الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي و البحث العلمي
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Mohamed Khider Biskra
Faculté des Sciences exactes et des Sciences de la nature et de la vie

Département d'Informatique

Filière :Informatique
Option :Image et vie artificielle

Réf: IVA8/ M2 /2018

Mémoire de fin d'Etudes
En vue de l'obtention du diplôme:

MASTER

Thème

Navigation réaliste en environnement virtuel informé

Présenté par :
Guerfi samia

Soutenu le 25/06/2018

Devant le jury composé de :

Pr.	**CHERIF Foudil**	**Pr**	**Président**
Mr.	**BOUCETTA Mebarek**	**MAA**	**Encadrant**
Mme.	**CHIGHOU1BE Rabia**	**MAA**	**Examinateur**

Année universitaire : 2017 / 2018

Remerciement

Nous tenons à remercier en premier lieu **Dieu** puissant qui nous a donné la force pour dépasser toutes nos années d'étude.

Nous exprimons toute gratitude à notre encadreur

Mr: boucetta mebarek pour ses conseils pendant notre travail de fin d'étude.

Toutes nos expressions de gratitude et de reconnaissance vont vers les membres du jury. Nos remerciements vont également à l'ensemble des enseignants du département d'informatique pour la formation qu'ils nous ont donné durant notre cycle d'étude

Enfin nous remercions, avec toute la suprême sincérité tout ceux qui nous ont aidé de prés ou de loin à terminer et réaliser notre projet de fin d'étude.

Introduction générale

La modélisation d'environnement sous forme de bases géométriques en trois dimensions devient de plus en plus courante. Toute personne, a déjà vu, et ce depuis de nombreuses années, un bâtiment, une ville, une maison ou toutes autres formes architecturales modélisées en 3D, et ce, dans les films d'animations (Pixar, Disney), dans les jeux vidéo ou tout simplement dans les effets spéciaux de films récents. Cette représentation en 3D est une aide précieuse aux métiers du génie civil ou aux génies du septième art.

Ces environnements virtuels sont apparus dans un premier temps dépourvus d'entités mobiles et purement géométriques. Puis sont apparus les premiers personnages virtuels, créant ainsi une pseudo-vie au sein-même de ces environnements. Les premiers personnages n'étaient qu'animation, ils suivaient une trajectoire déterminée au préalable par leurs créateurs respectifs. Au fur et à mesure, ces personnages se sont dotés d'une certaine autonomie, ils étaient désormais capable d'analyser brièvement le monde extérieur afin de choisir la meilleure stratégie à adopter. Puis, dans un souci d'automatisation des procédés d'animation, le problème posé fut de doter les entités dynamiques d'un comportement plausible. Cette automatisation de l'animation a ensuite permis de tendre vers la simulation d'humanoïdes autonomes.

Un environnement informé présente des informations concernant un endroit, des localisations d'objets ou de lieux. L'intérêt de disposer d'informations portées par les objets de l'environnement virtuel a été montré, par M. Kallmann et al.[14] avec la notion de Smart Objects. L'objectif est d'enrichir l'environnement par des annotations sur sa sémantique et de doter les objets d'informations permettant aux agents d'interagir avec eux (positions d'interaction, gestes, actions...), ainsi que de propriétés sémantiques manipulées par les actions des objets tout en ayant des comportements simples. Cette idée renforce l'autonomie des agents et facilitent la planification de leurs actions[19].

Le reste de ce document sera organisé comme suit :

1. Dans le premier chapitre nous présenterons une étude bibliographique du domaine de la navigation réactive, Représentation de l'environnement, Algorithmes de planification de chemin, Navigation réactive et l'environnement informé.

2. Dans le deuxième chapitre l'architecture de notre système sera présentée et la conception de notre solution sera détaillée.
3. Dans le troisième chapitre nous décrirons l'environnement de développement logiciel et nous présenterons les différents résultats de notre système.

