faudra organiser la génération afin de pouvoir produire en même temps les informations permettant de déterminer les bons objets occultants. Si une méthode image est utilisée, il faudra le plus possible générer la scène en s'éloignant du point de vue de la caméra pour que les méthodes du Z-Buffer hiérarchique ou de la carte d'occultation hiérarchique soient les plus efficaces possibles.

L'objectif du stage est de proposer une contribution originale sur le couplage génération procédurale et visibilité, une contribution dans le domaine de la visibilité n'est pas prévue dans l'état actuel des choses faute de temps. Une méthode adéquate issue de la littérature existante devra donc être choisie.

2.3 Conclusion de l'état de l'art

Nous avons dans cet état de l'art introduit les méthodes les plus connues dans les domaines de la génération procédurale et l'élimination des polygones non visibles par la caméra. C'est sur ces méthodes que s'est appuyé le travail effectué pendant le stage.

Un tour de la littérature à propos de ce concept de couplage génération procédurale et fonction visibilité était indispensable, aussi bien d'un point de vue informatif que par souci de travailler sur des bases qui n'ont pas encore été explorées. L'article[26] de Greuter, Parker, Stewart et Leach écrit en 2003 fait état d'une méthode de génération de villes pseudo-infinies. Le

principe est de se baser sur le volume de vision de la caméra, puis de le remplir avec des bâtiments et des routes selon l'endroit où l'on se situe. Les positions et différents paramètres des bâtiments sont déterminés par des nombres aléatoires distribués dans l'espace par une fonction de hachage qui à chaque coordonnées (x, y) associe un nombre pseudo-aléatoire. Les modèles sont générés en même temps que la progression de l'utilisateur dans l'espace. L'algorithme conserve une liste d'élements récemments produits, afin de pas avoir à les recréer dans l'immédiat pendant que l'utilisateur se trouve dans cette même région de l'espace. Ces éléments sont stockés dans une display list d'OpenGL. Après examen des résultats, il apparait que la qualité finale du rendu (voir figure 25) peut être améliorée et les performances chutent relativement rapidement. En effet dès 1000 bâtiments générés, la fréquence d'affichage tombe en dessous de 15 images par secondes. De plus, le plan des routes suit un schéma de grille régulière, ce qui n'offre pas un aspect réaliste et naturel à la ville générée. Enfin, il n'est pas fait mention d'utilisation d'une fonction de visibilité autre que la méthode du volume de vue, ni d'une possibilité de contrôler le niveau de détail dans cet article. Les objectifs visés par le stage apportent donc une réelle contribution par rapport à ce travail déjà publié.

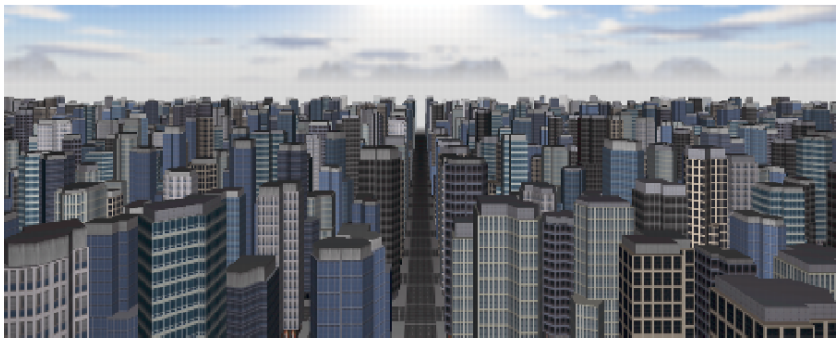

FIGURE 25 – Rendu final du travail effectué par Greuter dans son article[26]

3 Génération de villes par grammaire de forme

Ce stage est à la frontière entre deux domaines : la génération procédurale et l'accélération de rendu. Cette partie est consacrée au premier domaine et présente donc le travail réalisé sur la création automatique de géométrie appliquée aux environnements de type urbain. Une contribution a été apportée sur une méthode de découpage du terrain par grammaire de forme (*shape grammar* en anglais). L'algorithme génère la ville en trois étapes comme indiqué dans la littérature : génération du plan des routes, génération des blocs et parcelles de bâtiments puis génération des bâtiments. Après un rapide rappel sur les shape grammars, une partie est consacrée à chacune de ces phases de l'algorithme. Puis nous terminons cette section par une présentation et une réflexion sur les résultats obtenus.

3.1 Rappels rapide sur les shapes grammars

Comme ils ont été introduits brièvement dans la partie 2.1.1, les shapes grammars[4] sont un formalisme permettant de décrire comment itération après itération, un segment ou un groupe de segment sera transformé en un autre segment ou groupe d'autres segments. Ce procédé permet de créer de la géométrie 2D ou 3D. Dans son approche originelle, il est assez délicat d'appliquer les shape grammars à la problématique de ce stage. En revanche, le concept a été réutilisé en se servant de formes définies se transformant ou générant récursivement d'autres types de formes. Les règles de la grammaire ayant les même parties gauches sont choisies selon une distribution de probabilité. C'est de cette manière que le plan des routes et les bâtiments de l'implémentation proposée ont été générés.

3.2 Génération du plan des routes de la ville

Une grammaire, illustrée graphiquement en annexe figure 26, a été définie en appliquant le concept des shape grammars. Elle comporte actuellement deux types de forme, des triangles quelconques (notés T) et des quadrangles quelconques (notés Q). Huit règles ont été définies, dont deux règles d'arrêts. Pour l'axiome d'initialisation, la génération peut commencer à partir d'un T ou d'un Q. La subdivision continue jusqu'à ce que le polygone courant soit trop petit, dans ce cas la règle d'arrêt correspondante (la règle d'arrêt de Q ou de T selon le type du polygone courant) est invoquée.

Voici la grammaire utilisée (Q désignant un quadrangle et T un triangle, Q_{term} et T_{term} les états terminaux respectifs des formes Q et T) :

$$
\begin{array}{lcl}
Q & \longrightarrow & Q_{term} \\
Q & \longrightarrow & Q\ Q \\
Q & \longrightarrow & T\ Q \\
Q & \longrightarrow & T\ T
\end{array}
\qquad\qquad
\begin{array}{lcl}
T & \longrightarrow & T_{term} \\
T & \longrightarrow & Q\ Q\ Q \\
T & \longrightarrow & T\ Q \\
T & \longrightarrow & T\ T
\end{array}
$$

Toutes les règles de la grammaire sont illustrées graphiquement dans la figure 26.

À chaque itération, une règle est choisie au hasard selon une distribution de probabilité. Pour des raisons d'esthétisme, cette distribution dépend de l'*étirement* du polygone courant. L'étirement e est une fonction définie comme suit (en notant $\mathcal{T}$ l'ensemble des triangles et $\mathcal{Q}$ l'ensemble des quadrangles) :

$$
e : \begin{cases}
\mathcal{Q} \cup \mathcal{T} & \longrightarrow & [0,1] \\
q & \longmapsto & e(q) = \frac{s_c}{s_l},\ \forall\ q \in \mathcal{Q} \\
t & \longmapsto & e(t) = 2 * \frac{r_{ci}}{r_{cc}},\ \forall\ t \in \mathcal{T}
\end{cases}
$$

Avec s_c et s_l respectivement le segment le plus court et le segment le plus long du quadrangle, r_{ci} le rayon du cercle inscrit au triangle et r_{cc} le rayon du cercle circonscrit au triangle.

La figure 27 donne un exemple de distribution pour Q selon un étirement donné, un ensemble plus détaillé des distributions de probabilités est disponible en annexe figure 44.

Des règles additionnelles ont été définies pour améliorer le résultat visuel du plan, et éviter l'applatissement des polygones générés :

- Si la règle oblige à couper une arête en deux (cas en figure 29(a)), par exemple comme $Q \rightarrow QQ$ ou $T \rightarrow TT$ (exception pour la règle $T \rightarrow TQ$ présentée ci-après), alors c'est l'arête la plus longue du polygone qui sera coupée en deux. Le point de coupure est choisi au hasard entre 35% et 65% de la longueur du segment comme illustré dans la figure 28.
- Si la règle oblige à créer une arête qui relie deux sommets (cas en figure 29(b)), actuellement seule la règle $Q \rightarrow TT$ est concernée, alors les sommets les plus proches entre eux sont sélectionnés pour être reliés.
- Pour la règle $T \rightarrow TQ$ (cas en figure 29(c)), le Q généré contient la plus courte arête du triangle courant.

