Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider Biskra

Faculté des Sciences Exactes, des sciences de la Nature et de la Vie

Département d'Informatique

N° d'ordre :…...……
Série :……

Mémoire

Présenté en vue de l'obtention du diplôme de Magister en Informatique

Option**: Synthèse d'images et vie artificielle**

Titre :

Planification, navigation et comportements en temps réel d'une foule d'humains virtuels

Présenté par : **M. Mebarek BOUCETTA**

Soutenu le : / /2012

Devant le jury :

Djedi Noureddine	**Professeur Université de Biskra**	**Président**
Babahenini Med Chaouki	**MCA Université de Biskra**	**Examinateur**
Moussaoui Abdelwahab	**MCA Université de Setif**	**Examinateur**
Cherif Foudil	**MCA Université de Biskra**	**Rapporteur**

Remerciements

Tout d’abord, je souhaite remercier mon encadreur Dr Foudil. CHERIF de m’avoir accordé leur confiance afin de me permettre de travailler sur ce sujet de magister et pour ses précieux conseils et surtout pour sa disponibilité et sa grande aide.

Je remercie également Pr. Noureddine DJEDI, Dr. Mohammed Chaouki BABAHENINI et Abdelwahab MOUSSAOUI d'avoir accepté d’être les examinateurs.

En particulier, je remercie les collègues de la post graduation qui font que l'ambiance et l'atmosphère de travail soient de plus agréables.

Enfin, je remercie, ma famille dans tout son ensemble pour l’aide et le soutien qu’on m’a donné.

Table des matières

Remerciements 1
Table des matières 2
Liste de figures 4
Liste de Tableaux 6
Résumé 7
Abstract 8
ملخص 9
Introduction générale 10
Etat de l'art 16
1 Modèles de foules virtuelles 17
1.1 Modèles microscopiques 17
1.1.1 Modèles de force sociale 17
1.1.2 Modèles d'automate cellulaire 18
1.1.3 Le modèle à base de règles 19
1.2 Modèles macroscopiques 21
1.2.1 Le modèle gazeux 21
1.2.2 Le modèle hydraulique 21
1.3 Conclusion 22
2 Travaux Connexes 24
2.1 Rendu de foule 24
2.1.1 Représentations d'humains virtuels 24
2.1.2 Techniques De variété 26
2.2 Représentation de l'environnement et recherche de chemin 27
2.2.1 Représentations exactes de l'environnement 28
2.2.2 Représentations approximatives de l'environnement 31
2.2.3 Le modèle de champs de potentiels 35
2.2.4 Planification de chemin 36
2.3 Comportement de groupes 41
2.4 Conclusion 44
3 Modèles de foules existants 45
3.1 Un modèle de foule faible densité : Autonomous pedestrian 45
3.2 Modèle HIDAC 49
3.3 Dynamique continue 54
3.4 Dynamique globale 56
3.5 Patchs de foule 58
3.6 Foules par l'exemple 60
3.7 Conclusion 61
Modèle proposé 62
4 Planification de mouvement 63
4.1 Représentation de l'environnement 64
4.1.1 Grilles uniformes. 64
4.1.2 Grilles hiérarchiques. 67
4.2 L'algorithme A* pour choisir le meilleur voisin 73
4.3 Navigation 78

4.3.1 Evitement de collision 78
4.3.2 Comportement de groupes 83
5 Modèle de l'humanoïde 87
5.1 Représentations d'humains virtuels 87
5.1.1 Modélisation polygonale 88
5.2 L'animation squelettique 89
5.2.1 Squelette 89
5.3 Techniques de variété 91
6 Implémentation et résultats 93
6.1 Environnement de développement 94
6.2 Rendu 94
6.2.1 Moteur 3D 95
6.2.2 OGRE 3D 96
6.3 Interfaces graphiques 98
6.3.1 Fonctionnement de CEGUI 98
6.4 Technique d'accélération 100
6.4.1 Le Niveau de Détail (LOD) 100
6.4.2 Hardware skinning 100
6.4.3 Test de visibilité 102
6.5 L'architecture 105
6.6 Interface graphique de l'application 107
6.7 Résultats 109
6.7.1 Planification de chemin 109
6.7.2 Simulation 111
6.7.3 Evaluation 116
Conclusion générale 122
Références 124
Annexe A : Fonctionnement d'OGRE 130
Annexe B : Animation des personnages 132

Liste de figures

Figure 1: Modèle de Helbing la simulation avec 10.000 individus (Helbing et al. 2000). . 18
Figure 2: Simulation de foule : AC (à gauche) et (à droite) structure 2D à la base de grille (Tecchia et al 2001) 19
Figure 3: Les trois règles comportementales du modèles Flocks of Boids 20
Figure 4: (à gauche) l'utilisation des imposteurs permet à Tecchia et autres de simuler de grandes foules [TEC 02.b]. (à droite) Dobbyn et autres combine le maillage rigide et l'imposteurs pour avoir un rendu de plus haute qualité devant le camera [DOB 05]. 26
Figure 5: variété dans la foule à l'aide du canal alpha [DE 05]. 27
Figure 6: Exemple de triangulation de Delaunay contrainte. 29
Figure 7: Exemples de triangulations de Delaunay filtrées. 30
Figure 8: Exemple de grilles régulières. 32
Figure 9: Utilisation de grille en animation. 32
Figure 10: Exemple de carte de cheminement. A gauche, triangulation de Delaunay (gris) de l'environnement d'origine, et carte de cheminement déduite (bleu). A droite, un exemple de carte de cheminement obtenue. 33
Figure 11: Carte de champs de potentiels : en noir les obstacles (répulsion), en blanc les zones de navigation (attraction). Le gradient de gris exprime la valeur de potentiel dans l'environnement. Cet exemple contient 6 minima locaux : zones isolées à potentiel d'attraction maximal. 35
Figure 12: Fonctionnement de l'algorithme de Dijkstra dans des cas contraints ou non [101]. Le carré rose représente le nœud source, le mauve la destination. Le gradient de bleus correspond à la distance à l'origine, le plus clair étant le plus éloigné. 37
Figure 13: Fonctionnement de l'algorithme A*. 38
Figure 14: Les membres du groupe (triangles rouges et jaunes) à la suite de leur chef (en bleu foncé) [LOS 03]. 42
Figure 15: Les problèmes classiques rencontrés lorsque les membres d'un groupe chercher individuellement un chemin dans un environnement complexe. [KAM 04]. 43
Figure 16 : Vues de l'ancienne Pennsylvania Station recréée à partir de plans et de photos d'époque (à gauche la salle d'attente, à droite un des halls). 45
Figure 17 : modèle hiérarchique de l'environnement 46
Figure 18 : Comportements réactifs (routines) 47
Figure 19 : Architecture de sélection d'action 48
Figure 20 : Vue d'ensemble de l'architecture de HiDAC 50
Figure 21 : Rectangle d'influence et évitement de collision 51
Figure 22 : (à droite) Des agents évitant un piéton au sol (à gauche) Des agents piétinant un agent tombé 52
Figure 23 : Propagation de la panique 53
Figure 24 : Vue d'ensemble de l'algorithme 55
Figure 25 : 8 000 personnes en poursuivant 2 000 autres. 55
Figure 26 : Vue d'ensemble de l'algorithme de dynamique globale 57
Figure 27 : 10 000 personnages se déplaçant dans des directions opposées. 57
Figure 28 : Exemples de patchs. L'axe vertical représente le temps. Le patch de gauche contient un mobile endogène. Celui de droite de droite contient un mobile exogène : il sort du patch au temps t1 et y retourne au temps t2. 58

Figure 29 : Connexion de patch partageant des patterns miroirs. 58
Figure 30 : Deux manières d'assembler les patchs : approche bottom-up (haut) et approche top-down (bas). 59
Figure 31 : La méthode de foules par l'exemple 60
Figure 32: Le processus de génération de la grille uniforme 65
Figure 33 : Relation d'adjacence dans une grille régulière 66
Figure 34 : Décomposition en quadtree 69
Figure 35 : Une carte 2D représentant l'espace de navigation 69
Figure 36 : La décomposition de la carte 2D 70
Figure 37 : La représentation de la décomposition en quadtree 70
Figure 38 : Une carte 2D simple du monde 71
Figure 39 : Une carte 2D simple décomposée en régions 71
Figure 40 : une grille régulière et la grille hiérarchique correspondante 72
Figure 41 : Distance de Manhattan (chemins rouge, jaune et bleu) contre distance euclidienne en vert 74
Figure 42 : Planification 2D ½ 77
Figure 43 : Méthode de perception 79
Figure 44 : Les trois types de collision 80
Figure 45 : Alignement des membres de groupe (vert) suite au mouvement de leur chef (rouge) 84
Figure 46 : modèle 3D d'un humanoïde 88
Figure 47: méthode de rigging avec un squelette. 90
Figure 48 : technique de variété d'apparence 92
Figure 49 : pyramide de vue 103
Figure 50 : Architecture du système développé 105
Figure 51 : Temps de calcul de A* suivant la fonction h(c) 109
Figure 52 : modèles 3D utilisés 111
Figure 53 : le rendu 3D d'un environnement urbain 112
Figure 54 : Taux d'affichage moyen des différents scenarios 114
Figure 55 : une capture d'écran de simulation d'une foule de 500 agents 115
Figure 56 : En haut, l'évitement de collision est désactivé, de nombreuses collisions se produisent (cercle rouge). En bas, il est activé, la simulation est plus réaliste. 117
Figure 57 : En haut, la prise en compte de la hauteur du terrain est désactivé, de nombreuses collisions se produisent (cercle rouge). En bas, il est activé, les humanoïdes sont correctement repositionnés. 118
Figure 58 : La capture d'écran de gauche montre une capture d'écran où tout le monde marche seul, celle de droite, où certains piétons se déplacent en petit groupe, parait plus naturelle. (En haut une vue de dessus et en bas une vue en perspective) 119
Figure 59 : La formation de lignes entremêlées lorsque deux flux de piétons se rencontrent face à face 120

Liste de Tableaux

Tableau 1 : Tableau récapitulatif 61
Tableau 2 : Temps de calcul de A* (microseconde) 109
Tableau 3 : Taux d'affichage moyen des différents scenarios 113
Tableau 4 : nombre Max de collisions par frame 116
Tableau 5 : Tableau comparatif 121

Résumé

La simulation de foules en temps réel est un domaine de recherche qui propose de nombreux défis. Les foules virtuelles sont intégrées aux scènes de films, jeux vidéo, architecture, modélisation urbaine et de nombreux autres domaines. Peupler ces environnements virtuels avec de grandes foules d'humains virtuels est une tâche difficile, et pose des véritables problèmes tels que le rendu en temps réel et la simulation des comportements. Une simulation pour de tels environnements est difficile car un compromis doit être assuré entre le réalisme du rendu et le temps réel. Notre objectif principal est de concevoir un système capable de simuler une grande foule d'humains virtuels. Nous présentons dans ce travail un modèle capable de gérer la planification de chemin de milliers de piétons en temps réel, en tenant en compte de l'évitement de collisions entre les différents individus de la foule. Notre contribution consiste en la combinaison de deux méthodes de représentation approximative de l'environnement basées sur la subdivision de l'espace de navigation. Une planification globale efficace basée sur une grille hiérarchique est utilisée pour la planification à long terme permettant au piéton de trouver un chemin vers son but. Une planification locale basée sur une grille régulière est utilisée pour trouver des chemins à court terme permettant pour chaque piéton de rejoindre son itinéraire pendant l'évitement d'une éventuelle collision. Nous avons aussi étendu notre architecture de planification de mouvement avec un algorithme capable de simuler le comportement de groupes, ce qui améliore la perception de la scène par les utilisateurs.

Notre simulation est réalisée en utilisant OGRE 3D, et différentes techniques d'accélérations ont été exploitées, et les résultats expérimentaux montrent que nous sommes capables de simuler jusqu'à 2000 piétons en temps réel.

Mots clés : foule, planification, comportement, navigation, évitement de collision, animation temps réel.

Abstract

Crowd simulation is an important aspect of the computer graphics domain. Virtual crowds are integrated to film scenes, computer games, architecture, urban modelling and many other areas. Populating virtual environments with large crowds of virtual humans is a challenging task, and poses problems such as real-time rendering and behavior simulation. Simulation for such environments is difficult as a trade off needs to be found between realism and real-time simulation. Our main purpose is to design a system able to simulate a large crowd of virtual humans interacting in every-day urban environment conditions. We present a model able to handle the path planning of thousands of pedestrians in real time, while ensuring dynamic collision avoidance. Our contribution is the combination of two approximated methods of representation of environment based on space subdivision to solve the path planning problem of thousands of pedestrians in real time. Global path calculation based on an efficient hierarchical grid is used for long term path planning allows to the pedestrian to find a path to the goal, and realistic local path based on a regular grid is used for short term planning allows for each pedestrian to join his pathway during the avoidance of a possible collision. We have also extended our crowd motion planning architecture with an algorithm able to simulate group behaviors, which much enhances the user perception of the watched scene.

Our simulation is performed using OGRE 3D, and different accelerations techniques have been exploited, and the experimental results show that we are able to simulate up to 2000 pedestrian at real time.

Key words: crowd, planning, behavior, navigation, collision Avoidance, navigation, real-time animation.

ملخص

محاكاة الحشود هو جانب مهم جدا من ميدان الرسم بالحاسوب، مشاهد الحشود الافتراضية تدمج في الأفلام وألعاب الفيديو، والعمارة، و هندسة المناطق الحضرية والعديد من المجالات الأخرى. ملئ مثل هذه البيئات الافتراضية بالحشود الكبيرة من البشر يطرح مشاكل حقيقية مثل آنية العرض وواقعية محاكاة السلوك. محاكاة مثل هذه البيئات صعب جدا لأنه يجب أن يتم التوصل إلى التوفيق بين الواقعية و آنية المحاكاة. هدفنا الرئيسي هو تصميم نظام قادر على محاكاة حشود كبيرة من البشر الافتراضي يتفاعل في جميع ظروف الحياة العادية في المناطق الحضرية. في عملنا هذا نقدم نموذجا يمكن من معالجة تخطيط المسار للآلاف من المارة مع تحقيق آنية العرض، و ضمان ديناميكية تجنب الاصطدام. مساهمتنا تتمثل في الدمج بين طريقتين للتمثيل التقريبي للبيئة على أساس تقسيم فضاء الملاحة. تخطيط شامل يستند على تقسيم شبكي هرمي فعال يستخدم في التخطيط على المدى الطويل للمشاة لإيجاد الطريق إلى الهدف. تخطيط محلي يستند على تقسيم شبكي منتظم يستخدم في التخطيط على المدى القصير يمكن المشاة من العثور على المسار والعودة إليها أثناء عملية تجنب الاصطدام المحتل. كما قمنا كذلك بمحاكاة سلوك إمكانية الحركة ضمن مجموعات مصغرة، الشيء الذي يعزز من واقعية المحاكاة.

يتم إجراء المحاكاة الثلاثية الأبعاد باستخدام محرك العرض OGRE 3D إضافة إلى تقنيات تسريع يتم استخدامها لتحسين العرض، والنتائج التجريبية تبين أننا قادرون على محاكاة ما يصل إلى 2000 من المارة في وقت آني.

الكلمات المفاتيح: الحشد, السلوك, التخطيط, تجنب الاصطدام, الإبحار ,التحريك الآني

Introduction générale

Les foules virtuelles n'ont été principalement abordées en synthèse d'images que vers la fin des années 90, car leur simulation présente un challenge en termes d'utilisation de la mémoire et des temps de calcul nécessaires.

La communauté en intelligence artificielle s'est intéressée bien avant aux systèmes multi-agents mais on ne pouvait pas vraiment parler alors de simulation de foules. La complexité se trouve dans le nombre d'entités à simuler qui se chiffre en milliers voire millions et la complexité du comportement lui-même. La modélisation, le rendu et l'animation d'un seul humain virtuel (humanoïde) est déjà une tâche difficile et une foule peut linéairement multiplier les calculs. Souvent un humanoïde est modélisé à partir de plusieurs milliers de polygones sur lesquels des textures (images) peuvent être appliquées. L'animation demande souvent un squelette supplémentaire pour simuler le mouvement, les intentions et les actions. De plus, un humanoïde placé dans un environnement doit pouvoir le percevoir afin de réagir en conséquence et de façon cohérente. Tout ceci doit être combiné et contribue à un comportement complexe à simuler et à afficher.

Les méthodes de rendu et de simulation de foules virtuelles doivent tenir compte de ces difficultés et ne peuvent se contenter des résultats obtenus sur la simulation d'un seul humanoïde. Des méthodes sont apparues pour diminuer la mémoire et le temps de calcul nécessaires souvent en s'appuyant sur la similarité de chaque humain virtuel qui compose la foule. Chaque humanoïde a une apparence similaire ainsi qu'un comportement équivalent. Il est donc possible de partager de l'information stockée et d'utiliser les mêmes modèles géométriques pour représenter les humains virtuels. La recherche sur les foules virtuelles s'est intéressée aux différentes étapes suivantes :

1. La modélisation : la création et la composition du contenu : ceci permet de mettre en place par exemple les données nécessaires à la prise de décision de l'humanoïde

2. Le rendu: ajouter des milliers de personnes dans une scène revient à augmenter le nombre de polygones à afficher. Ceci peut réduire considérablement le temps d'exécution si aucun algorithme spécifique n'est appliqué.

Plusieurs algorithmes ont simplifié le rendu des foules en se basant sur le rendu à base d'images [TEC 02.b] [TEC 02.a] [AUB 00], sur le niveau de détails [COI 07] [DOB 05] ou sur la visibilité [TEC 02.b].

3. la planification de mouvement

Afin de simuler correctement une foule virtuelle il est très important de prévoir les mouvements de chaque individu : quel est le but ? Comment y arriver ? Y a-t-il les gens à rester près de ? Y a-t-il les individus à éviter ? Toutes ces questions sont gérées par la planification de mouvement. Nous distinguons trois aspects de la planification de mouvement qui doivent être abordés pour obtenir des résultats réalistes :

- La navigation (planification de chemin): Il existe plusieurs classes d'algorithmes pour décrire les décisions de trajectoire des individus d'une foule. Ils peuvent être basés sur des décisions locales avec pour but d'avoir un comportement visuellement cohérent [TEC 02.b] [LER 07], une distribution attendue [DES 03] [WON 07] ou une trajectoire liant un point à un autre [LOS 03] [WON 07]. La navigation peut aussi chercher une trajectoire globale, avec un point d'entrée et une destination [WON 07] [WON 08] [SUD 08] [ARI 01] [LAM 04]. Une approche différente considère la foule comme un groupe d'individus et simule une navigation plutôt de "troupeau" [REY 87].

- Evitement de collision: lorsqu'une foule se déplace, elle doit éviter les collisions avec son environnement. Aussi, chaque individu doit éviter ses voisins. De nombreuses méthodes existent pour traiter ce genre de situation. La détection doit se faire en temps réel, et la trajectoire des individus doit rester plausible.

- Cohésion de groupe : si un piéton se déplace dans un groupe avec ses amis ou sa famille, il essaye de rester aussi étroitement à eux que possible, tout en progressant et évitant les obstacles sur son chemin.

Domaines d'application

Animer un grand nombre de personnages simultanément et de manière cohérente répond aux besoins d'un grand nombre de secteurs, citons par exemple le cinéma (pour éviter d'engager des milliers de figurants), le jeu vidéo (peupler les mondes virtuels) mais aussi l'urbanisme (simuler les flux de piétons pour mieux adapter le mobilier urbain ou les transports en commun), la sécurité (simuler l'évacuation d'un stade) ou encore le domaine militaire.

L'exploration des domaines d'application de simulation de foule donne un aperçu sur la façon dont le concept est appliqué. Cet aperçu peut être utilisé pour concevoir un modèle applicable à une large gamme d'applications. Nous explorons en détail ces applications:

1. Divertissement

La simulation de foule est progressivement occuper un rôle méritant dans la communauté du divertissement. Il est spécialement utilisé dans les domaines suivants:

- La production de films:

Les films exigent habituellement des scènes avec de grands groupes de personnages dont l'apparence et le mouvement semble très réel. Plusieurs Films gagnants Oscar ont leur succès grâce à Massive logiciel de simulation de foule. Le système Massive est spécialement conçu pour la génération de scènes de bataille contenant des milliers d'agents (utilisé dans le Seigneur des Anneaux III, 2003 [MAS 11]). Un autre système de simulation de foule moins populaires est utilisé dans la production cinématographique est ALICE, un moteur de foule à base de Maya (utilisée dans le film l'épopée Troy, 2004 [ALI 03]). Dans les systèmes de simulation de foule existants pour les effets visuels, le mouvement de personnages n'est pas limité aux surfaces planes. Les agents peuvent effectuer des actions telles que l'escalade des sommets et des échelles en utilisant l'intelligence artificielle.

- La production du jeu:

Les foules sont également intégrées dans les jeux informatiques pour ajouter la vie et enrichir le contenu. Les jeux vidéo ont besoin de simuler une activité de vie dense pour immerger le joueur (s). SpirOps Crowd (utilisée dans le jeu Splinter Cell - Double Agent, 2006 [SPI 03]). Fournit des bibliothèques en mesure de gérer les comportements de base de chaque personnage dans une foule. De même pour les systèmes ci-dessus, SpirOps Crowd utilise l'intelligence artificielle pour simuler le mouvement d'agent.

Peu de systèmes de simulation de foule conçu pour le divertissement permettre la génération de foules dans des situations de vie normale et beaucoup moins encore avec la simulation des comportements des piétons.

2. Gestion des interventions d'urgence

La simulation de foule est aussi utilisée pour l'analyse du comportement dans des situations d'urgence avec des zones de navigation limitée [HEL 00] [SHE 08] pour aider à la conception des stratégies d'évacuation en toute sécurité. Les applications populaires de l'animation de foule dans ce domaine sont:

- L'incendie et l'intoxication au gaz:

Beaucoup de recherche se reproduisent des situations d'urgence avec des foules de haute densité dans un environnement fermé, sous réserve de grave incendie, la fumée ou un gaz toxique [LI 04] [COU 05]. Ils évaluent le nombre possible de blessés et des personnages capables d'atteindre des zones de sécurité. Ces simulations sont utilisées pour modéliser le nombre et les positions des sorties de sécurité.

- Foules surexcitées:

La simulation de foule est aussi utilisée pour modéliser les foules de très grand nombre de personnages assistés à un grand événement (match de football dans un stade, un concert dans un théâtre). Un paramètre d'urgence est ajouté à la simulation, en imitant l'excitation dangereuse de la foule ou les rumeurs de faux danger qui peut entraîner une foule à devenir incontrôlables [KLÜ 04] [HEL 00]. Ceci est également utilisé pour la planification d'évacuations sûres.

- Attentat à la bombe:

La simulation d'assemblée publique soumise à un attentat à la bombe est également très commun [SHE 06], [USM 07]. Il évalue le nombre de morts et de blessés lors d'un acte terroriste et peut être utilisé pour prédire les dégâts d'une attaque potentielle. Les systèmes ci-dessus généralement attachés des particules ponctuelles aux personnages et ces particules sont animées. Ceci est différent de l'approche d'intelligence artificielle utilisée dans les scènes de foule pour le divertissement.

3. Modélisation urbaine

L'architecture et la modélisation urbaine utilise souvent la simulation de foule pour donner la vie aux modèles ou aux plans. Les applications courantes sont le tourisme virtuel, la planification urbaine, la visualisation architecturale et le patrimoine culturel et ils impliquent de peupler un environnement urbain avec un grand nombre d'individus.

La simulation du trafic des piétons reproduit le comportement collectif des piétons dans un environnement urbain, dans des situations de vie normale. Des logiciels de simulation de piétons existent Micro-Pedsim [PED 06], Legion [LEG 07] mais peu d'eux traitent la prise de décision et la planification de la trajectoire qui font partie du comportement de tout piéton réel.

La simulation de foule pour la modélisation urbaine reflète la réalité et par conséquent aucun paramètre de panique n'est ajouté à la foule.

La planification de la sécurité publique est l'application la plus courante de simulation de foule et il n'est applicable que si les agents sont dans un état de panique, ce qui néglige le comportement collectif normal.

Motivation

Aujourd'hui la plupart des techniques de planification de mouvement offrent des solutions trop chères pour simuler de grandes foules. Pour garder un taux d'affichage élevé, de telles méthodes exigent de réduire le nombre d'individus, ou de simplifier leur rendu aux particules en 2D. Nous souhaitons fournir une solution pour la planification de mouvement d'une foule, de sorte que la simulation entière, c.-à-d., la planification, l'évitement de collision, l'animation, et le rendu de milliers d'humains virtuels fortement détaillés, fonctionne en temps réel et avec un degré de réalisme acceptable.

Contributions

Les principales contributions de notre travail peuvent être récapitulées comme suit :

- Afin de résoudre la planification de mouvement de foule en temps réel d'une façon réaliste, nous avons développé un système basé sur la combinaison de deux méthodes de représentation de l'espace de navigation, la grille hiérarchique (Quadtree) et la grille régulière. Une planification au niveau global basée sur une grille hiérarchique est utilisée pour permettre au piéton de trouver un chemin vers son but. Une planification au niveau local basée sur une grille régulière est utilisée pour trouver des chemins permettant pour chaque piéton de rejoindre son itinéraire pendant l'évitement d'une éventuelle collision.
- Nous avons aussi étendu notre architecture de planification de mouvement avec un algorithme capable de simuler le comportement de groupes, ce qui améliore la perception de la scène par les utilisateurs.
- Utiliser OGRE 3D un moteur de rendu temps réel permettant de facilité la simulation et d'optimiser le rendu.
- en plus d'utilisation d'OGRE 3D nous avons employé des techniques d'optimisations additionnelles afin d'augmenter les performances de notre système de simulation : le niveau de détail (LOD), le hardware skinning (animation squelettique) et le test de visibilité (pyramide de vue). Ces techniques nous permettent de simuler des milliers de personnages 3D fortement détaillés qui sont animés et qui se déplacent en temps réel.

Organisation du mémoire

Ce document est organisé en six chapitres regroupés en deux parties:

Partie I : Etat De L'art

Chapitre 1 : « Modèles de foules virtuelles »

Dans ce chapitre nous présenterons les deux principales méthodologies de simulation de foules, macroscopique et microscopique ainsi que les caractéristiques des différents modèles.

Chapitre 2 : « Travaux connexes»

Dans ce chapitre nous présenterons les travaux réalisés dans le domaine des foules virtuelles, en ce qui concerne le rendu, la représentation de l'environnement et le comportement.

Chapitre 3 : « modèles de foules existants»

Dans ce chapitre nous présenterons quelques systèmes de foule existant ainsi que leurs caractéristiques.

Partie II : Le modèle proposé

Chapitre 4 : « Planification de mouvement » :

Dans ce chapitre nous présenterons en détail la représentation de l'environnement utilisé affin de résoudre le problème de planification et de navigation.

Chapitre 5 : « Modèle de l'humanoïde» :

Dans ce chapitre nous présenterons les différentes techniques utilisés pour représenter et animer les piétons composant la foule.

Chapitre 6 : « Implémentation et résultats »

Dans ce chapitre nous présenterons les détails d'implémentation de notre système, le moteur de rendu utilisé OGRE 3D, ainsi que les différentes techniques d'accélération employées et enfin nous exposerons les résultats obtenus.

La conclusion mettra en avant les perspectives d'évolution du modèle proposé.

Etat de l’art

Chapitre 1 Modèles de foules virtuelles

1 Modèles de foules virtuelles

Beaucoup de modèles de simulation de piétonniers ont été développés au cours des années dans divers disciplines comprenant l'Infographie, la robotique, et la dynamique d'évacuation. Ceux-ci peuvent être groupés en deux méthodologies principales: macroscopique et microscopique.

Les modèles macroscopiques focalisent sur le système en son totalité (caractéristiques du flux plutôt que différents piétons), tandis que les modèles microscopiques étudient le comportement et la décision de différents piétons et de leurs interactions avec d'autres piétons dans la foule.

1.1 Modèles microscopiques

Les modèles microscopiques décrivent le comportement de différents piétons dans l'espace-temps. Il y a trois sous-catégories : les modèles de force sociale, les modèles à base de règles et les modèles d'automates cellulaires(AC). La différence entre eux est dans la discrétisation de l'espace et du temps. Les modèles de force sociale [HEL 00] décrivent le comportement piétonnier par les champs de forces virtuelles induits par le comportement social des individus. Dans les modèles à base règles Le déplacement de chaque individu est régit par des règles de comportement de la forme « si condition alors action. Dans l'approche d'automates cellulaires, l'espace de l'étude est représenté par une grille de cellules uniformes, chaque cellule à un état locale qui dépend d'un ensemble de règles décrit le comportement des piétons [CHE 04]. Ces règles calculent l'état d'une cellule particulière en fonction de son état précédent et des états des cellules adjacentes.

Les modèles microscopiques sont plus intéressants du point de vue d'animer les foules d'agents virtuelles avec des comportements réalistes et autonomes.

1.1.1 Modèles de force sociale

Le modèle de force sociale est une approche microscopique pour simuler le mouvement piétonnier. Une force sociale (virtuelle) donnée est analogue à de vraie force telle que l'interaction répulsive, force de frottement, dissipation, il résout les équations du mouvement de Newton pour chaque individu. Ce modèle peut être appliqué avec succès pour simuler les scénarios de mouvement piétonniers réels.

Relativement aux autres modèles, les modèles de force sociale décrivent le comportement piétonnier de façon réaliste. Cependant, ils sont conçus pour être aussi simples que possible. Chaque agent est représenté dans la locomotion plane par un cercle avec son propre diamètre et le modèle décrit les coordonnées, les vitesses, et les interactions continues avec d'autres agents. Chaque paramètre de force a une interprétation réelle, et il est différent pour chaque piéton. Les forces sociales modèlent le comportement de foule humain avec un mélange de facteurs socio-psychologique et physiques. Le modèle empiriquement dérivé le plus important de forces sociales est le modèle de Helbing.

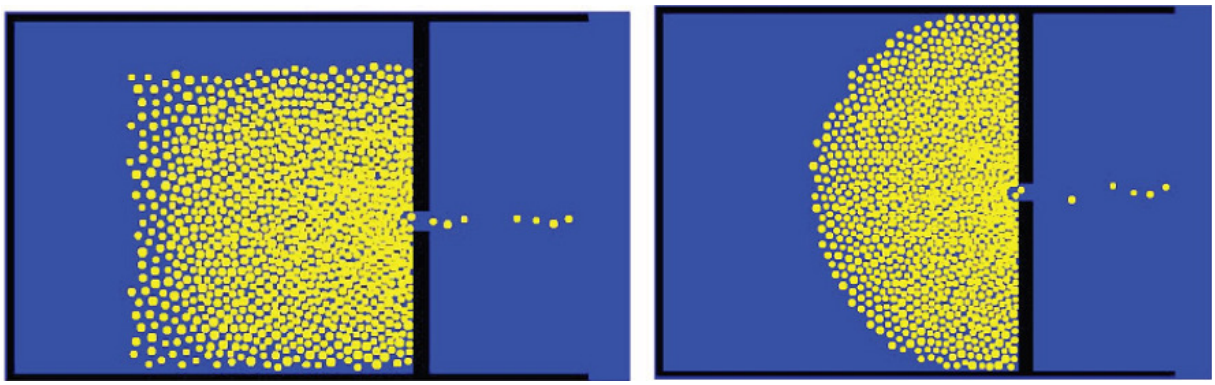


Figure 1: Modèle de Helbing la simulation avec 10.000 individus (Helbing et al. 2000).

1.1.2 Modèles d'automate cellulaire

L'automate cellulaire est une approche d'intelligence artificielle pour la modélisation de simulation défini comme formalisation idéalisations mathématique des systèmes physiques dans lesquels l'espace et le temps sont discrets et les quantités physiques prennent un ensemble fini de valeurs discrètes.

Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps t+1 est fonction de l'état au temps t d'un nombre fini d'état de cellules appelé son (voisinage). À chaque nouvelle unité de temps, un ensemble de règles locales [WOL 83] sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle (génération) de cellules dépendant entièrement de la génération précédente. Ces règles décrivent le comportement (intelligent) de prise de décision des automates, de ce fait créant et émulant le comportement réel. Chaque automate évalue ses occasions au cas par cas. Le comportement émergent global de groupe est un résultat des

interactions des règles locales comme chaque piéton examine les cellules disponibles dans son voisinage.

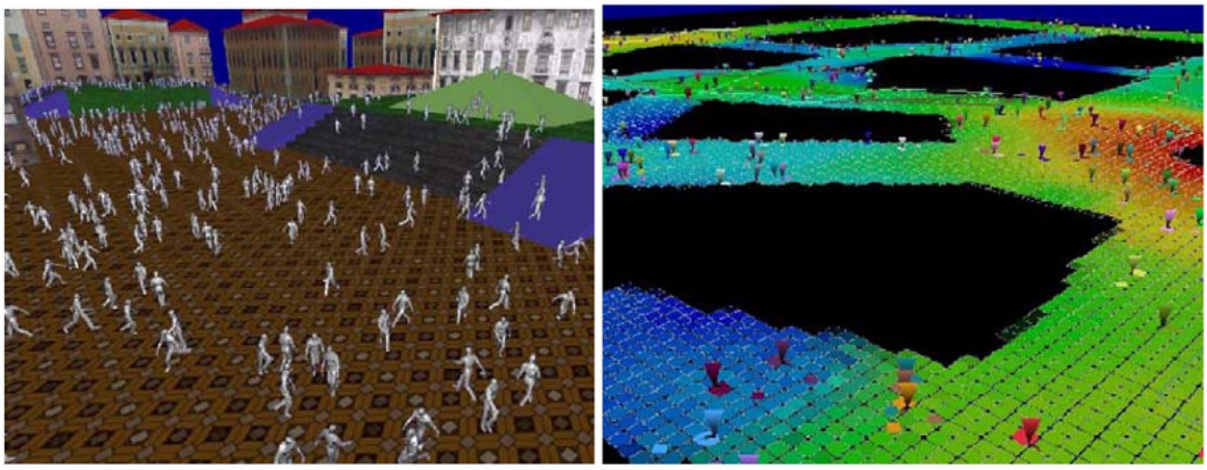


Figure 2: Simulation de foule : AC (à gauche) et (à droite) structure 2D à la base de grille (Tecchia et al 2001)

Les modèles d'AC [TEC 01] [KIR 03] [CHE 04] [TOR 07] bien que sont simple et rapide à mettre en œuvre, ne permettent pas de contact entre les agents. L'espace de navigation est discrète, et les individus peuvent seulement se déplacer aux cellules adjacentes libres. Cette approche de damier offre des résultats réalistes pour une foule de plus faible densité, mais des résultats peu réalistes quand des situations de haute densité sont exigés dans les cellules discrètes. De plus des chemins globaux plus réalistes dans la grille peuvent être obtenus en pré calculant les chemins vers les buts et en les stockant dans la grille [LOS 03].

1.1.3 Le modèle à base de règles

Craig Reynolds [REY 87], avec son modèle Flocks of Boids, introduit la notion de simulation microscopique à base de règles. Le déplacement de chaque individu est régit par des règles de comportement de la forme « si condition alors action».

Trois règles sont proposées par C.Reynolds dans le cadre de l'animation de nuées (Figure 3)

Séparation : afin d'éviter d'éventuelles collision avec ses voisins.

Alignement : afin de réguler sa vitesse par rapport à l'ensemble du groupe.

Cohésion : afin de rester proche de ses voisins.

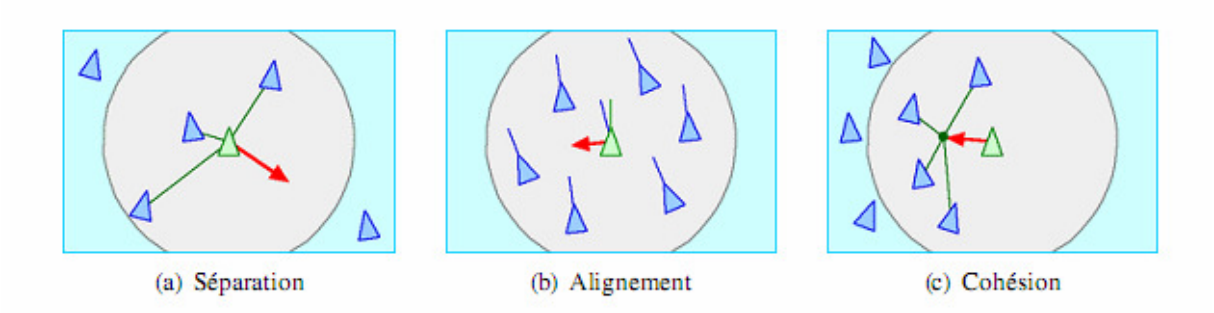


Figure 3: Les trois règles comportementales du modèles Flocks of Boids

De telles règles d'inter-relations conduisent ainsi à un comportement de groupe émergeant. Mais, dans ce modèle flocks of boids, l'autonomie de l'entité est bridée par le fait que son comportement est uniquement défini en fonction de celui de ses voisins. On a alors deux catégories d'individus: des leaders qui choisissent réellement la trajectoire, et des suiveurs qui se contentent d'évoluer suivant le mouvement général en appliquant les règles d'évitement. C.Hartman et B.Benes [HAR 06] proposent de ne pas fixer le leader lors de la simulation, mais plutôt d'introduire une nouvelle règle permettant à une entité standard de prendre la place du décideur actuel.

D'autre approches, comme celle de W.Shao et D.Terzopoulos [SHA 05] ou F.Lamarche et S.Donikian [LAM 04], exploitent tout de même le concept de règles sans avoir recours à un leader: les différentes possibilités d'adaptation de l'entité sont itérativement évaluées lors du déplacement, ce dernier étant cette fois-ci guidé par un processus de contrôle indépendant. Cette technique est aussi utilisée dans le monde du jeu vidéo [GRE 00], où les possibilités de contrôle offertes par les règles séduisent. De plus, la faible complexité d'évaluation des modèles à base de règles est exploitée par le domaine de l'animation [LOS 03] afin de peupler des environnements avec des milliers d'individus capables de se déplacer, et même d'accomplir certains comportements assez simples.

Les modèles à base de règles proposent une approche plus souple de la navigation microscopique, permettant d'introduire des adaptations variées dans différentes situations. Néanmoins, le caractère itératif de la décision rend difficile la fusion d'informations, les règles étant indépendantes les unes des autres. Une solution, exploitée par W.Shao et D.Terzopoulos [SHA 05] ou encore S.Raupp Musse [MUS 00], est de créer des règles gérant directement un ensemble d'informations, comme un groupe de personnes évoluant dans l'environnement. Mais cette approche a le désavantage de spécialiser le modèle, rendant encore une fois sa généralisation difficile : afin de gérer le gradient des situations

pouvant apparaître lors de la navigation, un modèle à base de règles devra spécifier l'ensemble des adaptations pouvant intervenir pour l'ensemble des configurations.

1.2 Modèles macroscopiques

Les modèles de simulation macroscopique sont historiquement les premiers modèles informatiques ayant permis des simulations de foules. En effet, ceux-ci simulent la totalité des individus par un ensemble de fonctions mathématiques, ne nécessitant qu'une puissance de calcul assez faible. Ainsi, ces modèles abandonnent tout comportement individuel évolué, au profit d'une simulation très rapide d'un grand nombre d'entités.

Le but de ces modèles réside dans la caractérisation d'un phénomène Particulier généralement reproductible dans des conditions environnementales figées.

1.2.1 Le modèle gazeux

L.F.Henderson [HEN 71] propose dès 1971 d'assimiler les déplacements de personnes dans des conditions de faible densité à ceux de molécules de gaz. En se basant sur la théorie cinétique des gaz de Maxwell-Boltzmann, il fait le parallèle entre la densité des piétons et celle des particules d'un gaz, en assimilant leur différence d'états (arrêt, marche, course) aux différences d'énergie des gaz. Ce modèle s'appuie sur deux hypothèses concernant les piétons, en laissant de côté toutes règles sociales (comme les affinités liant certaines personnes):

- Leurs positions et vitesses sont indépendantes les unes des autres;
- Leur vitesse est indépendante de leur position.

Ces premiers constats, même s'ils peuvent paraître évidents aujourd'hui, sont à la base de toute simulation d'humanoïdes virtuels.

1.2.2 Le modèle hydraulique

Un modèle hydraulique a été proposé par J.Archea [ARC 79]] pour simuler des foules de forte densité. Il assimile ici le mouvement des personnes au travers de couloirs, escaliers, et portes, à celui de l'eau au travers de tuyaux, vannes, ou autres.