Enfin nous terminerons notre étude par une conclusion générale et nous mentionnerons aussi les perspectives de notre travail.

Chapitre 1

Navigation réaliste en environnement virtuel informé

1. Introduction :

Le processus de la planification de chemin consiste à trouver globalement les meilleurs endroits par les quels l'entité doit passer pour atteindre une destination, nécessite une description topologique, permettant de représenter et d'organiser l'environnement de déplacement. L'environnement virtuel doit donc être représenté dans une structure de données appropriée pour avoir une interprétation rapide du point de vue des entités autonomes. Cette description est ensuite exploitée par des algorithmes de recherche de chemin. L'environnement Virtuel Sémantique (EVS) ajoute des informations sémantiques (connaissances sur l'environnement) nécessaires pour rendre possible une interaction plus intelligente entre les agents et leur environnement.

Pour cela, nous décrivons dans ce chapitre les différentes méthodes de représentation de l'environnement ainsi que les différents algorithmes de recherche de chemin capables d'extraire le chemin idéal vers le but même dans le plus complexe des cas. Ensuite, l'aspect navigation sera également traité dans ce chapitre. Enfin, nous parlons sur Les Objets sémantiques

2. Domaines d'application de la navigation réactive :

Les domaines d'application de la navigation réactive sont variés tels que les jeux vidéo, domaines de la médecine et des soins des phobies.

2.1. Jeux vidéo :

Avec l'évolution de la puissance de calcule des GPU, cette évolution permet aux concepteurs de jeux vidéo a concentre sur le comportement des personnages par exemple **Autodesk Gmeware Navigation** Autodesk a beaucoup investi dans la navigation dans les dernières années, la constitution d'une équipe qualifiée, basée à paris. La société a également concentré ses ressources de développement sur la refonte Kynapse, ce qui a entraîné des annonces relativement moins sur le produit (et de nouvelles fonctionnalités) jusqu'à ce que Gameware navigation a été introduite la semaine dernière.

2.2. Robotique :

Un robot est une machine équipée de capacités de perception de décision, et d'action qui lui permettent d'agir de manière autonome dans son environnement en

fonction de la perception. La navigation d'un robot consiste à déterminer une suite de coordonnées que ce dernier doit suivre pour atteindre une destination donnée. Les navigation réactive de robot supprime les problème dus aux différence entre la réalité et le modèle de l'environnement du robot.

2.3. Médecine :

Dans le développement de simulations chirurgicales en environnement immersif nouveau besoins se font sentir en ce qui concerne les outils d'interaction et de visualisation, dans ce cadre les navigations réactive utilisée pour améliorer la navigation dans réseaux vasculaire dense. [18]

3. Représentation de l'environnement :

L'environnement de déplacement des humanoïdes est une scène 3D.la structure de donnée utilisée la plupart du temps, est un ensemble de points placés en trois dimensions. Ces points sont associés pour former les facettes triangulaires qui composent les objets de la scène. Ces données sont très coûteuses à parcourir pour prédire les collisions et planifier un chemin. On va donc découper et simplifier cet espace pour en extraire des caractéristique qui nous intéresserons pour l'animation et plus particulièrement la gestion de la navigation des humanoïdes.

Les techniques de subdivision spatiale s'appliquent directement sur la scène 3D ou bien sur la surface où se situe le déplacement. Dans le dernier cas si l'on conserve les informations relatives à la hauteur on parle de représentation $2D^{\frac{1}{2}}$, la surface où l'on projette est alors un plan déformé, sinon il s'agit d'une représentation 2D et on projette sur un plan.

Nous allons d'abord présenter des techniques de représentation explicite de l'espace navigable, où l'intégralité de l'espace est décomposée en cellules. Puis nous présenterons des représentations implicites de cet espace par des cartes de cheminements.