À chaque fois qu'un segment est créé, il est étiqueté selon sa taille comme

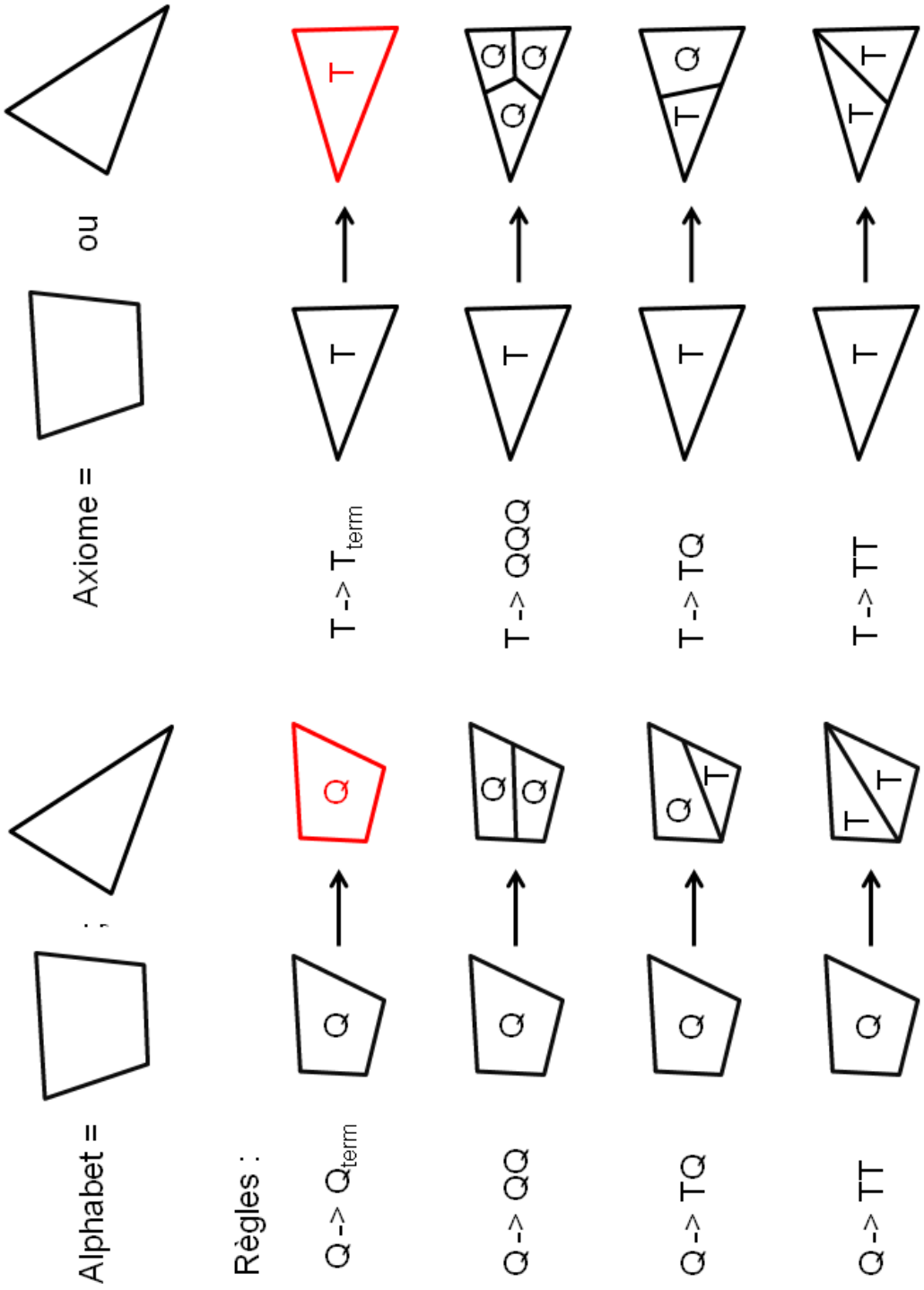

FIGURE 26 – Définition graphique de la grammaire utilisée

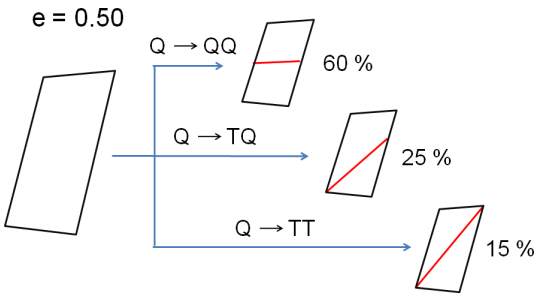

FIGURE 27 – Exemple de distribution de probabilité en fonction de l'étirement du quadrangle. Pour une valeur de l'étirement $e = 0.5$, il y a 60% de chance d'utiliser la règle $Q \rightarrow QQ$, 25% pour la règle $Q \rightarrow TQ$ et 15% pour $Q \rightarrow TT$. Ces probabilités ont pour but de préserver l'esthétisme du plan généré en évitant l'applatissement des polygones au fil des dérivations

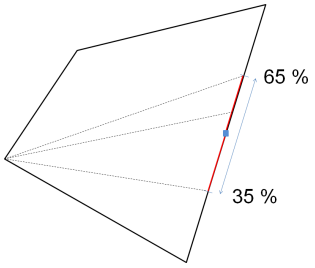

FIGURE 28 – Lorsqu'une règle impose de couper une arête en deux, la plus grande arête est choisie et elle sera coupée au hasard entre 35% et 65% de sa longueur comme illustré ci-dessus

étant de type *avenue* ou *rue*. Cela permet la variation automatique de paramètres comme par exemple la largeur des routes ou le style des bâtiments. Un exemple d'itérations de la grammaire est disponible dans les annexes figure 45. Le résultat final dans l'application est présenté en figure 30.

3.3 Génération des parcelles de bâtiments

Une fois que les routes ont été générées, il faut subdiviser l'espace des Q_{term} et des T_{term} afin de créer les quadrangles qui serviront de parcelles de construction des bâtiments. Actuellement, seul des bâtiments quadrangulaires peuvent être générés.

Dans le cas général, c'est-à-dire si la surface du polygone est suffisament grande, on découpe le bloc en quatres lignes de bâtiments disposées en quadrangle (formant ainsi un anneau rectangulaire). Si le bloc est trop petit, il ne contiendra qu'une seule rangée de bâtiments. On procède tout d'abord à un découpage du bloc par création d'un polygone identique mis à l'échelle inférieure (comme de (a) en (b) dans l'illustration figure 31) pour créer les

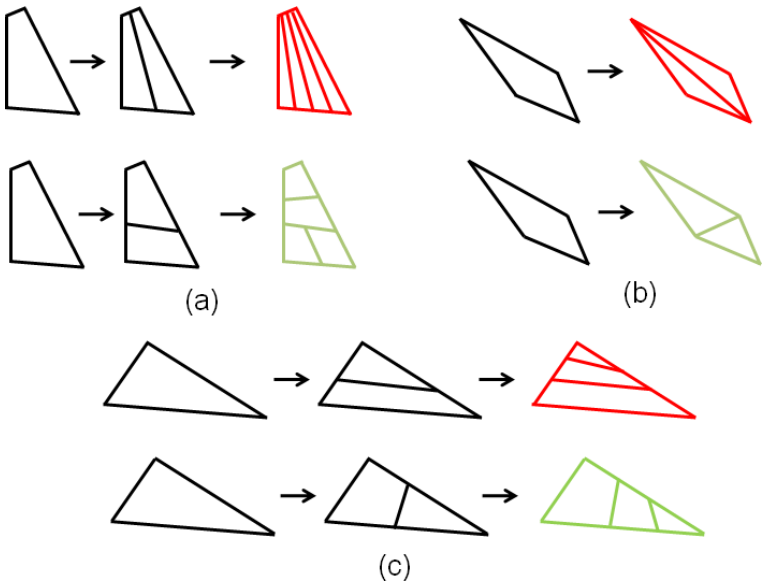

FIGURE 29 – En (a), lors d'une division impliquant de scinder une arête en deux, ne pas couper l'arête la plus longue du polygone peut conduire à un applatissement peu esthétique des polygones générés
En (b), lors d'une division impliquant de relier deux sommets par une nouvelle arête, ne pas relier les deux sommets les plus proches peut conduire au même problème
En (c), lors de l'application de la règle $T \rightarrow TQ$, si le Q généré ne contient pas l'arête la plus courte du triangle courant, alors cela peut entrainer la production de quadrangles trop applatis

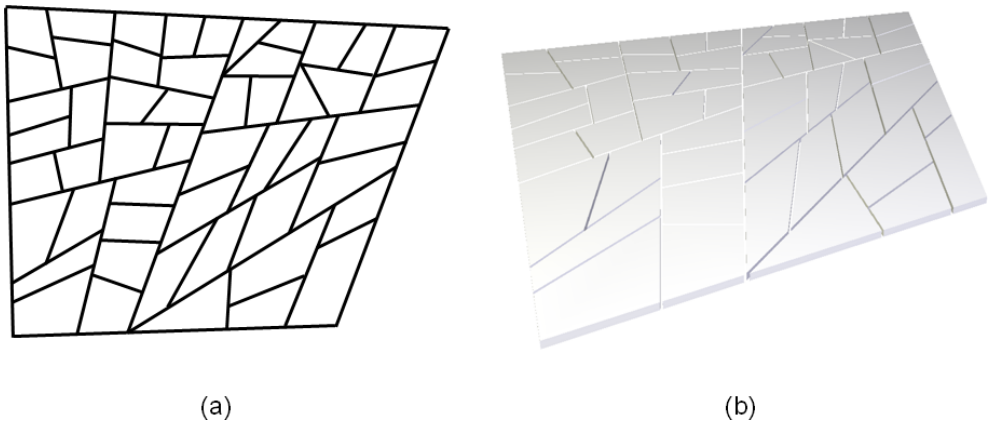

FIGURE 30 – En (a) le résultat final issu de la grammaire de génération et en (b) le rendu correspondant dans le programme

demi-routes (opération appelée *shrink* en anglais), les autres demi-routes correspondantes seront produites par les blocs voisins. On recommence la même opération de découpage par shrinking pour les trottoirs, l'anneau (ou

la ligne) des bâtiments. Dans le cas de découpage en anneau, le polygone qui reste à l'intérieur devient une cour intérieure.