La simulation de l'évacuation du bâtiment se fait en deux étapes:

1. Initialisation du mouvement de l'ensemble des individus recevant simultanément Un message d'alerte (ouverture des vannes);
2. Simulation de l'évacuation (écoulement de l'eau dû à la force de gravitation).

Ce modèle se base sur les hypothèses suivantes:

• Prise de conscience et décision de l'évacuation immédiate et simultanée de tous les individus;

• Répartition uniforme des occupants;

• Débits uniformes;

• Flux unidirectionnels;

• Trajets prédéfinis.

Même si ce modèle a trouvé de nombreuses applications, notamment du fait de sa facilité de mise en œuvre, il souffre de la simplicité des hypothèses sur les quelles il repose: en réalité les personnes ne partent pas en même temps, et leurs directions ne sont pas aussi facilement prévisibles.

1.3 Conclusion

Nous avons exposé dans ce chapitre les trois modèles microscopiques permettant de simuler individuellement le comportement des piétons. Le modèle de force sociale, le modèle à base de règles et le modèle d'automate cellulaire(AC).

La conclusion que l'on peut tirer des modèles présentés est qu'aucun de ceux-ci ne permet, seul, de gérer toutes les situations potentielles des mouvements de personnes. Certains sont plus axés sur l'obtention de performances de calcul, d'autres sur la robustesse de la résolution, certains donnent une grande part à l'organisation sociale, et d'autres appuient plus sur la décision individuelle. Le danger lors de la réutilisation de ces modèles est donc de les exploiter en dehors du contexte pour lequel ils ont été créés, et validés.

Nous avons présenté ensuite les modèles macroscopiques, qui sont les premiers modèles informatiques ayant permis de simuler des foules de piétons. Ces modèles se caractérisent tous par une abstraction totale du comportement individuel, pour se concentrer sur une approche macroscopique quantitative de la foule. C'est pourquoi ces modèles ont été très utilisés dans le cadre de simulations d'évacuation, où le comportement individuel peut être quasiment ignoré, se réduisant à sa plus simple expression : se déplacer vers la sortie la plus proche. C'est pourquoi nous pouvons penser que ces modèles seraient très difficilement exploitables dans des situations plus variées, où l'impact des décisions individuelles ne serait plus négligeable.

Néanmoins, la plupart de ces approches sont basées sur des études statistiques conséquentes, issues d'observations du réel indépendantes du modèle de synthèse. Ces études restent donc exploitables pour des simulations plus évoluées, ne serait-ce que pour valider globalement les résultats, ou encore comme données de configuration. Elles ne sont néanmoins pas suffisantes, ne couvrant pas un spectre assez large du comportement humain.

Chapitre 2 Travaux Connexes

2 Travaux Connexes

La simulation de foules en temps réel est un domaine de recherche qui propose de nombreux défis, notamment, car elle demande l'utilisation performante des processeurs (CPU et GPU) pour obtenir des résultats convaincants. Dans ce chapitre nous allons d'abord décrire les différentes techniques utilisées couramment pour le rendu de foules en temps réel, ainsi que les différentes techniques permettant une variété accrue dans la génération des individus composant la foule. Ensuite nous allons décrire les différentes méthodes de représentation de l'environnement utilisées pour la planification des mouvements de foules en temps réel, de telles représentations permettant de décrire leur comportement (foules), et leur interaction même avec cet environnement.

2.1 Rendu de foule

Le rendu d'un humain virtuel consiste en plusieurs étapes importantes : mettre à jour l'animation, déformer le maillage en conséquence, plaquer la texturer, faire le rendu de l'ombre etc. Les deux défis principaux pour le rendu des foules d'humains virtuels en temps réel sont :

- Garder un taux d'affichage élevé quelque soit la position du camera et quelque soit le nombre d'humains visibles.
- Rendre chaque individu de la foule unique dans son apparence.

2.1.1 Représentations d'humains virtuels

Le rendu des milliers d'humains virtuels au taux d'affichage élevés est aujourd'hui possible si une approche de niveau de détail (LOD) est employée. En effet, pour accomplir cette tache en utilisant un maillage déformables [LAN 98], même avec le matériel spécialisé, ne s'étend pas à de grandes foules.

Une première optimisation est de diminuer la complexité du modèle géométrique selon la distance du camera, qui est l'idée principale de l'approche de niveau de détail [HOP 96].

Une deuxième optimisation pour réduire le coût de mises à jour d'animation aussi bien que les déformations du maillage, est d'utiliser les maillages rigides [ULI 04] [COI 05]. Nous appelant les maillages rigides les maillages dont les sommets sont déformés dans un prétraitement en respectant un jeu d'animations. Ces postures sont stockées dans la mémoire pour être réutiliser pendant la simulation. Les maillages rigides ont l'avantage d'éviter d'effectuer la déformation coûteuse du maillage en temps réel. Cependant, elles requirent un grand espace mémoire, et ainsi, seulement un ensemble limité d'animations pré-calculées peut être employé.

Une troisième optimisation qui est couramment employé dans le domaine de rendu de foule est le rendu à base d'image. En effet, le rendu à base d'image n'a pas le besoin de mettre à jour l'animation et de déformer le maillage, comme il travaille directement sur l'image finale projetée sur l'écran.

Les imposteurs sont souvent utilisés dans les simulations de foules en temps réel [TEC 02.b] (voir à gauche de la figure 4).Un imposteur représente un humain virtuel avec seulement deux triangles texturés formant un rectangle, donnant ainsi l'illusion voulue à une certaine distance. Similairement à un maillage rigide, les animations d'un imposteur sont pré-calculées, car sa création lors d'exécution serait trop fastidieuse. En effet, chaque posture de l'animation est stockée sur le GPU dans une texture différente et contient tous les points de vue nécessaires pour conserver l'illusion lorsque la caméra ou l'humain bouge.

Dobbyn et autres parvenu de combiner efficacement des imposteurs et des maillages géométriques pour visualiser un nombre impressionnant d'humains virtuels [DOB 05], comme illustré sur la figure 4 (à droite).

Malheureusement, l'utilisation des imposteurs pour le rendu des humains virtuels est fortement restrictive car le rang des mouvements possibles est limité au contenu d'image; généralement, les comportements sont limités à la navigation et les humains virtuels exécutent seulement la locomotion. Ceci explique partiellement pourquoi les environnements interactifs peuplés d'humains exhibant des comportements riches et divers [SHA 05] ont une taille limitée de population.

Figure 4: (à gauche) l'utilisation des imposteurs permet à Tecchia et autres de simuler de grandes foules [TEC 02.b]. (à droite) Dobbyn et autres combine le maillage rigide et l'imposteurs pour avoir un rendu de plus haute qualité devant le camera [DOB 05].

L'approche développée dans le moteur de foule Yaq [MAÏ 09] combine les maillages déformables, maillages rigides, et les imposteurs pour visualiser larges foules des humains virtuels. Cette approche permis d'avoir des maillages fortement détaillé au premier plan, dynamiquement animé et ainsi capable d'effectuer l'animation faciale et de regarder l'un l'autre ou au camera.

Comme les imposteurs sont simplement des rectangles 2D, leur affichage est très rapide. Leur désavantage principal est le stockage des animations qui est très coûteux, à cause du nombre élevé d'images qu'il faut garder sur le GPU.

2.1.2 Techniques De variété

Pour rendre chaque individu de foule unique dans son apparence, une solution idéale serait de concevoir chaque acteur d'un maillage différent avec une texture différente. Il semble évident qu'une telle approche devient rapidement infaisable avec un nombre croissant d'humains virtuels à simuler. Ce qui nécessite une armée de concepteurs à effectuer une telle tâche, et une mémoire illimité pour stocker et utiliser toutes ces maillages et textures durant la simulation.

La première étape pour assurer la variété de rendu d'une foule est d'avoir un petit ensemble de maillages d'humains, et plusieurs textures pour chacun d'eux. Avec le moteur de foule Yaq [MAÏ 09], il peut en général employer jusqu'à 6 ou 8 humains différents, chacun est associé avec 2 à 5 textures différentes. À partir de plusieurs modèles, des techniques additionnelles pour augmenter la variété ont été développées. Tecchia et al [TEC 02.a] a proposé d'assigner pour chaque texture, différentes couleurs pour différentes parties du corps dans un processus à multi passe. De Heras et al [DE 05] a prouvé que la

variété de couleur peut être réalisée avec des shaders, utilisant le canal alpha comme outil pour différencier les parties du corps. En termes de performance, elles pouvaient simuler des centaines d'humains virtuels en temps réel.

Il est important de rappeler que créer des modèles d'humains virtuels demande beaucoup de temps à un artiste. C'est pourquoi l'ajout des accessoires permet de varier encore l'aspect des instances générées. En effet, dans la vie réelle, les gens ont des coupes de cheveux différentes, portent des lunettes et des sacs, utilisent des téléphones portables, etc. Ces particularités sont des détails tout aussi importants que la variété de couleur, car la somme de ces détails permet de rendre les humains virtuels composant la foule uniques.

Sur la figure 5, des résultats obtenus en différenciant des parties du corps à l'aide du canal alpha.

Figure 5: variété dans la foule à l'aide du canal alpha [DE 05].

2.2 Représentation de l'environnement et recherche de chemin

Comme nous l'avons évoqué dans la section précédente, l'évolution des modèles informatiques avec l'approche microscopique s'est intéressée à l'aspect individuel de la décision relative au déplacement. C'est à cet aspect que s'intéresse la planification de chemin. Le but de la planification de chemin est de trouver globalement les endroits par lesquels passer les individus pour atteindre une destination.

On voit donc apparaître avec ces modèles la nécessité d'une description topologique, permettant de représenter et d'organiser l'environnement de simulation. Cette description topologique peut être plus ou moins fine, exacte ou approximative, et peut éventuellement contenir des informations sémantiques. Les algorithmes utilisés en animation comportementale pour représenter l'environnement sont étroitement liés à ceux employés en robotique [BAR 91], domaine où cette représentation tient un rôle prépondérant. Cette description est ensuite exploitée par des algorithmes de recherche de chemin. Ceux-ci peuvent se baser sur différentes informations issues de l'environnement, qu'elles soient géométriques comme les distances, ou qualitatives comme la charge cognitive associée au chemin.

2.2.1 Représentations exactes de l'environnement

Les représentations exactes de l'environnement vont chercher à organiser les données spatiales tout en conservant intégralement les informations qu'elles contiennent à l'origine. La méthode utilisée consiste généralement à découper l'espace navigable en cellules convexes de différentes formes. Il existe plusieurs modèles pour effectuer cette subdivision, mais nous nous attacherons ici à la plus utilisée qu'est la triangulation de, Delaunay, ainsi qu'à ses évolutions.

- Triangulation de Delaunay

La triangulation de Delaunay [BOI 98] crée un ensemble de triangles en réunissant des points fournis en entrée. L'algorithme d'unification respecte pour contrainte que le cercle circonscrit à un triangle ne contienne aucun autre point que les trois sommets qui le composent. Une conséquence de cette propriété est que l'angle minimum d'un triangle produit est maximisé. La complexité de l'algorithme d'unification est en O(n.ln n), avec n le nombre de points fournis en entrée.

La propriété la plus intéressante de cette triangulation est que chaque point est relié à son plus proche voisin par l'arête d'un triangle. Ainsi, cette triangulation peut être utilisée pour représenter l'espace navigable, les points d'entrée étant issus des obstacles. Une autre utilisation possible de cette triangulation est cette fois dynamique lors de la simulation [LAM 04], pour calculer un graphe de voisinage entre les entités mobiles, qui serviront cette fois-ci de point d'entrée.

- Triangulation de Delaunay contrainte

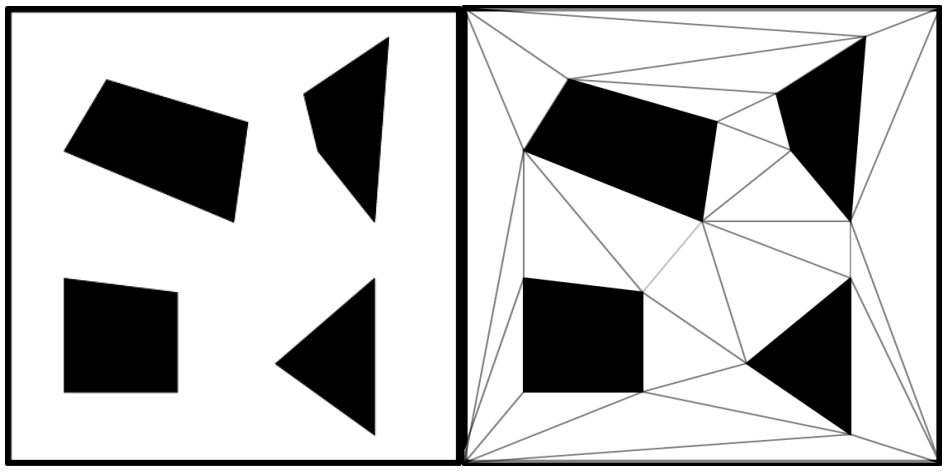
La triangulation de Delaunay contrainte [CHE 87] permet de modérer les contraintes de la version standard afin de conserver certaines arêtes de la définition graphique d'origine. Pour se faire, la contrainte du cercle circonscrit à un triangle est modifiée, pour spécifier que tout point inclus dans ce cercle ne peut être relié à tous les points du triangle sans intersectés un segment contraint.

Cette triangulation étend la propriété du plus proche voisin en permettant d'y associer une notion de visibilité. Si l'on considère que les arêtes contraintes coupent la visibilité d'un point à l'autre, la propriété de plus proche voisin devient : chaque point est relié à son plus proche voisin visible.

La triangulation de Delaunay contrainte est aussi utilisée pour obtenir une subdivision spatiale en triangles [KAL 03], en introduisant les obstacles à la navigation sous la forme d'autant de segments contraints (Figure 6). Il a été prouvé [BOI 95] que le nombre de triangles produits lors d'une telle subdivision est linéaire en fonction du nombre de points, indiquant que la discrétisation est donc directement proportionnelle à la complexité géométrique de l'environnement. Cette propriété présente un avantage certain comparativement aux méthodes approximatives, où la discrétisation est fonction de la précision désirée de la représentation.

- Triangulation de Delaunay filtrée

La triangulation de Delaunay filtrée [LAM 04] est une extension de la version contrainte. Deux types de filtrages sont proposés, l'un ayant pour effet d'augmenter le nombre de triangles produits afin d'affiner la représentation de l'environnement, l'autre de le diminuer afin de tenir compte de la visibilité pour l'élaboration d'un graphe de voisinage.



(a) Environnement d'origine **(b) Triangulation de Delaunay contrainte**

Figure 6: Exemple de triangulation de Delaunay contrainte.

Premièrement, concernant son application à la subdivision spatiale, la triangulation de Delaunay est filtrée par l'ajout progressif de contraintes représentant les goulets d'étranglement. Ainsi, en considérant l'ensemble des arêtes produites, on est sûr de représenter tous les rétrécissements présents dans l'environnement.

Deuxièmement, concernant son application aux graphes de voisinage, la triangulation de Delaunay est filtrée sur un principe équivalent à la triangulation contrainte, en supprimant les arêtes intersectées avec les obstacles de l'environnement (Figure 7). Ainsi, cette triangulation peut servir à déterminer trivialement quels sont les voisins directs visibles d'une entité, et avec un simple parcours de graphe peut sélectionner les entités visibles à une certaine distance.

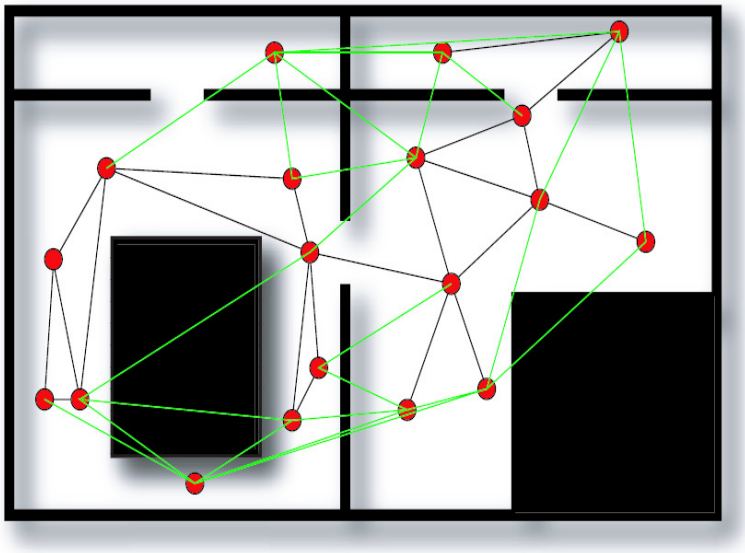


Figure 7: Exemples de triangulations de Delaunay filtrées.

Pour conclure concernant les représentations exactes de l'environnement, nous allons retenir deux points forts. Tout d'abord, ces représentations sont totalement indépendantes d'une quelconque précision désirée, reproduisant exactement l'environnement d'origine tout en l'organisant de manière plus accessible. Ainsi, il sera possible par ces représentations d'extraire la définition graphique source, pour son éventuelle utilisation dans des processus plus précis. Ensuite, ces représentations exposent plusieurs propriétés, comme le calcul des plus proches voisins, permettant par la suite d'automatiser un certain nombre de calculs à moindre coûts. Néanmoins, un point faible est associé à ces techniques, résidant dans la profusion des cellules produites. En effet,

pour des environnements à géométrie complexe, notamment contenant des courbes, le nombre de triangles obtenus augmente très rapidement. Même si l'effectif des triangles reste linéaire en fonction du nombre d'obstacles traités, cela réduira tout de même les performances d'algorithmes se basant sur cette définition.

Une solution proposée par F. Lamarche et al. [LAM 03][LAM 04] consiste à abstraire la subdivision obtenue, par des regroupements successifs, permettant de hiérarchiser les processus et ainsi de réduire le nombre de cellules à manipuler à chaque étape. Il faut tout de même noter que pour rester avantageuse en termes de réalisme, une telle abstraction doit permettre de conserver un maximum d'informations. Dans le cas contraire, elle risque de fausser les décisions prises par des algorithmes l'exploitant directement (comme la planification de chemin).

2.2.2 Représentations approximatives de l'environnement

Les représentations approximatives de l'environnement sont sans doute les plus utilisées en animation comportementale, du fait de leur facilité de mise en œuvre. Ces représentations vont décrire l'espace libre de navigation avec des formes géométriques simples telles que des carrés ou des segments. Deux modèles entrent dans cette catégorie : les modèles à base de grilles, et les cartes de cheminement.

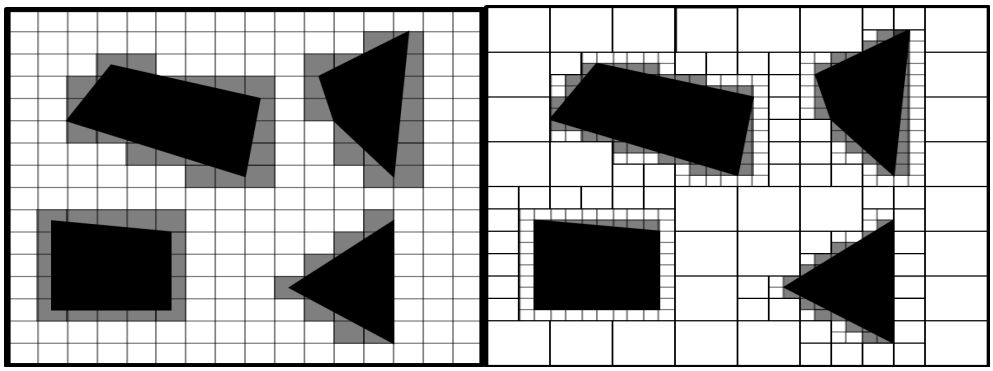
- Modèles à base de grilles

Le premier modèle de représentation approximative utilise des grilles régulières, formées de cellules carrées en deux dimensions (Figure 8 (a)) et cubiques en trois dimensions. L'environnement est donc pavé par ces cellules, qui peuvent avoir trois états : libre, partiellement obstruée, et obstacle.

La précision de la représentation obtenue dépend directement de la taille des cellules utilisées : plus les cellules sont grandes, moins la représentation est précise. Bien entendu, l'augmentation de la précision induit une augmentation proportionnelle de l'occupation mémoire. L'occupation mémoire de cette méthode constitue ainsi son premier point faible, se répercutant directement sur la complexité de recherche de chemin à l'intérieur de l'environnement [THR 96].

Pour réduire ce problème, une évolution de ce modèle est apparue sous la forme des grilles hiérarchiques [YAH 98] [SHA 05]. Cette méthode décrit l'espace navigable par une succession de grilles de plus en plus précises, organisées sous forme d'arbre. L'algorithme de construction va commencer par paver l'environnement avec les cases les moins précises,

puis découper récursivement les cases partiellement obstruées, en quatre parties pour la 2D (formation d'un quadtree – Figure 8 (b)), ou en huit pour la 3D (formation d'un octree). L'algorithme arrête les subdivisions dès qu'une précision fixée est atteinte, ou si la case produite n'est plus partiellement obstruée. Cette méthode est donc d'autant plus avantageuse que l'environnement est peu dense en obstacles.



(a) Grille régulière **(b) Quadtree de profondeur 3**

Figure 8: Exemple de grilles régulières.

Les méthodes à base de grilles sont fortement utilisées en animation comportementale [KUF 98] [TEC 00] du fait de leur simplicité de mise en œuvre et de leur rapidité d'exploitation. On peut ainsi voir des animations de milliers d'individus [TEC 02.b] basées sur ce mode de représentation (Figure 9).

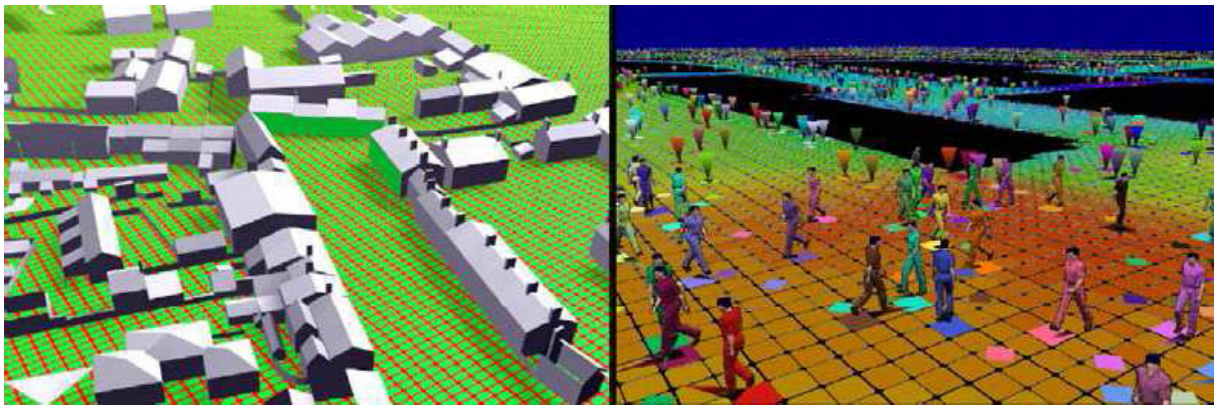


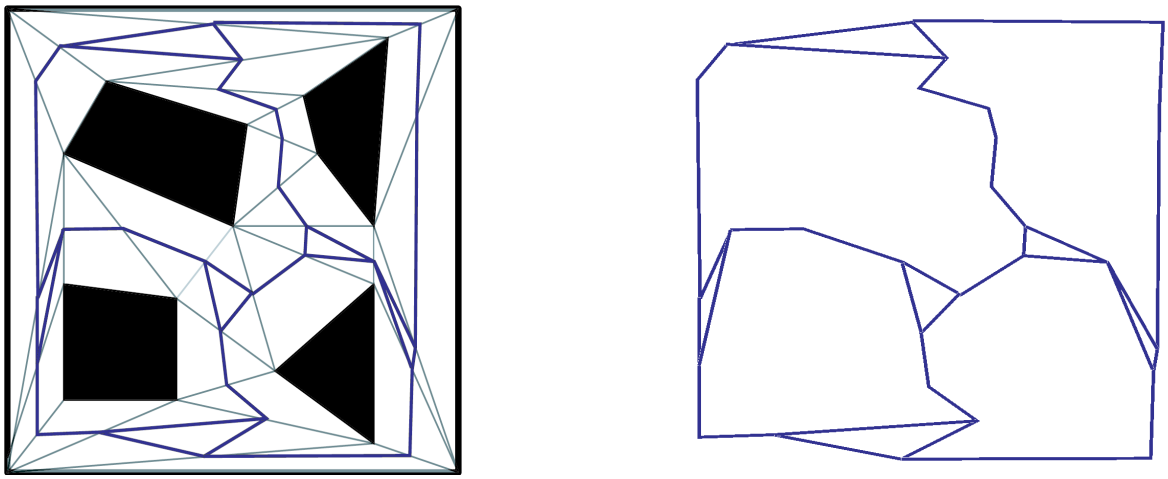
Figure 9: Utilisation de grille en animation.

Les grilles sont très difficilement généralisables à des environnements quelconques, notamment s'ils comportent des obstacles non alignés sur les axes, et dans un cadre de simulation où l'individu devra se déplacer finement. Néanmoins, la rapidité d'accès aux informations qu'elle produit fait de cette méthode un complément envisageable à une approche plus fine de représentation de l'environnement. Pour finir, il faut remarquer que

le niveau de discrétisation de la grille introduit implicitement une limite aux densités de populations que l'on peut simuler [AND 05]. Cela rend aussi le déplacement des entités discret, et les contraint généralement à avoir une taille standardisée.

- Cartes de cheminement

Les cartes de cheminement discrétisent l'espace navigable sous la forme d'un réseau de chemins. Ce réseau est obtenu en reliant des points clefs répartis à l'intérieur de l'environnement (Figure 10). Plusieurs méthodes existent pour créer des cartes de cheminement, différentes dans la manière de créer et de relier ces points clefs.



(a) Calculs sur l'environnement d'origine **(b) Exemple de carte de cheminement**

Figure 10: Exemple de carte de cheminement. A gauche, triangulation de Delaunay (gris) de l'environnement d'origine, et carte de cheminement déduite (bleu). A droite, un exemple de carte de cheminement obtenue.

Graphe de visibilité.

Cette méthode utilise les sommets des polygones représentant les obstacles comme points clefs. Les points clefs sont ensuite reliés deux à deux s'ils sont mutuellement visibles, c'est à dire si l'on peut tracer une ligne droite passant par les deux points sans rencontrer d'obstacle. Les graphes ainsi créés minimisent les distances parcourues [ARI 01], assurant d'obtenir des chemins de longueur minimale. La taille du graphe généré est ici dépendante du nombre de points mutuellement visibles, et est donc d'autant plus importante que l'environnement est ouvert avec des obstacles ponctuels et disparates.

Diagramme de Voronoï généralisé.

Cette méthode est basée sur une notion d'équidistance aux obstacles de l'environnement. Pour se faire, un ensemble de sites sont évalués au sein de l'environnement, correspondant aux obstacles, dont les intersections vont former les points clefs. Les chemins ainsi générés maximisent la distance aux obstacles. Ce calcul s'avère complexe, mais son approximation peut être obtenue rapidement par des méthodes utilisant les cartes graphiques [HOF 99], où le calcul est obtenu directement en effectuant le rendu des sites. Une autre méthode utilise la triangulation de Delaunay pour obtenir les points clefs du diagramme de Voronoï généralisé, en se servant du centre des triangles calculés. La carte de cheminement ainsi produite peut être considérée comme un condensé des informations de la triangulation, ne contenant plus la définition géométrique des obstacles de l'environnement.

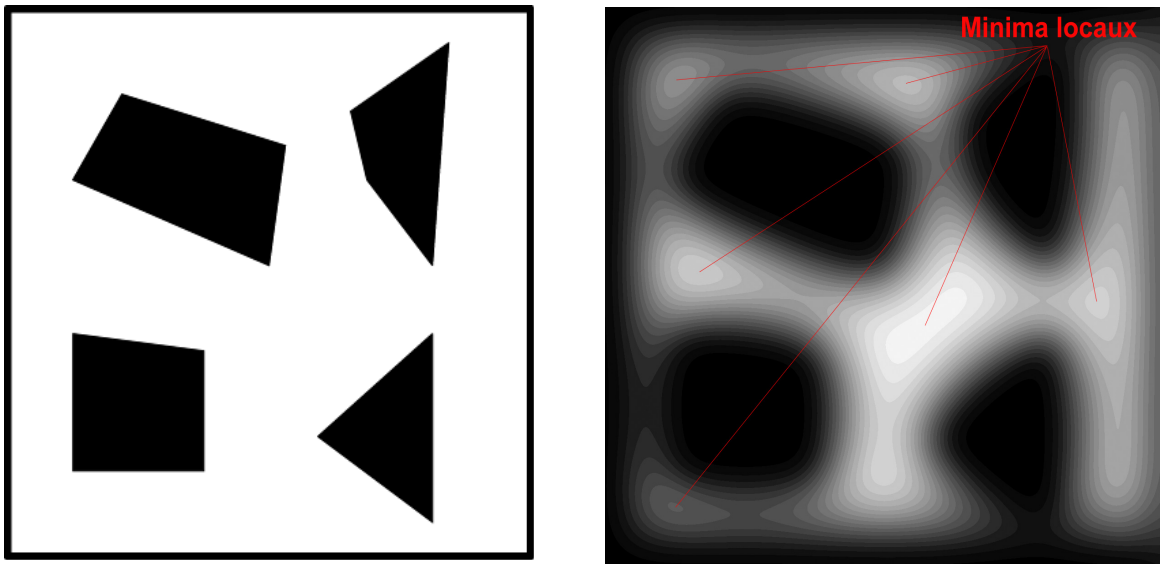
Cartes de cheminement probabilistes.

Avec cette méthode, la définition géométrique de l'environnement n'est pas utilisée comme support direct de la construction. En effet, les points clefs sont ici obtenus par une distribution aléatoire au sein de l'environnement navigable [OVE 02]. Deux points sont ensuite reliés s'il existe un chemin libre de collision entre eux. Ce test est effectué en se basant sur la géométrie de l'environnement, mais aussi sur la taille de l'entité qui se déplace. Cette méthode est plus largement utilisée en planification de mouvement [CHO 03] [PET 03] qu'en navigation, afin de générer des déplacements assez complexes (sauter, se baisser, marcher sur des piliers).

Pour conclure, les cartes de cheminement présentent le net avantage de fournir une description très condensée de l'environnement, ne nécessitant une prise de décision qu'au niveau des points clefs. Néanmoins, leur définition seule n'est pas suffisante pour gérer la complexité de la locomotion humaine. Par exemple, leur exploitation devient très difficile lorsqu'il s'agit de gérer le croisement des entités le long d'un même chemin. Certaines méthodes [PET 06] proposent de régler ce problème en subdivisant chaque chemin dans sa largeur, les gérant ainsi comme un ensemble de rails parallèles entre lesquels vont pouvoir transiter les entités en mouvement. Mais, même si une telle méthode produit des animations visuellement plausibles, son fonctionnement est trop éloigné du comportement humain pour être exploité dans des simulations réalistes. Les représentations sous forme de cartes de cheminement semblent donc bien adaptées à un processus de planification de chemin général, ne tenant pas compte des autres entités, mais beaucoup moins à un processus de navigation, gérant lui les mouvements locaux.

2.2.3 Le modèle de champs de potentiels

Le modèle à base de champs de potentiels consiste en une définition de l'environnement permettant directement de résoudre les déplacements de personnes. Les champs de potentiels caractérisent les obstacles de l'environnement par des forces de répulsion, et les buts par des forces d'attraction [REI 94] [TRE 06]. Un gradient de forces, ou champ de potentiel, est alors déduit en chaque point de l'environnement comme étant une somme pondérée, le plus souvent par la distance, des potentiels de répulsion et du potentiel lié au but (Figure 11). La navigation d'une entité est alors simulée par une descente de, gradient depuis sa position dans l'environnement.



(a) Environnement d'origine **(b) Exemple de champs de potentiels**

Figure 11: Carte de champs de potentiels : en noir les obstacles (répulsion), en blanc les zones de navigation (attraction). Le gradient de gris exprime la valeur de potentiel dans l'environnement. Cet exemple contient 6 minima locaux : zones isolées à potentiel d'attraction maximal.

Les méthodes basées sur les champs de potentiels s'avèrent simples et efficaces, mais posent le problème des minima locaux : zones de l'environnement où un potentiel minimal isolé apparaît (Figure 11(b)). Ainsi, la méthode de navigation associée va pousser l'entité à se déplacer vers le minimum local le plus proche, qui ne représente pas forcément son but. Pour pallier ce type de problème, des méthodes à base de marche aléatoire sont utilisées. Par exemple, dans RPP (Random Path Planner) [BAR 91], lorsqu'un minimum local est atteint, un ensemble de configurations aléatoires sont tirées puis testées avec une

phase de sortie du minimum et une phase de convergence vers le prochain minimum. Les informations sont alors stockées dans un graphe dont les nœuds sont les minima locaux et les arcs traduisent des chemins entre deux minima. D'autre méthodes existent à base de tirage aléatoire de direction à suivre [CAS 01], plutôt que de configuration, pour échapper au minimum local. Ces méthodes ne sont cependant pas très adaptées à l'animation comportementale. Les comportements induits par les tirages aléatoires, s'ils sont acceptables pour des robots, ne le sont pas pour des humanoïdes car ils sont en dehors de la logique de navigation humaine qui comporte une part importante de planification.

De manière générale, les méthodes à base de champs de potentiels ne sont pas applicables directement à la simulation d'humains, du fait du manque de souplesse dans la méthode de contrôle. On peut néanmoins trouver quelques cas d'application à grande échelle [GLO 04], où le comportement individuel est noyé dans la caractérisation globale des mouvements de milliers d'individus. Notons tout de même que cette technique a deux grands avantages qu'il serait bon de conserver dans une évolution permettant une décision individuelle plus importante. Premièrement, cette méthode permet la fusion de l'information au sein de l'environnement, traduisant tout ce qui intervient dans la navigation par des potentiels qui sont mélangés. Deuxièmement, et c'est une conséquence directe du premier point, ces méthodes introduisent une abstraction très forte de l'environnement, grâce à laquelle l'individu ne raisonne plus sur des entités perçues indépendantes (que ce soit la topologie des lieux ou les autres individus), mais directement sur une abstraction spécialisée pour sa tâche de déplacement.

2.2.4 Planification de chemin

Afin d'exploiter une représentation de l'environnement, qu'elle soit approximative ou exacte, les simulations microscopiques ont généralement recours à un procédé nommé planification de chemin. Ce procédé peut être comparé à un comportement exposé en psychologie, qui a pour but d'extraire un chemin reliant la position courante de l'entité à une destination donnée. Une méthode classique pour effectuer cette planification consiste à utiliser des algorithmes de parcours de graphe, bien que certaines méthodes utilisent d'autres techniques (comme la descente de gradient pour le cas des champs de potentiel. En effet, toutes les représentations topologiques que nous avons introduites sont exploitables sous la forme d'un graphe de points de passages, où les nœuds représentent les zones navigables et les arcs les connexions entre ces zones.

- Algorithmes de parcours

Différents algorithmes de parcours de graphe peuvent être utilisés dans le cadre de la planification de chemin. Ceux que nous décrirons ici se basent sur un critère de minimisation de coût pour extraire le plus court chemin. Ce coût représente souvent une notion de distance spatiale, mais nous verrons dans la section suivante que des heuristiques plus complexes peuvent être utilisées.

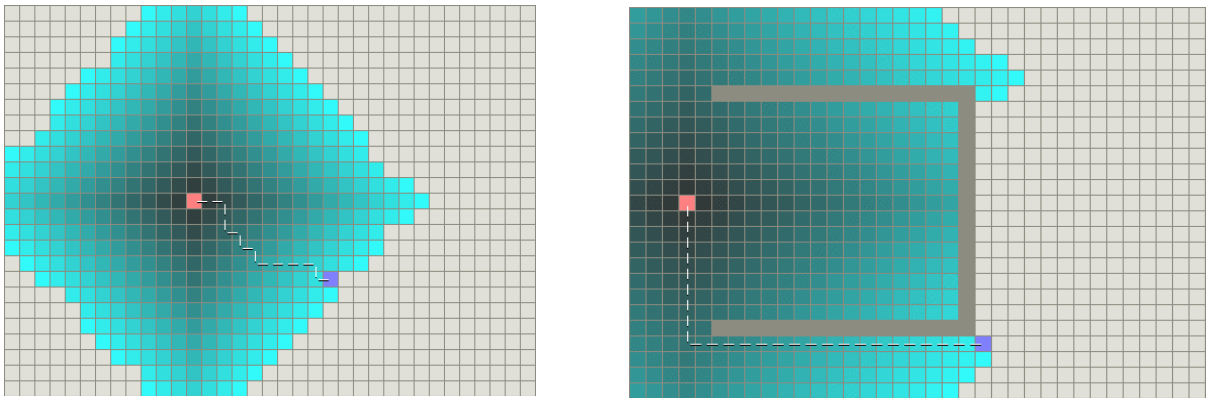
Dijkstra

L'algorithme de Dijkstra, aussi dénommé flood-fill, permet de trouver l'ensemble des meilleurs chemins entre deux nœuds du graphe. Cet algorithme utilise une file de priorité où il stocke pour chaque nœud exploré deux informations :

– la distance parcourue depuis le nœud de départ jusqu'au nœud courant ;

– le nœud précédent, i.e. parcouru avant le nœud courant.

L'algorithme commence avec un nœud source correspondant généralement à la position courante de l'entité, qui n'a donc aucun prédécesseur et un compteur de distance nul. Cet algorithme fonctionne directement sur le graphe topologique, qu'il soit explicite (par exemple avec les cartes de cheminement) ou implicite (par exemple avec les grilles). Afin de trouver un chemin entre deux nœuds, en admettant que la longueur de ce chemin soit l, l'algorithme explorera par propagation circulaire l'ensemble des nœuds se trouvant à une distance inférieure à l (Figure 12).



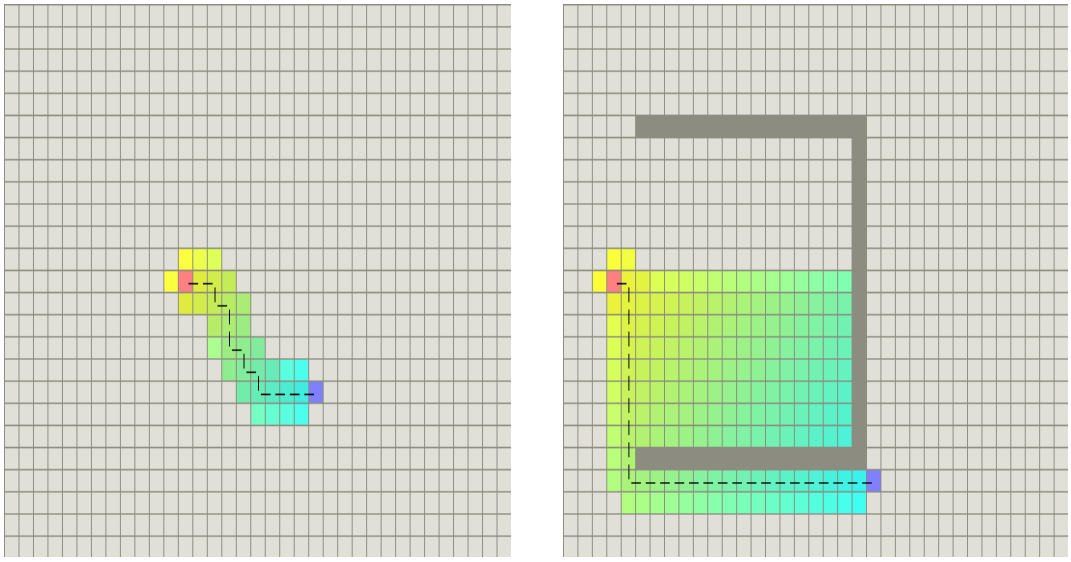
(a) Environnement non contraint **(b) Environnement avec obstacle concave**

Figure 12: Fonctionnement de l'algorithme de Dijkstra dans des cas contraints ou non [101]. Le carré rose représente le nœud source, le mauve la destination. Le gradient de bleus correspond à la distance à l'origine, le plus clair étant le plus éloigné.

Les intérêts majeurs de cet algorithme sont sa robustesse (si un chemin existe il sera forcément trouvé) et son abstraction totale de la destination, si ce n'est comme condition de fin. Son point faible est sa complexité calculatoire, étant de l'ordre de O(A+ N. log N) où A est le nombre d'arcs du graphe, et N son nombre de nœuds. Malgré tout, ce point faible peut aussi être un avantage, l'algorithme supportant des recherches guidées par un but plus conceptuel, associé à la sémantique des nœuds parcourus.