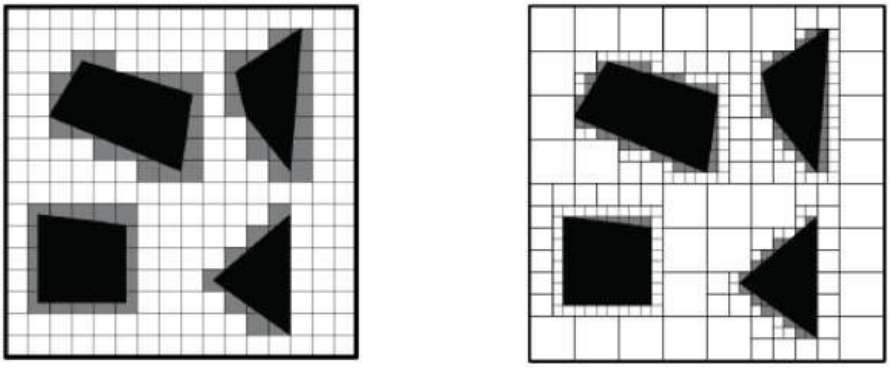
3.1. **Subdivisions spatiales :** nous allons présenter deux familles de subdivisions spatiales : les décompositions approchées et les décompositions exactes.

3.1.1. **décompositions approchées :** il s'agir dans ce cas de découper l'environnement en cellules de forme fixée, la plupart du temps hyper cubiques. Ces cellules

serrant soit totalement navigables, soit totalement obstruées, soit partiellement obstruée.

3.1.1.1. **Grille régulière :** on divise l'environnement avec une grille dont les cellules sont hypecubique et toutes la même dimension, cette méthode est aisée à utiliser car la grille est facile à parcourir et à construire. Les grilles utilisées sont souvent en 2D, en 3D , en $2D^{\frac{1}{2}}$ la décomposition en grilles régulières possède, toutefois, un défaut : la résolution de la grille de s'adapte pas aux détails de l' environnement. Plusieurs cellules pourront décrire de grandes zones vides, le rapport information/nombre de cellules n'est donc pas optimal. C'est pour pallier à ce défaut que les grilles hiérarchiques sont introduites.

3.1.1.2. **Grille hiérarchique :** cette méthode décrit la scène sous la forme d'un arbre, la taille des cellules décroît avec la profondeur dans l'arbre, il existe plusieurs technique de décomposition, nous présenterons ici les quad-tree (2D) et les oct-tree (3D), on ne peut pas exploiter les inégalités de répartition des obstacles dans l'environnement. [1]



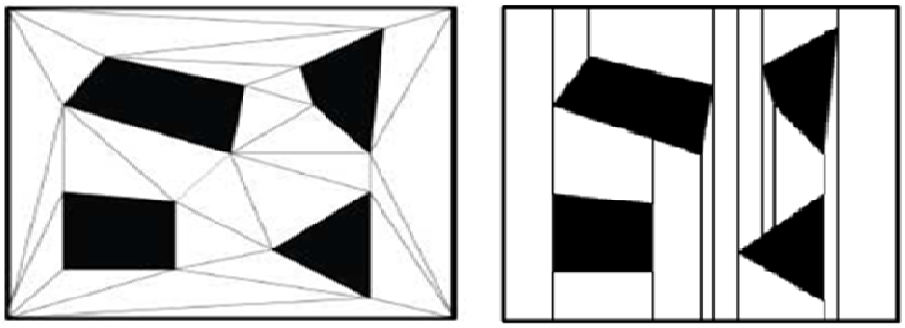
(a) Grille régulière **(b)**Quadtree de profondeur 3

Figure 1.1: Exemple à base de grilles.

3.1.2.**Décompositions exacte :** les méthodes de décomposition exacte ont pour but de représentation exactement l'espace libre, généralement sous la forme de cellules convexes de différentes formes(triangles, polygones , trapèzes…….)