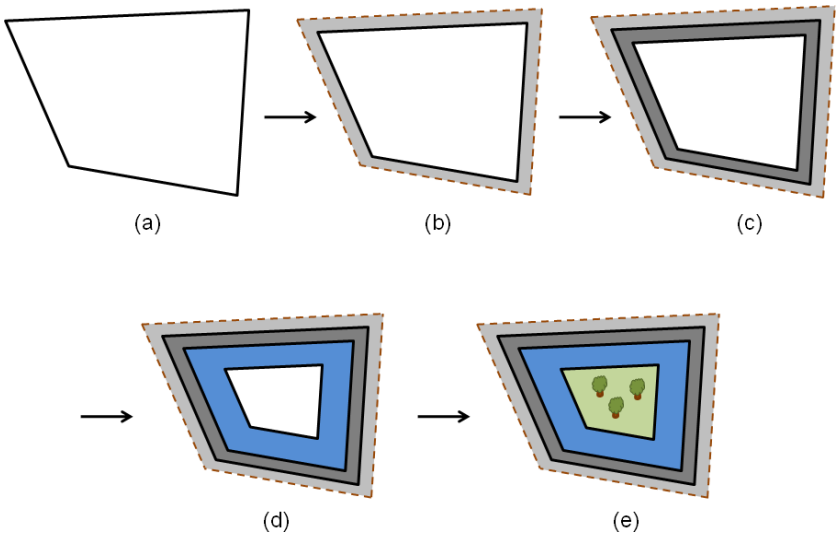

FIGURE 31 – Le bloc de départ (a) est subdivisé par rétrécissement une première fois pour générer les 4 demi-routes (b) (les 4 autres demi-routes seront générées par les 4 blocs voisins), puis une deuxième fois pour le trottoir (c). Ensuite, l'anneau (d) ou la ligne destiné(e) aux bâtiments est produit(e). En cas de découpage en anneau pour les bâtiments comme dans l'exemple ci-dessus, le polygone interne restant devient la cour intérieure (e) de la parcelle. En cas de découpage en ligne pour les bâtiments, il n'y a pas de cour intérieure.

Puis le programme définit précisément les parcelles de chaque bâtiment comme illustré dans 32. On projette chaque coin du polygone intérieur orthogonalement sur les deux bords adjacents du polygone extérieur. Une fois les coins créés, les espaces restants entre les coins sont divisés orthogonalement avec une largeur aléatoire comprise entre une largeur minimum et maximum. Il arrive que l'espace restant entre deux coins ne soit pas assez large pour accueillir un bâtiment d'une largeur convenable (définie par un seuil), dans ce cas les bâtiments de coin sont élargis et se partagent l'espace restant (cf figure 33).

3.4 Génération des bâtiments

La production des parcelles des bâtiments terminée, le programme passe alors à l'extrusion des bâtiments. Elle se fait là encore en utilisant une grammaire de forme, analogue à celle utilisée par Müller et Wonka[10], définie figure 34.

L'axiome de la grammaire est *Parcelle*. Les états terminaux sont en majuscule. Ils indiquent qu'il y aura un dessin de l'élément en question. Les

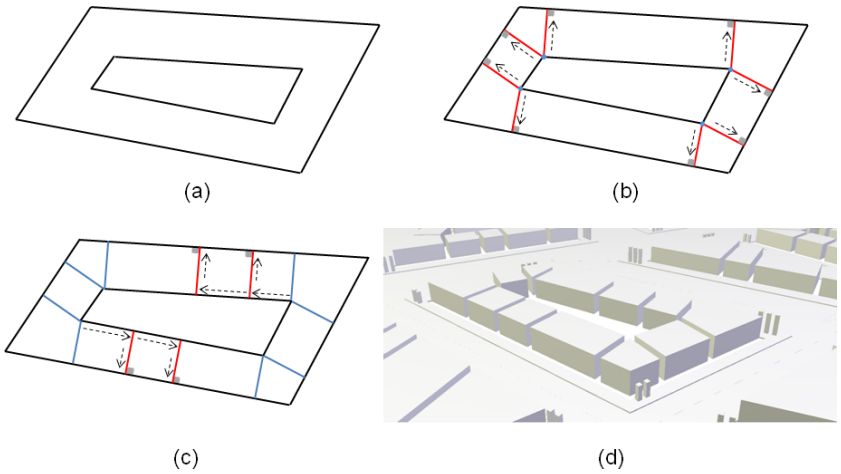

FIGURE 32 – Pour la génération des parcelles des bâtiments, on projette orthogonalement chaque coin du quadrangle interne sur les deux bords adjacents du polygone externe comme en (b). Ensuite, les portions restantes sont découpées avec une largeur choisie au hasard avec un seuil minimum, comme représenté en (c). Le résultat graphique correspondant est en (d).

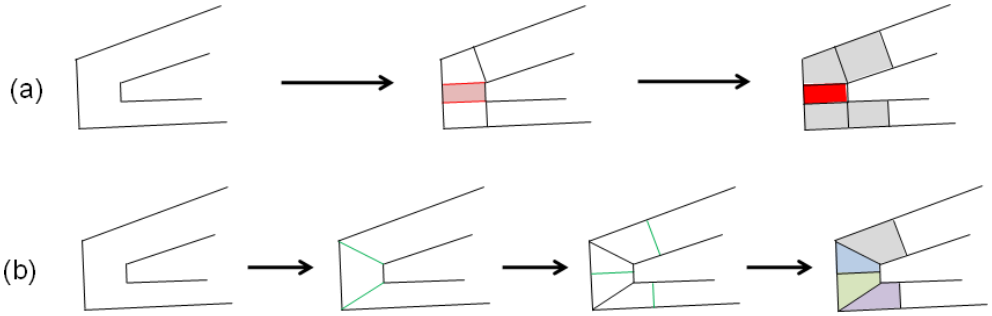

FIGURE 33 – En suivant le découpage classique (a), si une arête est trop petite, le bâtiment entre les deux coins ne sera pas assez large pour pouvoir y placer des éléments de façade. Une solution en (b) est donnée, consistant à diviser chaque coin en deux et à partager entre les deux coins l'espace restant sur l'arête trop petite.

choix des embranchements dans les règles $MurPremierEtage$ et $MurEtage$ se font sur des critères d'espace disponible sur la façade, afin que celle-ci ne soit pas trop surchargée. Si le mur est caché (entre deux bâtiments), alors on n'y ajoute pas d'élément.

Au total quatre styles de bâtiments ont été définis (voir modèles figure 35) : style *haussmanien*, style *immeuble*, style *pavillon* et style *maison de banlieue*. Le style d'un bâtiment et son nombre d'étages sont définis par une distribution de probabilité dépendant du type de voie (rue ou avenue) où ils sont implantés.

$$
\begin{aligned}
Parcelle &\longrightarrow PremierEtage \\
PremierEtage &\longrightarrow MurPremierEtage\ MurPremierEtage \\
&\quad\ MurPremierEtage\ MurPremierEtage\ SuiteEtage \\
MurPremierEtage &\longrightarrow MUR \mid MUR\ Porte \mid MUR\ Porte\ Fenetres \\
SuiteEtage &\longrightarrow Etage \mid Toit \\
Etage &\longrightarrow MurEtage\ MurEtage\ MurEtage\ MurEtage\ SuiteEtage \\
MurEtage &\longrightarrow MUR \mid MUR\ BALCON \mid MUR\ Fenetres \mid \\
&\quad\ MUR\ BALCON\ Fenetres \\
Toit &\longrightarrow TOIT \mid TOIT\ CHEMINEE \\
Porte &\longrightarrow PORTE \\
Fenetres &\longrightarrow FENETRE \mid FENETRE\ Fenetres
\end{aligned}
$$

FIGURE 34 – Grammaire de forme pour la génération des bâtiments

3.5 Résultats partiels et conclusion

Après implémentation, pour une ville d'environ 4000 habitants (en comptant une moyenne de 3 habitants par logement disponible) composée de 563 bâtiments répartis sur $80km^2$, le temps de génération est de 718 ms et le temps d'affichage de la totalité est de 109 ms. Ce qui empêche toute utilisation en temps réel dans l'état actuel du programme. La deuxième partie appliquant une fonction de visibilité prend donc tout son intérêt. Cependant, bien que ce résultat semblait évident à première vue, le travail de génération et de rendu en entier à été nécessaire d'un point de vue pédagogique afin de concentrer uniquement la difficulté sur l'élaboration et l'implémentation d'une méthode de génération procédurale. Et enfin, cela permettra de chiffrer l'apport de la fonction de visibilité.