A*

Dans les cas où la performance de la recherche est primordiale, comme en animation comportementale (dans le cas des jeux vidéos par exemple), l'algorithme A* est plus utilisé. Celui-ci fonctionne d'une façon similaire à l'algorithme de Dijkstra mais en ajoutant un coût prédictif aux nœuds correspondant au reste du chemin à parcourir. Le coût total est donc la longueur réelle du chemin jusqu'au nœud, incrémentée par une valeur prédite pour le chemin menant au but, calculée par une heuristique. L'algorithme va ainsi parcourir les nœuds dans l'ordre croissant de leurs coûts associés (Figure 13). La rapidité de convergence de cet algorithme est directement dépendante de la qualité de l'heuristique employée. Dans le cas de la recherche de plus court chemin, l'heuristique employée est une estimation de distance, généralement la norme géométrique séparant le nœud courant du but.



(a) Environnement non contraint **(b) Environnement avec obstacle concave**

Figure 13: Fonctionnement de l'algorithme A*.

L'avantage de cet algorithme réside dans sa rapidité calculatoire. Son inconvénient majeur réside dans le recours à une heuristique. En effet, plus l'évaluation de la longueur de chemin sera complexe, pouvant faire intervenir d'autres paramètres que la distance (par exemple un coût associé à la sémantique), plus cette heuristique sera difficile à expliciter. Ainsi, il est difficile d'utiliser cet algorithme pour des planifications de chemin dont le but n'est pas clairement identifié dans le graphe topologique, comme pour des recherches exploratoires où à buts multiples.

Évolutions du A*

Un certain nombre d'évolutions ont été proposées pour améliorer les propriétés du A*. B. Logan et N. Alechina [LOG 98] proposent l'algorithme ABC (pour A* with Bounded Cost), permettant d'ajouter des contraintes molles à respecter lors de la planification, comme des limitations de temps ou d'énergie.

L'algorithme IDA* [KOR 85] commence par effectuer une recherche en profondeur dans le graphe. Tout d'abord, une distance maximale de parcours D est estimée comme étant la distance au but. L'exploration commence alors avec le nœud d'origine, puis évalue récursivement les voisins jusqu'à ce que l'une des deux conditions suivantes soit vérifiée : soit un chemin est trouvé, il est alors minimal, soit un nœud est parcouru dont le coût est supérieur à D. Dans ce dernier cas, la recherche continue avec une borne D remise à jour par la plus petite estimation de la distance au but évaluée précédemment. Dans les faits, cet algorithme est plus lent que le A* traditionnel, mais nécessite moins de mémoire.

L'algorithme HPA* [BOT 04] (pour Hierarchical Pathfinding A*) consiste à redécouper le graphe de l'environnement afin de hiérarchiser la planification. Ensuite, l'algorithme propose d'associer des pré-calculs de plus courts chemins entre des points clefs de chaque partie abstraite. Ainsi, cet algorithme va pouvoir évaluer un chemin de haut niveau en ne manipulant que peu de nœuds.

Critères de planification de chemin

Différents critères peuvent être évalués pour le coût des algorithmes précédents. Les plus anciennes méthodes, ainsi que la plupart de celles utilisées en animation, se basent uniquement sur la distance séparant les nœuds. Mais, comme l'ont montré les études en psychologie cette information de distance est nécessaire, mais certainement pas suffisante pour rendre compte de la complexité du raisonnement humain. Ainsi, dans des simulations se voulant plus réalistes, d'autres critères d'évaluation sont proposés.

Les calculs de ces chemins reposent sur l'algorithme de graphes, ainsi, pour une entité ayant une destination unique, on calcule les chemins possibles grâce à A* (pour des destinations multiples, on pourra utiliser un algorithme de type "flood fill"). Dans [LAM 04], Lamarche et Donikian proposent un algorithme hiérarchique de calcul de chemins exploitant la structure d'abstractions hiérarchique introduite. On commence par calculer les chemins au niveau d'abstraction le plus haut, puis on descend dans la hiérarchie. Le coût du calcul est alors significativement diminué.

Dans [PDB06], Paris et al. proposent une planification de chemin basée sur une heuristique multicritères. Ce coût est alors ramené à une somme pondérée de critères (distance, densité de population, changements de direction, sens des flux de population) convertis en unités temporelles. Ainsi, le chemin choisi sera celui dont le coût temporel est le plus faible. On obtient alors un bon compromis entre tous les critères à prendre en compte. La planification se fait ici aussi de manière hiérarchique de façon à tirer profit des deux niveaux d'abstraction décrits précédemment et ainsi réduire les coûts de calcul.

2.3 Comportement de groupes

Dans les villes réelles, de nombreux piétons font partie d'un groupe, qu'ils soient assis, debout ou marchant vers leur but commun. Ils se comportent différemment que s'ils étaient seuls: ils adaptent leur rythme aux autres membres, attendre les uns des autres, ils peuvent se séparer dans les endroits serrés pour éviter les collisions, mais ils se regroupent ensuite. Plusieurs approches ont été prises afin de simuler de tels comportements. La première approche qui offre des résultats impressionnants est celui de Reynolds [REY 87] [REY 99] [REY 06], qui a conçu des règles intelligentes pour simuler des nuées d'oiseaux et de poissons.

Bayazit et al. [BAY 03] a créé des comportements supplémentaires basés sur les travaux de Reynolds pour améliorer encore le comportement de groupe. Ils ont introduit trois comportements de groupe:

- Le comportement de base est localement similaire au comportement flocking de Reynolds. Cependant, globalement le groupe est dirigé vers son but par une approche à base de carte de cheminement.

- Le comportement de passer par des couloirs étroits nécessite de désigner un chef, qui suit le chemin assigné. Les autres membres disposés dans une file d'attente le suivent.

- Le comportement berger nomme un membre du groupe comme le chien de berger. L'objectif du chien est de déplacer les autres membres du groupe (les moutons) vers un but, alors que la seule règle des membres du groupe est de s'éloigner du chien. Si un sous-groupe se sépare du troupeau, c'est le travail du chien de les regrouper.

Musse et Thalmann directement définies foules comme un ensemble de groupes. L'information est hiérarchiquement partagés [MUS 01]: l'information des positions des obstacles statiques et des buts sont partagés par toute la foule, tandis que l'intelligence, la mémoire, l'intention et la perception sont centralisées dans chaque structure de groupe, placés dans leur chef. Afin de rester proches les uns des autres, les membres du groupe marchent à des vitesses similaires, suivent le même chemin, et attendre les autre à l'arrivée d'un but. Enfin, les agents ont une structure plus simple que de groupes. Ils évitent les collisions les uns avec les autres et peut décider de changer de groupe ou de devenir un leader. O'Sullivan et al. présente ALOHA, une architecture capable de simuler des

comportements de groupe avec une approche de niveau de détail [OSU 03]. Leur technique est axée sur les interactions sociales, à savoir, agents parlant et écoutant les uns aux autres.

Niederberger et Gross [NIE 03] a introduit un système générique pour des agents autonomes et réactifs, organisés en groupes hiérarchiques. Les groupes sont définis avec des modèles spéciaux, et instancier avec un nombre donné de membres. Ce processus peut être répété pour obtenir une hiérarchie de groupes, par exemple, une famille est composée du père, la mère et les enfants: le père est le chef, et la mère suit le père, mais elle est aussi le chef des enfants.

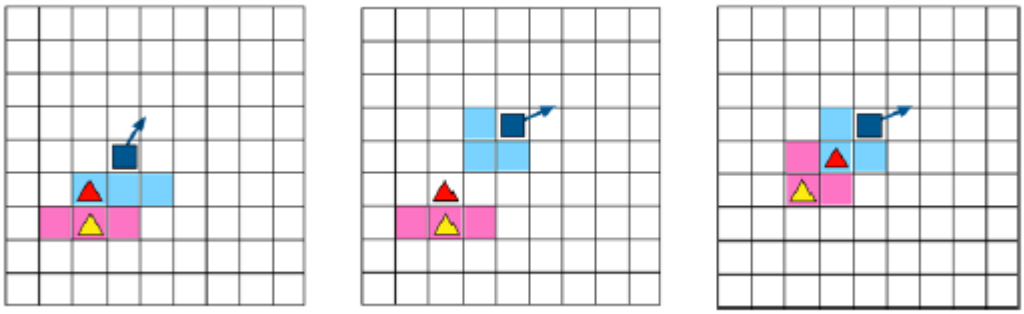


Figure 14: Les membres du groupe (triangles rouges et jaunes) à la suite de leur chef (en bleu foncé) [LOS 03].

Loscos et al. [LOS 03] Introduit un modèle simple pour simuler des groupes de piétons, basée sur une décomposition de l'environnement en cellules régulières. Groupes de 3 à 10 personnes sont créés et un chef est désigné pour chaque groupe. A chaque pas de temps, le leader se déplace d'abord dans une cellule voisine, et marque 3 cellules derrière lui pour informer ses suiveurs, où ils doivent se déplacer. Puis les membres suivants tentent d'atteindre les cellules conseillées, et marque de son tour 3 cellules derrière eux pour montrer la voie aux membres prochain. Ce processus est illustré à la figure 14.pour empêcher le chef de régulièrement marcher devant tout le monde, ils ont décidé de ne pas l'afficher.

Si les membres d'un groupe agissent individuellement (pas de leader qui prend les décisions), ils peuvent décider d'atteindre un objectif commun en utilisant des chemins différents, résultant une division du groupe. Un tel cas est illustré à la figure 15. Pour éviter ce problème, Kamphuis et Overmars [KAM 04] a maintenu une cohésion de groupe, en limitant la dispersion longitudinale et latérale de ses membres. Le mouvement du groupe vers son but et l'évitement des collisions sont gérés en utilisant les forces sociales.

Plus récemment, Kwon et al. [KWO 08] ont proposé une solution pour modifier le mouvement (animation et de navigation) d'un groupe tout en préservant sa formation. Ils créent un graphe dont les sommets sont des positions de personnages à Frames précis. Arête définissent la formation de voisinage entre les individus et leurs trajectoires. Lorsqu'un utilisateur modifie ce graphe, on prend soin de minimiser la distorsion des arrangements locaux entre les sommets adjacents.

Figure 15: Les problèmes classiques rencontrés lorsque les membres d'un groupe chercher individuellement un chemin dans un environnement complexe. [KAM 04].

2.4 Conclusion

Nous avons exposé dans ce chapitre premièrement les différentes techniques de rendu de foule d'humains virtuels, ensuite les techniques permettant la représentation de l'environnement, et son exploitation lors de la planification de chemin et la gestion du groupe.

Le premier point que l'on peut soulevé concernant le rendu est que l'exploitation de la technique de niveau de détail, et la combinaison de différentes méthodes de représentation (à base d'image, maillage statique, maillage simplifie), permet de simuler de grandes foules suffisamment détaillés même si le point de vue est proche de l'individu.

Le deuxième point est la difficulté d'associer réalisme et performances. Notamment concernant les représentations de l'environnement, on a pu voir que les versions approximatives offrent un accès rapide à la description topologique, alors que les approches exactes permettent la conservation maximale de l'information géométrique. Il est indéniable que dans le cadre de simulations devant produire des résultats comparables au réel, le réalisme prend la place la plus importante.

Le troisième point c'est que ces différentes techniques se basent, la plupart du temps, sur une projection en 2D de l'environnement pour rendre les calculs performants, mais cela introduit aussi une perte d'informations. En effet, la notion de hauteur des obstacles permet mieux de simuler les comportements des humanoïdes. Tout d'abords, elle permet de ne pas contraindre le déplacement des humanoïdes à des sols plans. Cette prise en compte de la hauteur permet d'éviter correctement les obstacles de faible hauteur tels qu'un muret qui contraint le déplacement mais pas la visibilité des humanoïdes. L'utilisation d'une structure en 2D1/2 qui intègre cette notion de hauteur peut donc générer des simulations plus réalistes.

Le dernier point concerne les paramètres devant être pris en compte pour évaluer un chemin. La planification de chemin repose sur les connaissances à priori de l'entité sur l'environnement, sur les conditions dans lesquelles elle se trouve (densité, appartenance à un groupe, situation de panique ...etc) sur les informations qu'elle peut extraire de son environnement (visibilité) et donc sur un "coût" associé à chaque chemin possible.

Chapitre 3 Modèles de foules existants

3.1 Un modèle de foule faible densité : Autonomous pedestrian

Les travaux sur les poissons artificiels de Demetri Terzopoulos [TUX 94] ont inspiré de nombreux chercheurs pour la simulation de foules d'humains. Wei Shao et lui ont développé un modèle visant à rendre autonomes les humains artificiels [SHA 05]. Ils s'appuient sur une vision individualiste, où chaque piéton est doté d'une composante décisionnelle (cognitive) en plus de son comportement réactif habituel. Leur but n'était pas de simuler une foule à proprement parler, mais de peupler un environnement urbain d'individus autonomes présentant un comportement réaliste.

Figure 16 : Vues de l'ancienne Pennsylvania Station recréée à partir de plans et de photos d'époque (à gauche la salle d'attente, à droite un des halls).

La première étape a été de construire l'environnement : l'ancienne Pennsylvania Station de New York a servi comme modèle. Un ensemble hiérarchique de cartes permet une perception optimale de l'environnement (Figure : 17). Les nœuds de la carte topologique correspondent aux grandes pièces de la gare et les arêtes représentent les accès entre ces pièces. Chaque nœud contient deux cartes pour la perception (une pour les objets statiques, une autre pour les mobiles) et deux cartes pour la planification de chemin (une grille quadtree pour la planification de chemin longue distance et une grille régulière plus détaillée pour les courtes distances).

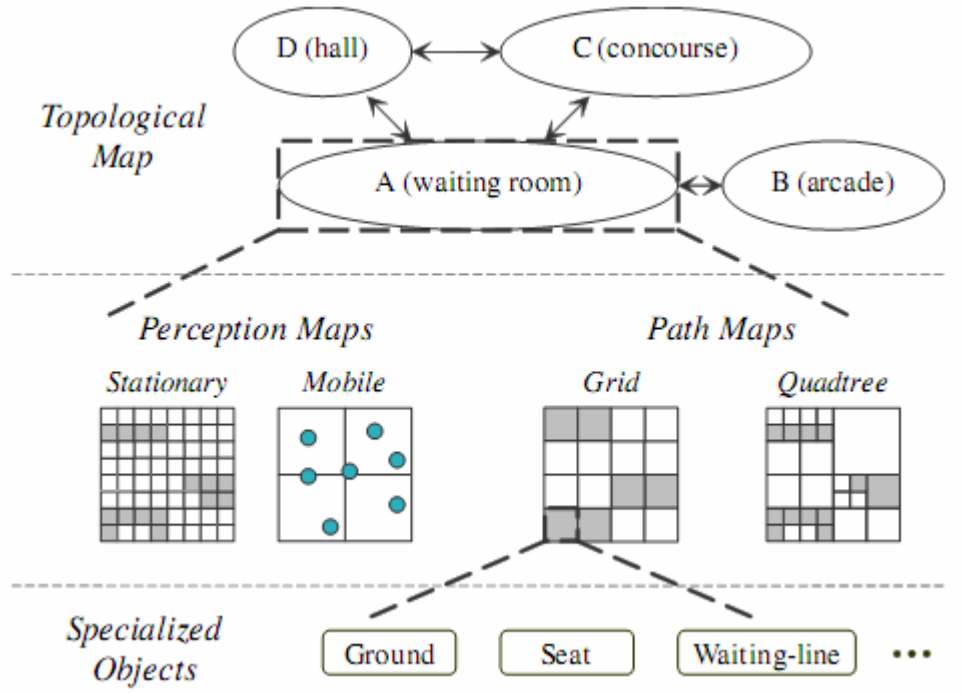


Figure 17 : modèle hiérarchique de l'environnement

Les fonctionnalités bas niveau (l'apparence des piétons, leur animation corporelle et leur locomotion) sont assurées par DI-Guy, un logiciel payant.

Les piétons doivent percevoir trois informations importantes. La première, la hauteur du sol, est contenue dans la carte d'objets statiques. Une fois perçue, la hauteur du sol est envoyée aux couches bas niveau pour que le piéton pose le pied à la hauteur appropriée. Les objets statiques sont perçus en lançant un éventail de rayons dont la longueur et le nombre déterminent l'acuité visuelle de l'agent. Chaque cellule de la carte traversée par un rayon est interrogée, elle renvoie alors des informations sur les objets qu'elle contient. Pour percevoir les objets mobiles, le piéton examine en priorité les cases les plus proches de lui avant de passer aux suivantes. La perception se termine lorsque seize objets mobiles ont été repérés, ainsi le coût en temps de calcul est constant. Ce choix est justifié par le fait qu'à un instant donné les gens portent de l'attention sur un nombre limité d'autres personnes, généralement les plus proches.

Chaque piéton possède un ensemble de capacités motrices, comme rester immobile, avancer, tourner, accélérer et ralentir. Les comportements dits « réactifs » font le lien entre les perceptions et ces capacités. Six routines en gèrent le commencement, la terminaison et le séquençage à court terme en se basant à la fois sur les perceptions et sur l'état interne de l'agent :

Routine A : éviter les obstacles statiques

Routine B : éviter les obstacles statiques lors d'un virage

Routine C : maintenir une séparation avec les piétons voisins

Routine D : éviter les piétons

Routine E : éviter les piétons dangereusement près

Routine F : vérifier les nouvelles directions

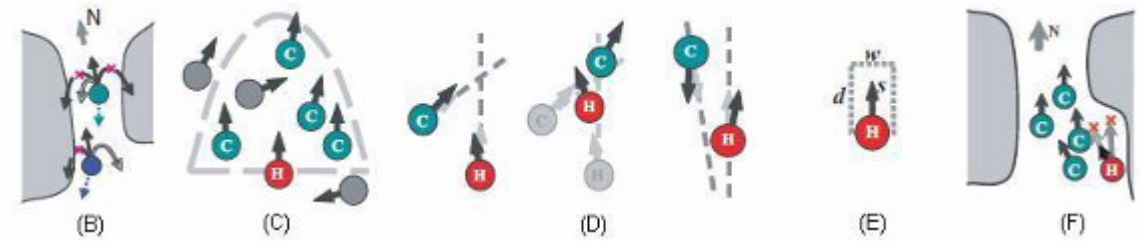


Figure 18 : Comportements réactifs (routines)

La situation vécue par un piéton est toujours une combinaison des six situations clés couvertes par ces routines, le problème est alors de les séquencer correctement. La solution apparaissant comme la meilleure en termes de nombre de collisions est la suivante : C-A-B-F-E-D.

Ces comportements réactifs permettent aux piétons de déambuler en évitant (presque) toutes les collisions. À un niveau au-dessus, ils sont capables de naviguer dans la gare, c'est-à-dire de se rendre là où ils le désirent tout en choisissant un chemin optimal. Par exemple ils choisissent le portail ou la cage d'escalier à emprunter en fonction de sa proximité, mais aussi en fonction de la densité de personnes présentes autour. Si, en suivant un chemin, ils détectent un raccourci, ils modifient leur trajectoire pour l'emprunter. Et enfin, en arrivant à destination ils calculent un chemin plus précis pour atteindre la cible de manière plus réaliste.

D'autres comportements ont été implémentés pour permettre une occupation réaliste de la gare par les agents, comme s'asseoir sur un siège inoccupé, s'approcher et regarder un spectacle de rue, rencontrer un ami et bavarder avec, faire la queue à un distributeur ou à un guichet, etc.

Chaque piéton entretient une pile de buts, le sommet étant le but courant. Il possède également un ensemble de variables représentant ses besoins physiologiques, psychologiques et sociaux (état interne), tels que la fatigue, la soif, la curiosité, le besoin

d'acheter un billet, etc. Lorsque la valeur d'un de ces besoins dépasse un certain seuil, un mécanisme de sélection d'action (Figure 19) va choisir le comportement adéquat pour le satisfaire et faire diminuer la variable. Une tâche en cours d'exécution (par exemple « se rendre au guichet et acheter un ticket ») peut être interrompue si certaines conditions sont réunies (par exemple « avoir soif » et « présence d'un distributeur de boissons à proximité », dans ce cas le but « acheter une boisson » est placé au sommet de la pile). Les conditions ne proviennent pas forcément du module cognitif (ici la présence d'un distributeur).

Concrètement, la pile de buts représente ce qui doit être fait et l'état interne pourquoi cela doit-il être fait. Le module cognitif décide comment le faire à un haut-niveau (aller ici, acheter ceci) tandis que le module comportemental décide comment le faire concrètement (et le fait).

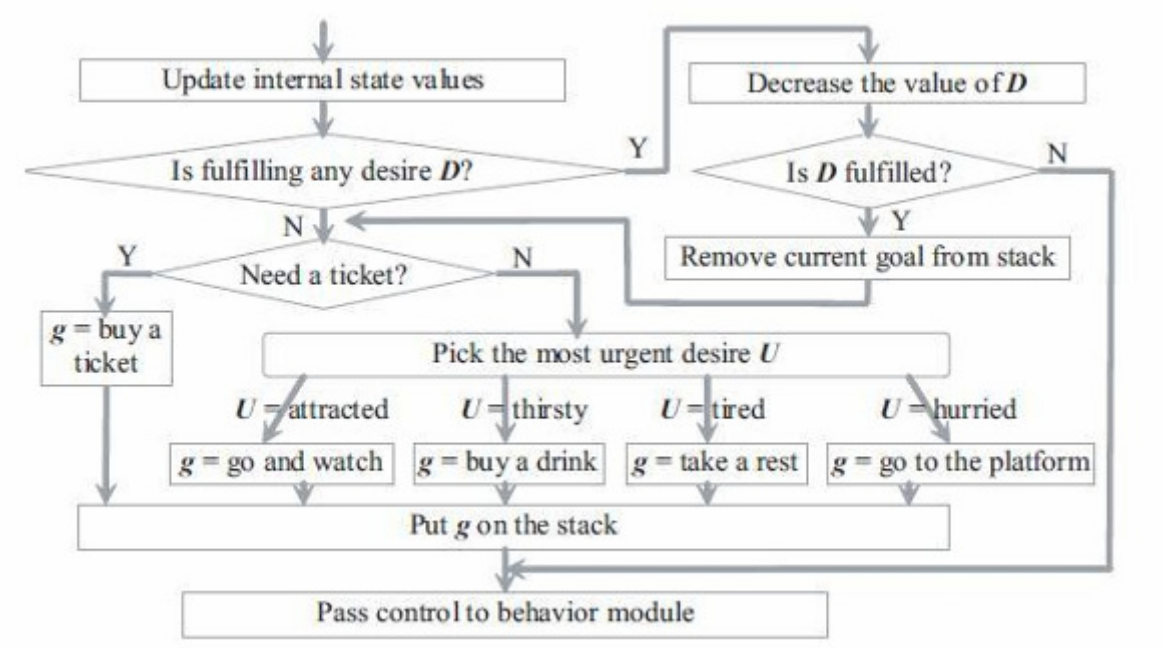


Figure 19 : Architecture de sélection d'action

Ce modèle est décrit dans [SHA 05] et [SHA 05B]. La simulation seule peut tourner en temps réel (30 fps) avec 1200 piétons, mais pas si le rendu 3D est effectué en même temps. Il est complet, robuste et réaliste cependant il ne gère pas de « vraie » foule puisque celle-ci est plutôt éparse. Il manque également un point important, Shao et Terzopoulos se sont concentrés sur la notion d'individu et n'ont pas pris en compte les petits groupes de piétons. Si deux personnes (ou plus) se connaissent, elles devraient marcher ensemble et coordonner leurs choix sur une période donnée.

3.2 Modèle HIDAC

Avec HiDAC (pour High Density Autonomous Crowds), Nuria Pelechano et al. Mêlent règles psychologiques et géométriques, ainsi que forces sociales et physiques [PEL 07] en espérant ne garder que les bons côtés de chaque approche. Ils se concentrent sur les déplacements locaux et la recherche de chemin dans un environnement dynamique (sa topologie peut changer en cours d'exécution), mais fermé (toujours limité par des murs).

Chaque agent a son propre comportement selon ses variables personnelles représentant ses facteurs physiologiques et psychologiques. Celui-ci est calculé à deux niveaux. Le module haut niveau traite la navigation, l'apprentissage, la communication avec les autres agents et la prise de décision tandis que le module bas niveau s'occupe de la perception, de l'évitement d'obstacles et d'autres comportements réactifs. Le module haut niveau reçoit les changements de l'environnement perçus et prend une décision en fonction de cette information et de ses connaissances enregistrées. Lorsqu'il a décidé dans quelle pièce se rendre, il envoie un point d'attraction au module bas niveau pour que celui-ci gère le trajet. Une fois l'agent rendu à la cible, le module bas niveau demande au module supérieur le prochain point d'attraction.

Le module bas niveau est composé de trois sous-modules : perception, déplacement et locomotion. En fonction des informations qui lui sont envoyées par le sous-module de perception et de l'état interne de l'agent (panique, impatience, etc) le sous-module de déplacement calcule sa vitesse et sa position suivante. Celles-ci sont alors communiquées au sous-module de locomotion pour qu'il prenne en charge les mouvements à effectuer.

Les deux modules sont influencés par l'état psychologique et physiologique de l'agent. Par exemple les prises de décisions (module haut niveau) sont altérées par la panique, mais la vitesse l'est aussi (bas niveau). L'état interne est en retour modifié par ces deux modules, d'une part par les perceptions (bas niveau) et de l'autre par la communication (haut niveau). La figure 20 résume cette architecture.

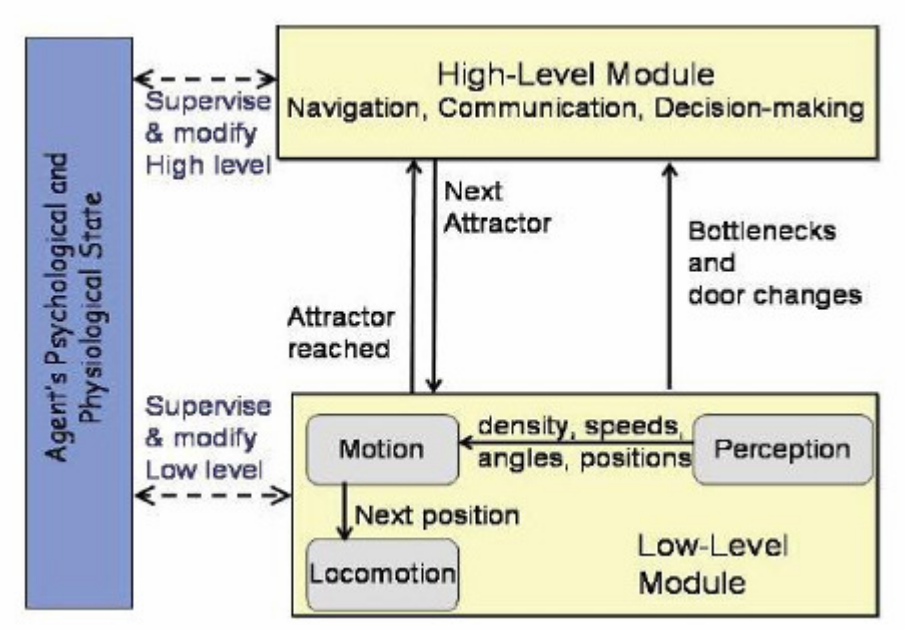


Figure 20 : Vue d'ensemble de l'architecture de HiDAC

La force exercée sur un piéton à un instant donné résulte de considérations à la fois sociales (autres agents), géométriques (murs) et motivationnelles (cible). La force subie par un piéton i (F_i) est donc une combinaison du point d'attraction (F^{At}_i), de la répulsion des murs m (F^{Mu}_{mi}), des obstacles k (F^{Ob}_{ki}), des autres agents j (F^{Ag}_{ji}). Il faut également empêcher les changements de direction trop brutaux ($F_i[n-1]$). Toutes ces forces sont additionnées selon différents poids ω qui sont calculés en fonction de règles psychologiques et géométriques.

$$F_i[n] = F_i[n-1] + F_i^{At}[n]\omega_i^{At} + \sum_m F_{mi}^{Mu}[n]\omega_i^{Mu} + \sum_k F_{ki}^{Ob}[n]\omega_i^{Ob} + \sum_{j\neq i} F_{ji}^{Ag}[n]\omega_i^{Ag} \qquad (1)$$

La nouvelle position est ensuite calculée à partir de cette force, de la vitesse maximale de l'agent, de la gestion de l'interpénétration, de la poussée par les autres agents, de l'évitement des personnes tombées au sol et bien sûr du temps entre deux pas de simulation.

$$P_i[n+1] = P_i[n] + \alpha_i[n]v_i[n]\left((1-\beta_i[n])f_i[n] + \beta_i[n]F_i^{AgT}[n]\right)T + r_i[n] \qquad (2)$$

Où

α représente si l'agent est poussé ou bien si il se déplace dans la direction désirée,

v est sa vitesse,

f est le vecteur force calculé en (1) normalisé

$\mathbf{F^{AgT}}$ est la force de répulsion des agents tombés au sol et **β** la priorité qui leur est accordée (car ils peuvent être enjambés)

r i est la correction de l'interpénétration

T l'incrément de temps entre deux pas de simulation.

La figure 21 donne une idée des paramètres pris en compte : la distance, l'orientation (tangente) et le vecteur vitesse des obstacles (ici d'autres agents). Elle montre également le rectangle d'influence dans lequel les obstacles sont pris en compte.

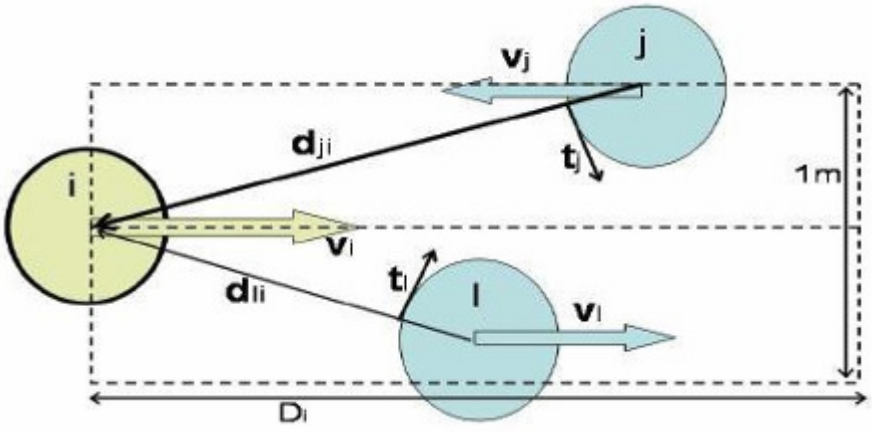


Figure 21 : Rectangle d'influence et évitement de collision

Les agents disposent également d'une petite aire en forme d'ellipse (dont la taille varie selon leur personnalité) qu'ils tentent de garder vide. Elle sert notamment pour les comportements d'attente en file. HiDAC détecte les collisions, ce qui permet de réagir à celles-ci, les agents peuvent alors se pousser entre eux pour dégager un passage. Si un agent subi de trop fortes poussées, il peut tomber. Il devient alors un obstacle que les autres doivent éviter (Figure : 22 à droite). Dans certains cas ils peuvent être amené à l'enjamber (en cas d'urgence, de manque d'espace, etc) (Figure : 22 à gauche). En voyant les autres courir (ou en entendant une alarme, etc), les piétons peuvent entrer en mode panique. Leur tendance à paniquer dépend de critères liés à leur personnalité. Une fois dans ce mode, ils se déplacent plus rapidement, ont tendance à pousser et sont agités (Figure : 23).

En cas d'embouteillage, le module bas niveau des agents impatients demande au module haut niveau de trouver un chemin alternatif. De la même manière, lorsqu'un changement dans l'environnement est perçu par le module bas niveau (comme une porte qui se ferme) il en informe haut niveau qui recalcule alors un nouveau point d'attraction : les agents s'adaptent en temps réel aux changements de leur environnement.

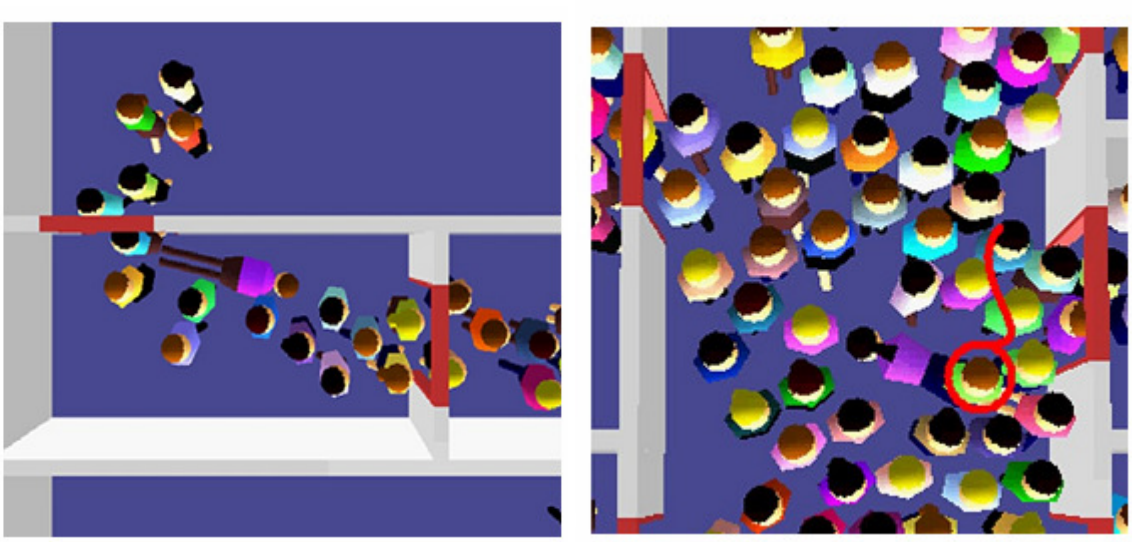


Figure 22 : (à droite) Des agents évitant un piéton au sol (à gauche) Des agents piétinant un agent tombé

Pelechano propose plusieurs comportements.

1. Le queuing (les agents se mettent en file indienne), comportement qui se retrouve lors d'un niveau de stress bas.

2. Le pushing , où des agents dépassant un niveau de stress ou d'exitation voient leur espace personnel nécessaire diminuer de par la nécessité grandissante de survie. En se rapprochant d'un autre agent ayant un espace personnel plus grand, ils vont entrer dans son espace personnel sans pour autant faire entrer cet autre agent dans la leur. Cela aura pour effet de pousser un agent calme, et ainsi de faire un passage à l'agent plus stressé.

3. La chute d'un agent et la possibilité de se faire piétiner. Chaque agent a un certain niveau d'équilibre, et si les forces de pushing lui parviennent et dépassent un certain seuil dans la même direction, l'agent tombe et devient un obstacle piétinable si les agents environnants ont un niveau de stress élevé.

4. Propagation de la panique. Soit par communication directe entre agents, soit par perception de l'´etat de panique des agents environnants. La panique doit se propager selon un taux de diffusion, avec un seuil à atteindre. Si le seuil est atteint alors l'agent panique (augmentation de la vitesse, de la force .... et comportement de panique).

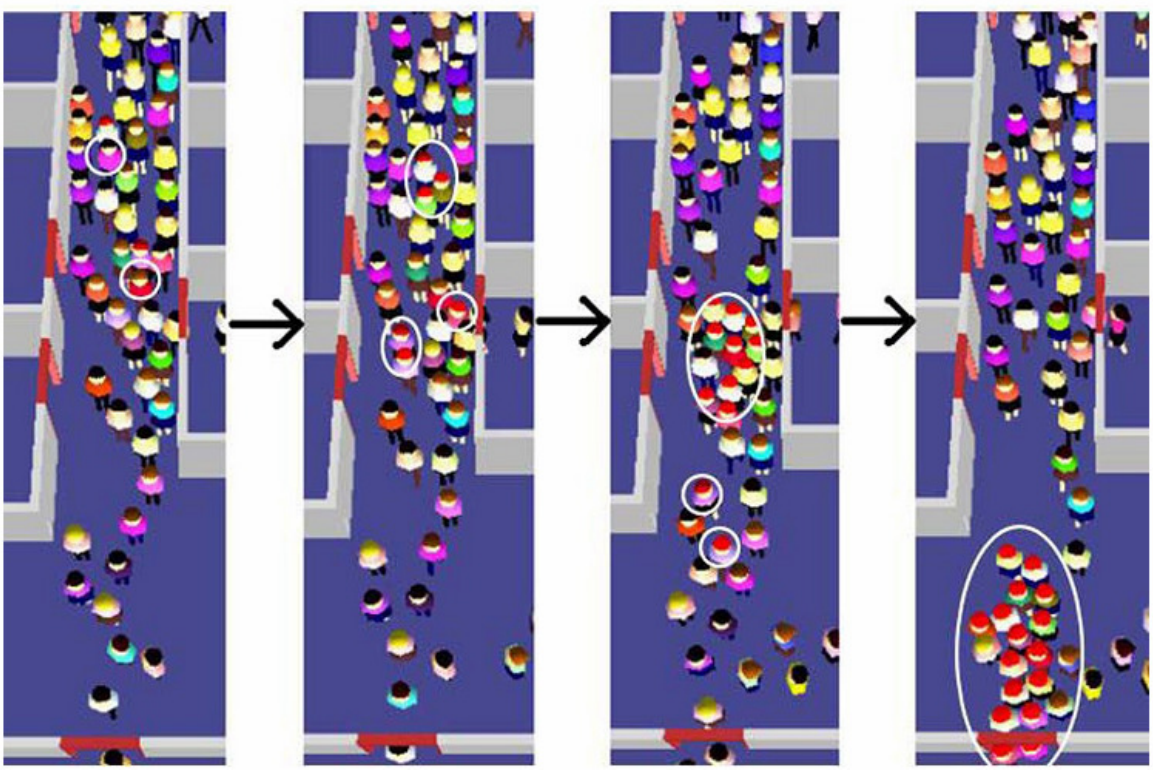


Figure 23 : Propagation de la panique

Ce modèle peut maintenir une simulation à 25 fps (rendu 3D compris) avec 600 agents. Il est possible de monter jusqu'à 1800 agents en n'effectuant pas le rendu. Il représente clairement les deux niveaux réactifs et cognitifs, aussi appelé respectivement ici module bas-niveau et module haut-niveau. La partie plus réactive ayant des relations avec la partie cognitive pour annoncer un changement d'´etat du monde déjà connu comme une modification des chemins possibles (une porte s'est fermée, un chemin est obstrué...), ou pour demander un nouveau point de passage. Les deux niveaux d'abstraction ont accès à des connaissances à la fois physiologique comme l'âge ou le poids et à des connaissances psychologiques comme le stress.

Bien que populaire, l'approche discrète n'est pas la seule utilisée. Nous allons dans ce que suit présenter d'autres méthodes. Nous commencerons par une approche à l'opposé des agents, où la foule est vue comme un système continu (continuum crowds). Nous parlerons ensuite de l'union de ces deux approches avec la dynamique globale (aggregate dynamics) avant de terminer sur deux approches plus originales : les patchs de foules et l'imitation de vidéos.

3.3 Dynamique continue

Avec les foules à dynamique continue, l'objectif de Treuille, Cooper et Popovic est de simuler un très grand nombre de piétons homogènes sans avoir à expliciter de règles pour l'évitement des collisions. Se basant sur les travaux de Hughes, ils voient la foule comme un système continu [HUG 03]. Dans leur modèle la planification globale de chemin et l'évitement local de collision sont unifiés dans une seule et même structure appelée champs de potentiel dynamique [TRE 06]. Leur modèle mathématique est basé sur trois hypotheses :

- Chaque personne essaie d'atteindre un point géographique précis.
- Les gens se déplacent le plus rapidement possible.
- Il existe un champ d'inconfort g qui fait que, si les autres conditions sont équivalentes, un personne préfère être au point x plutôt qu'au point x' quand g(x) < g(x').

Ainsi les personnes se déplacent en essayant de minimiser la longueur totale du chemin, le temps total pour arriver à destination et l'inconfort ressenti le long du trajet. Ceci revient à minimiser l'expression suivante :

$$\alpha \int_p fds + \beta \int_p fdt + \gamma \int_p fdt$$

Le premier terme correspond à la longueur du chemin, le second au temps total et le dernier à l'inconfort. ***P*** est l'ensemble des chemins entre un point x donné et la destination, ***ds*** signifie que l'intégrale est calculée en fonction de la longueur, tandis que ***dt*** veut dire qu'elle l'est par rapport au temps. Ces deux valeurs sont reliés par ***ds = f.dt*** où ***f*** est la vitesse. Chaque intégrale est pondérée par un coefficient (α,β,γ) en fonction de son importance. Cette (expression peut être simplifiée par : $\alpha \int_p fds$ où $c = \frac{\alpha f + \beta + \gamma g}{f}$

C est appelé champ de coût unitaire. Si on définit une fonction de potentiel $\Phi : R^2 \rightarrow R$ qui vaut 0 dans la zone cible et qui satisfait $\|\nabla\Phi(x)\| = C$ alors tous les chemins optimaux vers la cible suivent le gradient de cette fonction.

Treuille, Cooper et Popovic discrétisent ce modèle à l'aide de grilles régulières. A chaque étape de simulation, il faut construire une grille de densité à partir des positions courantes des piétons, puis construire une grille de coût unitaire C, combiner ces grilles avec d'autres

(représentant les zones cibles, les limites, etc) pour former la grille de potentiel et enfin utiliser ce champ de potentiel pour mettre à jour les positions des piétons. Tout ceci est résumé par la figure 24

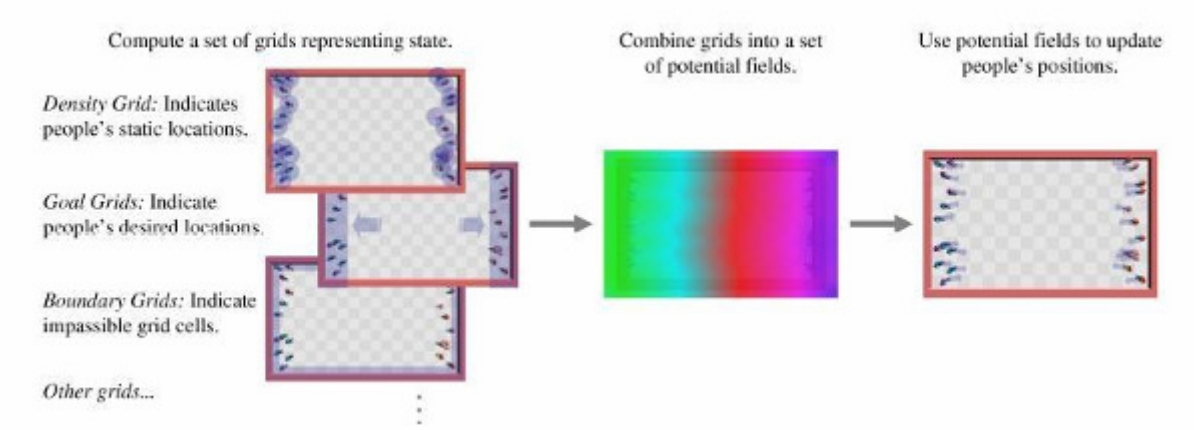


Figure 24 : Vue d'ensemble de l'algorithme

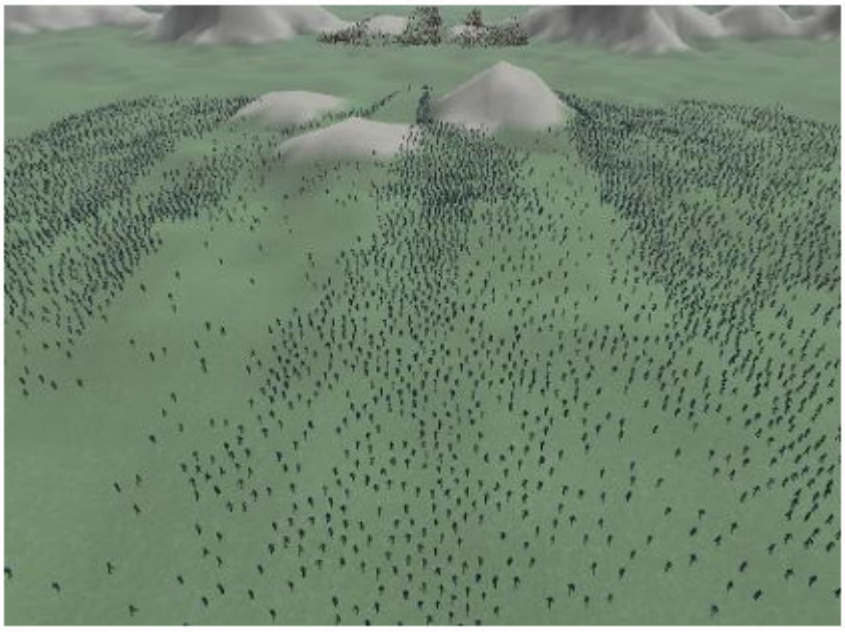


Figure 25 : 8 000 personnes en poursuivant 2 000 autres.

Cette méthode parvient à simuler 10 000 personnes (une armée de 8 000 soldats poursuivant un plus petit groupe de 2 000, figure 25) à 12 fps. Elle n'a pas la flexibilité des approches agents mais est conçue pour modéliser de grands groupes homogènes.

3.4 Dynamique globale

Rahul Narain et ses collègues ont quant à eux adopté les deux approches à la fois : à la représentation classique d'une foule par des agents ils ont ajouté une représentation continue de la densité et de la vitesse du flux de piétons [NAR 09]. Ils ont en fait mappé l'évitement de collision local dans le domaine continu afin d'obtenir une contrainte unilatérale d'incompressibilité (UIC). Cette contrainte agit à grande échelle et permet d'accélérer la simulation. L'idée est de reproduire le fait que, à haute densité, les piétons ont une liberté de mouvement réduite.

La première étape consiste à calculer le vecteur vitesse désiré $\mathbf{v^d_i}$ (prefered velocity) de chaque agent ***i***. La méthode utilisée pour ce calcul importe peu sur le résultat final, cela peut-être une approche continue, un module de navigation ou même un script. Ensuite, à partir de ces vitesses et de la densité est construite une grille correspondant à une discrétisation de la surface sur laquelle les marcheurs se déplacent (splatting). Cette grille est appelée champ de flux. La contrainte unilatérale ne peut pas être supérieure à un certain ρ exprime le fait que la densité maximum qui est fonction de la distance minimale entre deux piétons $\mathbf{d_{min}}$ et d'une constante $\boldsymbol{\alpha < 1}$ (qui assure le fait que l'état où la foule est parfaitement serrée est rarement atteint).

$$\rho < \rho_{\max} = \frac{2\alpha}{\sqrt{3}d_{\min}^2}$$

Il faut maintenant appliquer cette contrainte au champ de flux. En supposant que le nombre total d'agents est constant et que chaque agent parcourt le plus de distance possible dans la direction de son vecteur vitesse désiré, la solution est alors de la forme :

$$v = v_{\max} \frac{v^d - \nabla p}{\left\| v^d - \nabla p \right\|}$$

La vitesse maximale est notée $\mathbf{v_{max}}$, $\mathbf{v^d}$ est la vitesse du champ de flux avant la projection de la contrainte et **p** est la « pression » scalaire, $\mathbf{p} > 0$ signifiant que la densité a atteint le maximum autorisé par la contrainte d'incompressibilité. Une fois le nouveau champ de flux calculé, la dernière étape consiste à retrouver pour chaque agent **i** le vecteur vitesse final $\mathbf{v_i}$. On le définit le comme une interpolation du vecteur vitesse désiré et du champ de flux contraint à la position $\mathbf{x_i}$ de l'agent.

$$v_i = v_i^d + \frac{\rho(x_i)}{\rho_{\max}}\left(v(x_i) - v_i^d\right)$$

Les piétons vont alors se déplacer en suivant ce nouveau vecteur vitesse. L'algorithme est résumé par la figure 26.

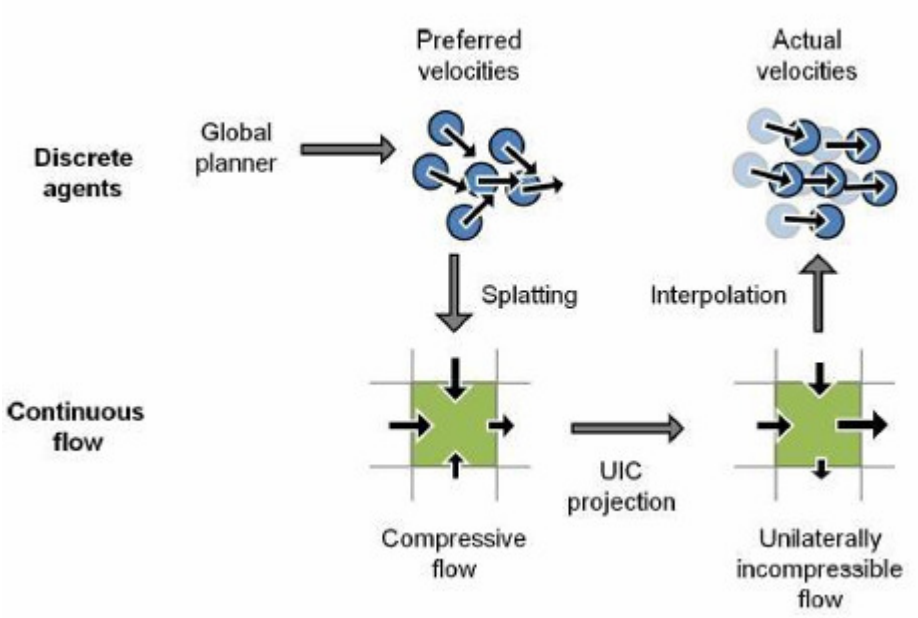


Figure 26 : Vue d'ensemble de l'algorithme de dynamique globale

Ce modèle permet de générer des foules comptant jusqu'à 100 000 agents mais à seulement 2 fps. Pour conserver un rafraichissement d'environ trente images par seconde il faut se limiter à environ 10 000 agents (figure 27). Les performances varient considérablement selon que l'environnement contienne des obstacles ou non.

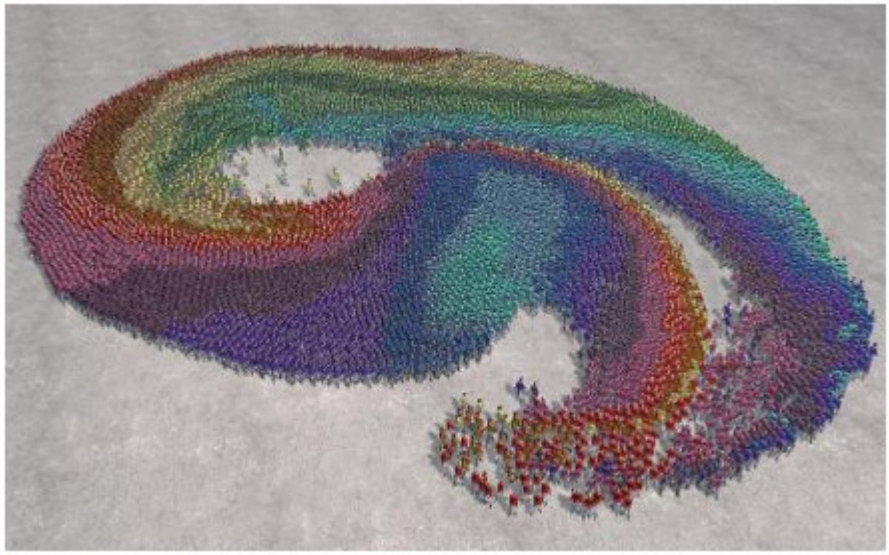


Figure 27 : 10 000 personnages se déplaçant dans des directions opposées.

3.5 Patchs de foule

Dans le but de peupler des environnements virtuels pouvant atteindre de très grandes dimensions, Yersin, Pettré et Thalmann ont développé la technique des patchs de foule (crowd patches) [YER 09]. Cette méthode pré-calcule un certain nombre d'éléments (comme les trajectoires) et les réutilise à la volée, permettant ainsi la génération du monde et de ses « habitants » en temps-réel pendant l'exécution. Les données pré-calculées se présentent sous forme de patchs, c'est-à-dire un morceau de surface contenant l'ensemble des positions des agents en fonction du temps. Toutes les trajectoires sont cycliques, leur point d'arrivée et leur point de départ sont confondus. De cette manière elles peuvent être jouées périodiquement. On distingue deux types de mobile dans un patch : les objets endogènes dont la trajectoire est entièrement compris à l'intérieur du patch, et les objets exogènes qui peuvent sortir du patch (figure 28).

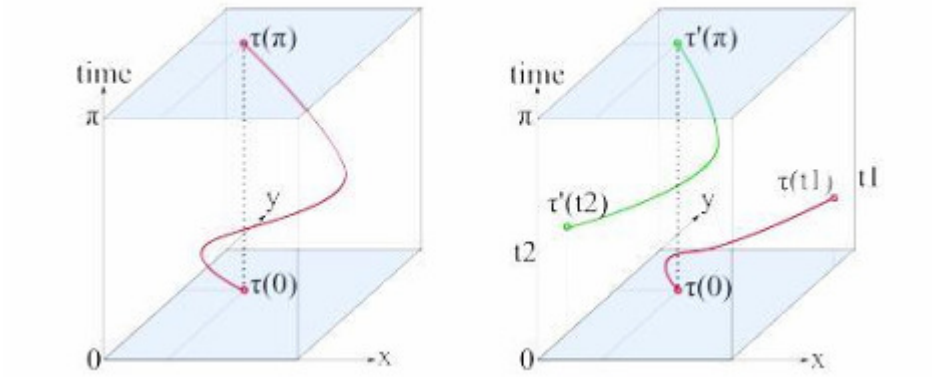


Figure 28 : Exemples de patchs. L'axe vertical représente le temps. Le patch de gauche contient un mobile endogène. Celui de droite de droite contient un mobile exogène : il sort du patch au temps t1 et y retourne au temps t2.

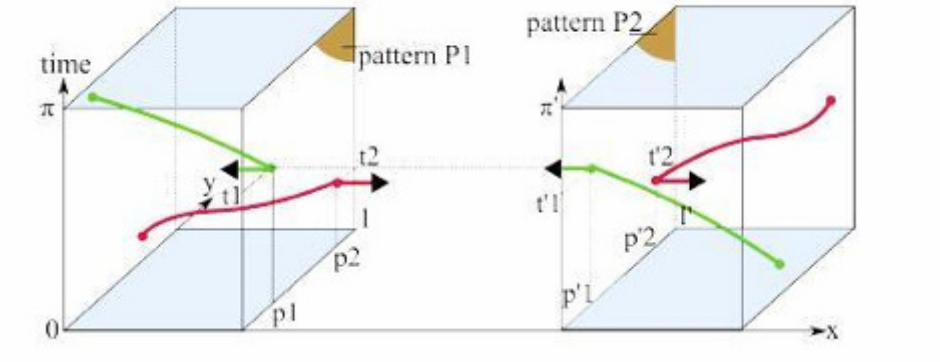


Figure 29 : Connexion de patch partageant des patterns miroirs.

Des patterns sont définis afin de permettre la connexion des patchs entre eux. Un pattern est défini pour chaque face du patch par sa longueur, sa durée et ainsi que par un ensemble

de couples (position,temps) de points entrants et un autre de points sortants. Deux patterns sont dits miroirs si ils sont de même longueur et de même durée, et si l'ensemble des points entrants du premier est égal à l'ensemble des points sortants du deuxième et inversement. Si deux faces (de deux patchs différents) présentent des patterns miroirs, alors elles peuvent être adjacentes et les deux patchs sont connectables (figure 29).

Les patchs peuvent avoir la forme de n'importe quel polygone convexe. Le monde est construit en les assemblant. Il y a deux manières de le faire : l'approche bottom-up où on les assemble itérativement, et l'approche top-down où on découpe une zone prédéfinie (figure 30).
Pendant l'exécution, les patchs sont introduits dans la scène là où la caméra regarde. On cherche d'abord dans la bibliothèque pré-calculée, si aucun ne correspond, il est créé à la volée.

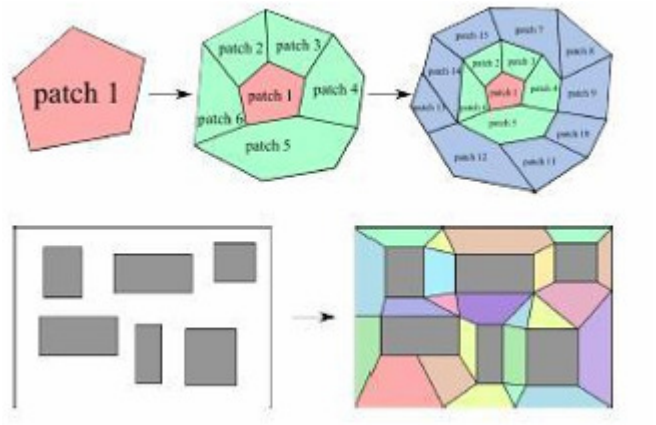


Figure 30 : Deux manières d'assembler les patchs : approche bottom-up (haut) et approche top-down (bas).

Cette méthode permet de générer des environnements peuplés de taille potentiellement infinie mais les humains virtuels n'ont aucune autonomie. Près de 3 000 piétons ont pu être simulés, à 20 fps, sachant que la simulation ne consomme que 5 à 10 % des ressources (le reste étant pour le rendu).

3.6 Foules par l'exemple

Lerner, Chrysanthou et Lischinski parviennent à générer des comportements très variés qui ne sont généralement pas pris en compte, comme s'arrêter quelques secondes pour regarder une vitrine, avoir une trajectoire chaotique parce qu'on est perdu, faire brusquement demi-tour, etc [LER 07].

L'idée est de recopier le comportement de piétons réels directement à partir de vidéos. Dans un premier temps, il faut construire une base de données dans laquelle seront stockés des exemples. Ces exemples sont construits d'après les trajectoires extraites des vidéos, ils forment un ensemble de situations. Pendant l'exécution, à chaque pas de simulation, les piétons perçoivent une situation vécue (position, orientation et vitesse des voisins, obstacles). Ils construisent et envoient alors une requête à la base de données pour trouver un exemple correspondant à la situation vécue. Il ne leur reste plus qu'à copier et à suivre la trajectoire donnée par l'exemple (figure 31).

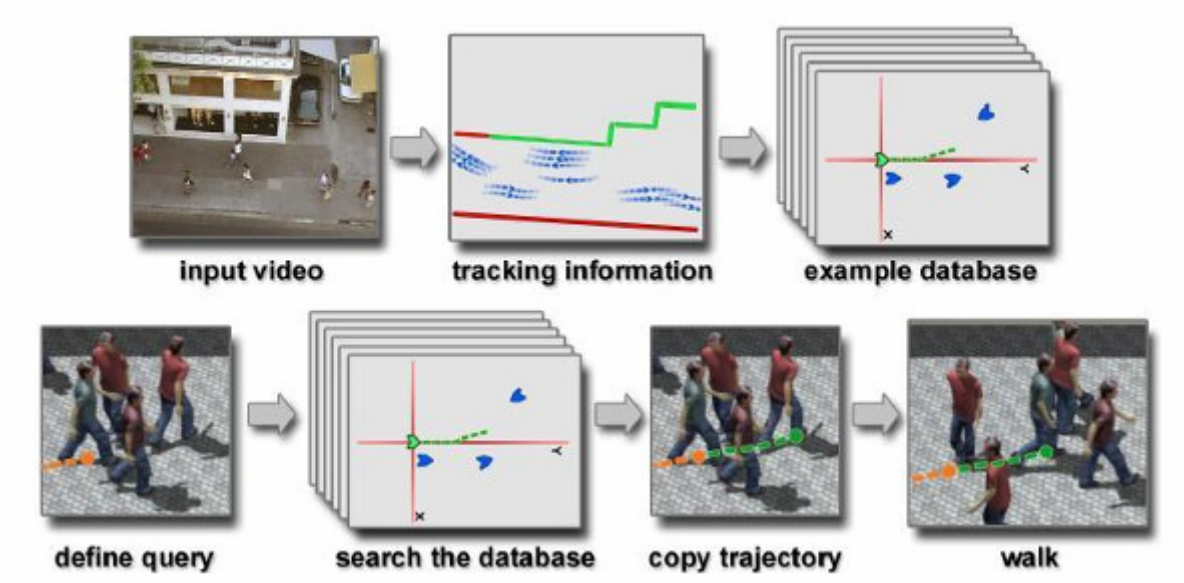


Figure 31 : La méthode de foules par l'exemple

Les points clés de cette méthode sont la construction des exemples, la construction de la requête et surtout la définition des critères de ressemblance entre situation vécue et exemples. Plus la base de données contient de situations, plus les comportements seront précis et réalistes, mais la recherche sera d'autant plus longue à effectuer.

Cette technique donne des résultats très réalistes, mais pour un nombre très limité de piétons (une quarantaine), et ne tourne pas en temps réel.

3.7 Conclusion

Plusieurs éléments manquent encore à la majorité des simulations existantes, quelle que soit leur approche : des piétons formant des petits groupes et se déplaçant à plusieurs, des piétons fortement hétérogènes (autant dans le comportement que dans l'apparence), et enfin des perturbations singulières dans le comportement (arrêt impromptu, personne « indisciplinée », etc). Le tableau 1 récapitule les caractéristiques principales des différents modèles présentés précédemment. La colonne Qualité correspond à la plausibilité des déplacements observés et la colonne Interactivité aux possibilités d'édition et de contrôle manuel.

	Nb. de piétons max	**Temps réel**	**Interactivité**	**Qualité**	**Hétérogénéité**	**Groupes**
Shao & Terzopoulos	1200 (sans rendu 3D)	Oui (sans rendu 3D)	Moyenne	Assez Bonne	Moyenne	Non
HiDAC	600	Oui	Bonne	Bonne	Moyenne	Non
Dynamique continue	10000	Non	Faible	Faible	Nulle	Non
Dynamique globale	100000	Entre 0 et 10000 agents	Faible	Faible	Nulle	Non
Patchs de foule	3000	Quasi	Nulle	Faible	Moyenne	Non
Foules par l'exemple	40	Non	Nulle	Bonne	Faible	Non

Tableau 1 : Tableau récapitulatif

L'approche par agents permet de faire des foules très réalistes et sophistiquées, où chaque individu prend des décisions qui lui sont propres en fonction de ses buts personnels. Cependant, calculer les mouvements et les décisions de chaque agent séparément est très coûteux, il n'est pas possible de simuler des foules de plusieurs dizaines de milliers de personnes en temps réel. Il est également difficile de gérer des foules ayant un but commun, comme une armée, car il faut expliciter les comportements de chaque individu. A l'inverse, les méthodes s'appuyant sur la dynamique continue permettent de simuler facilement de larges foules agissant de concert tout en permettant, si ce n'est le temps réel, au moins l'interactivité. Mais il ne faut pas que l'œil s'attarde sur les détails, le résultat est bien moins réaliste.

Modèle proposé

Aujourd'hui, malgré la grande variété de méthodes, aucun modèle ne permet de générer des foules à la fois réalistes et de très grande taille (sans même parler du temps-réel). Comme souvent, il faut faire un compromis entre quantité et qualité et s'adapter aux besoins. Par exemple, simuler une grande armée pour un plan large d'un film serait plutôt réalisable avec une foule continue, tandis que peupler une ville pour un jeu vidéo nécessiterait des agents autonomes et bien différenciés. La navigation des piétons est un phénomène complexe qui reste encore mal connu des scientifiques. Cette méconnaissance constitue à la fois la principale difficulté et le principal enjeu de sa modélisation. Les simulations sont difficiles à réaliser mais elles aident à mieux comprendre la réalité.

Dans notre système nous nous intéressons plutôt à la simulation de chaque individu composant la foule, nous cherchons de concevoir un système capable de simuler une grande foule réaliste d'humains virtuels en temps réel tous en restant peu couteux.

Dans cette partie nous présentons les détails de notre modèle d'abord nous commençant par l'architecture de planification de mouvement comprenait la représentation de l'environnement, la recherche de chemin, l'évitement dynamique de collision et la formation du petit groupe. Ensuite nous présentons le modèle de l'humanoïde et les techniques d'animation utilisées pour la simulation 3D. Enfin nous finissant par les détails de l'implémentation et les différents résultats aboutis

Chapitre 4 Planification de mouvement

Dans ce chapitre, nous allons expliquer les détails du modèle proposé, un modèle basé sur une approche hybride, exploite deux types de représentation de l'environnement, la grille régulière et la grille hiérarchique. La grille hiérarchique est utilisée pour la planification de chemin à long terme permettant à chaque individu composant la foule de raisonner sur cette représentation et d'extraire un itinéraire vers son but (destination) depuis sa position courante. La grille régulière est utilisée d'une part pour la planification de chemin à court terme permettant pour chaque piéton de rejoindre son itinéraire pendant l'évitement d'une éventuelle collision et d'une autre part pour la formation du petit groupe.

4 Planification de mouvement

Deux choses contraignent le déplacement: la structure intrinsèque de l'environnement et, les autres entités peuplant ce même environnement. La structure de l'environnement représente l'ensemble des obstacles statiques qui contraignent la navigation en bouchant des passages directs d'un point à un autre. Une grande densité d'obstacles contraint donc grandement la navigation et rend ce processus complexe à gérer. Un humanoïde a donc besoin d'une structure de données représentant l'environnement, qui lui permet de détecter rapidement les obstacles statiques gênant sa progression. D'autre part, dans le cadre d'environnements complexes, tels que les villes, les intérieurs de maisons ou d'immeubles, il doit pouvoir raisonner sur leur structure pour trouver un chemin, exempt d'obstacles, lui permettant d'atteindre un but fixé. Enfin, lors de la navigation, les autres humanoïdes jouent eux aussi le rôle d'obstacles. L'entité doit donc être capable de détecter, dynamiquement, les éventuelles collisions avec ses voisins pour pouvoir changer sa route en vue de les éviter. Ces points soulèvent donc deux problèmes de détection de collision : l'un statique, inhérent à la structure de l'environnement, l'autre dynamique, dépendant des autres humanoïdes en déplacement.

4.1 Représentation de l'environnement

L'environnement dans lequel les entités évoluent est représenté par une géométrie statique. Cette géométrie traduit la structure de l'environnement et, donc les contraintes imposées lors de la navigation. De manière générale, cette géométrie représente les obstacles sous la forme de polygones en 2d et sous la forme de polyèdres en 3d. Cette hypothèse sert, de base à un certain nombre de méthodes de représentation de l'espace et s'avère être corrélée avec les méthodes de modélisation d'environnement. Que cet, environnement ait été produit avec un logiciel de modélisation 3d ou un outil dédié, pour permettre une interprétation rapide du point de vue des entités le peuplant, il doit être représenté dans une structure de données appropriée. Dans notre modèle proposé, on exploite deux méthodes de représentation Ces méthodes consistent à discrétiser l'espace de manière à identifier les zones dans lesquelles l'entité peut, naviguer. Cette discrétisation peut ensuite être utilisée pour représenter la topologie des lieux au travers de la notion d'accessibilité entre zones. Cette propriété s'avère nécessaire pour la recherche d'un chemin entre deux points.

4.1.1 Grilles uniformes.

La méthode de décomposition en grille uniforme se base sur la notion de cellules carrées en deux dimensions. Le principe est donc de quadriller l'espace, de manière uniforme, avec ces cellules. Les cellules sont alors vides si elles appartiennent à la zone de navigation et obstruées si elles sont inclues à l'intérieur d'un obstacle.

La génération de la grille se fait en deux phases :

- La génération d'une carte 2D de l'environnement représentant la scène de simulation : La méthode consiste à effectuer un rendu de la scène en vue de dessus avec une projection orthogonale. L'image rendue permet alors d'obtenir une représentation de l'environnement sous la forme d'une grille régulière en 2d en identifiant les zones de navigation et les obstacles.
- La construction du graphe topologique correspondant a la grille où chaque nœud représente une cellule alors que chaque arc représente une relation d'accessibilité entre deux cellules.

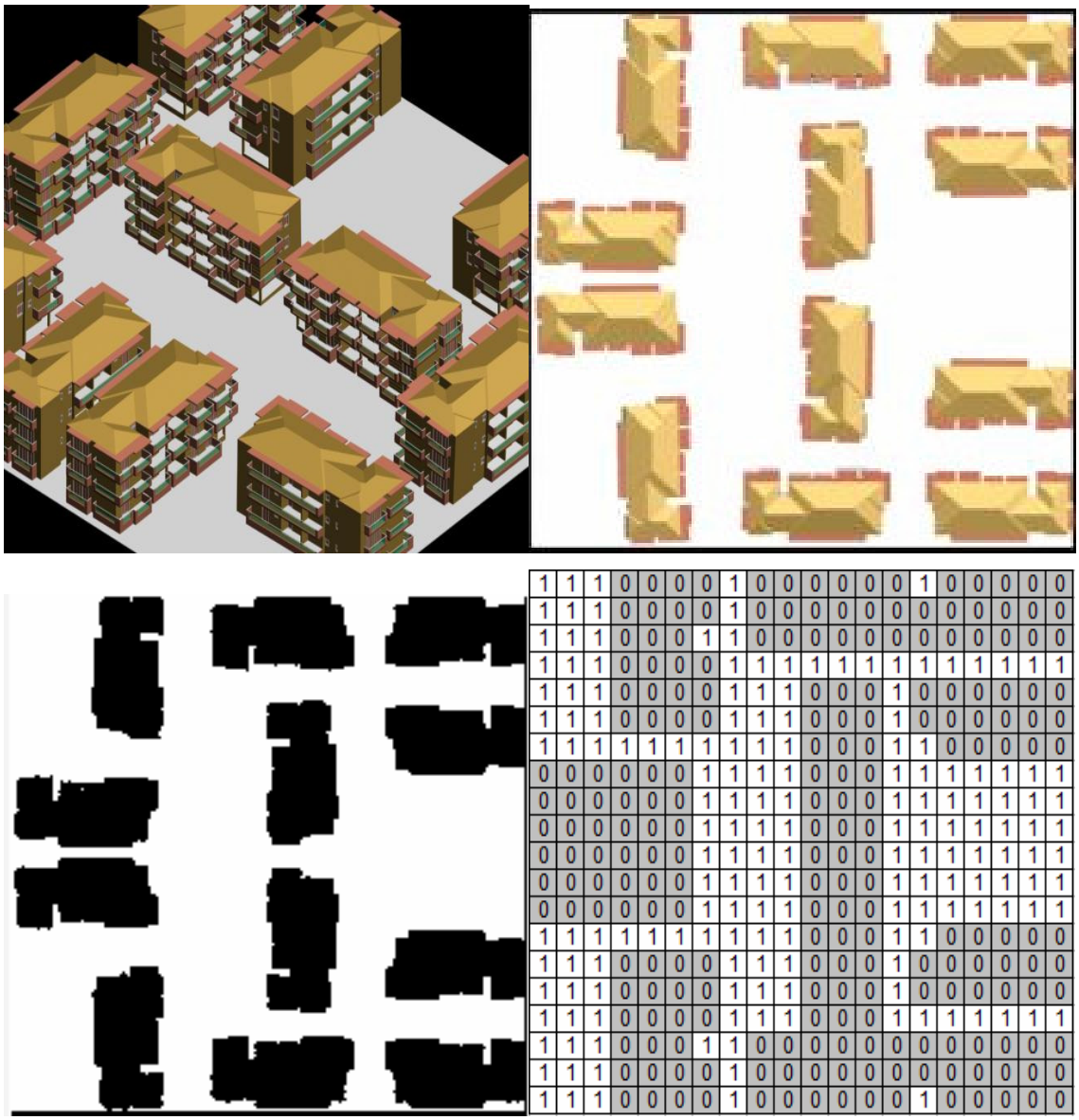


Figure 32: Le processus de génération de la grille uniforme

La précision de ce mode de représentation de l'environnement dépend de la taille de la cellule de base. Plus les cellules sont grandes moins la représentation est précise et inversement. Supposons que l'environnement soit un carré de n cellules de coté, la taille mémoire nécessaire pour le stockage de la grille est en $O(n^2)$. La complexité mémoire constitue l'un des problèmes majeurs de cette méthode pour laquelle il faut trouver un compromis précision / taille mémoire dépendant de chaque environnement à représenter. Outre son impact sur le coût mémoire, l'augmentation de précision grossit d'autant la taille du graphe topologique et, possède donc un impact non négligeable sur le coût de la recherche d'un chemin.

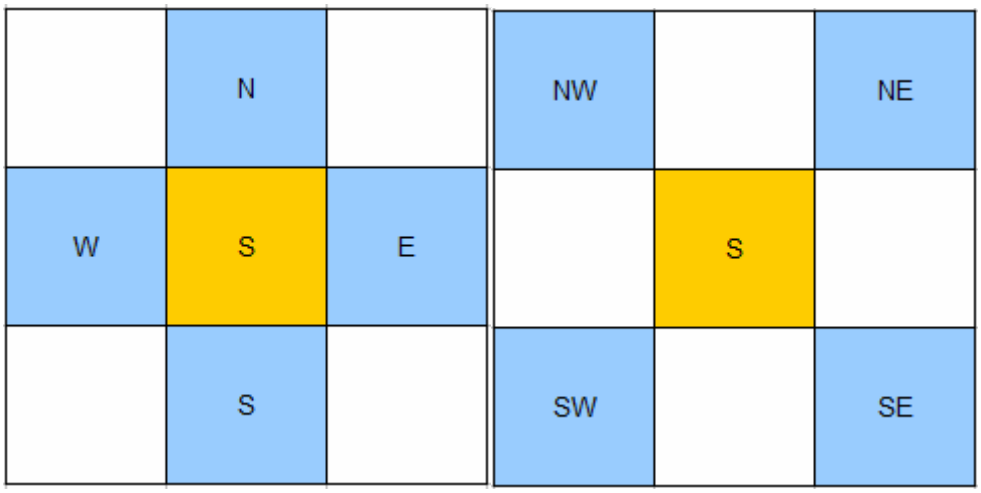


Figure 33 : Relation d'adjacence dans une grille régulière

L'approche de décomposition en cellules régulières consiste à subdiviser un environnement en cellules discrètes d'une forme et d'une taille prédéfinies, telles qu'une place, et puis extraire un graphe non orienté basé sur les relations de contiguïtés entre les cellules (figure 33). Cette approche a l'avantage de pouvoir produire des chemins précis, bien qu'ils soient inefficaces quand les environnements contiennent des vastes zones de régions obstacle-libres. Son coût de planification de chemin augmente avec la taille de grille, plutôt qu'en présence du nombre d'obstacles. A cet effet nous l'employons pour la planification de chemin à court terme seulement.

4.1.2 Grilles hiérarchiques.

Le principe des grilles hiérarchiques consiste à décrire l'espace sous forme d'un arbre de cellules qui subdivisent récursivement l'espace. Dans le cas en deux dimensions, le quadtree (voir figure : 34) subdivise récursivement l'espace en décrivant une cellule carrée à un niveau comme la réunion de quatre cellules carrées disjointes recouvrant tout l'espace de la cellule mère. Dans le cas en trois dimensions. L'octree fait de même avec une forme de base cubique subdivisée en huit sous cubes. Lors de la phase d'Initialisation de cette structure, une subdivision s'arrête à un niveau donné de l'arbre si la profondeur de la cellule est égale à la profondeur maximale de l'arbre (directement corrélée à la taille maximale de la cellule la plus précise) ou si toutes les sous cellules sont libres pour la navigation ou toutes à l'intérieur d'un obstacle. En terne d'occupation mémoire, au pire cas il est du même ordre de grandeur que pour les grilles régulières. Dans les faits, elle s'avère souvent moins coûteuse pour les environnements peu denses en obstacles, ce qui explique, en partie son utilisation en robotique.

Le graphe topologique associé à ce type de représentation est du même type que pour la grille régulière mais est uniquement constitué des feuilles de l'arbre (Qnadtree/Octree). Au pire, la taille du graphe est donc égale à celle du graphe associé à la grille régulière. Cependant, dans un très grand nombre de cas. la précision de la grille régulière est augmentée pour obtenir une meilleure approximation de la géométrie des obstacles. Cette augmentation du nombre de cellules est uniforme sur tonte la carte, la même augmentation de précision dans une grille hiérarchique, d'après le mode de construction n'augmente la précision que sur le bord des obstacles en conservant des cellules les plus grandes possibles pour des zones homogènes. L'utilisation de cette structure permet donc de réduire considérablement, par rapport à la grille régulière, le nombre de cellules et donc la taille du graphe topologique associé.

- Optimisation de chemin par l'approche de Quadtree

Le quadtree est un arbre, où chaque nœud a quatre nœuds enfant. N'importe quelle carte bidimensionnelle peut être représentée sous forme de quadtree par décomposition récursive. Chaque nœud dans l'arbre représente un bloc carré de la carte donnée. La taille du bloc carré peut être différente du nœud au nœud. Les nœuds dans le quadtree peuvent être classifiés dans trois groupes i.e. nœuds libres, nœuds d'obstacle et nœuds mélangés.

- Un nœud libre est un nœud où aucun obstacle n'est présent dans la région carrée.
- Un nœud d'obstacle est un nœud dont la région carrée est totalement remplie d'obstacles.
- Un nœud mélangé est un nœud dont la région carrée est partiellement remplie avec des obstacles.

La structure de données requise pour représentée un nœud est donné ci-dessous en syntaxe de langage de programmation C.

```
Struct Nœud
{       Nœud * NE;          // fils du nord est
        Nœud * SE;          // fils du sud est
        Nœud * SW;          // fils du sud ouest
        Nœud * NW;          //fils du nord ouest
        Nœud * parent;
        Nœud * liste_ adjacence;
        int Etat;
};
```

- Exemple de génération du Quadtree

Pour générer le quadtree la carte 2D montrée dans la figure : 34 (a) est d'abord divisé en quatre régions carrées secondaires (quatre nœuds enfant), à savoir NW, NE, SW, SE selon les directions. Ici les régions NW, SW sont entièrement occupés avec des obstacles (régions grises) et ils s'appellent les « nœuds d'obstacles », le nœud NE n'a aucun obstacle dans lui et il s'appelle « un nœud libre ». Le nœud SE est partiellement rempli avec l'obstacle et il s'appelle « un nœud mélangé ». La décomposition est montrée dans la figure : 34 (b).

Les nœuds libres et les nœuds d'obstacle ne sont pas décomposé plus loin et rester comme nœuds de feuille. Mais le nœud mélangé est subdivisé en quatre sub-quadrants, qui forment des enfants de ce nœud. Le processus de décomposition est répété jusqu'à ce que l'une ou l'autre des conditions mentionnées ci-dessous soit satisfaisante.

1. Le nœud est un nœud libre ou un nœud d'obstacle.
2. La taille de la région carrée représentée par les nœuds d'enfant est inférieure ou égal à la taille du piéton.

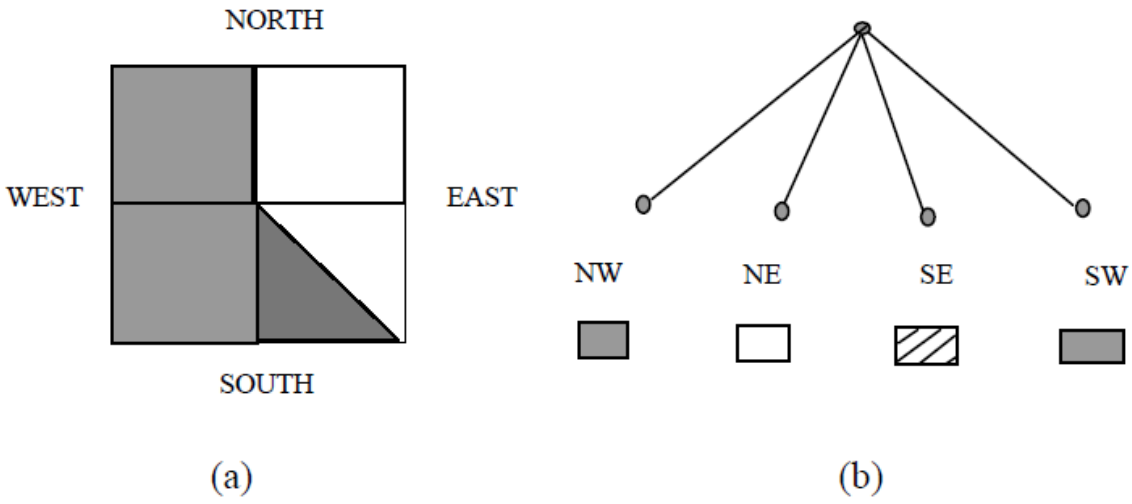


Figure 34 : Décomposition en quadtree

La petite boîte carrée sous chaque nœud représente le statut du nœud, où la boîte blanche, représente un nœud libre, boîte grise représente un nœud d'obstacle et boîte hachés représente un nœud mélangé, respectivement.