Pour de futurs développement, il est possible d'apporter plusieurs améliorations à l'algorithme de génération. Au niveau de la génération des routes, il serait intéressant de rendre le code concernant les triangles T et les quadrangles Q général afin de pouvoir ajouter de nouvelles formes plus facilement (par exemple des pentagones, hexagones, ...). Il serait également possible de sophistiquer la grammaire de génération pour qu'elle puisse générer des types de quartiers avec des modèles de subdivision différents selon que ce soit des quartiers résidentiels, des zones commerciales, etc. Au niveau des parcelles de bâtiments, il faudrait introduire de nouveaux schémas de découpage, que l'unique schéma déjà implémenté qui reste somme tout très basique. Enfin au niveau des bâtiments, il reste possible de créer de nouveaux styles ou d'améliorer ceux déjà existants, en ajoutant par exemple des modèles 3D pour les greffer sur la géométrie générée de manière procédurale pour donner plus de réalisme au résultat final.

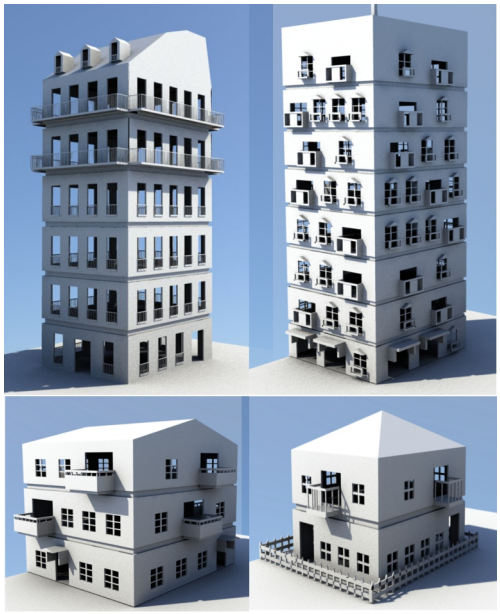

FIGURE 35 – Aspect graphique des quatres styles définis. En haut à gauche, le style *Haussmannien*, en haut à droite le style *immeuble*, en bas à gauche le style *pavillon* et en bas à droite le style *maison de banlieue*

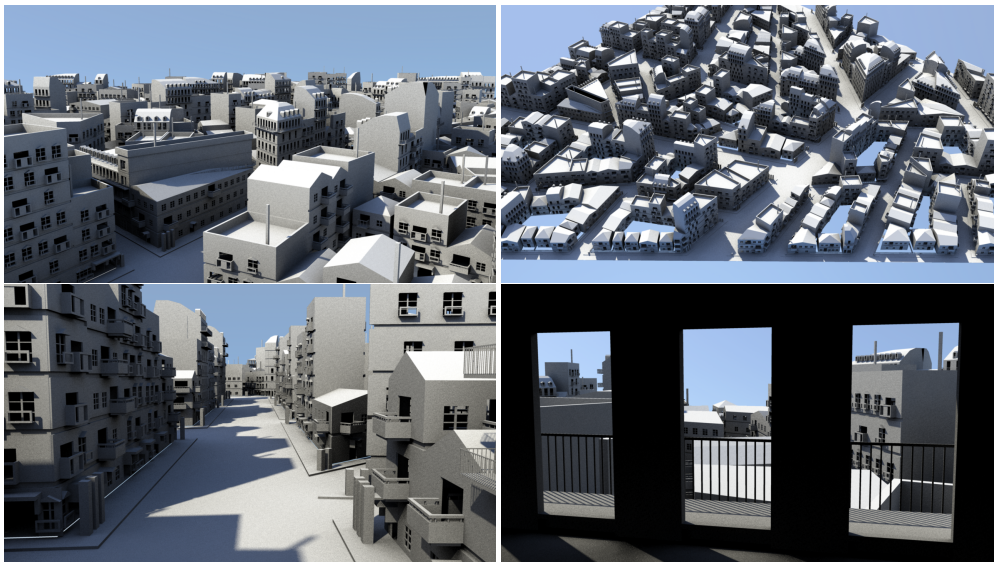

FIGURE 36 – Résultats de la ville générée

4 Génération en fonction du point de vue

Cette partie du rapport est consacrée au deuxième domaine de ce stage : l'accélération du rendu. Aucune contribution n'a été apportée à ce domaine, mais le couplage des deux aspects de génération procédurale et d'accélération de rendu est une approche originale en elle-même. La section précédente nous a permis de générer 563 bâtiments en 827 ms. Il est donc nécessaire d'utiliser l'accélération pour espérer obtenir une génération en temps réel. Dans cette partie, un rappel des objectifs et de leurs mises en oeuvre est d'abord donné, puis ces mises en oeuvre sont détaillées une à une, à savoir : la mise en place de la carte de graines, la génération en fonction du volume de vision et l'implémentation du Z-Buffer hiérarchique.

4.1 Rappel des objectifs

L'objectif final du stage est d'effectuer la génération et le rendu de la scène selon le point de vue donné en 16 ms, correspondant à la fréquence d'affichage de 60 images par secondes (ou 60 *FPS* pour *Frame Per Second* en anglais). Pour atteindre cet objectif, l'algorithme initial de génération a été modifié pour permettre le couplage avec les différentes techniques d'accélération de rendu. Le principe est donc d'ajouter un paramètre, à savoir le point de vue, et de le prendre en compte lors de la production de la géométrie. Comme il a été exposé dans l'état de l'art partie 2.2.3, la méthode du volume de vision est très efficace pour supprimer une grande partie des polygones non visibles, il s'agira donc de l'exploiter au mieux lors du couplage. Les éléments présents dans le volume de vision peuvent être encore relativement nombreux et une partie d'entre-eux peut encore être éliminée. Une autre méthode d'accélération de rendu plus fine doit être choisie. Dans la conclusion de l'état de l'art, il apparaît que les méthodes de la carte d'occultation hiérarchique[23] et du Z-Buffer hiérarchique[22] semblent être les plus appropriées. Pour des raison de simplicité d'implémentation, la méthode du Z-Buffer hiérarchique a été choisie. Afin qu'elle soit la plus efficace possible, il est important que les objets les plus occultants soient traités en premier, c'est-à-dire en général les objets les plus proches de la caméra. L'algorithme de génération devra donc être modifié pour que les éléments générés le soient en s'éloignant du point de vue.

4.2 Ajout de la carte de graines

Afin que l'algorithme de génération puisse générer plusieurs versions différentes de villes, plusieurs décisions sont prises et des valeurs de paramètres sont choisies selon une distribution aléatoire. L'algorithme utilisé est en fait *pseudo-aléatoire*, dans le sens où les valeurs générées ne sont pas prévisibles mais elles restent déterministes. Ce déterminisme est fixé par un entier de

départ appelé *graine*. Ainsi pour deux tirages à deux moments différents, si l'algorithme a été initialisé par la même graine, alors les valeurs produites seront exactement les mêmes et dans le même ordre. Ce concept pose en fait un problème dans notre cas. En effet, en imaginant deux parcours différents dans la ville (deux chemins différents), les valeurs de l'algorithme pseudo-aléatoire seront les mêmes mais distribuées différement dans l'espace (voir schéma 37). Donc si par exemple l'utilisateur revient sur ses pas les bâtiments auront un aspect différent puisqu'il est très probable qu'un paramètre ait été déterminé avec une valeur différente du premier passage.

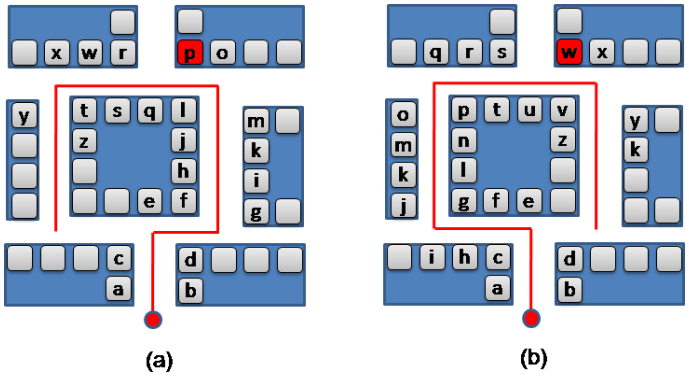

FIGURE 37 – Pour deux parcours différents donnés, la distribution des nombres aléatoires (notés, pour une séquence ordonnée, de a à z) sera différente à cause de la conservation de la séquentialité des tirages. En particulier, dans le cas où le nombre aléatoire est utilisé pour choisir le style de bâtiment, on a par exemple le bâtiment en rouge qui sera de type probablement différent selon que l'on emprunte le parcours (a) ou le parcours (b). La mutation se produit aussi si l'on revient sur ses pas. D'où l'intérêt de créer une carte de graines.