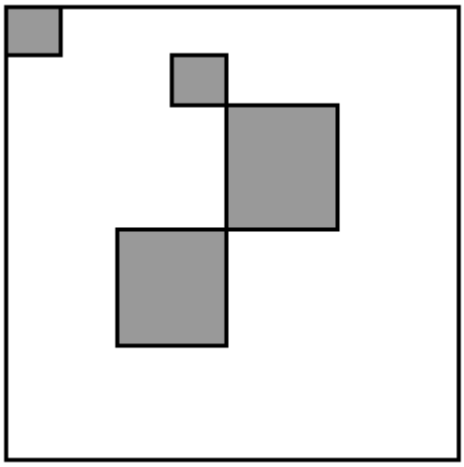


Figure 35 : Une carte 2D représentant l'espace de navigation

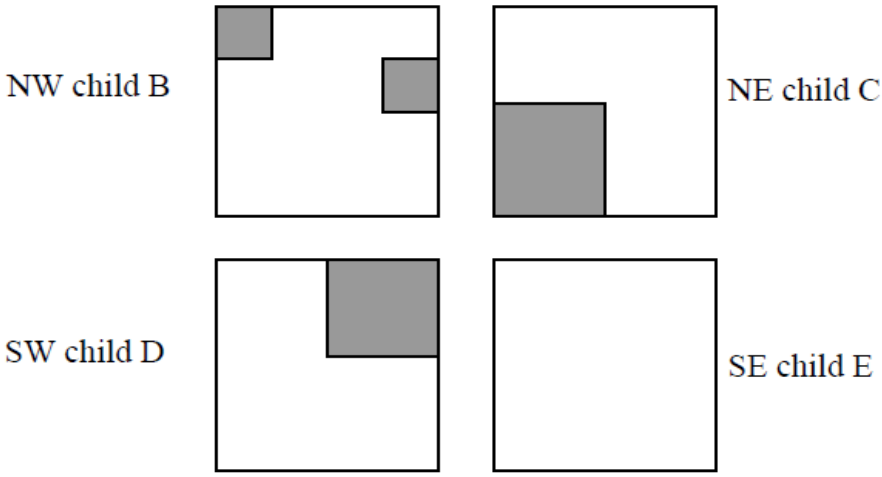


Figure 36 : La décomposition de la carte 2D

Les régions carrées gris dans la figure : 35 sont des régions occupées par des obstacles. Dans la première phase de décomposition, la carte est divisée en quatre régions carrées de taille égale suivant les indications (la figure : 36) La racine du quadtree est la carte elle-même et elle est dénotée par A.

Dans la décomposition ci-dessus l'enfant E ne contient aucun obstacle et il reste comme nœud de feuille. Les nœuds restants B, C, et D contiennent des obstacles et ils sont traités en tant que nœuds mélangés. Le quadtree après la première décomposition est représenté dans la figure : 37.

Les nœuds d'obstacle sont décomposés plus loin, suivant les indications.

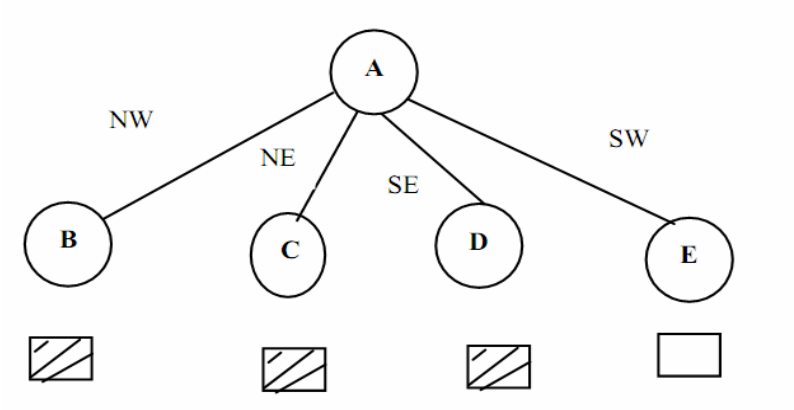


Figure 37 : La représentation de la décomposition en quadtree

- Calcul de Voisinage dans un Quadtree

La carte bidimensionnelle est divisée en un certain nombre de blocs carrés (de différentes tailles) tout en produisant le quadtree. Pour la planification de chemin nous devons nous déplacer particulièrement entre les blocs adjacents. Par conséquent, une certaine technique est nécessaire pour trouver ces blocs adjacents, appelés les « voisins » pour un bloc donné.

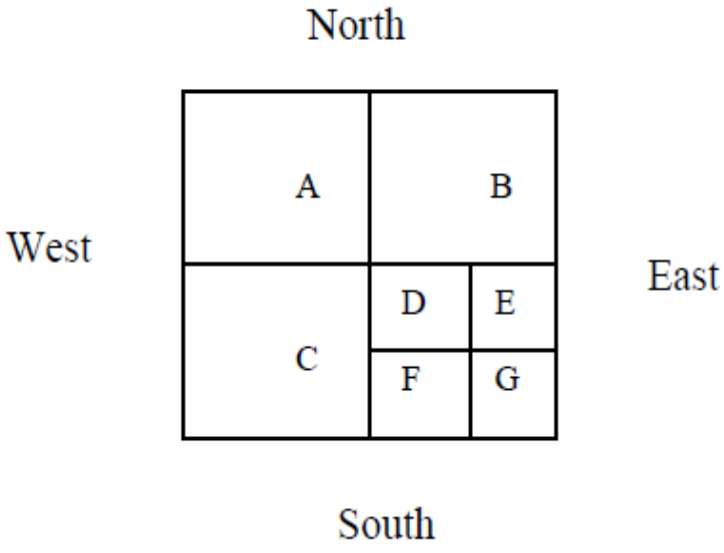


Figure 38 : Une carte 2D simple du monde

Dans la figure : 38. Les voisins du nœud D sont les régions B, E, F, et C dans les directions de nord, est, du sud, et occidentales, respectivement. Dans notre approche nous ne prenons pas le voisinage faisant le coin tel que D et G, en raison de la possibilité de l'absence d'un chemin entre les voisins faisant le coin.

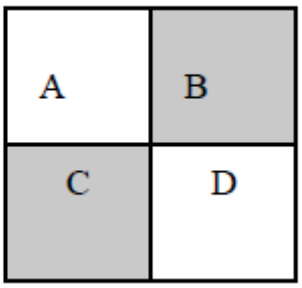


Figure 39 : Une carte 2D simple décomposée en régions

Dans la figure : 39, nous supposent que le piéton est dans la région carrée A, et le but à atteindre est la région carrée D. Si nous tenons compte du voisinage faisant le coin, les régions A et D iront bien des voisins. Mais il n'y a aucun chemin à entrer dans la région D

à partir de la région A, puisque les régions B et C sont occupées avec des obstacles. C'est la raison pour laquelle les voisins faisant le coin sont négligés.

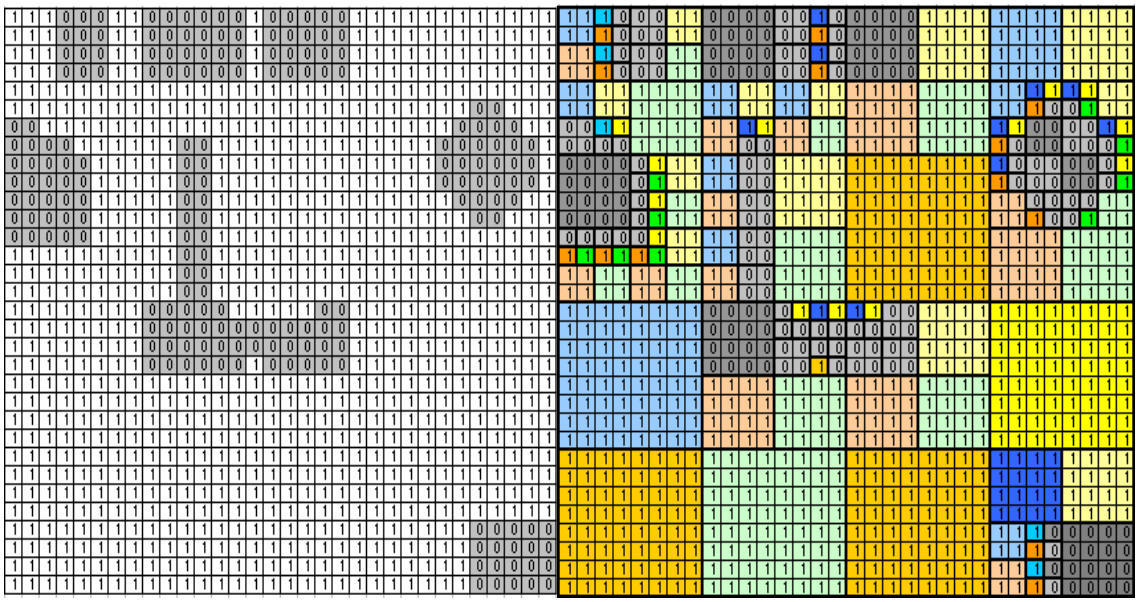


Figure 40 : une grille régulière et la grille hiérarchique correspondante

Dans la planification de chemin à base de quadtree, le temps de la recherche est très plus petit, car le nombre de nœuds à rechercher est considérablement plus petit (figure 40). En fait, le nombre de nœuds de feuille dans un quadtree d'une carte d'environnement ayant des obstacles polygonaux est approximativement *2/3 O(p)* d'où p est la somme des périmètres des obstacles en termes de l'unité la plus basse résolution. La recherche avec A* devra seulement traiter environ *O(p)* des nœuds dans le cas d'un quadtree, au lieu de n^2 dans le cas d'une méthode de recherche dans une grille régulière. D'ailleurs une hiérarchie de différents niveaux de la description de l'espace qui est disponible avec des quadtrees nous permet de rechercher un chemin près des obstacles seulement si nécessaire. On évite les chemins fermés en considérant seulement des voisins dans des directions horizontales et verticales. Pour ces raisons nous utilisons la représentation en quadtree pour planifier le chemin globale ver le but. Mais le point faible de cette méthode est que le chemin produit n'est pas optimal, il est en ce qui concerne les blocs carrés divisés et pas le chemin exact qu'un piéton peut traverser.

Après l'obtention de tous les nœuds voisins, le meilleur nœud voisin est choisi parmi eux, utilisant l'algorithme A*, ce qui est expliqué dans la prochaine section.

Ici la tâche est de trouver le chemin de coût minimum entre le nœud source et le nœud but.

4.2 L'algorithme A* pour choisir le meilleur voisin

Pour trouver un chemin d'un point à un autre il faut commencer par se diriger vers la destination. C'est justement cette idée qu'utilise l'algorithme A*. L'idée est très simple : à chaque itération on va tenter de se rapprocher de la destination, on va donc privilégier les possibilités directement plus proches de la destination, en mettant de côté toutes les autres.
Toutes les possibilités ne permettant pas de se rapprocher de la destination sont mises de côté, mais pas supprimées. Elles sont simplement mises dans une liste de possibilités à explorer si jamais la solution explorée actuellement s'avère mauvaise. En effet, on ne peut pas savoir à l'avance si un chemin va aboutir ou sera le plus court. Il suffit que ce chemin amène à une impasse pour que cette solution devienne inexploitable.
L'algorithme va donc d'abord se diriger vers les chemins les plus directs. Et si ces chemins n'aboutissent pas ou bien s'avèrent mauvais par la suite, il examinera les solutions mises de côté. C'est ce retour en arrière pour examiner les solutions mises de côté qui nous garantit que l'algorithme nous trouvera toujours une solution si elle existe bien sûr.
On peut donc lui donner un terrain avec autant d'obstacles qu'on veut, aussi tordus soient-ils, s'il y a une solution, A* la trouvera.
A* utilise deux listes, ces listes contiennent des nœuds d'un graphe, lequel graphe représentant notre terrain.
La première liste, appelée liste ouverte, va contenir tous les nœuds étudiés. Dès que l'algorithme va se pencher sur un nœud du graphe, il passera dans la liste ouverte (sauf s'il y est déjà).
La seconde liste, appelée liste fermée, contiendra tous les nœuds qui, à un moment où à un autre, ont été considérés comme faisant partie du chemin solution. Avant de passer dans la liste fermée, un nœud doit d'abord passer dans la liste ouverte, en effet, il doit d'abord être étudié avant d'être jugé comme bon.
Pour déterminer si un nœud est susceptible de faire partie du chemin solution, il faut pouvoir quantifier sa qualité. À cette fin l'algorithme A* est utilisé avec la fonction d'évaluation **F** d'un nœud C qui est défini comme suit :

$$F(c) = G(c) + H(c)$$

- le cout G (le cout pour aller du point de départ au nœud considéré)
- le cout H (le cout pour aller du nœud considéré au point de destination)
- le cout F (somme des précédents mais mémorisé pour ne pas le recalculer à chaque fois)

H (c) représente l'évaluation heuristique du coût du chemin restant du point courant (*c*) à la destination (*dest*).

Nous pouvons calculer ce distance de la manière que nous voulons, distance euclidienne, distance de Manhattan ou autre, elles peuvent convenir.

La distance de Manhattan

Entre deux points *C* et *Dest*, de coordonnées respectives (X_c,Y_c) et (X_{Des},Y_{Des}), la distance de Manhattan est définie par :

$$H(c)=P*[\text{abs}(x_{Dest}-x_c)+\text{abs}(y_{Dest}-y_c)]$$

La distance Euclidienne

$$H(c)=P*\sqrt{(x_{Dest}-x_c)^2+(y_{Dest}-y_c)^2}$$

P : est Le coût de déplacement d'une cellule à une cellule adjacente.

On calcule donc la distance entre le point étudié et le dernier point qu'on a jugé comme bon. Et on calcule aussi la distance entre le point étudié et le point de destination. La somme de ces deux distances nous donne la qualité du nœud étudié. Plus un nœud à une qualité faible, meilleur il est.

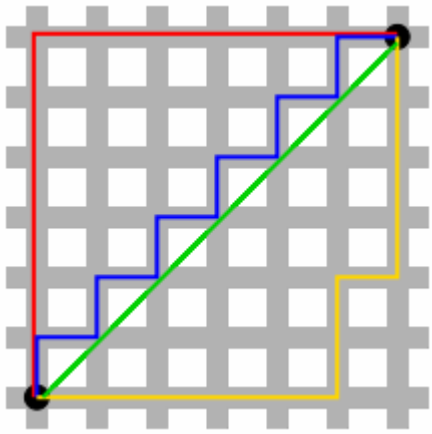


Figure 41 : Distance de Manhattan (chemins rouge, jaune et bleu) contre distance euclidienne en vert

L'algorithme A*

Début

Nœud_courant ← *Nœud_de_départ* ;

Répéter

Pour *chaque élément c de la liste des voisins* **faire**

Si *c.Etat = libre et c* ∉ *liste fermée* **alors**

Evaluer le nœud (F(c) = G(c) +H(c)) ;

Si *c* ∉ *liste ouverte* **alors**

Ajouter le nœud à la liste ouverte ;

Met le *Nœud_courant* comme parent du nœud c ;

Sinon *// déjà dans la liste*

Si *le nouveau cout est meilleur* **alors**

Met à jour la liste ouverte ;

Met à jour le parent du nœud ;

Fin si

Fin si

Fin si

Fin pour

Si *la liste ouverte est vide* **alors**

Pas de solution ;

Sinon

Nœud_courant ← le meilleur nœud de la liste ouverte ;

Retirer le meilleur nœud de la liste ouverte ;

Le met dans la liste fermée ;

Fin

Jusqu'à (*Nœud_courant= Nœud_destination*)

Fin

On ne sait pas grand chose du chemin solution si ce n'est que le point qui pourra nous rapprocher de la solution est un point voisin du point que nous étudions. On va donc étudier chacun des nœuds voisins du nœud courant pour déterminer celui qui a le plus de chances de faire partie du chemin solution.

La recherche du chemin commence par le point de départ, en étudiant tous ses voisins, en calculant leur qualité, et en choisissant le meilleur pour continuer. Chaque point étudié est

mis dans la liste ouverte et le meilleur de cette liste passe dans la liste fermée, il va servir de base pour la recherche suivante.

Pour déterminer le meilleur point pour continuer le chemin, il ne faut pas uniquement chercher celui qui a la meilleure qualité dans ses voisins, mais dans toute la liste ouverte. C'est comme ça qu'on pourra abandonner un chemin qui avait l'air bon au début et qui ne l'était pas.

Ainsi, à chaque itération on va regarder parmi tous les nœuds qui ont été étudiés (et qui n'ont pas encore été choisis) celui qui a la meilleure qualité. Et il est tout à fait possible que le meilleur ne soit pas un voisin direct du point courant. Cela signifiera que le point courant nous éloigne de la solution et qu'il faut corriger le tir.

L'algorithme s'arrête quand la destination a été atteinte ou bien lorsque toutes les solutions mises de côté ont été étudiées et qu'aucune ne s'est révélée bonne, c'est le cas où il n'y a pas de solution.

Une fois que la destination a été atteinte, il faut retrouver le chemin en suivant à chaque fois les parents des nœuds présents dans la liste fermée. On remonte le fil jusqu'à arriver au point de départ.

Du 2D au 2D1/2

Pour résoudre le problème de la planification de mouvement nous avons proposé d'utilisé deux structures pour représenter l'environnement de navigation de la grille régulière et la quadtree ces représentations d'environnement travaillent en 2D. Elles ne prennent en compte aucune notion de hauteur, et sont donc contraintes à manipuler des zones de navigation planes. Nous proposons une solution en deux dimensions et demi (2D1/2) consistant à ajouter une notion de hauteur aux zones de navigation ce qui offre la possibilité de conserver la simplicité des algorithmes en 2D tout en permettant de bien positionner les personnages au sol. La technique adoptée donc est la méthode de lancer de rayon.

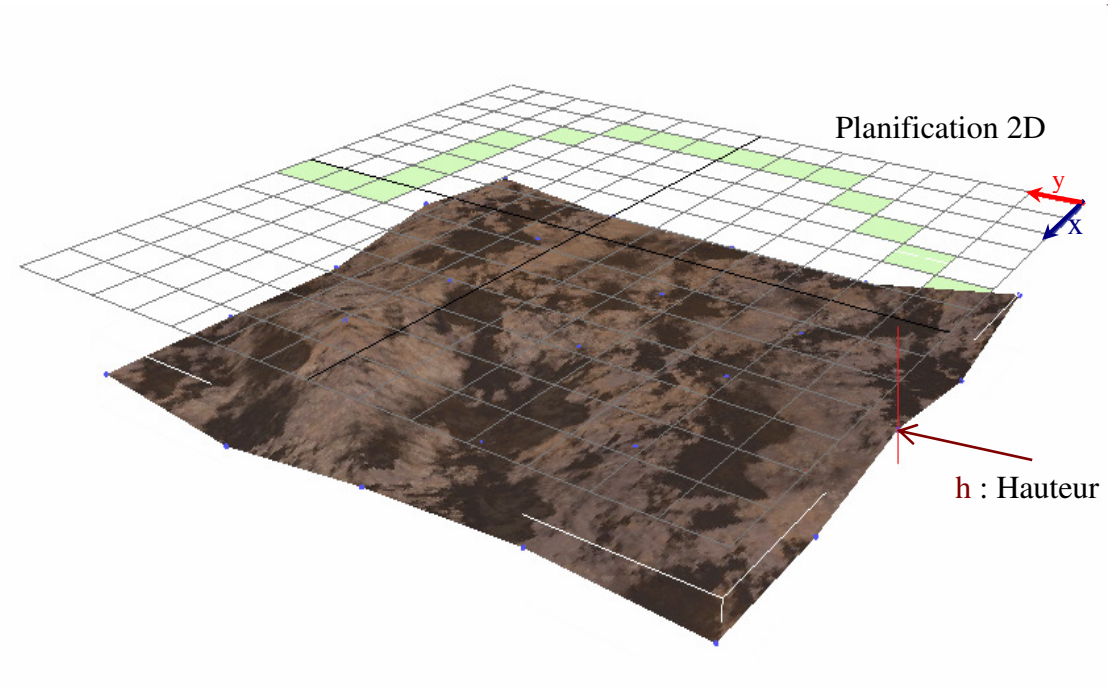


Figure 42 : Planification 2D ½

Pour positionner correctement les personnages au sol nous utilisant la méthode de lancer de rayon le principe de cet algorithme est de lancer un rayon de la position courante du personnage ver le bas et on recherchant le point d'intersection de ce dernier avec le terrain, ce point représente la hauteur du terrain de navigation. Après l'obtention de ce dernier on va donc repositionner le personnage à la bonne hauteur. Pour conserver un taux d'affichage élevé (framerate) cet algorithme est exécuté seulement pour les individus dedans le frustum de la caméra (c.à.d. dans la pyramide de vue) chaque frame durant toute la simulation.

4.3 Navigation

Dans la partie précédente, nous sommes partis d'un modèle géométrique 3d d'un environnement pour créer une subdivision spatiale permettant à une entité de se représenter les obstacles statiques avec lesquels elle peut entrer en interaction. Une entité est donc à même de planifier son chemin dans l'environnement et de détecter rapidement les obstacles dont elle est proche. Si ces informations sont suffisantes pour gérer la navigation d'une seule entité, elles ne permettent pas de gérer la navigation des centaines d'entités qui peuvent, par exemple, peupler une ville. En effet, les interactions que cela implique, ne sont plus seulement de la, forme d'une entité avec son environnement statique mais aussi de la forme d'une entité avec les autres entités. Un nouveau problème est alors soulevé, quel modèle peut-on utiliser pour traduire un comportement de navigation réaliste et permettant de gérer des centaines de piétons en temps réel? Cette question contient plusieurs sous-problèmes :

1. une entité ayant préalablement planifié un chemin, doit être à même de le suivre.
2. une entité, lorsqu'elle navigue doit avoir conscience des entités voisines pour être à même de prédire d'éventuelles collisions et être en mesure de les éviter.
3. lorsqu'une collision est prédite, l'entité doit réagir de manière à l'éviter.

4.3.1 Evitement de collision

Puisque les piétons se déplacent près les un au autres nous devons développer une méthode pour prévoir des éventuels collisions et laissons chaque individu éviter correctement d'autres unités à n'importe quel point donné.

La détection de la collision avec des agents est effectuée en utilisant l'information obtenue en percevant l'environnement. Pour percevoir l'environnement nous employons des requêtes scene queries du moteur l'OGRE 3D (c'est-à-dire, en obtenant l'information lier à la scène) différent type de requêtes de scène (scene queries) existent : ray queries, sphere queries, bounding-box queries, et bounding-plane queries.

Ray queries fournit les informations au sujet dont des objets dans la scène sont intersectés par un rayon donné (ligne imaginaire tracée entre deux points dans l'espace 3D). Sphere queries fournissent des informations au sujet des objets qui se trouvent à l'intérieur d'une

sphère donnée (décrite par un point d'origine dans l'espace 3D et un rayon). bounding-box queries utilisent des boîtes englobantes alignées sur les axes: nous fournissons deux vecteurs 3D qui définissent les coins opposés dans l'espace 3D, et Ogre fournit tous les objets qui se trouvent à l'intérieur de cette boîte.

bounding-plane queries fournissent des informations au sujet de tous les objets qui se trouvent en dessous d'un volume constitué par un ensemble de plans donnés.

Dans notre système nous utilisons Sphere queries paramétré comme suit :

Sphère (***O***, ***R***) / ***O*** : Point d'origine, ***R*** : rayon de la sphère

$O(x,y,z) = P(x,y,z) + d$

$$R = d * \tan g(\frac{\alpha}{2})$$

P(x,y,z) : position courante de l'agent ;

d : la distance de vue ;

α : est l'angle de vue ;

d et α sont des paramètre constant choisis par expérimentation

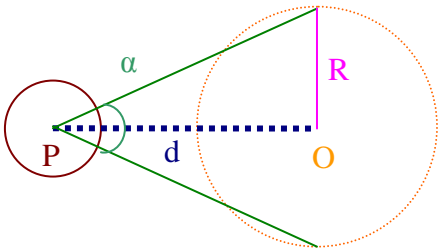


Figure 43 : Méthode de perception

Sphere queries fournissent des informations au sujet des objets qui se trouvent à l'intérieur de cette sphère et la décision est prise selon cette information.

Nous considérons trois types de collision entre deux agents

• **Collision de face:** se produit si les agents se déplacent l'un vers l'autre;

• **Collision en arrière:** quand l'agent est derrière un autre agent;

• **Collision de côté:** se produit si les deux agents marchent presque dans la même direction;

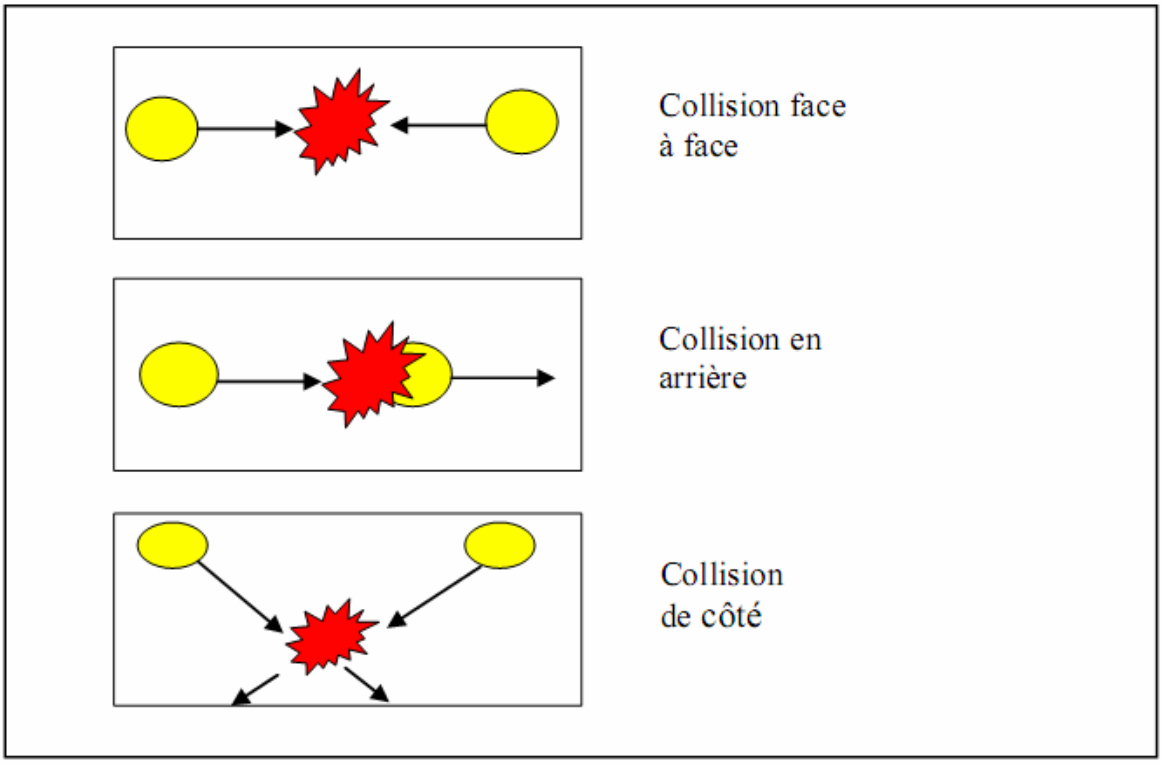


Figure 44 : Les trois types de collision

Les réactions aux collisions peuvent être classées dans deux grandes catégories de comportement : d'une part, les comportements agissant sur la vitesse de l'entité (s'arrêter, ralentir, accélérer), d'autre part, les comportements agissant sur la direction (éviter à gauche, éviter à droite). L'ensemble des ces comportements forme une base permettant d'exprimer tout type de réaction aux collisions.

• **Collision de face:** Si les deux agents se déplacent dans une direction opposée alors l'agent doit dévier avec un angle approprié (éviter à gauche ou bien éviter à droite)

- **Collision en arrière:** Si les deux agents ont la même direction alors on distingue deux cas de réactions possible :

  - ✓ l'agent adapte sa vitesse à l'agent qui le précède et marcher derrière lui.
  - ✓ l'agent dépasse l'agent qui le précède en choisissant le type d'évitement approprie.

- **Collision de côté:** on distingue deux cas de réactions possible :

  - ✓ Ralentir : Si la distance entre l'agent et la position attendue de la collision est assez suffisante.
  - ✓ Arrêter : Si la distance entre l'agent et la position attendue de la collision est moins suffisante.

Algorithme d'évitement de collision :

Pour *chaque frame* **faire**

Pour *chaque agent* **faire**

Perception de l'environnement ;

Si *le chemin est libre* **alors**

L'agent suit son chemin ;

Sinon

Si *collision de face* **alors**

Evitement à gauche (ou à droite) ;

Fin si

Si *collision en arrière* **alors**

Adapter la vitesse ;

Ou

Dépassement ;

Fin si

Si *collision de côté* **alors**

Ralentir ou Arrêter ;

Fin si

Fin si

Fin pour

Fin pour

Quand l'agent dévie de sa trajectoire ils doivent la joindre par la suite utilisant la planification locale de chemin.

Pour économiser le temps de calcul l'évitement de collision est garanti seulement pour les agents rendus (les agents dedans le frustum de vue)

4.3.2 Comportement de groupes

Dans le vrai environnement urbain, beaucoup de piétons font partie d'un groupe, ils marchent vers leur but communs, ils adaptent leur vitesse aux autres membres, attend l'un l'autre, et peuvent se séparés dans les endroits serrés pour éviter des collisions, mais se regroupent après.

La définition d'un groupe est souvent donnée pour un ensemble de personnes qui ont le même but ou liste de buts et les mêmes paramètres de mouvement, vitesse moyenne et accélération.
Dans notre système nous avons ajouté des petits groupes de deux à trois personnes, un tel comportement pour augmenter en plus le réalisme de la simulation. Nous avons utilisé la grille régulière pour implémenter la structure du groupe, notre méthode est basée principalement sur le modèle de flocks of boids de Reynolds. Un agent du groupe seulement, que nous appellerons le chef, doit faire calculer sa trajectoire en avant ; les autres agents, appelés membres, juste doivent le suivre. Donc l'utilisation des groupes peut être un moyen de conserver le temps de calcul nécessaire pour la planification de chemin

En plus des règles de navigation locale assurant l'évitement de collision, nous considérons quatre règles moins prioritaires que les premiers pour assurer la gestion du groupe :

• **Alignement:** Les membres de groupe essaient toujours d'atteindre leurs positions par rapport à la position du chef (figure 45);

• **Avancement :** si la distance entre la position actuelle de l'agent et sa destination est égale à l'espace personnel alors l'agent va s'avancer;

• **Séparation:** si la distance entre la position actuelle de l'agent et sa destination est inférieur à l'espace personnel alors l'agent va s'avancer après qu'il ralentisse ;

• **Cohésion:** si la distance entre la position actuelle de l'agent et sa destination est supérieur à l'espace personnel alors l'agent va s'avancer après qu'il accélère ;

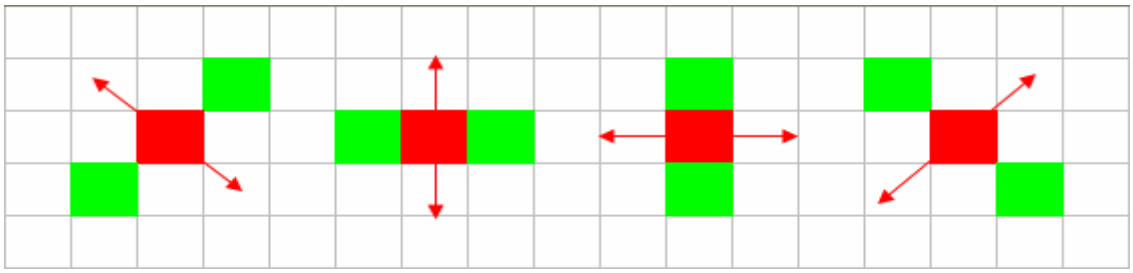


Figure 45 : Alignement des membres de groupe (vert) suite au mouvement de leur chef (rouge)

Algorithme de gestion de groupe :

Pour *chaque groupe* **faire**

Pour *chaque agent du groupe* **faire**

Perception de l'environnement ;

Si *le chemin est libre* **alors**

Calculer la nouvelle direction ; *// alignement*

Calculer la distance entre la position actuelle et la destination ;

Si *distance = espace personnel* **alors** *// avancement*

Avancement;

Fin si

Si *distance > espace personnel* **alors** *// cohésion*

Accélérer et Avancement;

Fin si

Si *distance < espace personnel* **alors** *// séparation*

Ralentir et avancement ;

Fin si

Sinon

Evitement de collision;

Fin si

Fin pour

Fin pour

Algorithme global de navigation :

Initialisation,

Pour *chaque groupe* **faire**

Planifier le chemin global ver le but (quadtree) ;

Fin pour

Boucle de Rendu,

Pour *chaque frame* **faire**

Pour *chaque groupe* **faire**

Perception de l'environnement par le chef ;

Si *le chemin est libre* **alors**

Avancer ;

Sinon

Evitement de collision ;

Si *le chemin direct vers l'itinéraire est libre* **alors**

Corriger la direction

Sinon

Planifier le chemin local (grille)

Fin si

Fin si

Pour *chaque membre du groupe* **faire**

Perception de l'environnement ;

Si *le chemin est libre* **alors**

Formation de groupe ;

Sinon

Evitement de collision

Fin si

Fin pour

Fin pour

Fin pour

Le changement des positions n'est pas suffisant pour fournir un aspect visuel crédible d'un piéton. Les piétons modifient non seulement leur positions et leur directions, ils effectuent également des actions principales : « Marcher », « tourner », « courir » …etc.

Dans la section suivante nous présentant les techniques utilisée pour représenter et animer les humains virtuels constituant la foule.

Chapitre 5 Modèle de l'humanoïde

5 Modèle de l'humanoïde

Historiquement, des personnages ont été dessinés et animés en faisant les images multiples montrant le personnage à des poses légèrement différentes. Ces images sont alors affichées en boucle pour donner l'impression du mouvement. Aujourd'hui Avec la puissance des cartes graphiques, il est possible d'avoir de pleins personnages tridimensionnels suffisamment détaillés et les animer en temps réel avec différentes techniques.

Dans ce qui suit nous allons détailler la représentation 3D d'humains virtuels, la technique d'animation permettant d'obtenir des mouvements crédibles en temps réel et enfin les techniques de variétés permettant de rendre chaque individu unique dans son apparence.

5.1 Représentations d'humains virtuels

Dans le cas idéal, pour visualiser des foules d'humains virtuels, nous utiliserions simplement des milliers de différents maillages de triangles déformables. Malheureusement, en dépit des avancées technologiques dans le domaine du hardware graphique programmable, nous sommes toujours limités par un certain budget de triangles à dépenser pour chaque image. Donc nous utilisant un nombre limité de différents maillages polygonaux très détaillés d'humains qui nous servent de modèles À partir de ces modèles, grâce aux techniques de variété d'apparence et de mouvement que nous décrirons par la suite, nous générons des milliers d'instances uniques. Cette méthode permet de générer des foules convaincantes de milliers d'individus tout en conservant un bon compromis entre la qualité et les ressource exploités en termes de mémoire nécessaire.

5.1.1 Modélisation polygonale

Dans la modélisation polygonale le modèle est assimilé à un ensemble facettes appelés polygones : ce polyèdre est donc décrit par la liste des sommets et des arêtes. Les polygones sont orientés pour différencier l'extérieur et l'intérieur du modèle. Souvent ces polygones sont des triangles, on parle de maillage triangulaire.

Dans notre système nous exploitons des modèles polygonaux texturés, représentants des humains virtuels on utilisant le logiciel de modélisation 3DS MAX. Ce type de représentation 3D est le plus approprié aux humains virtuels, il a l'avantage d'être afficher en temps réel et de produire des animations souples sur les machines courantes (le GPU ne prend que des triangles)

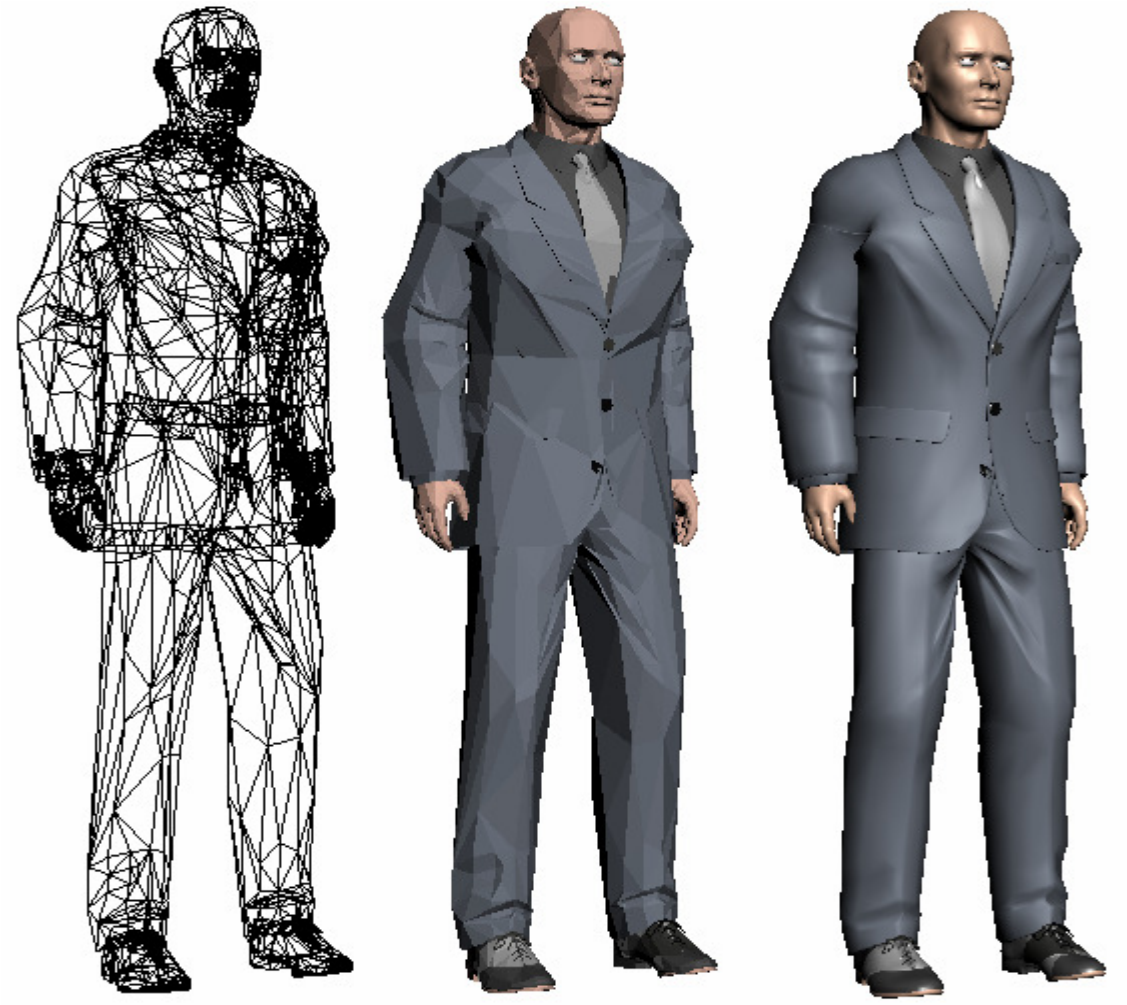


Figure 46 : modèle 3D d'un humanoïde

5.2 L'animation squelettique

L'animation squelettique est une technique d'animation par ordinateur dans laquelle un caractère est représenté dans deux parts : une représentation extérieure employée pour dessiner le personnage (appelé peau ou la maille) et un ensemble hiérarchique d'os reliés ensemble (appelés le *squelette*) utilisés pour animer (*pose* et *intrigue*) la maille. Cette technique est employée souvent pour animer des humains virtuels ou d'autres formes organiques d'une façon plus intuitive. Une animation peut être définie par les mouvements simples des os, au lieu du sommet par sommet (dans le cas d'une maille polygonale).

5.2.1 Squelette

On parle d'animation par squelette lorsque l'on a un mesh qui est déformé par un squelette. On peut voir ce squelette comme un squelette humain. on va y retrouver des os tels que : le torse, les bras, avant bras, les cuisses (fémur), les tibias etc.

Dans le corps humain, ce sont les muscles qui donnent le mouvement, dans l'animation par squelette, ce sont les os qui inculquent le mouvement au mesh. Bien sûr on ne créé un os que pour les parties que l'on désire animer, c'est-à-dire où l'on veut placer une articulation. Si les doigts d'un mesh n'ont pas besoin d'être animés, on ne placera pas les os les représentants.

Cette technique est employée en construisant une série des « os, » parfois désigné sous le nom du *rigging*. Chaque os a une transformation tridimensionnelle (qui inclut sa position, échelle et orientation), et un os parent facultatif. Les os forment donc une hiérarchie. La transformation finale d'un nœud enfant est le produit des matrices de transformations de son parent et ses propres transformations. Ainsi le déplacement d'un fémur déplacera la jambe inférieure aussi. Comme le personnage est animé, les os changent leur matrice de transformation au fil du temps, sous l'influence d'un certain contrôleur d'animation.

Chaque os dans le squelette est associé à une certaine partie de la représentation morphologique du personnage. *Le skinning* est le processus de créer cette association. Dans le cas le plus commun d'un personnage de maillage polygonal, l'os est associé à un groupe de sommets ; par exemple, dans un modèle d'un être humain, l'os de « cuisse » serait associé aux sommets composant les polygones dans la cuisse du modèle. Des parties de la

peau du personnage peuvent normalement être associées aux os multiples, pour chaque sommet on associer des facteurs de cadrage appeler poids de sommet, ou poids de mélange. Le mouvement de la peau près des joints de deux os, peut donc être influencé par les deux os.

Pour un maillage polygonal, chaque sommet peut avoir un poids de mélange pour chaque os. Pour calculer la position finale du sommet, la transformation de chaque os est appliquée à la position de sommet, mesurée par son poids correspondant.

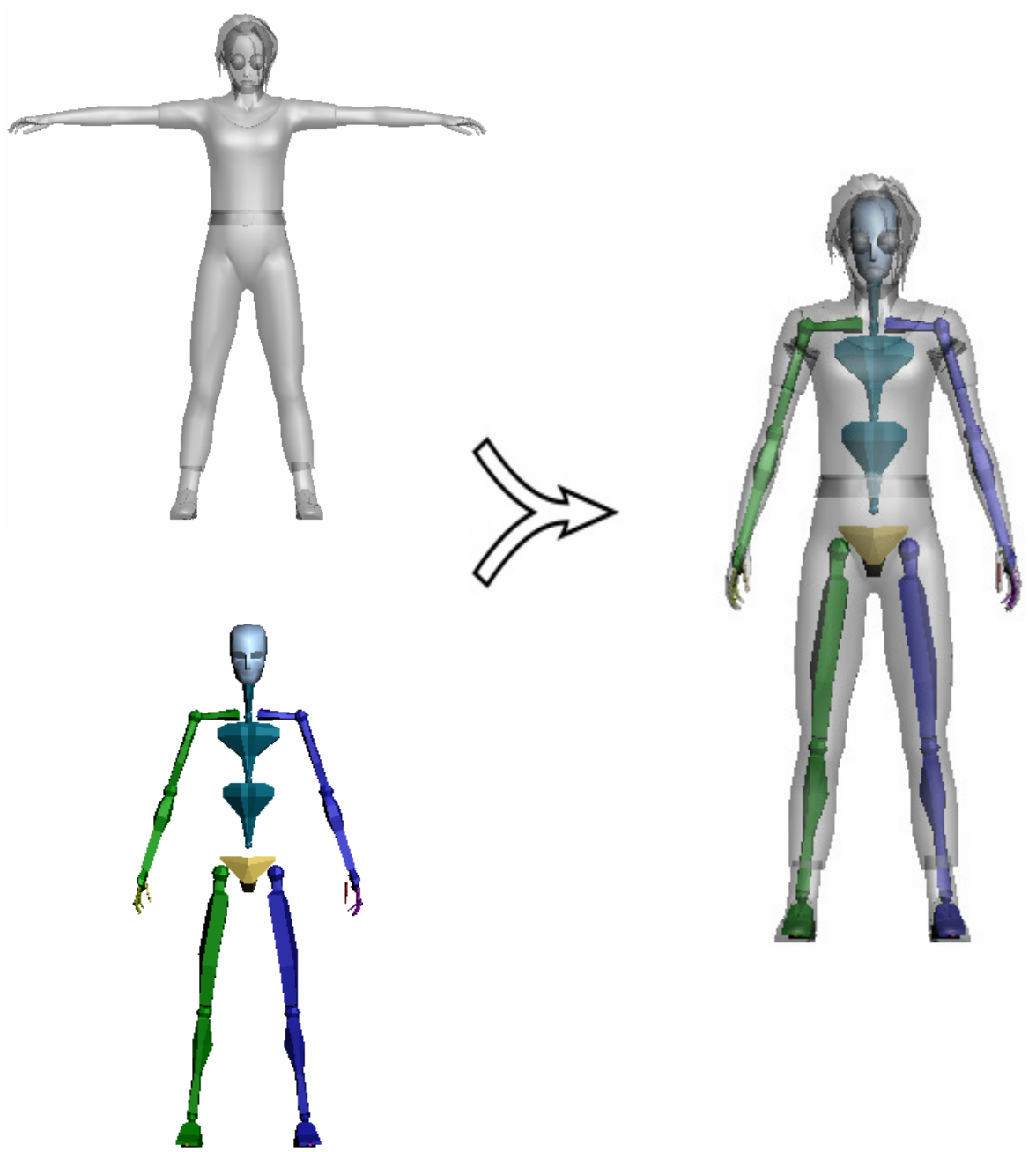


Figure 47: méthode de rigging avec un squelette.

5.3 Techniques de variété

Chaque modèle d'humain virtuel est utilisé pour instancier plusieurs milliers d'individus. Il est nécessaire de pouvoir varier les couleurs et la forme de chaque instance. La variation de couleur est basée sur l'idée de segmenter un humain virtuel en plusieurs parties et d'utiliser un nombre de textures différentes. Cette segmentation permet d'associer à diverse parties du même modèle des textures différentes. Plus précisément, chaque partie de corps est lui t'affecter une partie de texture correspondante permis l'ensemble des textures. Avec l'exploitation de segmentation en peut obtenir une plus grande variété d'apparence en utilisant un nombre restreint de modèles et de textures.

Il est important de noter que créer des modèles d'humains virtuels demande beaucoup de temps. C'est pourquoi l'ajout des accessoires nous permet de varier encore l'aspect des instances générées. En effet, dans la vie réelle, les gens ont des coupes de cheveux différentes, portent des lunettes et des sacs, utilisent des téléphones portables, etc. Ces particularités sont des détails tout aussi importants que la variété de couleur, car la somme de ces détails permet de rendre uniques les humains virtuels composant la foule, ce qui augmente le réalisme de la simulation. Nous ajoutons des accessoires simples, c'est-à-dire des accessoires pouvant être juste posés sur l'humain virtuel et n'impliquant aucun changement dans son comportement. Par exemple, un chapeau posé sur la tête du personnage, ou une montre à son poignet, un sac à dos à ses épaules. Chaque accessoire est attaché à un endroit spécifique de l'humain virtuel. Prenons l'exemple d'un chapeau : il est généralement attaché au crâne des personnages et peut être orienté différemment selon le modèle d'humain virtuel le portant.

Une des actions principale des foules est la locomotion. Pour accentuer encore la variété de la foule nous sommes aussi intéressés à pouvoir introduire la variété du mouvement des individus la composant. Pour cela, nous utilisons un moteur de locomotion capable de générer des cycles de mouvements différents comme la marche et la course à des vitesses variables. Cela donne déjà une impression de variété, car les gens ne marchent pas tous de même façon et au même rythme.

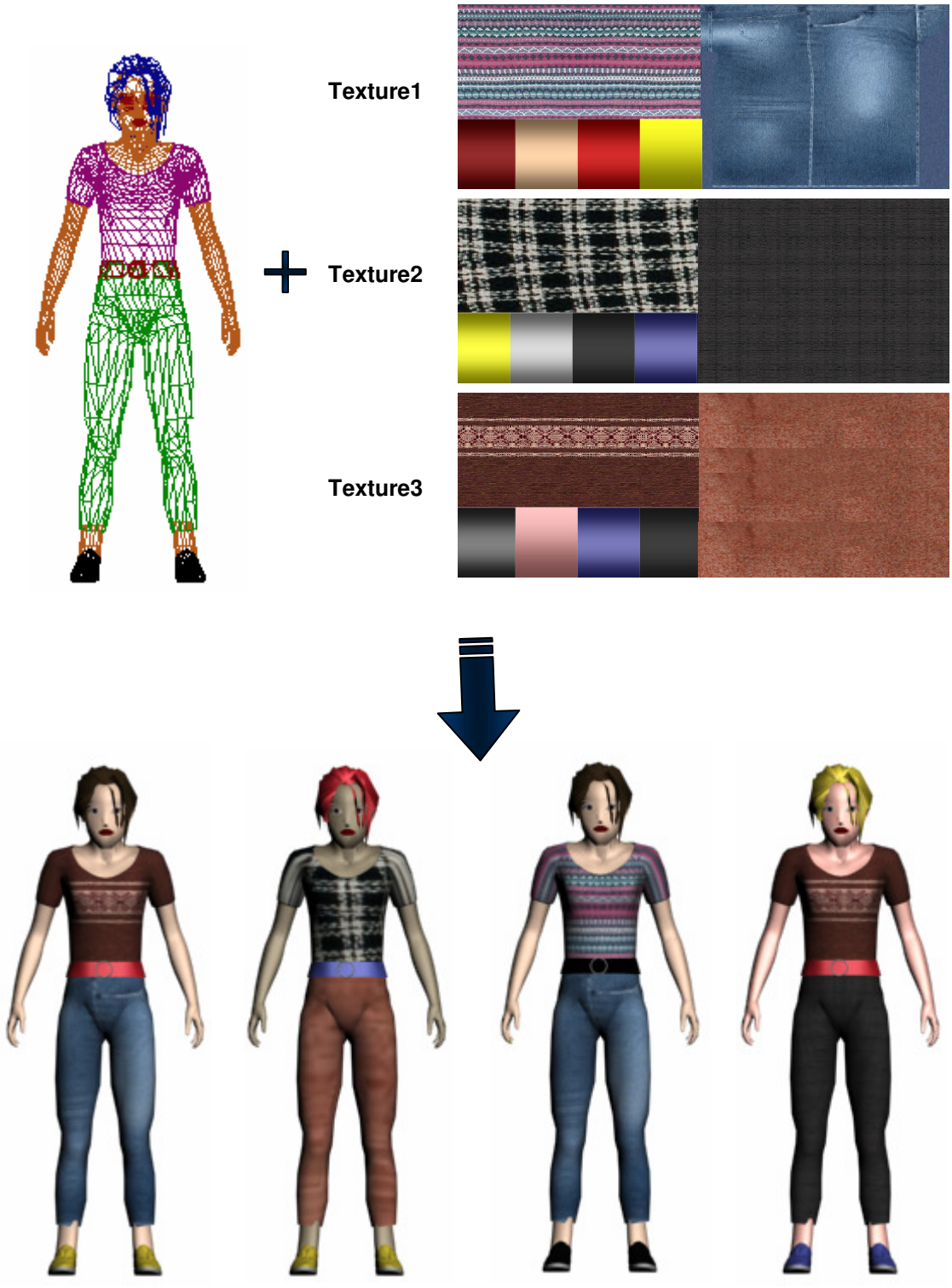


Figure 48 : technique de variété d'apparence

Chapitre 6 Implémentation et Résultats

6 Implémentation et résultats

Ce chapitre à pour but de présenter les résultats expérimentaux, à travers un ensemble de simulations, utilisant le modèle proposé dans le chapitre précédent. Nous allons présenter les résultats concernant la planification de chemin, ensuite nous présentons les résultats de simulation de différents scenarios de foule avec différente taille. Mais d'abord un aperçu des systèmes utilisés est donné en bref.

6.1 Environnement de développement

L'application a été développée avec le langage de programmation C++. En utilisant l'environnement de développements intégré visuel studio 2005 (SP1) de Microsoft. En plus de la bibliothèque standard de C++, nous utilisant le moteur de rendu OGRE 3D qui s'occupe du rendu 3D et CEGUI pour la création d'interface graphique.

Les personnages et la géométrie de l'environnement virtuel, y compris tous les objets détaillés sont modelés avec l'application 3DS MAX studio d'Autodesk. Les animations de personnages et les cartes de la ville sont également obtenues avec 3DS MAX. Tous les essais et les exécutions de l'application sont faits sur une machine avec un processeur Intel Core 2 Duo 2GHz 2GHz , 4Gb RAM, carte graphique (Unité GPU) d'ATI Radeon HD3430.

3DS MAX

Anciennement connu sous le nom "3D Studio," 3ds Max est un programme de modélisation 3D, d'animation et de rendu d'Autodesk, Inc. Largement utilisé dans les domaines des jeux interactifs, des effets visuels pour les films et des modèles de conception industrielle. L'application comprend un module d'animation qui utilise la cinématique inverse, qui relie les composants afin qu'ils se déplacent ensemble, en ajoutant à l'effet de mettre un personnage à la vie.

6.2 Rendu

Même si les APIs de bas niveau telle que OpenGL ou DirectX permettent de dessiner des objets à l'écran, celles-ci nécessite un énorme d'effort pour modélisé des scènes plus complexes. Pour la simulation d'environnement virtuel, il est ainsi préférable d'utiliser un moteur 3D qui nous permettra d'obtenir beaucoup plus facilement le rendu que nous désirons. Pour cela dans notre système nous utilisons le moteur de rendu Ogre 3D, ce moteur fournit notamment des fonctions nous permettant de charger des fichiers de différents formats, d'animer les modèles en fournissant uniquement le nom de l'animation et ainsi de suite, son but étant de simplifier au maximum le rendu de notre scène virtuelle.

6.2.1 Moteur 3D

Pour faire de la programmation 3D, il existe principalement deux librairies : DirectX et OpenGL. DirectX est un format propriétaire détenu par Microsoft et OpenGL est un format multiplateforme, libre et développé par Silicon Graphics. Ce sont deux librairies de rendu qui fournissent un ensemble de fonctions permettant l'affichage des objets en trois dimensions, en se servant directement des fonctionnalités offertes par la carte graphique. C'est ce qui s'appelle de la programmation bas niveau. L'utilisation de ces librairies est longue pour un rendu visuellement correct. C'est à ce moment là que le moteur 3D entre en jeu.

Intérêt d'un moteur 3D

L'intérêt du moteur 3D réside dans le fait qu'il s'agit d'une surcouche de haut niveau pour des librairies graphiques, telles qu'OpenGL ou DirectX. Ceci permet à la plupart des moteurs de fonctionner de la même façon. Un tel moteur fournit des structures de données adaptées aux applications 3D, et toute une batterie de fonctions, dans le but de simplifier la tâche des développeurs, et surtout d'optimiser le rendu graphique.

Fonctionnement

Généralement, un moteur 3D permet de gérer l'affichage sous plusieurs points de vues possibles (parfois même simultanément) d'un ensemble d'objets dans un espace virtuel appelé « scène ». L'opération d'affichage est appelée le « rendu ». Celui-ci est calculé en temps réel, chaque image calculée étant communément appelée « frame ».

Le moteur 3D va optimiser le rendu grâce à des algorithmes. Il en existe principalement deux. Le ray-tracing, qui pour chaque pixel de l'image à générer, tire un rayon partant de la caméra et calcule ce qu'il traverse. Il détermine de cette façon la couleur du pixel en fonction des objets traversés et de leurs propriétés visuelles tel que leur transparence, leur réflexion, leur réfraction, etc. Le second algorithme est le rendu de polygones. Cette méthode part du principe que les objets sont représentés sous forme de triangles, auxquels le moteur fait subir différentes transformations géométriques (transformations dans l'espace puis projection). Il les affiche ensuite en fonction de leur éclairage et/ou de leurs couleurs ou textures. Ces algorithmes permettent au moteur de savoir quelles sont les faces des objets à afficher dans la scène, afin d'économiser des calculs inutiles d'éléments qui ne seront pas visibles à l'écran par l'utilisateur. Le moteur 3D gère nativement un grand nombre de phénomènes graphiques, tels que les ombres, les lumières et autres effets graphiques basiques.

Pour résumer, un moteur 3D est là pour permettre à des développeurs de pouvoir réaliser plus efficacement une application 3D, sans avoir forcément une grosse base de connaissances en développement graphique. Toutefois, il requiert une maîtrise du langage de programmation dans lequel le moteur est écrit.

6.2.2 OGRE 3D

OGRE est un moteur 3D libre et gratuit qui permet, à partir d'objets à facettes, de réaliser un environnement en 3D. Celui-ci sera perçu par un rendu 2D au travers d'une caméra. OGRE est une surcouche utilisant les APIs DirectX et OpenGL, qui permettent l'utilisation de cartes accélératrices 3D (OGRE ne fournit pas de moteur de rendu 3D logiciel, il faut une carte 3D ou un émulateur de cartes 3D).

Caractéristiques

Multiplateforme : OGRE fonctionne aussi bien sous Windows, Linux ou MAC.
– Compatible OpenGl et DirectX.
– Très performant : il gère les effets graphiques les plus récents et affiche un Frame Rate très satisfaisant!
– Offre une architecture 'professionnelle' : son conception est tel qu'il peut parfaitement s'intégrer à des applications graphiques professionnelles. Le travail en équipe ne pose pas de problème grâce à la POO et sa modularité offre de grandes possibilités de personnalisations.
– Très bien documenté : OGRE offre une API de très bonne qualité, un manuel complet et des tutoriaux tous très bien fait! Le forum est aussi très actif et peu beaucoup aider.
Fonctionnalités d'OGRE

Affichage 3D

Le but principal d'une application 3D est d'afficher un agencement d'objets 3D (des modèles) à l'écran. OGRE gère divers formats d'objet, qui peuvent être préalablement modélisés sous différents logiciels, open source ou payant, tels que 3DStudio Max, Blender, Maya, etc. Il offre de nombreuses fonctions afin de gérer ces objets : déplacement, rotation, agrandissement… Un format de fichier spécifique est utilisé par OGRE pour ces modèles, ainsi que pour leurs textures.
Il est également possible d'afficher les polygones qui composent un objet, généralement dans le but de repérer facilement les erreurs dans une application.

Animation

Si certains objets doivent être immobiles, d'autres en revanche nécessitent des mouvements. Pour cela, OGRE supporte différents types d'animations. Les animations de type « squelettes » sont prévues dès la modélisation de l'objet. Les mouvements sont préalablement créés par le modeleur et calculés par le logiciel de modélisation, puis exportés dans un fichier. OGRE se contentera alors de charger ce fichier, et de lancer la ou les animations choisies. Il est également possible de programmer des animations directement avec OGRE. En effet, il suffit d'indiquer à un objet un mouvement (translation, rotation), ainsi que le temps qu'il lui est accordé pour effectuer ce mouvement. Le moteur se chargera de mettre à jour la position de notre objet pour chaque frame.

Interaction avec l'utilisateur

Une fonctionnalité indispensable pour une application graphique 3D est sans doute la gestion des entrées utilisateur. OGRE propose une gestion simplifiée des périphériques classiques (les biens connus souris et clavier) mais également une gestion manette de jeu vidéo.

Caméra

Afin d'effectuer le rendu de la scène, OGRE a besoin que le programmeur lui spécifie un, ou plusieurs angles de vue. Pour cela, des objets spéciaux sont insérés dans l'application : les caméras. Une application peut contenir une seule caméra et tout de même changer d'angle de vue durant son exécution. De même, plusieurs caméras peuvent rendre différents points de vue à l'écran simultanément : dans le cas d'un jeu vidéo, le cas le plus courant est l'écran divisé en deux lors d'une partie multi-joueurs sur une seule machine. Dans la première image, une caméra est utilisée pour regarder le sol, tandis que dans la seconde, on regarde à la fois le sol et le ciel, via deux caméras.

Éclairage

OGRE offre quatre types d'éclairage pour les scènes finales (à noter que la couleur de la lumière est laissée à l'appréciation du développeur).

Lumière ambiante : toutes les faces des objets sont éclairées de manière uniforme et équitable.

Point : la lumière est envoyée dans toutes les directions, à partir d'un point précis.

Spot (image de gauche) : lumière de style « lampe torche », qui comporte donc une direction et un halo de lumière en cône.

Directionnel : simule un point lumineux étant situé à une distance infinie de la scène, ce qui équivaut à notre soleil.

Ombre

Il existe également plusieurs sortes de rendus d'ombres dans OGRE.

Texture Modulative : crée une texture en noir et blanc appliquée à la scène, en tenant compte seulement d'une seule source de lumière.

Stencil Modulative : calcule toutes les ombres de la scène, de toutes provenances, puis l'applique sur le rendu.

Stencil Additive : calcule chaque ombre séparément puis les fusionne sur la scène. Cette dernière technique s'avère être la plus coûteuse mais la plus réaliste.

6.3 Interfaces graphiques

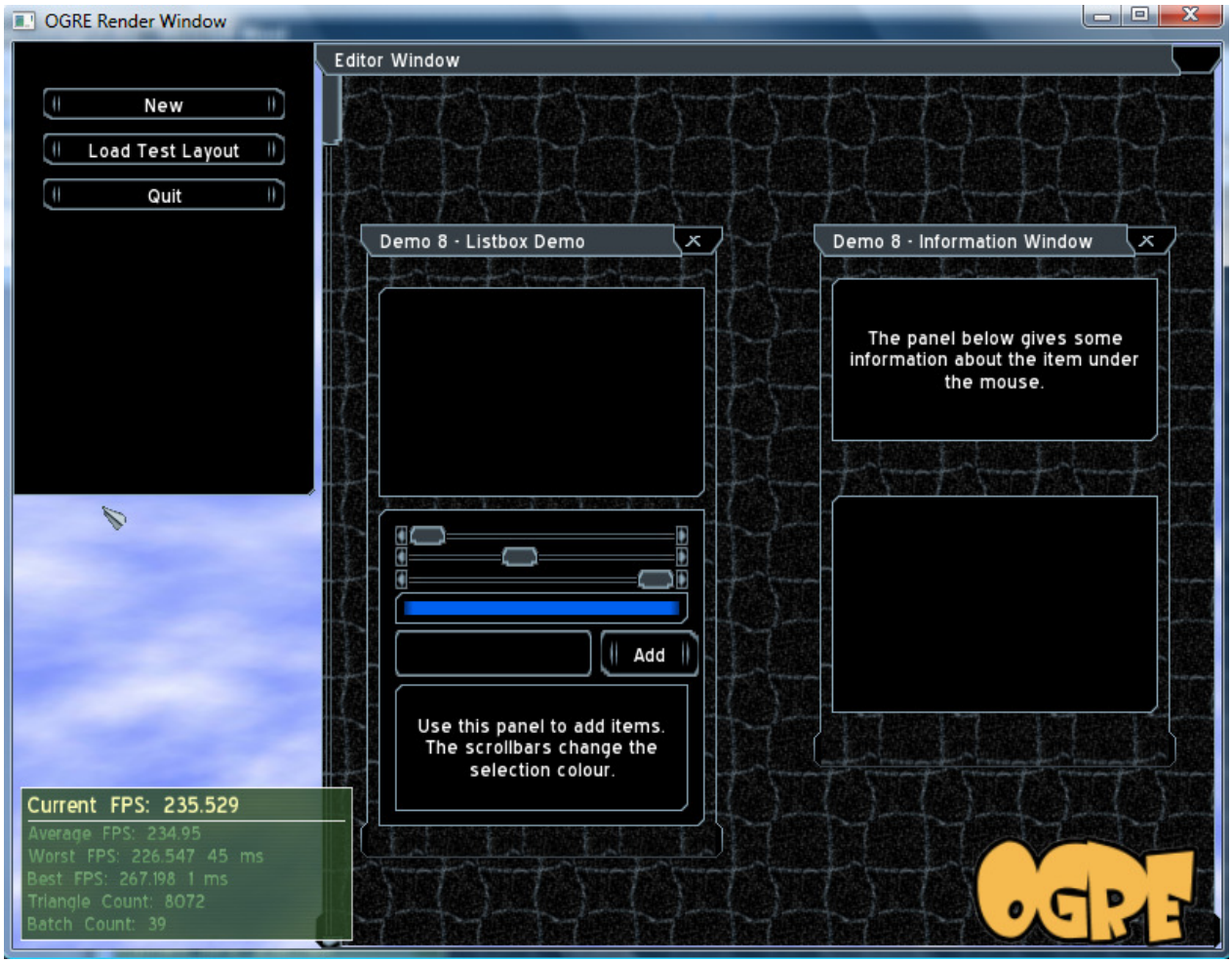
L'interface d'une application 3D est un élément très important. En effet, une mauvaise conception d'interface laissera un gout d'inachevé à une application. Elle entrainera du retard, de l'incohérence et nuira à l'expérience utilisateur concernant notre jeu.

Il existe de divers outils permettant de réaliser des interfaces graphiques. Ceux-ci permettent de faciliter aux développeurs d'application 3D l'implémentation d'interfaces graphiques, afin de les laisser se focaliser sur l'application en elle-même.

Nous avons choisi de réaliser notre interface en CEGUI (Crazy Eddie Graphic User Interface). Il s'agit d'une librairie libre permettant de créer des widgets et des fenêtres par l'intermédiaire d'un moteur 3D. Cette librairie orientée objet est écrite en C++ et supporte DirectX ainsi que l'OpenGL. Elle est directement intégrée à OGRE, c'est pourquoi nous avons décidé de nous tourner vers elle.

6.3.1 Fonctionnement de CEGUI

Comme toutes les librairies de création d'interfaces graphiques, CEGUI sert à créer simplement toute sorte de boutons, de fenêtres, de menus déroulants, de panneaux ou encore de barres déroulantes, et de permettre à l'utilisateur d'interagir avec ceux-ci. CEGUI permet de créer des interfaces qui ressembleront visuellement à l'application 3D qui y sera associé. Ainsi une telle interface est parfaitement intégrable dans un environnement utilisant OGRE. On pourra donc avoir, par exemple, un menu CEGUI, avec une application OGRE en arrière plan.



Cependant, CEGUI étant une surcouche graphique, il ne peut pas s'occuper de la gestion des entrées clavier et souris. Nous devons donc, dans un premier temps, lui communiquer l'évènement produit par l'utilisateur, puis lui transmettre le caractère ou le bouton correspondant. Concernant les entrées clavier, il faut veiller à propager le bon caractère à CEGUI, car il ne fait pas la distinction entre les différentes configurations des touches (clavier français, anglais, etc.).

6.4 Technique d'accélération

Le temps de rendu en 3D temps réel est imperceptible, il doit être inférieur à la persistance rétinienne, ce qui explique le besoin de l'accélération matérielle. Les moteur 3D mettant en œuvre des algorithmes optimisés. Dans notre système en plus des fonctionnalités offerte par le moteur de rendu OGRE 3D permettant une animation fluide et en temps réel d'humain virtuel, nous utilisons des technique d'accélération supplémentaire, LOD, Hardware skinning, Test de visibilité (view frustum test) permettant d'avoir des foules de milliers d'humains virtuels où chaque individu le composant est capable de planifier son chemin et d'éviter des éventuelles collisions.

6.4.1 Le Niveau de Détail (LOD)

LOD est littéralement le niveau de détail; ce niveau de détail dépend de la distance entre un objet et l'objectif de la caméra. Il s'agit d'avoir plusieurs versions d'un même mesh qui sont des versions simplifiées de ce mesh. Par exemple, si le mesh atteint les 5.000 triangles et qu'il se trouve très loin de la caméra, ce mesh représentera seulement quelques pixels à l'écran, mais les 5000 triangles du mesh seront quand même envoyés à l'affichage. Nous pouvons donc remplacer le mesh original par un mesh similaire mais simplifié. Ce qui fait que les objets les plus éloignés sont rendus avec moins de triangles. En effet, de toute façon, il n'y a besoin que de quelques pixels pour rendre les objets distants, rendre de tels objets avec un maximum de polygones reviendrait à gaspiller considérablement de la puissance de calcul.

6.4.2 Hardware skinning

L'animation squelettique est typiquement un bon candidat pour l'accélération sur GPU dans un programme Vertex shader. L'augmentation des performances de rendu par rapport du software skinning est substantielle (particulièrement pour plus qu'un juste peu d'objet skinnés et animés dans une scène).

Dans l'animation squelettiques vertex skinning est le processus de transformation de la position et de la normale des sommets d'un maillage en basant sur la matrice de transformation d'os animés que les sommets sont pondérés aux. Avant l'introduction du pipeline programmable dans le matériel graphique, il été nécessaire de calculer la position et le normale de chaque sommet d'un maillage sur le CPU et envoyez les informations de

chaque sommet au GPU avant que le modèle animé pourrait être rendu correctement. Avec l'utilisation d'un vertex shader programmable, on peut transmettre les informations des sommets du maillage vers la mémoire GPU, puis pour les subséquents rendus, nous passons la matrice de transformation d'os du squelette animée pour le GPU et permettre au programme vertex shader de calculer la position et la normale des sommets animées. L'avantage est qu'au lieu d'envoyer des milliers de sommets au GPU chaque frame, seule une petite fraction des données doit être envoyée pour animer l'ensemble du modèle.

Dans notre système, nous utilisons des vertex buffer afin de stocker des informations des sommets du modèle directement dans la mémoire du GPU et de faire le rendu de modèle animée à l'aide d'un vertex shader écrit dans le langage de shader Cg.

Le programme vertex shader va calculer la position et la normale de chaque sommet de l'objet animé pour chaque frame de rendu.

```
void hardwareSkinningOneWeight_vp(
        float4 position : POSITION,
        float3 normal   : NORMAL,
        float2 uv       : TEXCOORD0,
        float  blendIdx : BLENDINDICES,
        out float4 oPosition : POSITION,
        out float2 oUv       : TEXCOORD0,
        out float4 colour          : COLOR,
        // Support up to 24 bones of float3x4
        // vs_1_1 only supports 96 params so more than this is not feasible
        uniform float3x4   worldMatrix3x4Array[24],
        uniform float4x4 viewProjectionMatrix,
        uniform float4   lightPos[2],
        uniform float4   lightDiffuseColour[2],
        uniform float4   ambient)
```

```
{

// transform by indexed matrix

float4 blendPos = float4(mul(worldMatrix3x4Array[blendIdx], position).xyz, 1.0);

// view / projection

oPosition = mul(viewProjectionMatrix, blendPos);

// transform normal

float3 norm = mul((float3x3)worldMatrix3x4Array[blendIdx], normal);

// Lighting - support point and directional

float3 lightDir0 = normalize (lightPos[0].xyz -  (blendPos.xyz * lightPos[0].w));

float3 lightDir1 = normalize (lightPos[1].xyz -  (blendPos.xyz * lightPos[1].w));

oUv = uv;

colour = ambient + (saturate(dot(lightDir0, norm)) * lightDiffuseColour[0]) +

                (saturate(dot(lightDir1, norm)) * lightDiffuseColour[1]);

}
```

6.4.3 Test de visibilité

Afin de visualiser une scène sous différents angles une caméra virtuelle est souvent utilisée. La configuration de la caméra virtuelle, fait couramment avec de différentes fonctions permettant de détermine ce qui est visible à l'écran.

Le frustum (pyramide de vue) est le volume qui contient tout ce qui est potentiellement (il peut y avoir des occlusions) visibles sur l'écran. Ce volume est défini selon les paramètres de l'appareil, et quand on utilise une projection en perspective prend la forme d'une pyramide tronquée.

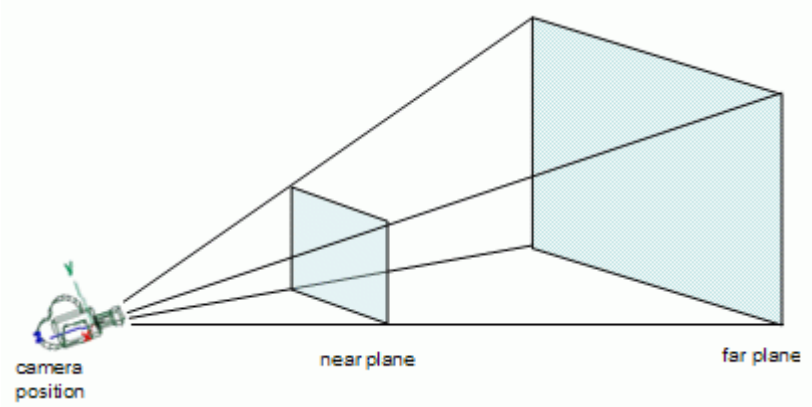
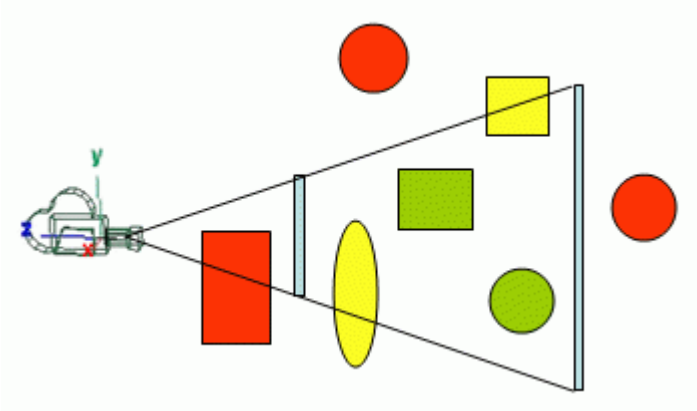


Figure 49 : pyramide de vue

Le sommet de la pyramide est la position de la caméra et la base de la pyramide est le plan distant. La pyramide est tronquée au plan proche, d'où le nom de frustum parvient.

Tous les choses qui seront potentiellement visibles à l'écran sont à l'intérieur, au moins partiellement de la pyramide tronquée, donc il n'est pas nécessaire d'essayer de faire le rendu de ce qui est en dehors du tronc, car il ne sera pas visible en tout cas.



Dans la figure ci-dessus toutes les choses verts (totalement à l'intérieur du frustum vue) et tous les trucs jaunes (partiellement à l'intérieur) seront rendus, alors que les trucs rouges ne seront pas rendus. Notez que la sphère verte n'est pas visible (il est occulté par l'ellipse jaune), mais il sera rendu quand même parce qu'il est à l'intérieur du frustum vue.

L'objectif de frustum vue culling est donc d'être capable d'identifier ce qui est à l'intérieur du frustum (totalement ou partiellement), et d'abattre tout ce qui n'est pas à l'intérieur. Seule la substance qui est à l'intérieur du frustum, même partiellement, est envoyée vers le matériel graphique. En fin de compte, tout ce qui est demandé du matériel graphique est de rendre ce qui est potentiellement visible, des économies sur le traitement de tous ces

sommets qui ne sont pas visibles en tout cas. Par ailleurs, cela peut potentiellement améliorer les performances de l'application, depuis que les sommets qui font partie de la partie visible de l'univers en 3D sont conservés dans la mémoire de carte graphique, et ce sont plus susceptibles d'ajustement que le monde entier en 3D.

Ce test n'a de sens que si la partie du monde en 3D qui est à l'intérieur du frustum est sensiblement plus petit que le monde lui-même. Dans notre système comme nous allons voir OGRE s'occupe automatiquement de cette fonction.

Partant de cette aidé d'affichage qui nous l'étendons pour gérer l'animation de la même manière que le rendu, alors dans notre système on s'occupant seulement que des piétons qui sont à l'intérieur du pyramide de vue c.à.d. les piétons du frustum doivent éviter les collisions avec les objets statique (obstacle) et éviter aussi les inters collision les un avec les autre pendant leur déplacement vers le but. Alors que pour les piétons non visibles l'évitement de collision et l'inter collision n'est pas nécessaire, sauf leur progression qui nous intéresse, donc pour ces derniers on mise à jours seulement leur position

6.5 L'architecture

L'architecture logicielle proposée s'organise sous la forme de quatre modules intégrés au sein d'une même application (Figure 50) :

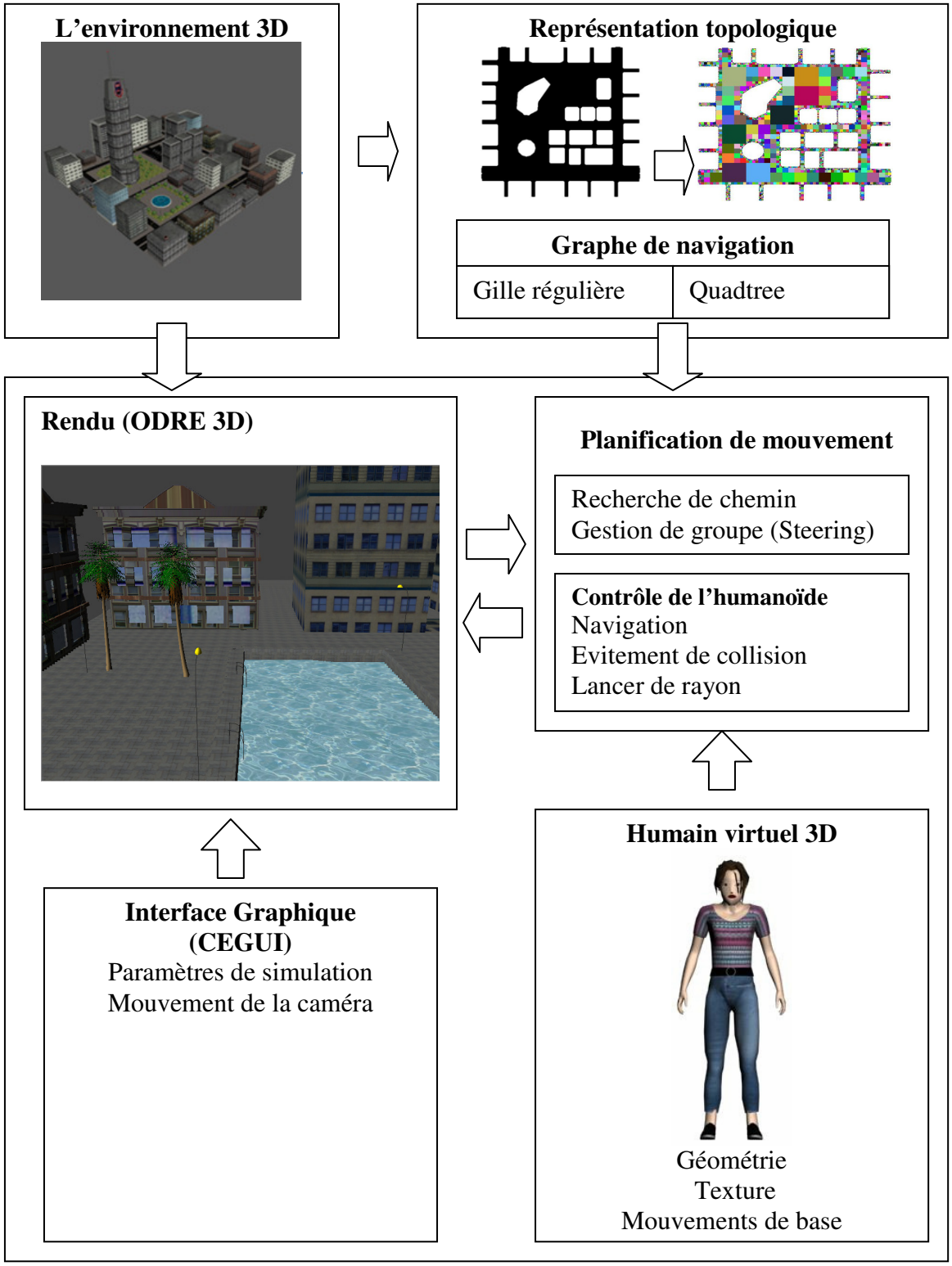


Figure 50 : Architecture du système développé

Le module de représentation topologique : utilise la carte de l'environnement obtenue dans une phase de prétraitement pour générer deux graphe de navigation, la grille régulière et le Quadtree ce module est lancer au démarrage de l'application

Le module d'humain virtuel 3D:

- Instanciation du personnage.
- Technique de variété (textures et accessoires).

Le module de planification de mouvement :

- Rechercher de chemin (A*)
- Positionnement du personnage au sol
- Navigation (lancer l'animation, suivie de la trajectoire).
- Evitement de collision
- Gestion de groupe

Le module d'interface graphique:

- Paramétrer la simulation.
- Contrôler les mouvements de la caméra.

6.6 Interface graphique de l'application

1 : Spécification des paramètres de la simulation.

- Nombre totale d'agents.
- Prise en compte de l'évitement de collision.
- Prise en compte de la gestion du groupe.

2 : Choix de type de Rendu.