Pour éviter ces mutations, une carte de graines a été mise en place sous la forme de deux fonctions *int seedMap(float x, float y)* et *int seedMap(float x, float y, float z)*, qui pour une position 2D ou 3D renvoient une graine dépendant uniquement de la position et plus de l'ordre du tirage. Afin d'économiser de la mémoire, la carte de graine est elle-même procédurale, c'est-à-dire qu'elle n'est pas pré-générée sous forme de tableau, mais produite lors de l'appel (comme la technique présentée dans l'article de Greuter[26]). Les deux fonctions reposent sur un entier général qui est la graine globale du programme (choisie avant le lancement du programme). En pratique, en prenant comme exemple le choix d'une règle de la grammaire de forme pour générer le plan des routes, le choix sera effectué en fonction de la graine présente à la position du sommet 0 du quadrangle courant. Actuellement, la carte de graines est utilisée pour :

- le choix des règles de subdivision des polygones pour le plan des routes ($Q \longrightarrow QQ$, $Q \longrightarrow TQ$, ...)

- le choix de la largeur des bâtiments lors du découpage des blocs en parcelles
- le choix des types des bâtiments et de leurs nombres d'étages (influencés par le type de rue)
- la détermination de la présence d'éléments sur les façades (comme les balcons, les fenêtres, ...)

4.3 Génération en fonction du volume de vision

La méthode d'élimination des polygones hors du volume de vision (ou *VFC* pour *View Frustum Culling* en anglais) a été présentée dans l'état de l'art, partie 2.2.3. Le volume de vision est constitué de 6 plans : plan haut, plan bas, plan gauche, plan droit, plan lointain et plan proche. Tout élément ne se trouvant pas entre ces 6 plans sera éliminé. Dans l'application, chaque élément susceptible d'être généré est capable de calculer sa boîte englobante, et celle-ci est donc testée avec le volume de vision. Cette méthode a été réellement couplée dans le processus de génération dès la génération des routes. En effet, comme expliqué sur les schémas 38 (pour le plan des routes) et 39 (pour la division des blocs en parcelles), si un quadrangle ou triangle n'est pas dans le volume de vision, alors il est écarté du processus de subdivision (jusqu'à sa réinitialisation pour dessiner l'image suivante du programme). Ainsi, les polygones rejetés ne seront pas subdivisés jusqu'à devenir des blocs ni des parcelles, ni des bâtiments : de nombreux tests et calculs sont ainsi évités. Une optimisation proposée dans l'article d'Assarsson[18] a été utilisée : lors des tests, on sépare les boîtes englobantes totalement à l'intérieur du volume de vision et les boîtes englobantes en intersection avec celui-ci. Si une boîte englobante est totalement à l'intérieur du volume, alors les sous-objets inférieurs hiérarchiquement qui sont par définition inclus spatialement dans cette boîte englobante n'auront pas besoin d'être testés. On continue donc à tester seulement les objets intersectés avec le volume de vision, ce qui permet d'éviter de nombreux calculs.

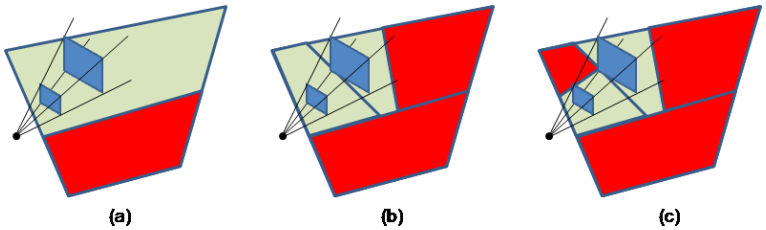

FIGURE 38 – Pour éviter la génération inutile, on ne subdivise pas les polygones du plan des routes qui sont hors du volume de vision. Les polygones en rouges sont ceux qui ont été rejetés par le test du volume de vision. Au final on obtient seulement les polygones nécessaires.

Le test du volume de vision n'est plus rentable quand on se trouve trop

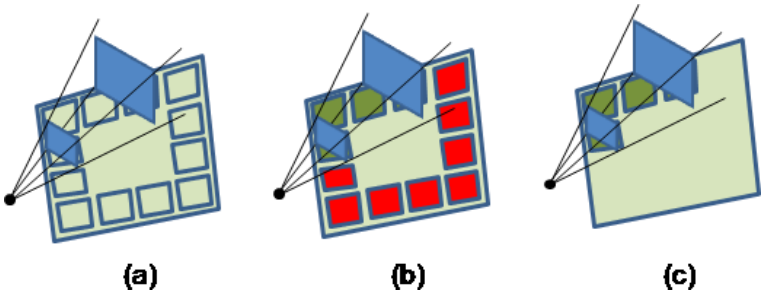

FIGURE 39 – Pour éviter la génération inutile, on ne subdivise pas les parcelles des bâtiments qui sont hors du volume de vision. Les parcelles en rouges sont celles qui ont été rejetées par le test du volume de vision.

bas dans la hiérarchie de génération. Par exemple, il est moins coûteux de générer directement une fenêtre même si elle n'est pas visible plutôt que de tester d'abord sa boîte englobante pour ensuite la dessiner ou non selon le résultat. Par essais successifs, il s'est averé que le niveau le plus bas de la hiérarchie pour lequel le test était rentable est au niveau des façades d'un étage. Les gains obtenus par cette méthode permettent d'éliminer au moins 50 % de la géométrie. Le test est présenté sous forme graphique en figure 40.

4.4 Z-Buffer hiérarchique au sein du frustum de vue

Après l'élimination d'une partie de la géométrie par la méthode du volume de vision, il s'agit d'utiliser un algorithme plus fin pour supprimer les éléments occultés et donc invisibles pour l'utilisateur. L'approche du Z-Buffer hiérarchique[22], introduite dans l'état de l'art partie 2.2.4 a été choisie. Son fonctionnement est d'abord rappelé puis les détails de son implémentation sont expliqués.

4.4.1 Rappel du fonctionnement du Z-Buffer hiérarchique

Le Z-Buffer hiérarchique est une approche pour accélérer le très connu algorithme du Z-Buffer. En effet, le Z-Buffer ne permet de ne tester la propfondeur qu'au niveau du pixel tandis que le Z-Buffer hiérarchique permet de tester directement les objets. Le principe du Z-Buffer consiste à stocker les valeurs des profondeurs des objets dessinés dans un tableau représentant l'écran. A chaque fois qu'on objet doit être dessiné, on teste quels pixels de sa projection sur l'écran sont visibles ou masqués en comparant les valeurs des profondeurs. On dessine ensuite les pixels visibles. Pour chaque pixel de dessiné, on met à jour le Z-Buffer avec les nouvelles valeurs de profondeurs (plus proches que celles déjà stockées). Ainsi de suite jusqu'à avoir parcouru l'ensemble des objets à tester.

Le Z-Buffer hiérarchique établit une pyramide hiérarchique de Z-Buffers (voir le schéma simplifié en figure 41). Le niveau le plus bas de la hiérarchie

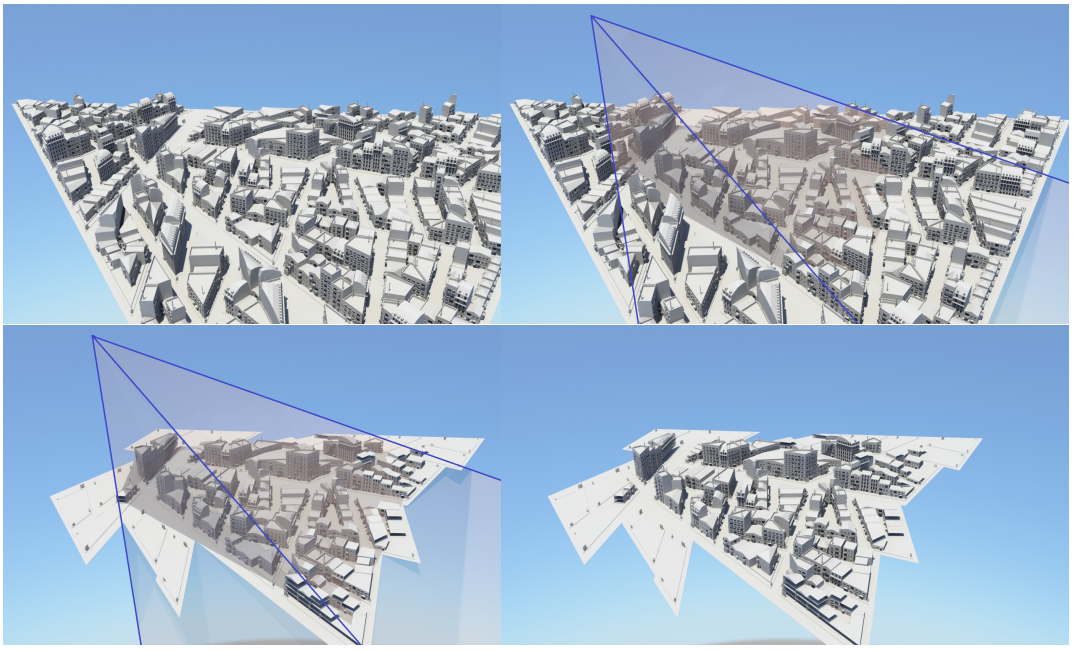

FIGURE 40 – En haut à droite, la ville entière a été générée et en haut à gauche on a ajouté le volume de vision : il a beaucoup de géométrie inutile. En bas à gauche, la même ville avec le même point de vue avec application de la méthode du volume de vision. En bas à droite, le résultat final sans le volume de vision dessiné.