- Rendu solide avec lissage.
- Rendu en filaire.
- Rendu en sommet (points)

3 : Contrôler la simulation.

- Lancer la simulation.
- Met la simulation en pause.
- Quitter la simulation.

4 : Contrôler les mouvements de la caméra.

- Touche «flèche du haut » : fait avancer la caméra.
- Touche «flèche droite » : déplace la caméra vers la droite.
- Touche «flèche du bas » : fait reculer la caméra.
- Touche «flèche gauche » : déplace la caméra vers la gauche.

5 : Contrôler la Rotation de la caméra.

- Mouvement horizontale de la souris : Rotation de la caméra autour de l'axe des Y.
- Mouvement verticale de la souris : Rotation de la caméra autour de l'axe des X.

6 : Information sur l'état de la simulation.

- Taux moyen d'affichage.
- Nombre de triangles affichés.

6.7 Résultats

6.7.1 Planification de chemin

Dans cette section nous présentons les résultats de la planification de chemin en utilisant l'algorithme A* avec différente méthode d'évaluation (fonction heuristique).

Fonction heuristique H(c)	Nombre des nœuds visités								
	20	40	50	100	150	200	400	500	750
Distance de Manhattan	56	93	147	341	709	1122	5406	7725	15949
Carré de la Distance Euclidienne	58	127	256	614	1235	2005	5918	9788	30000
Distance Euclidienne	59	189	492	938	1605	3011	7399	14341	32219

Tableau 2 : Temps de calcul de A* (microseconde)

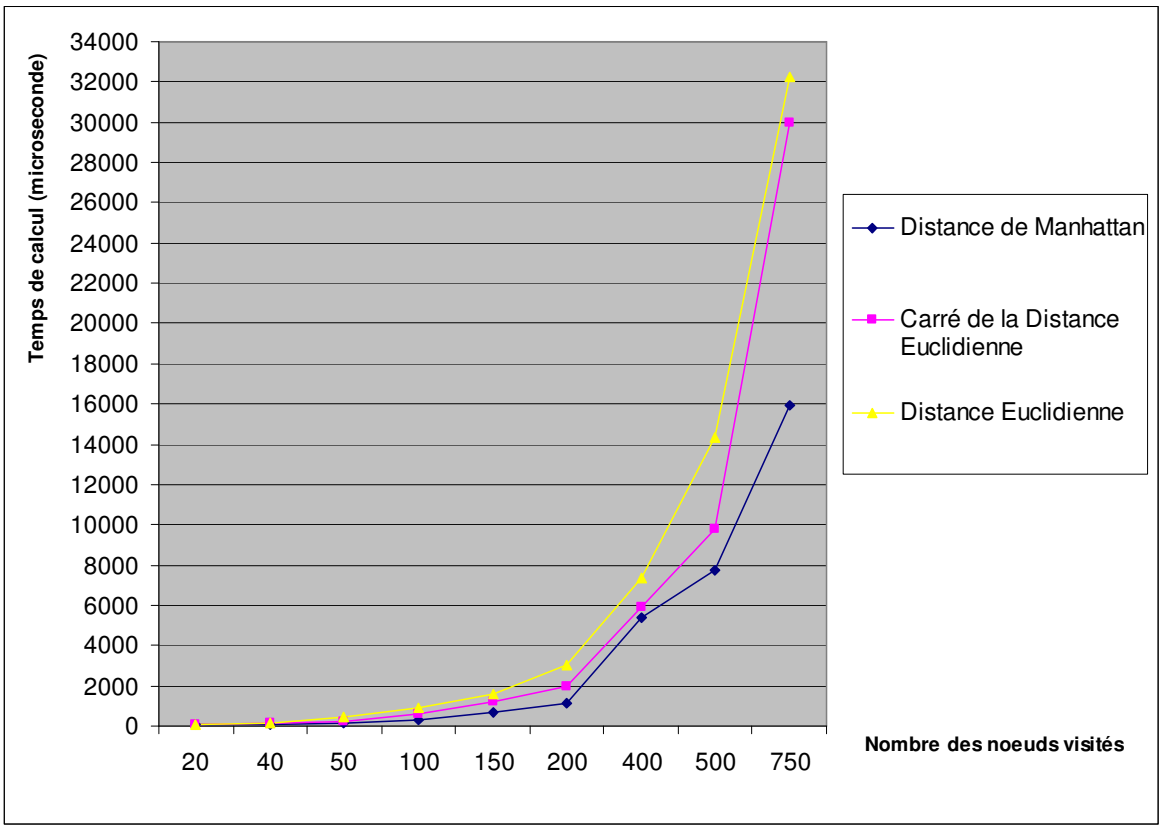


Figure 51 : Temps de calcul de A* suivant la fonction h(c)

Le tableau 2 résume les résultats du temps de calcul de l'algorithme A* en utilisant trois fonction pour évaluer le chemin (I) la distance de Manhattan, (II) le carré de la distance

Euclidienne, (III) la distance Euclidienne, toutes ces résultats sont prêt sur le même nombre de nœud visités On observe que

Les résultats montrent que l'utilisation de A* avec la fonction de distance de Manhattan est plus rapide par rapport les deux autres.et que il est plus lent avec la fonction de distance Euclidienne (due à l'utilisation de la racine carré). Alors que son utilisation avec la fonction le carré de la distance Euclidienne offre des résultats situés entre ces deux premiers.

Puisque nous autorisant les déplacements en diagonale, l'utilisation de la distance de Manhattan n'offre pas dans cas des chemins optimaux.

Pour ce raison nous utilisons le carré de la distance Euclidienne qui offre des chemins de même qualité que l'utilisation de la distance Euclidienne avec un temps de calcul accepté

6.7.2 Simulation

Dans cette section nous présentons les résultats pour l'exécution de différents scénarios de simulation qui incluent 1, 100, 500, 1000, 2000, 3000, 5000 et 10000 agents respectivement. Les agents sont créés au début de la simulation sur les points de départ pré- spécifiés dans la ville virtuelle et sont assignés des tâches globales d'atteindre (buts).

- Modèle d'humanoïdes

Trois modèles différents de personnages (homme et femme et une fille) avec des nombres variables de complexité polygonale (environs 5000 polygones) sont aléatoirement assignés aux agents à l'instanciation. Chaque agent a un ensemble large d'animations squelettiques comprenant : marcher, courir, tourné,… etc.

Les agents sont enlevés de la simulation une fois qu'ils atteignent leurs buts.

Figure 52 : modèles 3D utilisés

- La ville virtuelle

Le bloc de ville virtuel se compose de bâtiments et d'un certain nombre d'autres objets environnementaux, tels que des poteaux de lumières, des panneaux de signalisation, arbres, sièges. Ces objets sont également reconnus comme obstacles par la foule. Par conséquent, ils sont également tenus compte pendant la planification locale de chemin pour l'évitement de collision.

Figure 53 : le rendu 3D d'un environnement urbain

- Taux d'affichage

Des statistiques de simulation sont évaluées pour différents scénarios. Ces scénarios sont définis en ce qui concerne la taille maximum des foules qu'ils les constituant. Ces statistiques incluent (i) le taux d'affichage moyennes images par seconde (fps), et (ii) la taille de la foule. En ce qui concerne le temps de simulation. Toutes les statistiques sont recueillies durant une même durée de simulation. Sur la figure 54, ces résultats sont illustrés.

Taille de foule	50	100	500	1000	2000	3000	5000	10000
Rendu 3D	300	230	45	25	12	8,5	5,5	2,5
Rendu 3D + Optimisation	380	325	115	55	30	20,8	12	6
Simulation	355	315	110	53	26,5	19	8,7	4
Simulation + Evitement de collision	310	250	77	32	17	11	7,5	2,8

Tableau 3 : Taux d'affichage moyen des différents scenarios

Pour les résultats d'essai nous employons différents scénarios de taille différente de 1 agent à 10000 agents.

Pour le premier scénario (Rendu 3D) : nous employons des piétons statiques sans optimisations et les résultats prouvent que le système fonctionne au temps réel pour jusqu'à 1000 agents (25 fps)

Pour le deuxième scénario (Rendu 3D + Optimisation) : nous employons les piétons statiques avec deux genre d'optimisation, nous employons les modèles virtuels d'humains avec quatre niveau de détail (LOD) et nous utilisons aussi skinning accéléré (avec GPU). Les résultats montrent les améliorations et le système fonctionnent au temps réel pour jusqu'à 3000 agents (20.8 fps)

Pour le troisième scénario (Simulation) : nous employons des piétons avec différente sorte d'animations, les agents essayent d'atteindre des buts aléatoires mais sans prise en compte de l'évitement de collision, et nous utilisons le rendu 3D avec optimisation. Comme il est montré dans le tableau 3, le système fonctionne aux taux proche du temps réel pour jusqu'à 3000 agents (19 fps).

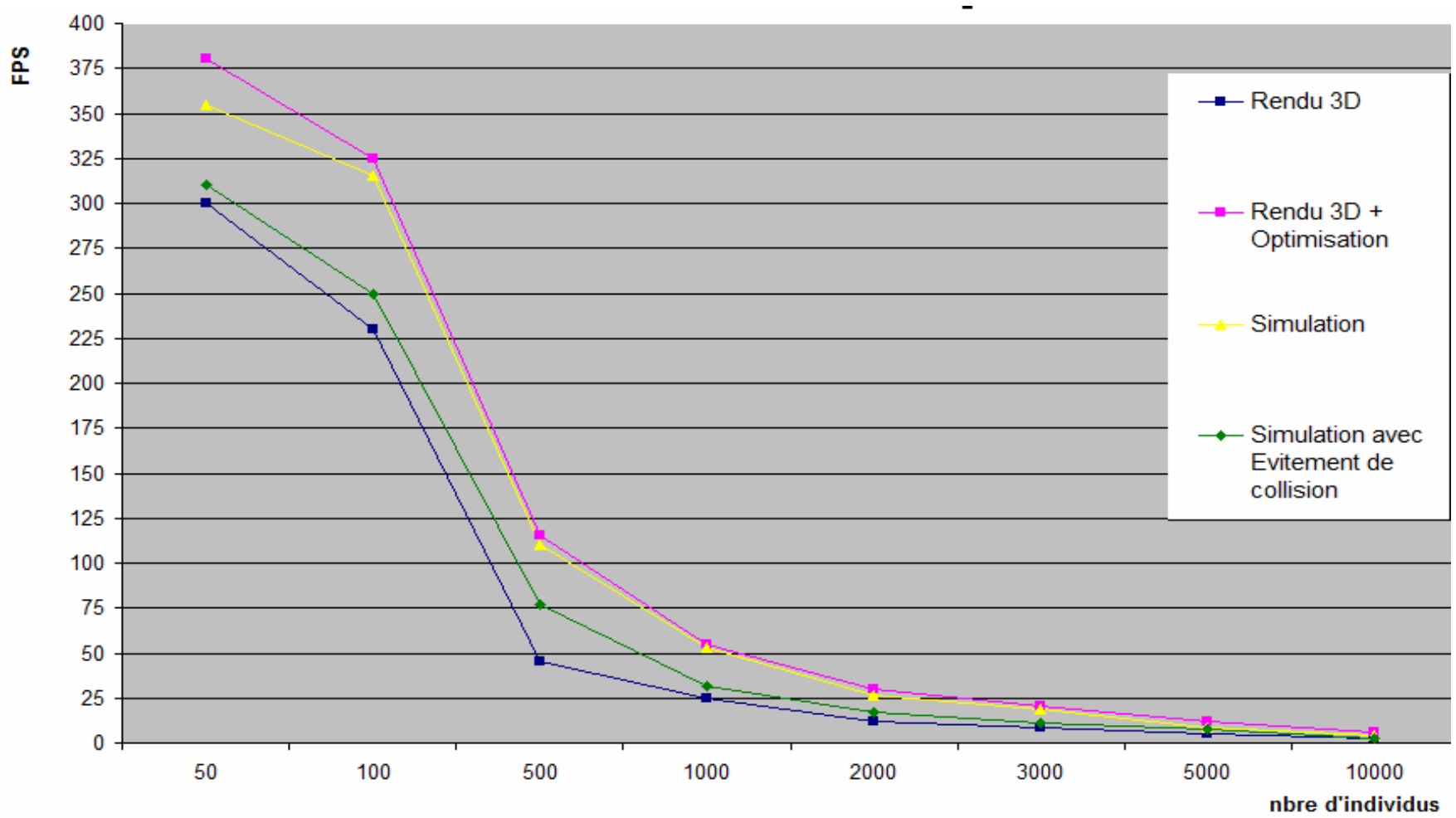


Figure 54 : Taux d'affichage moyen des différents scenarios

Le quatrième scénario (Simulation avec évitement de collision) : nous employons des piétons avec différente sorte d'animations, les agents essayent d'atteindre des buts aléatoires tout en assurant l'évitement de collision, et nous utilisons le rendu 3D avec optimisation. Comme il est montré dans le tableau 3, le système fonctionne aux taux proche du temps réel pour jusqu'à 2000 agents (17 fps).

Figure 55 : une capture d'écran de simulation d'une foule de 500 agents

- Evitement de collision

Des statistiques sur le nombre de collision durant une simulation sont résumés dans le tableau 4 pour deux scénarios, (i) simulation sans prise en compte de la gestion du groupe et (ii) simulation avec la gestion du groupe. Ces statistiques incluent le nombre max de collision par frame pour des foules de différentes tailles. En ce qui concerne l'environnement et le temps de simulation. Toutes les statistiques sont recueillies sur un environnement représenter avec une grille de 150 / 150 cellules et durant une période de simulation de deux minutes (2 min).

Taille de foule	10	20	40	50	60	80	100	200	300	400	500	1000
Simulation sans gestion de groupe	0	0	1	1	1	2	3	4	7	10	14	35
Simulation avec gestion de groupe	0	0	1	2	2	2	3	5	8	10	15	37

Tableau 4 : nombre Max de collisions par frame

On observe que le nombre max de collision par frame augmente avec l'augmentation de la taille de la foule pour les deux scénarios ceci est évident car la foule devienne progressivement dense tandis que l'espace de navigation libre se réduit dans ce cas.
La deuxième remarque est que le nombre max de collision par frame pour le scénario (ii) est un peu plus grand par rapport le scénario (i), cela est justifié par le fait que l'utilisation des règles additionnelles pour la gestion de groupe laissent les membres du même groupe se déplacent tout essayant de rester aussi étroitement les un aux autres, ce qui rend l'évitement de collision avec un groupe un peu difficile.

6.7.3 Evaluation

Afin d'évaluer la pertinence de certains choix, quelques tests ont été effectués. Le principe était le suivant : deux simulations sont lancées avec les mêmes conditions initiales, mais une des deux est privée de la fonctionnalité que l'on souhaite évaluer. L'impact visuel de l'évitement de collision, le repositionnement au sol et des déplacements en petit groupe a ainsi été évalué.

- Evitement de collisions

La figure 56 montre des captures d'écran d'une simulation avec ou sans évitement de collision.

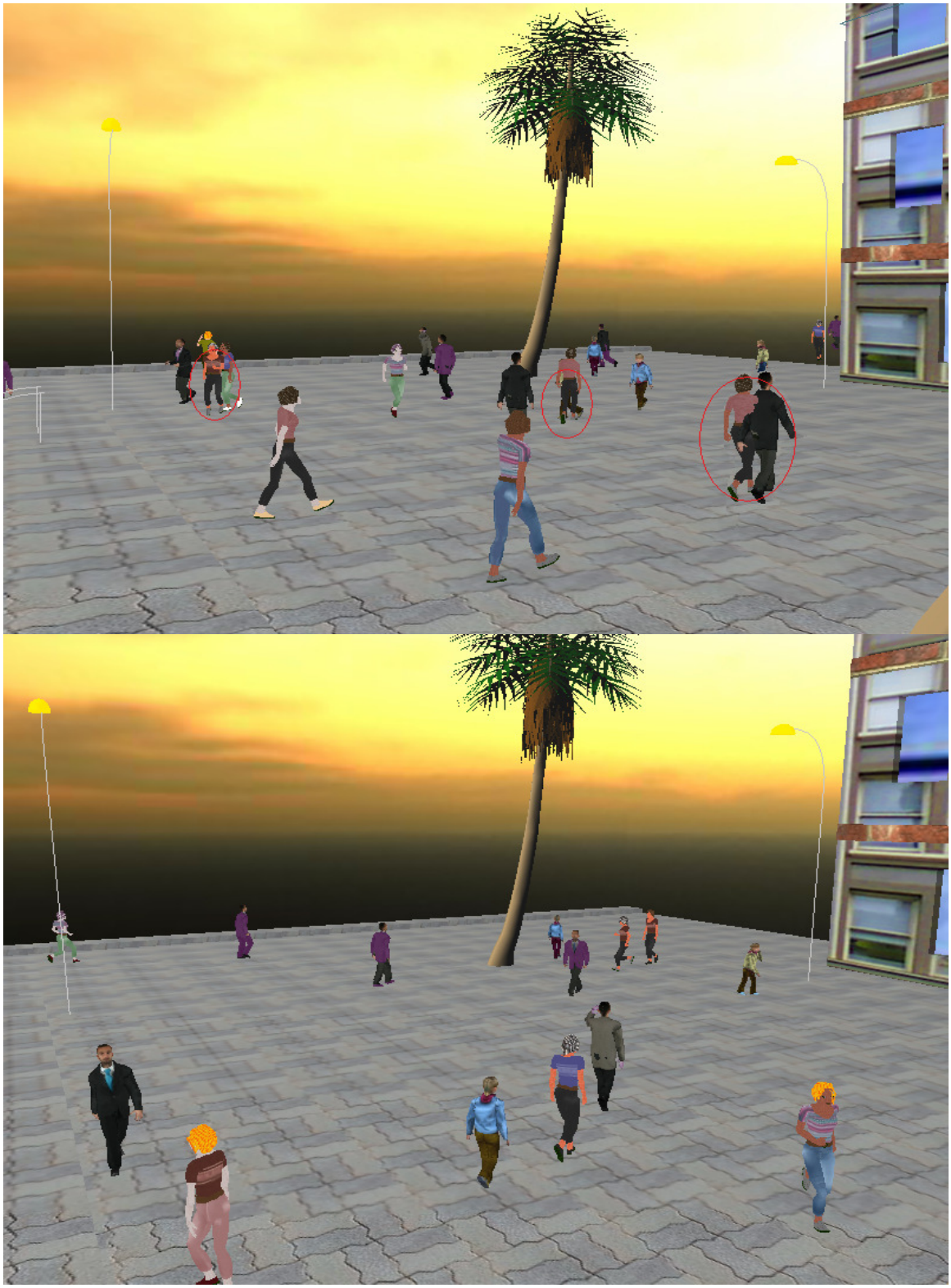


Figure 56 : En haut, l'évitement de collision est désactivé, de nombreuses collisions se produisent (cercle rouge). En bas, il est activé, la simulation est plus réaliste.

- Repositionnement au sol

La figure 57 montre des captures d'écran d'une simulation avec ou sans prise en compte de la hauteur du terrain.

Figure 57 : En haut, la prise en compte de la hauteur du terrain est désactivé, de nombreuses collisions se produisent (cercle rouge). En bas, il est activé, les humanoïdes sont correctement repositionnés.

- **Déplacements en groupe**

La figure 58 montre la différence visuelle entre une simulation sans groupe de piétons et une autre où certains personnages se déplacent à plusieurs.

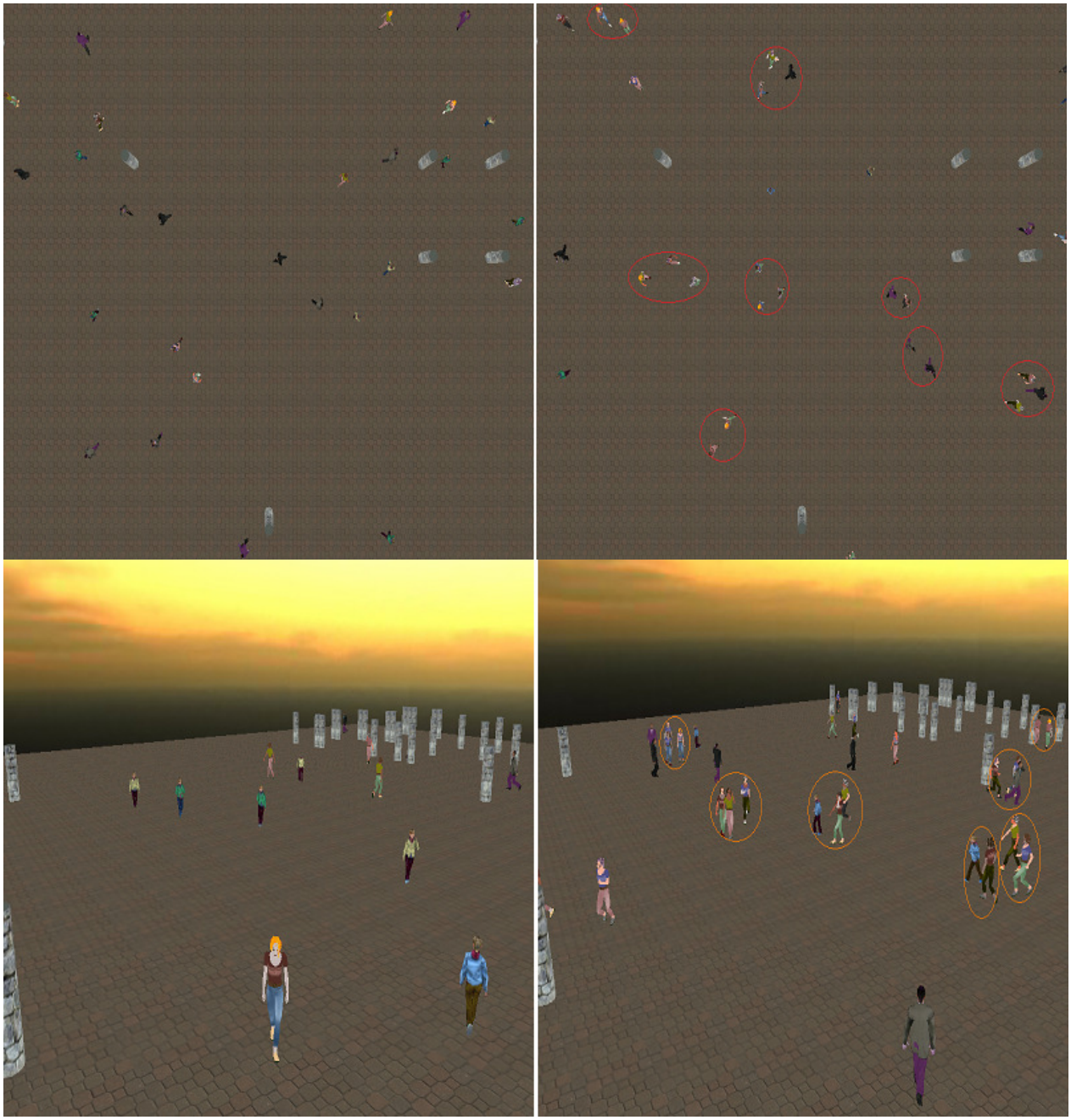


Figure 58 : La capture d'écran de gauche montre une capture d'écran où tout le monde marche seul, celle de droite, où certains piétons se déplacent en petit groupe, parait plus naturelle. (En haut une vue de dessus et en bas une vue en perspective)

- Comportement émergent

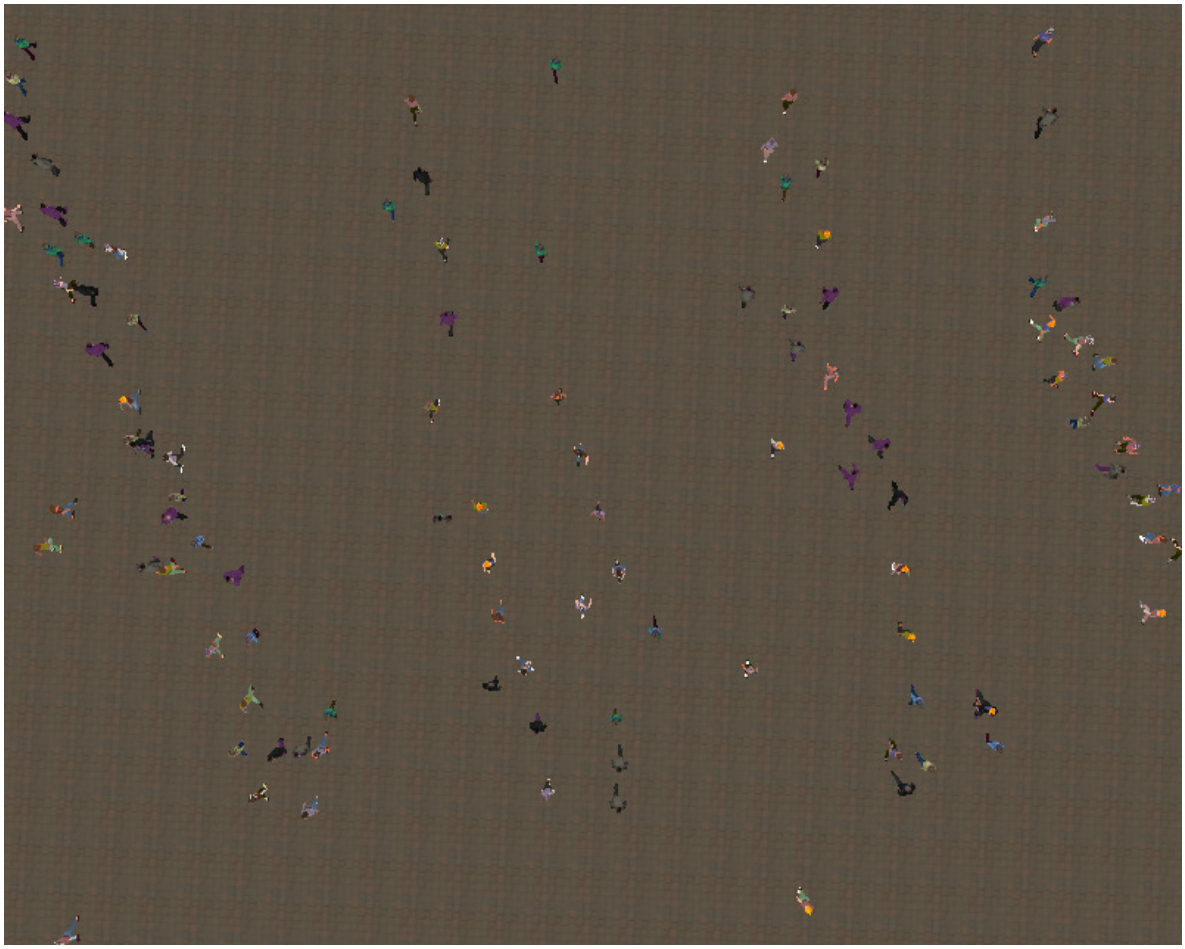


Figure 59 : La formation de lignes entremêlées lorsque deux flux de piétons se rencontrent face à face

Performance

Sur un ordinateur équipé d'un Intel Core 2 Duo 2 GHz, de 4 Go de RAM et d'une carte graphique ATI HD 3430, la simulation peut gérer jusqu'à 1000 de piétons sans qu'il n'y ait de ralentissement notable à l'image (32 fps). Avec 2000 piétons la simulation tourne à 17 fps. On conserve un taux interactif jusqu'à environ 3000 agents (11 fps). Le tableau 4 compare les résultats de notre méthode avec celles présentées dans le Chapitre 3. On peut voir qu'elle réunit des critères que peu d'autres techniques remplissent.

	Nb. de piétons max	**Temps réel**	**Interactivité**	**Qualité**	**Hétérogénéité**	**Groupes**
Shao & Terzopoulos	1200 (sans rendu 3D)	Oui (sans rendu 3D)	Moyenne	Assez Bonne	Moyenne	Non
HiDAC	600	Oui	Bonne	Bonne	Moyenne	Non
Dynamique continue	10000	Non	Faible	Faible	Nulle	Non
Dynamique globale	100000	Entre 0 et 10000 agents	Faible	Faible	Nulle	Non
Patchs de foule	3000	Quasi	Nulle	Faible	Moyenne	Non
Foules par l'exemple	40	Non	Nulle	Bonne	Faible	Non
Notre modèle	3000	2000	Bonne	Bonne	Moyenne	Oui

Tableau 5 : Tableau comparatif

Conclusion générale

Conclusion

La simulation de foules virtuelles est un sujet très difficile car on doit gérer des milliers voir des millions d'individus. Certaines méthodes approchent les foules en tant qu'une seule entité, tel un flot de personnes. Dans notre système nous nous intéressons plutôt à la simulation de chaque individu composant la foule. Nous cherchons à simuler de grande foule en temps réel, les algorithmes choisis ne peuvent provenir directement de l'intelligence artificielle. Il est considéré que les problèmes suivants sont à résoudre : le rendu, la navigation, le traitement des collisions, l'animation.

Nous avons présenté une approche pour résoudre le problème de planification de mouvement de milliers d'humains virtuels en temps réel. Cette approche comprend la combinaison de la représentation en grille régulière et en grilles hiérarchique, une approche permet de réduire et de conserver le temps de calcul nécessaire pour la planification de chemin pour les foules virtuelles simulées. Nous avons aussi étendu notre architecture de planification de mouvement avec un algorithme capable de simuler le comportement de groupes, ce qui améliore la perception de la scène par les utilisateurs. Dans notre système de simulation nous avons utilisés le moteur de rendu OGRE 3D, et plusieurs techniques d'optimisations sont employées : niveau de détail(LOD), GPU - skinning et le test de visibilité (frustum de vue). Les résultats obtenus prouvent que notre système peut fonctionner au temps réel pour jusqu'à 2000 agents.

Perspectives

Pour la planification de chemin nous avons pris en compte un seul critère pour évaluer le coût du chemin qui est la distance entre le point de départ et le point d'arrivé. L'ajout d'autres critères comme la distribution de la densité de population, la nature de la zone à traverser, sa difficulté sera un objectif d'amélioration sollicité.

En plus des techniques d'accélération utilisées nous souhaitons employer une nouvelle méthode « geometry instancing » pour instancier les individus de la foule en exploitant en plus les performances des cartes graphiques actuelles.

Concernant l'animation, enrichir la base des mouvements avec d'autres mouvements de base comme le mouvement de la tête, la possibilité de faire des conversations, des activités habituelle, etc. Ce qui permet d'ajouter des comportements intelligents qui seront déclenchés selon la situation et la position des piétons.

Références

[AND 05] Rasmus Andersen, Jean Louis Berrou, et Alex Gerodimos. On some limitations of grid-based (ca) pedestrian simulation models. Dans First International Workshop on Crowd Simulation (V-Crowds'05). VRlab, EPFL, 2005.

[ARC 79] J. Archea. The evacuation of non-ambulatory patients from hospital and nursing home fires : A framework for a model. Technical report, novembre 1979. NBSIR 79-1906.

[ARI 01] O. Arikan, S. Chenney, et D. A. Forsyth. Efficient multi-agent path planning. Dans Computer Animation and Simulation '01, pages 151–162. Springer-Verlag, 2001.

[AUB 00] A. Aubel, R. Boulic, and D. Thalmann. Real-time display of virtual humans : levels of details and impostors. *IEEE Transactions on Circuits and Systems for Video Technology*, 10(2) :207- 217, 2000.

[BAR 91] J. Barraquand et J.-C. Latombe. Robot motion planning : a distributed representation approach. Internation Journal of Robotics Research, 10(6) :628–649, 1991.

[BAY 03] O. Burchan Bayazit, Jyh-Ming Lien, and Nancy M. Amato. Better group behaviors in complex environments using global roadmaps. In ICAL 2003, pages 362–370, 2003.

[BOI 98] J.-D. Boissonnat et M. Yvinec. Algorithmic Geometry. Cambridge University Press, 1998.

[BOI 95] J.-D. Boisonnat et M. Yvinec. Géométrie algorithmique. Collection Informatique. EDISCIENCE international, 1995.

[BOT 04] Adi Botea, Martin Müller, et Jonathan Schaeffer. Near optimal hierarchical pathfinding. Journal of Game Development, 1(1) :7–28, 2004.

[CAS 01] S. Caselli, M. Reggiani, et R. Rocchi. Heuristic methods for randomized path planning in potential fields, 2001.

[CHE 87] L. P. Chew. Constrained delaunay triangulations. Dans Proceedings of the third annual symposium on Computational geometry, pages 215–222. ACM Press, 1987.

[CHE 04] S. Chenney. Flow Tiles. In. Proc. ACM SIGGRAPH/ Eurographics Symposium on Computer Animation, Grenoble, France: 233–242, 2004.

[CHO 03] M. G. Choi, J. Lee, et S. Y. Shin. Planning biped locomotion using motion capture data and probabilistic roadmaps. ACM Transactions on Graphics, 22(2) :182–203, 2003.

[COI 07] Jean-Marc Coic, Céline Loscos, and Alexandre Meyer. Three lod for the realistic and real-time rendering of crowds with dynamic lighting. Research Report RN/06/20, Université Claude Bernard, LIRIS, France, April 2007.

[COI 05] J.-M. Coic, C. Loscos, and A. Meyer. Three LOD for the realistic and real-time rendering of crowds with dynamic lighting. Research Report LIRIS, France, 2005.

[COU 05] N. Courty and S. R. Musse. Simulation of large crowds in emergency situations including gaseous phenomena. In CGI '05: Proceedings of the Computer Graphics International 2005, pages 206–212, Washington, DC, USA, 2005. IEEE Computer Society.

[DE 05] P. de Heras Ciechomski, S. Schertenleib, J. Maïm, D. Maupu, and D. Thalmann. Real-time shader rendering for crowds in virtual heritage. In VAST'05, pages 1–8, 2005.

[DES 03] Michael Despina and Chrisanthou Yiorgos. Automatic high level avatar guidance based on the affordance of movement. European Association for Computer Graphics, pages 87- 98, September 2003.

[DOB 05] S. Dobbyn, J. Hamill, K. O'Conor, and C. O'Sullivan. Geopostors : A real-time geometry/impostor crowd rendering system. Proceedings of the ACM SIGGRAPH 2005 Symposium on Interactive 3D Graphics and Games (SI3D), ACM Press, pages 95 -102, 2005.

[GLO 04] Christian Gloor, Pascal Stucki, et Kai Nagel. Hybrid techniques for pedestrian simulations. Dans 4th Swiss Transport Research Conference, Monte Verità, Ascona, 2004.

[HAR 06] Christopher Hartman et Bedrich Benes. Autonomous boids : Research articles. Computer Animation and Virtual Worlds, 17(3,4) :199–206, 2006.

[HEL 00] Dirk Helbing, Illes Farkas, and Tamas Vicsek. Simulating dynamical features of escape panic. Nature, 407:487–490, 2000.

[HEN 71] L. F. Henderson. The statistics of crowd fluids. Nature, 229 :381–383, 1971.

[HOF 99] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, et T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. Computer Graphics, 33 :277–286, 1999.

[HOP 96] H. Hoppe. Progressive meshes. SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 99–108, 1996.

[HUG 03] R.L. Hughes. The Flow of Human Crowds. Annual Review of Fluid Mechanics, vol. 35, 2003.

[KAL 03] M. Kallmann, H. Bieri, et D. Thalmann. Fully dynamic constrained delaunay triangulations. Geometric Modelling for Scientific Visualization, 2003.

[KAM 04] A. Kamphuis and M.H. Overmars. Finding paths for coherent groups using clearance. SCA'04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 19–28, 2004.

[KIR 03] Kirchner, A., Namazi, A., Nishinari, K. and Schadschneider, A. Role of Conflicts in the Floor Field Cellular Automaton Model for Pedestrian Dynamics. In. Proc. 2nd International Conference on Pedestrians and Evacuation Dynamics. (PED), London, UK: 51–62, 2003.

[KLÜ 04] Hubert Klüpfel and Tim Meyer-König. Simulation of the evacuation of a football stadium. InTraffic and Granular Flow ’03, pages 423–430, Berlin, Germany, 2004. Springer.

[KOR 85] R. E. Korpf. Depth-first iterative-deepening : An optimal admissible tree search. Artificial intelligence, 27(1) :97–109, 1985.

[KWO 08] T. Kwon, K. Hoon Lee, J. Lee, and S. Takahashi. Group motion editing. In SIGGRAPH’08: ACM SIGGRAPH 2008 papers, pages 1–8, 2008.

[KUF 98] J. J. Kuffner. Goal-directed navigation for animated characters using real-time path planning and control. Lecture Notes in Computer Science, 1537 :171–179, 1998.

[LAM 04] Fabrice Lamarche and Stephane Donikian. Crowd of virtual humans : a new approach for real time navigation in complex and structured environments. Computer Graphics Forum, 23(3) :509- 518, 2004.

[LAM 03] Fabrice Lamarche. Humanoïde virtuels, réaction et cognition : une architecture pour leur autonomie. Thèse de Doctorat, I.F.S.I.C., Université de Rennes I, IRISA-INRIA, Campus de Beaulieu, 35042 Rennes cedex, France, décembre 2003.

[LAN 98] J. Lander. Skin them bones. Game Developer Magazine, pages 11–14, May 1998.

[LER 07] A. Lerner, Y. Chrysanthou, and D. Lischinski. Crowds by example. Computer Graphics Forum (Proceedings of Eurographics), 26(3), 2007.

[LI 04] Huibo Li,Wen Tang, and Derek Simpson. Behaviour based motion simulation for fire evacuation procedures. Theory and Practice of Computer Graphics, 0:112–118, 2004.

[LOG 98] B. Logan et N. Alechina. A* with bounded costs. Dans Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98), pages 444– 449, 1998.

[LOS 03] C. Loscos, D. Marchal, and A. Meyer. Intuitive crowd behaviour in dense urban environments using local laws. Theory and Practice of Computer Graphics, pages 122–129, 2003.

[MAÏ 09] J. Maïm. Generating, Animating, and Rendering Varied Individuals for Real-Time Crowds. PhD thesis, EPFL, 2009.

[MUS 00] Soraia Raupp Musse. Human crowd modelling with various levels of behaviour control. Thèse de Doctorat, EPFL : École Polytechnique Fédérale de Lausanne, Lausanne, Suisse, 2000.

[MUS 01] S. Raupp Musse and D. Thalmann. A hierarchical model for real time simulation of virtual human crowds. IEEE Transactions on Visualization and Computer Graphics, 7(2):152–164, 2001.

[NAR 09] R. Narain, A. Golas, S. Curtis, M. C. Lin. Aggregate Dynamics for Dense Crowd Simulation. SIGGRAPH, 2009.

[NIE 03] C. Niederberger and M. H. Gross. Hierarchical and heterogeneous reactive agents for realtime applications. Computer Graphics Forum, 22(3):323–331, 2003.

[OSU 03] C. O'Sullivan, J. Cassell, H. Vilhjámsson, J. Dingliana, S. Dobbyn, B. McNamee, C. Peters, and T. Giang. Levels of detail for crowds and groups. Computer Graphics Forum, 21(4): 733–741, 2003.

[OVE 02] M. H. Overmars. Recent developments in motion planning. Dans International Conference on Computational Science (3), pages 3–13, 2002.

[PEL 07] N. Pelechano, J.M. Allbeck, N.I. Badler. Controlling Individual Agents in High-Density Crowd Simulation. SIGGRAPH Symposium on Computer Animation, 2007.

[PET 03] J. Pettré, J.-P. Laumond, et T. Siméon. A 2-stages locomotion planner for digital actors. Dans Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA'03), pages 258–264, 2003.

[PET 06] Julien Pettré, Pablo de Heras Ciechomski, Jonathan Maïm, Barbara Yersin, Jean-Paul Laumond, et Daniel Thalmann. Real-time navigating crowds : scalable simulation and rendering. Computer Animation and Virtual Worlds, 17(3-4) :445– 455, 2006.

[REI 94] B. Reich, H. Ko, W. Becket, et N. I. Badler. Terrain reasoning for human locomotion. Dans Computer Animation, pages 996–1005. IEEE, 1994.

[REY 87] C. W. Reynolds. Flocks herds and schools : A distributed behaviour model. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques, pages 25- 34. ACM Press, 1987.

[REY 99] C. W. Reynolds. Steering behaviors for autonomous characters. Proc. of Game Developers Conference, pages 763–782, 1999.

[REY 06] C. W. Reynolds. Big fast crowds on ps3. In sandbox '06: Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames, pages 113–121, 2006.

[SHA 05] Wei Shao et Demetri Terzopoulos. Autonomous pedestrians. Dans SCA '05 : Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 19–28, New York, NY, USA, 2005.
[SHA 05b] W. Shao, D. Terzopoulos. Environmental Modeling for Autonomous Virtual Pedestrians. SAE Symposium on Digital Human Modeling for Design and Engineering, 2005.