est un Z-Buffer classique comme décrit ci-avant. Chaque niveau $n+1$ est un tableau dont les deux dimensions a_{n+1} x b_{n+1} sont celles du niveau n divisées par deux, c'est-à-dire que $a_{n+1} = a_n/2$ et $b_{n+1} = b_n/2$. En effet, pour un carré de quatre pixels du tableau au niveau n, on conserve la valeur la plus éloignée de la caméra (ayant son Z le plus grand) pour la stocker dans la case correspondante au niveau $n+1$. La construction s'arrête lorsque l'on atteint le niveau où le tableau est de taille 2x2 (où un pixel du tableau représente en fait un quart de l'écran). Lors des tests, ceci permet de décider rapidement si un objet est masqué par d'autres. On se place au niveau où la zone représentée par un pixel du tableau est de la même taille que la projection de la boîte englobante. On compare le Z de la boîte englobante testée avec la valeur de Z stockée dans le tableau. Si la valeur de Z du polygone testé est supérieure à la valeur stockée, alors le polygone est masqué et n'a donc pas besoin d'être affiché. Si la valeur de Z du polygone testé est inférieure à la valeur stockée, alors on descend dans le niveau inférieur $n-1$ pour comparer le Z a un endroit plus précis. On continue de descendre jusqu'à trouver un niveau où la boîte englobante est masquée, ou alors jusqu'à arriver à la dernière couche et obtenir que le Z de la boîte englobante testée soit encore plus petit que le Z stocké. Ce dernier cas indique que le polygone doit être affiché et la pyramide

de Z est ensuite mise à jour. Cette méthode permet d'écarter rapidement les polygones non visibles mais oblige à parcourir tous les niveaux si un objet se trouve être finalement visible.

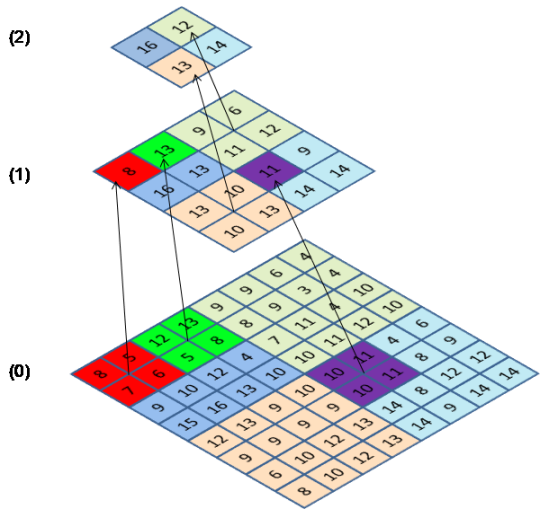

FIGURE 41 – Exemple avec un Z-Buffer hiérarchique simplifié. Le niveau (0) contient un Z-Buffer classique, les valeurs représentant la profondeur de l'objet déjà dessiné. Le niveau (1) est construit en gardant la valeur de profondeur la plus grande parmi un carré de quatre pixels du buffer en (0) (par exemple pour les carrés sur fond rouge, vert et violet). Puis le niveau (2) est construit sur le même principe à partir du niveau (1). La pyramide s'arrête quand la dimension du buffer est de 2x2.

4.4.2 Implémentation en hardware

Cet algorithme a été implémenté sur la carte graphique afin de profiter de la puissance de calcul du GPU pour les tests de visibilité. Les concepts de l'implémentation sur carte graphique sont relativement simples. Chaque frame, un premier groupe d'objets considérés comme bons occultants, c'est à dire les plus proches de la caméra, est dessiné directement sans test de visibilité. Puis la hiérarchie est construite sur GPU à partir de ce premier ensemble d'objets. La pyramide est ensuite récupérée. On prend ensuite les autres élements générés et on teste leurs englobants sur CPU avec le Z-Buffer hiérarchique obtenu juste avant. S'ils sont visibles, on stocke leurs références dans un tableau intermédiaire pour les dessiner après. Quand le nombre d'objets dans le tableau est suffisant, on transfère et dessine tous les éléments visibles en même temps afin d'amortir le coût du transfert. La pyramide est à nouveau mise à jour et récupérée. On continue ainsi de suite jusqu'à avoir traité tous les éléments de la scène en les envoyant par lots au GPU. Dans l'état actuel, cet algorithme n'a pas encore été totalement

intégré au programme, mais des tests ont été conduits sur un programme indépendant du programme principal. Ces tests se sont avérés très concluants et apporteront une accélération sensible à la génération de la ville, même si cet apport n'est pas encore chiffrable pour le moment. Les résultats chiffrés pour cette partie du stage seront donc présentés lors de la soutenance.

4.4.3 Génération par éloignement croissant de la caméra

Pour rendre la méthode du Z-Buffer hiérarchique réellement efficace, il faut que les objets testés le soient par éloignement croissant de la caméra. En effet, un buffer de profondeur est maintenu à jour pour tester un à un les objets et le pire des cas correspond à la situation où les objets les plus loins sont d'abord testés : le buffer serait mis à jour avec des valeurs de profondeur très grandes, et lors du test d'objets plus proches, ces derniers seraient détectés comme visibles puis dessinés, puis le buffer serait à nouveau mis à jour. Les objets les plus loins auraient donc été dessinés pour rien. L'efficacité de cet algorithme dépend donc grandement de l'ordre de dessin (bien qu'il ne soit pas toujours possible de le contrôler selon les applications). L'algorithme de génération a donc été modifié dès la génération des routes pour que lors de la division d'un quadrangle A en deux quadrangles B et C (règle $Q \longrightarrow QQ$), la subdivion du quadrangle B ou C le plus proche de la caméra soit lancée en premier (auparavant le choix avait été fixé arbitrairement). Pour la subdivision des blocs en parcelles, la parcelle la plus proche de la caméra est détectée puis le lancement des subdivisions est donné en s'éloignant de la caméra par alternance droite-gauche par rapport à la parcelle de départ (voir schéma explicatif 42(a)).

Même si la méthode pour procéder à l'éloignement croissant de la caméra reste grossière et peu précise (voir cas pathologique du schéma figure 42(b)), elle a l'avantage d'être très peu coûteuse. Un essai a été réalisé en tentant de stocker les différents objets à subdiviser dans des listes représentant des intervalles de distances, puis en les évaluant en commençant par la liste la plus proche. Mais les copies et la gestion des listes engendrent un ralentissement important du programme.

5 Résultats et discussion

La première partie du stage, portant sur la génération procédurale de ville, a été menée à son terme et a donné des résultats graphiquement intéressants (voir rendus figure 43). Le programme réalisé permet en lui-même à partir d'une graine de départ (pour initialiser le moteur pseudo-aléatoire) et d'un triangle ou quadrangle initial, de générer une ville entière sans intervention de l'utilisateur. Il utilise les grammaires de formes pour générer le plan des routes et les bâtiments. La grammaire inclut pour l'instant deux types de polygones : le triangle et le quadrangle. Dans le modèle généré, on peut

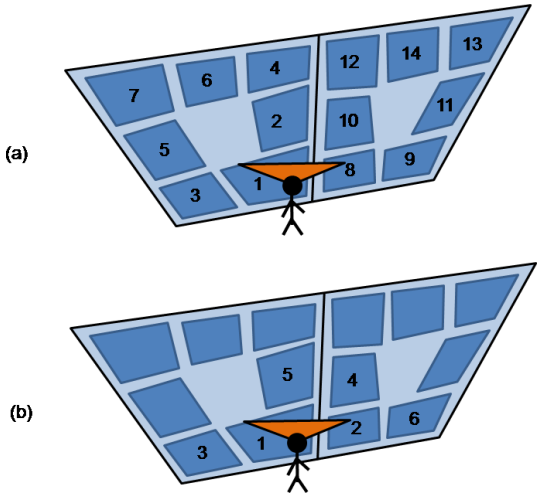

FIGURE 42 – En (a), la méthode de subdivision actuellement implémentée. La parcelle la plus proche de l'utilisateur est détectée, puis on génère les bâtiments en alternance droite-gauche, la numérotation indique l'ordre de génération. Cette méthode de génération n'assure pas de toujours obtenir une génération en s'éloignant strictement de la caméra. En (b), la numérotation indique l'ordre qu'il aurait fallu appliquer pour s'éloigner strictement de la caméra. Comme il a été expliqué, cette méthode est trop coûteuse pour être rentable.

distinguer quatres styles différents de bâtiments, deux types de voies (rues et avenues) et une certaine quantité de décorations qui sont générées procéduralement aussi. Pour la génération d'environ 560 bâtiments, l'algorithme a besoin d'environ 710 ms. Une contribution a été apportée avec l'implémentation d'une méthode à base de grammaire de formes pour la génération du plan des routes.

La deuxième partie du stage portait sur le couplage de l'algorithme de génération de la ville avec des méthodes de visibilité, dans le but de limiter au maximum la génération de géométrie inutile (dans le sens non visible par l'utilisateur). Après avoir mis en place une carte de graines pour éviter les mutations lors du parcours de la ville, la méthode du volume de vision a été couplée à la génération du plan des routes. Afin d'améliorer l'efficacité d'une méthode basée image, le processus de génération a été adapté pour que la géométrie soit générée par éloignement croissant de la caméra. Les gains obtenus, variables selon le point de vue, sont d'au moins 50%. Enfin, afin d'éliminer les bâtiments encore occultés par d'autres au sein du volume de vue, le Z-Buffer hiérarchique a été testé dans un programme indépendant au programme principal et a donné des résultats concluants qui devraient accélérer sensiblement la génération. Son intégration dans le programme principal permettra de présenter des gains chiffrés lors de la soutenance.

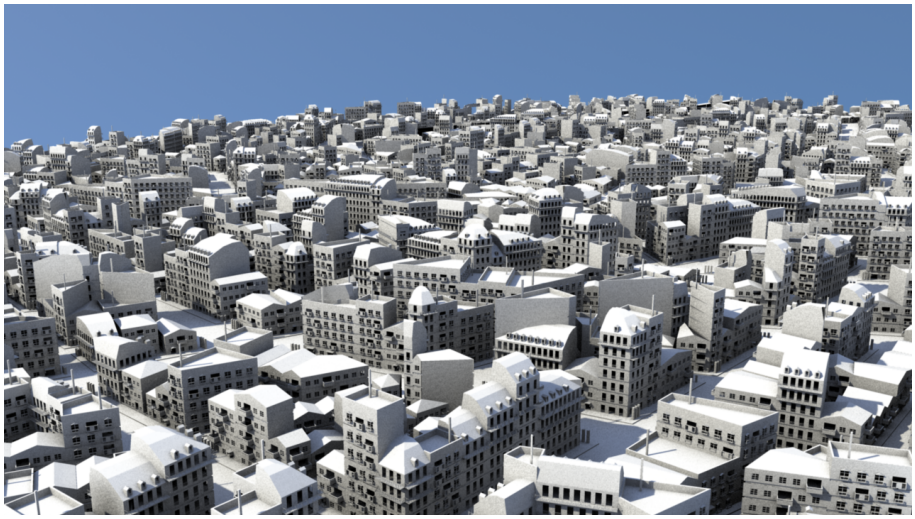

FIGURE 43 – Rendu final de la ville générée

6 Conclusion et travaux futurs

L'objectif du stage était d'explorer le domaine du couplage génération procédurale et fonction de visibilité. Les résultats obtenus jusqu'ici sans l'implémentation du Z-Buffer hiérarchique semblent encore loin du temps réel, mais restent peu représentatifd étant donné que le gain apporté évitera une grande partie de la génération des bâtiments et donc de leurs façades, qui est le goulet d'étranglement du programme. De plus, la direction choisie pour le stage est de recommencer en entier la production à chaque image, pour ne rien stocker en mémoire. Aucun modèle 3D n'a été greffé sur la géométrie existante, alors que la génération des décorations des fenêtres et des balcons est réellement coûteuse. Il semble donc raisonnable de juger sur la possibilité d'obtenir du temps réel lorsque le programme aura été amélioré. Enfin, même si le temps réel est obtenu, le concept est encore loin d'être applicable dans l'industrie. En prenant l'exemple des jeux vidéos, quand les modèles sont générés hors-ligne, les phases de pré-calcul permettent de pré-générer des informations sur le rendu (éclairage, textures, ...) mais également sur des éléments propres au jeu (intelligence artificielle, éléments du scénario, ...). Il resterait donc encore à développer cet aspect pour que ce concept soit utilisé par les studios de jeux vidéos. Les résultats seraient néanmoins l'ouverture d'une voie pour cette industrie.

Les pistes de poursuite du sujet sont nombreuses. Du côté de la génération procédurale, plusieurs sophistication de l'algorithme sont envisageable. Le premier serait d'introduire les polygones génériques pour plus de diversité, notamment lors de la division du plan des routes. Il serait intéressant de mettre en place plusieurs grammaires selon les échelles de génération :

par exemple avoir une grammaire pour découper en arrondissement, puis une autre pour découper en type de quartiers (commerciaux, résidentiels, industriels, ...), puis d'autres adaptées selon le type de quartier. Le découpage des blocs en parcelle pourrait s'étendre et proposer d'autres schémas de division. Une amélioration de taille qui permettrait d'accélérer la génération serait de mettre en place des modèles 3D *étiquetés* : en prenant l'exemple d'une fenêtre, il serait possible d'avoir un modèle générique avec certaines de ses arêtes étiquetées comme étant extensibles (sans dénaturer le modèle) avec des limites inférieures et supérieures. Ce genre de modèles étiquetés éviterait de produire jusqu'à un niveau de détail trop précis comme les décorations devant les fenêtres ou garde-fous des balcons, dont la génération est très longue et multipliée par autant d'éléments sur les façades. La mise en place de textures et la génération des intérieurs sont encore deux autres grandes pistes de poursuite.

Du côté de la visibilité, on pourrait imaginer d'autres méthodes tirant un plus grand profit de la connaissance *a priori* des mécanismes du processus de génération. La génération en s'éloignant des routes, au lieu de s'éloigner de la caméra, semble être plus efficace pour choisir de bons occultants. En effet, les meilleurs bâtiments occultants sont ceux qui sont au bord des routes. Il est également envisageable d'essayer d'autres méthodes existantes dans la littérature pour comparer les résultats et choisir l'approche la plus rapide.

Annexes

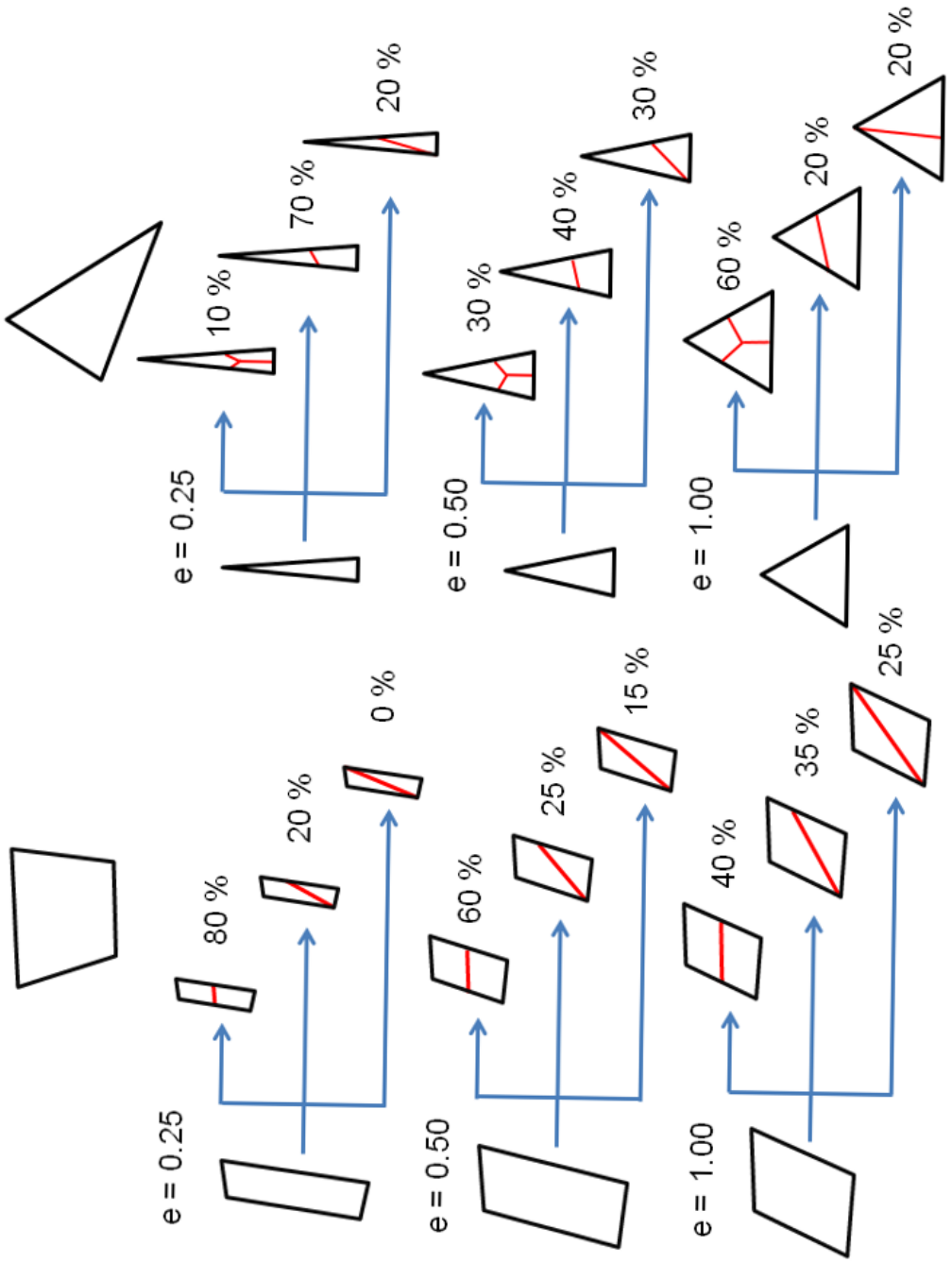

FIGURE 44 – Détails des distributions de probabilités des règles concernant Q et T pour des valeurs d'étirement types. Pour les autres valeurs une extrapolation est réalisée