[SHE 08] Ameya Shendarkar, Karthik Vasudevan, Seungho Lee, and Young-Jun Son. Crowd simulation for emergency response using bdi agents based on immersive virtual reality. Simulation Modelling Practice and Theory, 16(9):1415–1429, June 2008.

[SHE 06] Ameya Shendarkar, Karthik Vasudevan, Seungho Lee, and Young-Jun Son. Crowd simulation for emergency response using bdi agent based on virtual reality. In

WSC ’06: Proceedings of the 38th conference on Winter simulation, pages 545–553. Winter Simulation Conference, 2006.

[SUD 08] Avneesh Sud, Erik Andersen, Sean Curtis, Ming C. Lin, and Dinesh Manocha. Real-time path planning in dynamic virtual environments using multiagent navigation graphs. IEEE Transactions on Visualization and Computer Graphics, 14(3) :526- 538, 2008.

[TEC 00] F. Tecchia et Y. Chrysanthou. Real time rendering of densely populated urban environments. Dans RenderingTechniques ’00 (10th EurographicsWorkshop on Rendering), pages 45–56, Brno, Czech Republic, 2000. Springer-Verlag.

[TEC 01] Tecchia, F., Loscos, C., Conroy, R. and Chrysanthou, Y. Agent Behavior Simulator (ABS): A Platform for Urban Behavior Development. In. Proc. ACM/EG Games Technology Conference (2001).

[TEC 02.b] Franco Tecchia, Céline Loscos, et Yiorgos Chrysanthou. Visualizing crowds in real-time. Computer Graphics Forum, 21(4) :753–765, 2002.

[TEC 02.a] Franco Tecchia, Céline Loscos, and Yiorgos Chrysanthou. Image based crowd rendering. IEEE Computer Graphics and Applications, 22(2) : 36 - 43, 2002.

[THR 96] S. Thrun et A. Bücken. Integrating grid-based and topological maps for mobile robot navigation. Dans Proc. of the AAAI Thirteenth National Conference on Artificial Intelligence, pages 944–951. AAAI Press / MIT Press, 1996.

[TOR 07] Torrens, P. M. Behavioral intelligence for geospatial agents in urban environments. In. Proc. IEEE Intelligent Agent Technology, Los Alamitos, CA, IEEE: 63–66, 2007.

[TRE 06] Adrien Treuille, Seth Cooper, et Zoran Popovi´c. Continuum crowds. Dans SIGGRAPH ’06 : ACM SIGGRAPH 2006 Papers, pages 1160–1168, New York, NY, USA, 2006. ACM Press.

[TUX 94] Tu, X. and Terzopoulos, D. Artificial Fishes: Physics, Locomotion, Perception, Behavior. In. Proc. ACM SIGGRAPH, ACM Press. New York, USA: 43–50, 1994.

[ULI 04] B. Ulicny, P. de Heras, and D. Thalmann. Crowdbrush: Interactive authoring of real-time crowd scenes. SCA ’04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation, pages 243–252, 2004.

[ULI O1] Ulicny, B. and Thalmann, D. Crowd Simulation for Interactive Virtual Environments and VR Training Systems. In. Proc. Eurographics Workshop on Animation and Simulation, Springer-Verlag. Berlin: 163–170, 2001.

[USM 07] Zeeshan-ul-hassan Usmani, Andrew English, and Richard Griffith. Modeling the effects of a suicide bombing: Crowd formations. In Proceedings of the Interservice/Industry Training, Simulation, and Education Conference (I/ITSEC) 2007, page 7399, Orlando, FL,USA, 2007.

[WON 07] Kai Yip Wong, Mary-Ann Thyvetil, Andriana Machaira, and Céline Loscos. Density distribution and local behaviour for realistic crowd simulation. In IADIS International Computer Graphics and Visualization. Springer Berlin / Heidelberg, 2007.

[WON 08] Kai Yip Wong and Céline Loscos. Hierarchical Path Planning for Virtual Crowds, volume 5277 of Motion in Games. Springer Lecture notes in Computer Sciences, Edited by Arjan Egges and Arno Kamphuis and Mark Overmars, 2008.

[WOL 83] S.Wolfram. Statistical Mechanics of Cellular Automata. Reviews of Modern Physics. 55(3): 601–644, (1983).

[YAH 98] A. Yahja, A. Stentz, S. Singh, et B. L. Brumitt. Framed-quadtree path planning for mobile robots operating in sparse environnements. Dans IEEE Conference on Robotics and Automation (ICRA), Leuven, Belgium, Mai 1998.

[YER 09] B. Yersin, J. Maïm, J. Pettré, D. Thalmann. Crowd Patches : Populating Large-Scale Virtual Environments for Real-Time Applications. Proceedings of Symposium on Interactive 3D Graphics and Games, 2009.

[PED 06] Micro-Pedsim.
http://people.revoledu.com/kardi/research/pedestrian/micropedsim/download.htm, 2006. [Accédé le 01 avril 2012].

[GRE 00] Robin Green. Steering behaviours. Dans Tutorial at SigGraph'00, 2000.
http://www.red3d.com/siggraph/2000/course39. [Accédé le 01 avril 2012].

[LEG 07] Legion. http://www.legion.com, 2007. [Accédé le 01 avril 2012].

[MAS 11] Massive Software Inc. http://www.massivesoftware.com, 2011. [Accédé le 01 avril 2012].

[ALI 03] Alice. http://www.kolve.com/vfxwork/vfxwork.htm, 2003. [Accédé le 01 avril 2012].

[SPI 03] SpirOps Crowd. http://www.spirops.com, 2003. [Accédé le 01 avril 2012].

Annexe A : Fonctionnement d'OGRE

Voyons un peu plus en détail le fonctionnement d'OGRE. Dans un premier temps, nous verrons les étapes nécessaires à l'initialisation de l'application, puis le système de scène, le lien entre entités et nœuds, et enfin la boucle événementielle.

1. Ressources

OGRE utilise beaucoup de fichiers annexes, comme les modèles des objets 3D ou les textures. Il est nécessaire de lui indiquer dans quels répertoires il doit chercher les différentes ressources. Pour cela, plusieurs fichiers de configuration doivent être créés et analysés lors du chargement de l'application.

2. Objet racine

La première chose à réaliser pour une application OGRE est de créer un objet racine. Cet objet est la base de l'application, car tous les autres objets lui seront rattachés.

3. Choix du système de rendu

Il faut ensuite choisir le mode d'affichage, le type de rendu (DirectX ou OpenGL), la résolution et autres paramètres graphiques parmi ceux supportés par la machine.

4. Scène

Une scène dans OGRE est un élément qui va contenir des éléments graphiques. Elle comporte une hiérarchie similaire aux arbres : une racine « père » à laquelle vont se rattacher d'autres objets, qui pourront à leur tour devenir parents. Un objet parent transmet certaines données à tous ses fils. Ainsi, un fils nouvellement créé sera positionné dans la scène par rapport aux coordonnées de son parent. De même, une transformation de type translation, rotation ou redimensionnement se répercutera sur tous les fils d'un objet. L'inverse n'est cependant pas vrai : un fils subissant une transformation ne la communique pas à son parent.

5. Nœuds et Entités

Un élément important du fonctionnement d'OGRE est le lien entre les nœuds et les entités. Une entité est un élément graphique chargé ou généré. Il contient donc l'objet à afficher, la texture associée, ainsi qu'une possible animation. Néanmoins, il est impossible de le visualiser tel quel. Il doit être associé à un nœud de la scène, qui contiendra toutes les informations nécessaires à son affichage, notamment sa position dans celle-ci. Toutes les

transformations que doit subir l'objet doivent être appliquées au nœud auquel il est rattaché. Un nœud peut posséder plusieurs entités, en revanche une entité n'a qu'un seul nœud.

6. Caméra et point de vue

Une fois les objets graphiques correctement chargés, il faut décrire la portion de scène à afficher. Autrement dit, un point de vue doit être spécifié à l'application, grâce à l'ajout d'une caméra dans celle-ci. La caméra va nous permettre de voir nos objets, de changer l'angle de vue ou encore la distance. Elle doit être associée à un nœud de la scène, et peut changer dynamiquement de nœud, ce qui permet de changer d'angle de vue durant l'exécution. S'il n'y a généralement qu'un seul point de vue à la fois, il est toutefois possible d'en rajouter d'autres et de les afficher simultanément à l'écran.

7. Boucle d'évènements

Une fois que tous les objets OGRE sont initialisés, il faut lancer la boucle d'évènements. A chaque tour de cette boucle, les événements sont captés et traités puis, en fonction des traitements ainsi fait, on calcule l'image à afficher. Lorsque cette boucle est interrompue, volontairement ou non dans le cas d'une erreur du programme, l'application se termine.

Annexe B : Animation des personnages

Un moteur 3D permet de simplifier grandement la réalisation d'une application graphique. Néanmoins, il est nécessaire de disposer d'objets 3D : des modèles, qui peuvent être directement créés par le moteur graphique. Des logiciels sont toutefois dédiés à cette tâche, permettant ainsi de faciliter grandement leur création. Ce que nous allons vous présenter ici est la solution que nous avons choisi pour la création de nos modèles : 3D Studio Max.

1. Présentation de 3D Studio Max

3D Studio Max est un outil puissant de modélisation, d'animation et de rendu tridimensionnel développé par l'entreprise Autodesk. De nombreux jeux professionnels ont bénéficié de ce logiciel, tels que les récents Fallout, Gears of War ou encore Warhammer Online.

Ce présent document va expliquer comment préparer l'animation du personnage 3D à l'aide du logiciel de modélisation 3ds max 7 pour pouvoir l'utilisé dans le moteur de rendu OGRE 3D.

2. Import du Mesh

Ce que l'on appelle mesh, c'est l'objet en 3 dimensions que forme un ensemble de faces. On dit faces, mais on devrait même dire vertex. Les vertex sont en fait les points dans l'espace 3D qui sont reliés entre eux pour former des faces. En premier lieu, ce sont donc eux qui composent le mesh. Il faut savoir que tout objet 3D est dessiné grâce à des faces ; une face étant formée par 3 vertex, donc si nous dessinons un carré, nous aurons 2 faces.

Dans un premier temps, nous allons importer notre mesh de personnages « humanoïdes », c'est-à-dire à la morphologie proche de celle de l'être humain à 3Ds max et dans ce qui suit on va voir comment le préparer pour l'animation.

3. Animation de personnages avec Character Studio

Caractère studio est un outil très performant intégré à 3D studio Max et dédié à l'animation de personnages en 3D. Il se décompose en trois composants principaux :

Biped qui permet de créer et animer le squelette d'un personnage

Physique qui est utilisé pour appliquer l'animation d'un squelette sur le maillage réel du personnage

Crowd qui permet d'animer des groupes de personnages avec différents comportements ainsi que des interactions.

La grande force de Character Studio est en effet de fournir de puissants outils d'abstraction et d'encapsulation des animations afin de permettre leur réutilisation aisée.

3.1. rigging

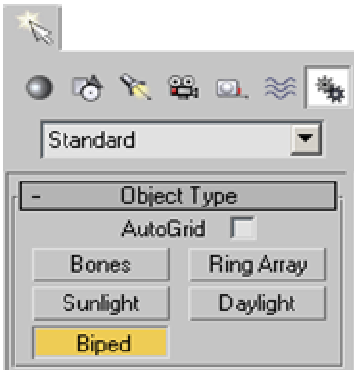
Il s'agit de la partie qui consiste à créer un « squelette » à l'intérieur d'une modélisation pour pouvoir l'animer par la suite. Character Studio permet de créer rapidement un squelette hiérarchique (plus de 10 niveaux de hiérarchie) de base automatiquement « riggué » sur un modèle par défaut qu'il a fallu adapter à la morphologie « particulière » des personnages. Une attention particulière doit être portée à ce squelette car sa morphologie déterminera fortement les mouvements qui seront par la suite appliqué sur le personnage.
Le maillage peut être appliqué sur le Biped soit à l'aide du modificateur Skin utilisé également avec les Bones soit avec le Modificateur Physique spécifique à Character Studio. Physique à l'avantage de permettre des simulations complexes de muscles et de tendons par exemple.

Nous allons voir comment créer un biped et assigner les vertex de notre mesh à ce dernier pour pouvoir l'animer. Pour nous faciliter la tâche du placement du biped, nous allons nous empêcher de déplacer notre mesh.

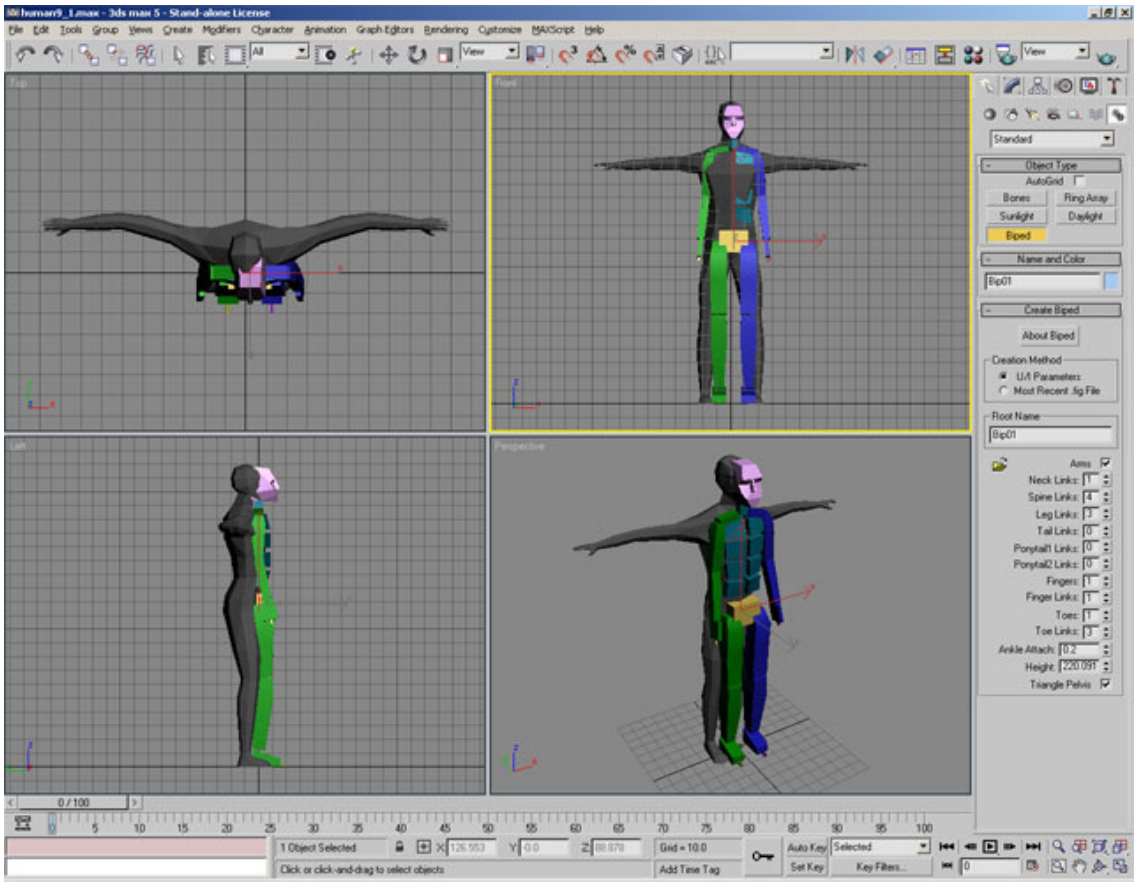
Sélectionnons donc notre mesh puis faisons un clic droit et cliquons sur 'Freeze selection'. A ce moment, notre mesh devient gris et nous ne pouvons plus le bouger.

3.1.1. Créer le biped

Bien logique, nous allons donc créer notre 'biped'. Pour cela, allons dans l'onglet 'Create' puis cliquons sur le bouton 'Systems' et enfin 'Biped' :



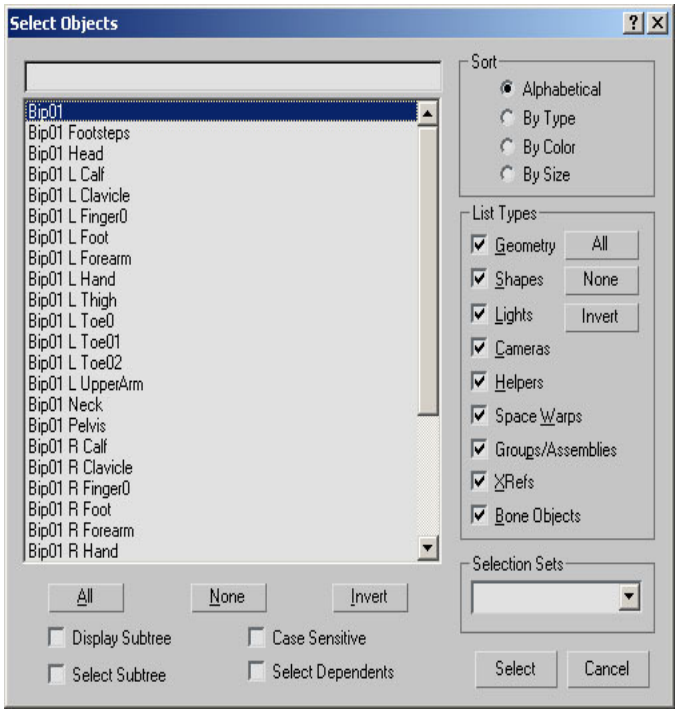
Dessinons ensuite notre biped simplement en maintenant le bouton gauche enfoncé et en déplaçant la souris. Essayons de faire en sorte que le "cadre" rouge soit à peu près à la même échelle que notre mesh. Lorsque nous lâchons le bouton, 3dsmax se charge de dessiner toutes les parties du squelette :



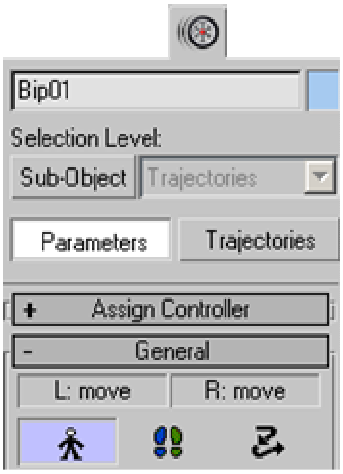
Si cela ne correspond pas tout à fait à la forme de notre mesh, nous devons le corriger.

3.1.2. Redimensionner le biped

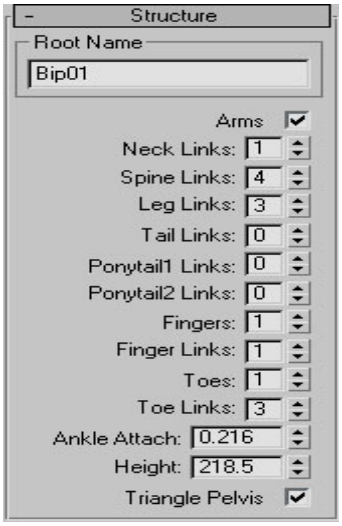
Par défaut, on peut voir que notre biped n'a, par exemple, qu'un seul doigt à chaque main. On va devoir remédier à cela en sélectionnant le biped, Cliquant sur le bouton de sélection par nom ('Select by Name'), dans la liste et double clique sur 'Bip01' :



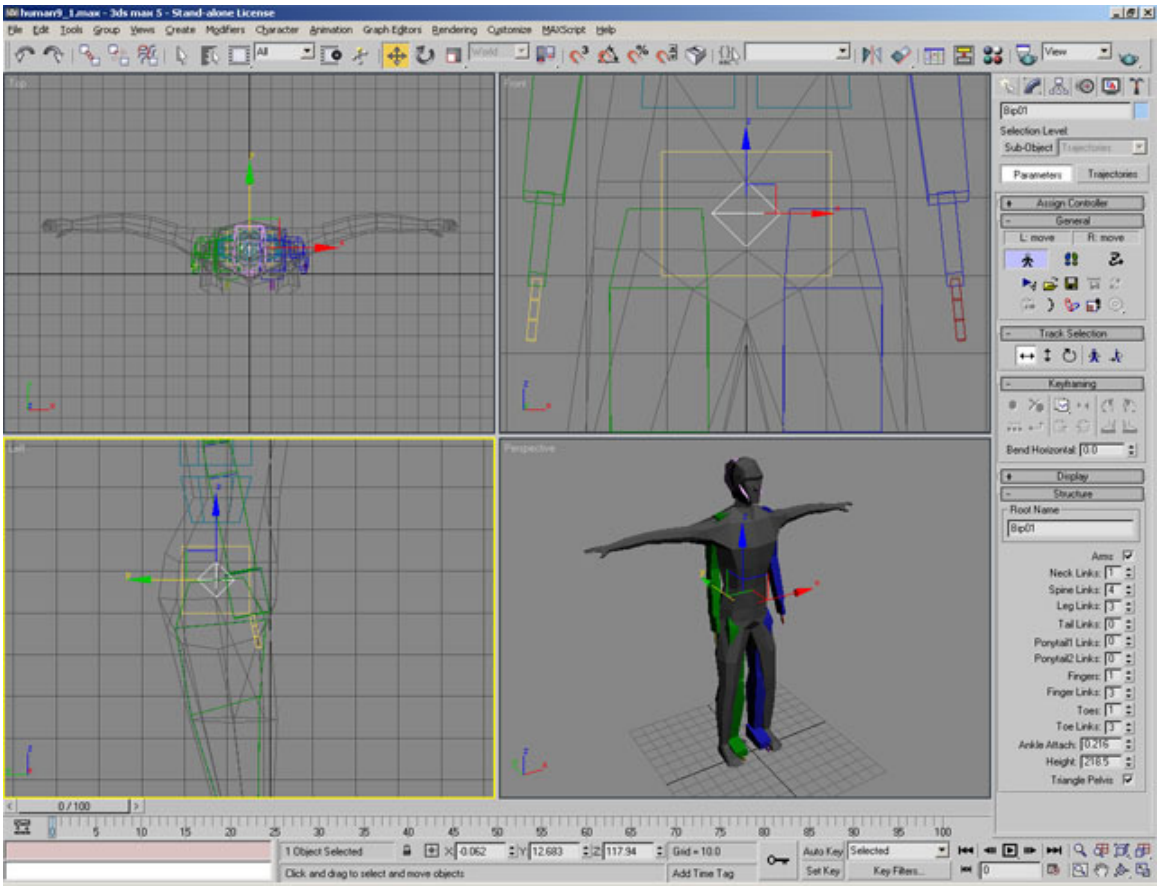
Ensuite, allons dans l'onglet 'Motion' et cochons le bouton 'Figure Mode' :



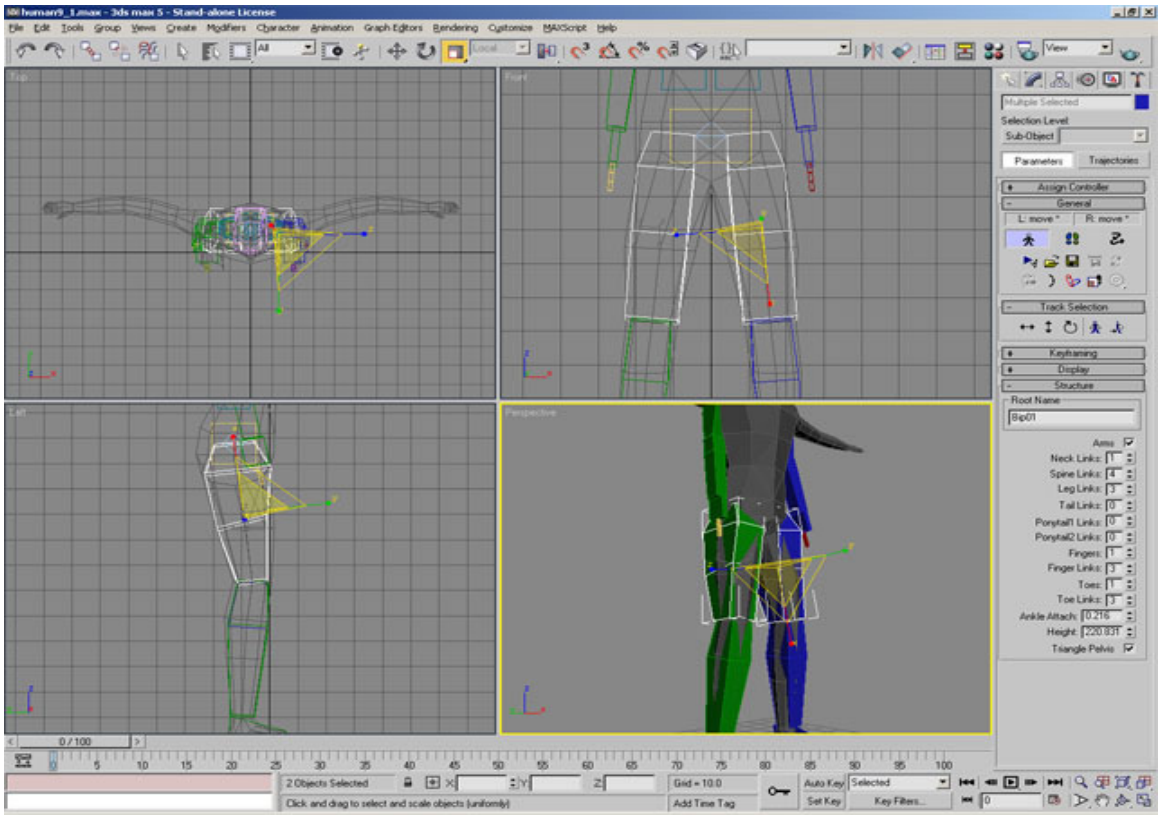
C'est en fait par ce mode que nous allons pouvoir faire tous les "réglages" du biped, y compris le redimensionnement du squelette. Nous pouvons donc maintenant modifier la liste 'Structure' :

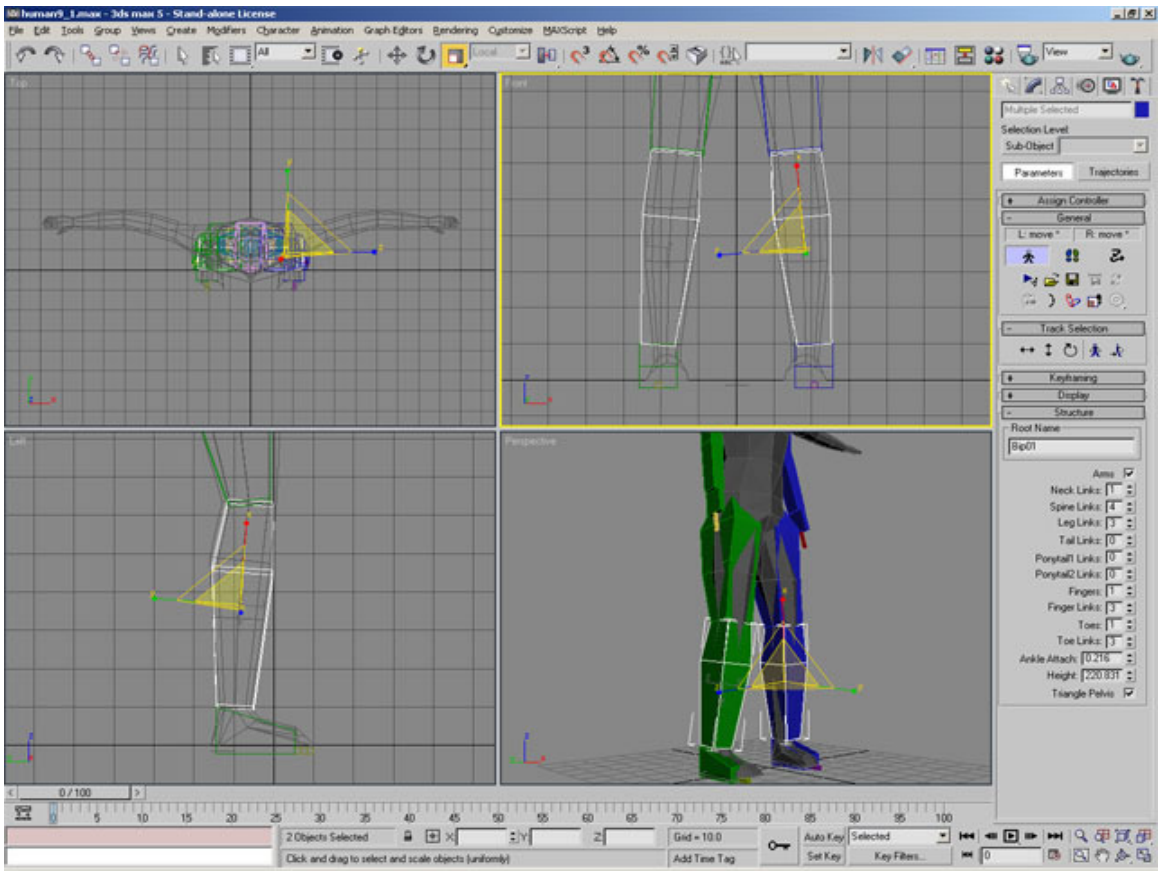


Par exemple, pour les doigts, on va changer la valeur de 'Fingers' ; pour les orteils, c'est 'Toes'. Il faut par la suite placer les éléments du biped en fonction de notre mesh. Déplaçant le comme on le fera avec n'importe quel objet :

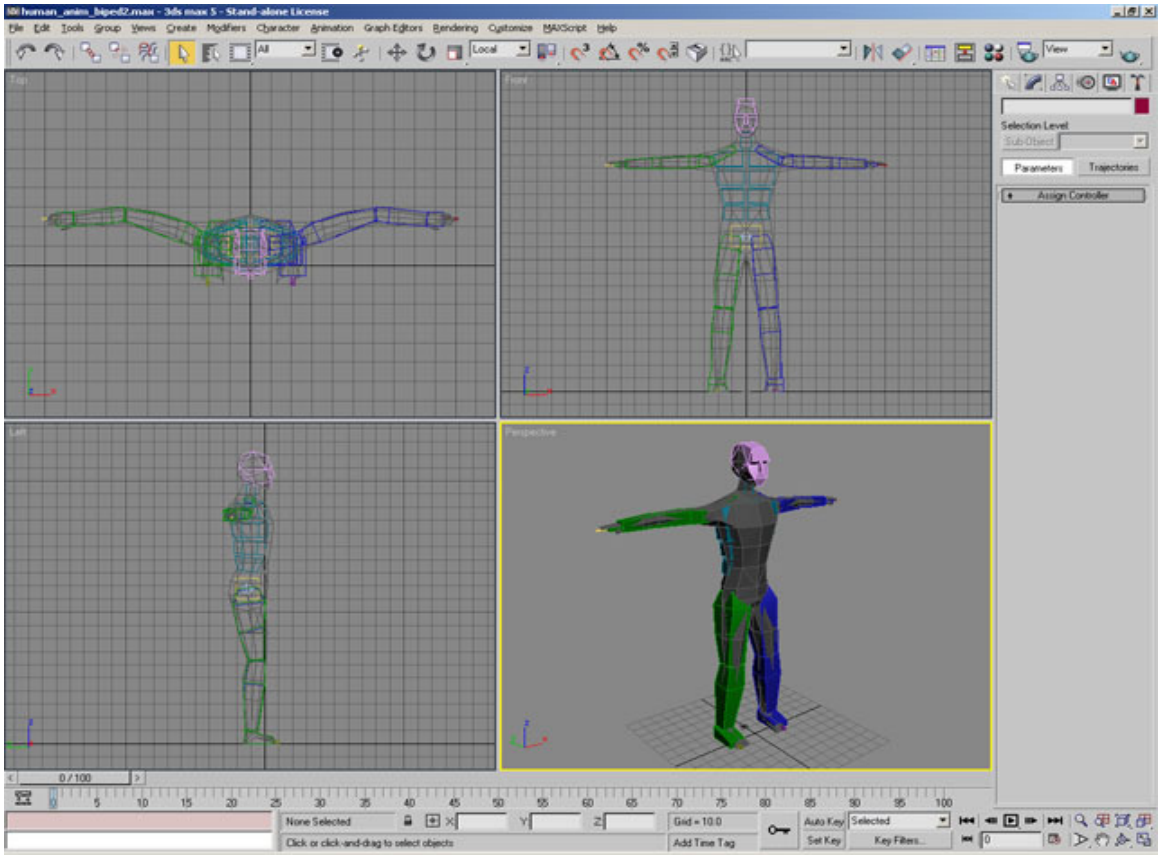


Ensuite, utilisons les outils 'Scale' et 'Rotate' pour placer touts les membres convenablement :





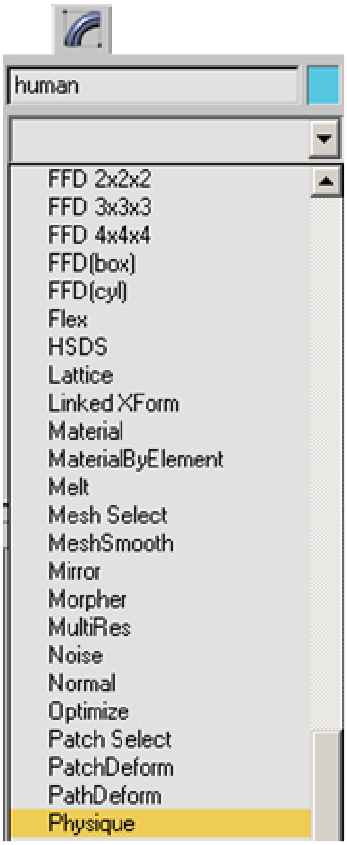
Après avoir réglé tous les membres du biped on arriver à quelque chose qui ressembler à cela :



3.1.3. Ajouter le modificateur Physique

Bien, maintenant, on va reprendre notre mesh. Pour cela, il faut d'abord le débloquer :

clic-droit dans une vue puis 'Unfreeze All'. Le mesh devrait reprendre ses couleurs. Nous allons donc lier le biped au mesh de façon à ce que quand on bouge un membre du biped, une partie du mesh suive ce même mouvement. Sélectionnons notre mesh puis appliquons lui le 'Modifier' intitulé 'Physique' :

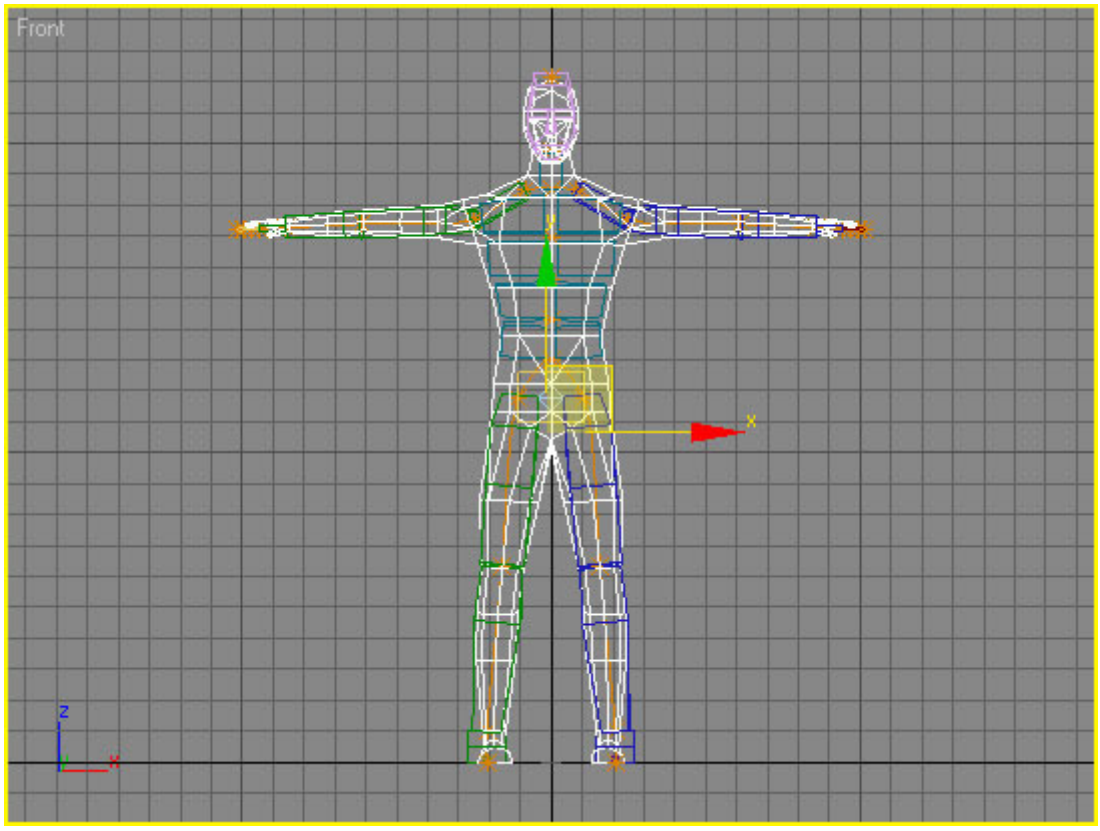


Cochons le bouton 'Attach to Node' (), cliquons sur le bouton 'Select by Name'.

La fenêtre 'Pick Object' s'ouvre. Elle a le même design que celle de la sélection d'objets. Là, sélectionnons 'Bip01' et cliquons sur le bouton 'Pick'.

La fenêtre 'Physique initialization' apparaît. Cliquons sur le bouton 'Initialize'. 3dsmax, En fait, il calcule les vertex qu'il va pouvoir joindre au biped en fonction de la position du biped. C'est pour cette raison qu'il faut autant que possible placer le biped correctement, car cela nous évite de définir manuellement quel vertex sera affecté par le squelette.

Normalement, si nous sélectionnons notre mesh maintenant, nous devrions voir qu'une "ligne" orangée parcourt tout le mesh :



Ensuite, on va tester les mouvements. Oui, car même si 3dsmax calcul en grande partie les vertex affectés par le squelette, il arrive souvent que l'on ait des surprises !

Faisons une sélection par nom, sélectionnons l'objet 'Bip01' et, si ce n'est déjà fait, décochons le bouton 'Figure Mode'.

Maintenant, sélectionnons un des membres du biped, par exemple 'Bip01 R Thigh', et déplaçons le dans une des fenêtres pour tester (Ctrl + Z est là pour annuler ce mouvement).

3.2. Ajouter l'animation

L'étape suivante est l'animation, c'est tout simplement faire bouger le personnage, lui donner vie. La seule contrainte est l'imagination ou le talent de l'animateur en question

Pour cela, il y a un moyen très simple : charger les animations de type capture de mouvement. Ce sont des fichiers ".bip".

Pour avoir une meilleure visibilité de notre mesh, on va cacher tous les objets SAUF le 'Bip01' et notre mesh.

Pour charger l'animation : Sélectionnons le 'Bip01', puis allons dans l'onglet 'Motion'. Si le bouton 'Figure Mode' est coché, décochons-le.

Cliquons ensuite sur le bouton 'Load File' (). Allons chercher un fichier .bip et nous aurons l'animation chargée !

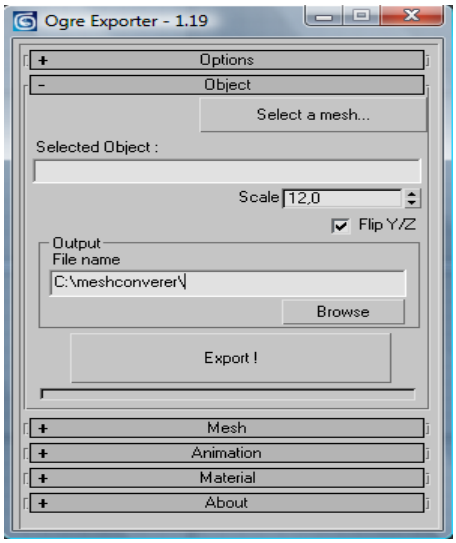
3.3. Exportation

Une fois que tout est fini, que les animations sont terminées, on passe enfin à l'export vers le moteur 3D pour voir le résultat.

Bien que 3D Studio Max n'intègre pas nativement la gestion des maillages supportés par OGRE, il est très facile de convertir les modèles ainsi créés dans le format désiré. Il existe de nombreux plugins qui réalisent cette liaison 3D Studio Max-OGRE, tout en s'intégrant parfaitement dans l'environnement de modélisation.

Nous avons choisi d'utiliser Ogre3DSExporter version 1.2.2.

Cliquons sur le bouton 'Ogre Exporter' ().



Ce petit plugin permet de transformer le modèle animer 3ds max en fichiers binaire pour Ogre 3D mesh, material et skeleton.