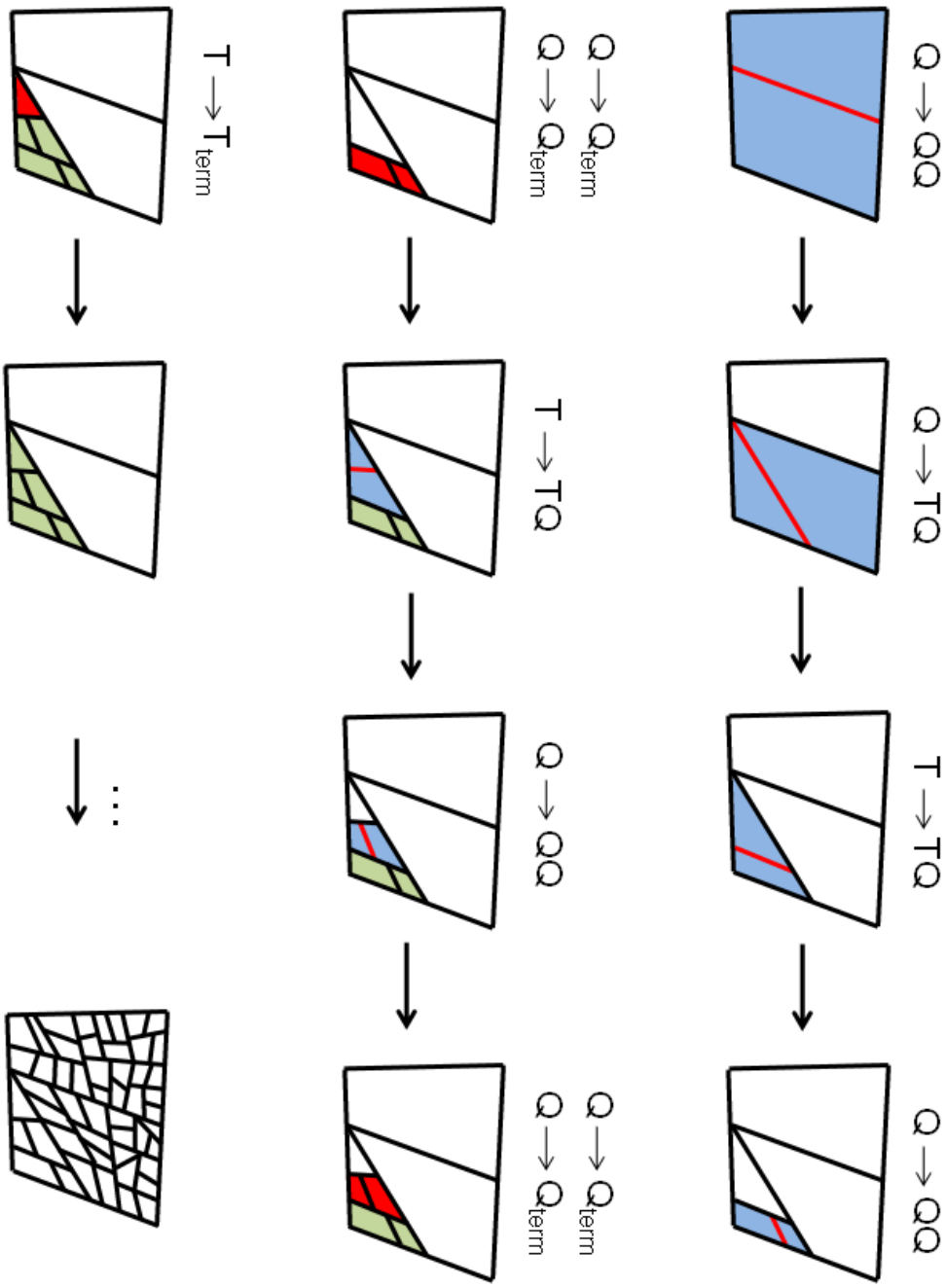

FIGURE 45 – Exemple de quelques itérations de la grammaire de forme pour générer le plan des routes

Références

[1] A. Lindenmayer, "Mathematical models for cellular interactions in development," *Journal of Theoretical Biology I & II*, vol. 18, no. 3, pp. 280–315, 1968.

[2] "http ://el.media.mit.edu/logo-foundation/."

[3] J.-E. Marvie, J. Perret, and K. Bouatouch, "The fl-system : a functional l-system for procedural geometric modeling," *The Visual Computer*, vol. 21, pp. 329–339, 2005.

[4] G. Stiny, "Introduction to shape and shape grammars," *Environment and Planning B*, vol. 7, no. 3, pp. 343–351, 1980.

[5] C. A. Vanegas, D. G. Aliaga, P. Wonka, P. MÃ$\frac{1}{4}$ller, P. Waddell, and B. Watson, "Modeling the appearance and behaviour of urban spaces," *Computer Graphics Forum*, vol. 29, no. 1, pp. 25–42, 2010.

[6] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *SIGGRAPH '01 : Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 301–308, 2001.

[7] G. Chen, G. Esch, P. Wonka, P. Müller, and E. Zhang, "Interactive procedural street modeling," *ACM Trans. Graph.*, vol. 27, pp. 103 :1–103 :10, August 2008.

[8] G. Kelly and H. McCabe, "Citygen : An interactive system for procedural city generation," *GDTW 2007 : The Fifth Annual International Conference in Computer Game Design and Technology*, pp. 8–16, 2007.

[9] T. Lechner, B. Watson, and U. Wilensky, "Procedural city modeling," in *In 1st Midwestern Graphics Conference*, 2003.

[10] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. V. Gool, "Procedural modeling of buildings," *Proceedings of ACM SIGGRAPH 2006 / ACM Transactions on Graphics*, vol. 25, no. 3, pp. 614–623, 2006.

[11] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *ACM Trans. Graph.*, vol. 22, pp. 669–677, July 2003.

[12] P. Müller, G. Zeng, P. Wonka, and L. V. Gool, "Image-based procedural modeling of facades," *Proceedings of ACM SIGGRAPH 2007 / ACM Transactions on Graphics*, vol. 26, no. 3, 2007.

[13] P. Merrell, E. Schkufza, and V. Koltun, "Computer-generated residential building layouts," *ACM Trans. Graph.*, vol. 29, pp. 181 :1–181 :12, December 2010.

[14] D. Luebke and C. Georges, "Portals and mirrors : simple, fast evaluation of potentially visible sets," in *Proceedings of the 1995 symposium on Interactive 3D graphics*, I3D '95, pp. 105–106., ACM, 1995.

[15] D. W. Eggert, K. W. Bowyer, and C. R. Dyer, "Aspect graphs : state-of-the-art and applications in digital photogrammetry," in *Proc. ISPRS 17th Congress : Int'l Archives Photogrammetry Remote Sensing*, pp. 633–645, 1992.

[16] D. Cohen-Or, Y. L. Chrysanthou, C. T. Silva, and F. Durand, "A survey of visibility for walkthrough applications," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 9, pp. 412 – 431, July-Sept. 2003.

[17] F. Durand, G. Drettakis, and C. Puech, "The 3d visibility complex," *ACM Trans. Graph.*, vol. 21, pp. 176–206, April 2002.

[18] U. Assarsson and T. Möller, "Optimized view frustum culling algorithms for bounding boxes," *Journal of Graphics Tools*, vol. 5, pp. 9–22, 2000.

[19] C. Saona-Vázquez, I. Navazo, and P. Brunet, "The visibility octree : a data structure for 3d navigation," *Computers & Graphics*, vol. 23, no. 5, pp. 635 – 643, 1999.

[20] T. Hudson, D. Manocha, J. Cohen, M. Lin, K. Hoff, and H. Zhang, "Accelerated occlusion culling using shadow frusta," in *Proceedings of the thirteenth annual symposium on Computational geometry*, SCG '97, (New York, NY, USA), pp. 1–10, ACM, 1997.

[21] P. Wonka and D. Schmalstieg, "Occluder shadows for fast walkthroughs of urban environments," *Computer Graphics Forum*, vol. 18, no. 3, pp. 51–60, 1999.

[22] N. Greene, M. Kass, and G. Miller, "Hierarchical z-buffer visibility," in *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, pp. 231–238, ACM, 1993.

[23] H. Zhang, D. Manocha, T. Hudson, and K. E. Hoff, III, "Visibility culling using hierarchical occlusion maps," in *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '97, pp. 77–88, ACM Press/Addison-Wesley Publishing Co., 1997.

[24] P. Wonka, M. Wimmer, and D. Schmalstieg, "Visibility preprocessing with occluder fusion for urban walkthroughs," in *EUROGRAPHICS WORKSHOP ON RENDERING*, pp. 71–82, Springer-Verlag, 2000.

[25] V. Koltun, Y. Chrysanthou, and D. Cohen-or, "Virtual occluders : An efficient intermediate pvs representation," 2000.

[26] S. Greuter, J. Parker, N. Stewart, and G. Leach, "Real-time procedural generation of 'pseudo infinite' cities," in *GRAPHITE '03 : Proceedings of the 1st international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, (New York, NY, USA), pp. 87–ff, ACM, 2003.