3.1.2.1. **Triangulation de delaunay contrainte en 2d** prend en entrée un ensemble P de points du plan et fournit en sortie un ensemble de triangles dont les sommets sont formée par les points fournis en entrée. Ces triangles respectent la contrainte suivante : le cercle circonscrit au triangle ne contient pas de points de p autre que les sommets de ce même triangle.

3.1.2.2. **Décomposition en trapèzes** cette décomposition permet de décomposer l'environnement sous forme de cellules trapézoïdales. Elle est basée sur un algorithme de balayage. Les points délimitant la géométrie de l'environnement sont triés suivant l'axe des ordonnées. Puis un maximum de deux segments est généré, ayant pour origine le point sélectionné et pour extrémité la prochaine intersection avec le bord d'un polygone délimitant un obstacle.



(a)Décomposition de Delaunay contrainte **(b)**Décomposition de trapèze

Figure 1.2: Représentation exacte de l'environnement avec la triangulation de Delaunay contrainte et de trapèze.

3.2. **Cartes de cheminement** :nous allons présenter deux familles de Cartes de cheminement .

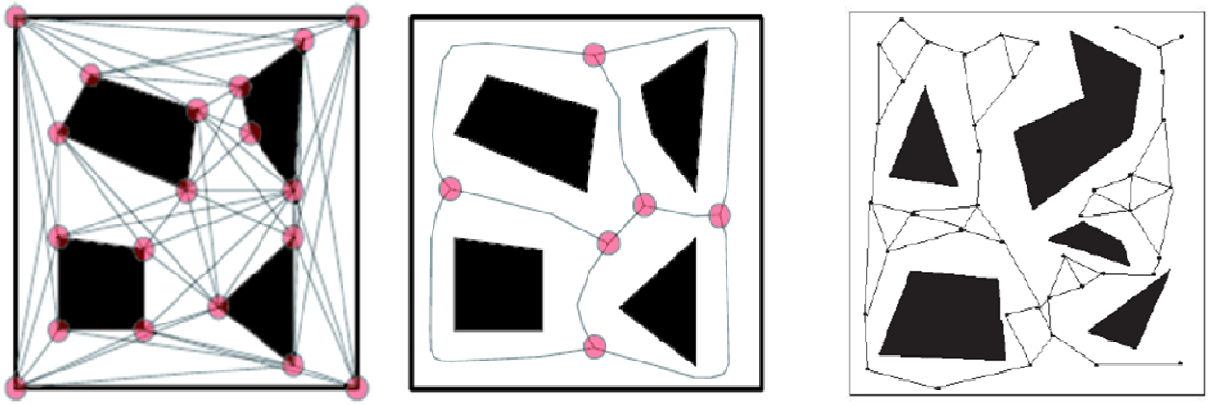
3.2.1.**Cartes de cheminement déterministe :** les méthodes à bases de cartes de cheminement représentent l'espace sous la forme d'un réseau de chemins permettant aux entités de naviguer en évitant les obstacles. Pour ce faire un certain nombre de points clefs sont répartis à l'intérieur de l'environnement et connectés s'il existe un chemin en ligne droite les reliant sans rencontrer d'obstacle. Différentes méthodes, basées sur ce concept existent .elles diffèrent principalement par le mode de génération des points clefs et par la façon de les relier.

3.2.1.1. **Graphe de visibilité** le graphe de visibilité utilise les sommets des polygones décrivant les obstacles de l'environnement, comme points clefs de la carte de cheminement. Par la suite, deux points clefs sont reliés si et seulement si ils sont mutuellement visibles. [2]

3.2.1.2. **Diagramme de voronoï généralisé** le principe de cet algorithme est de créer des chemins équidistants des obstacles de la scène.la plupart du temps on extrait ce diagramme de la projection de la scène en 2D. les sommets du graphe

résultant correspondent aux points d'où partent au moine trois chemins. Les chemins crées de cette manière maximisent la distance aux obstacles. [1]

3.2.2. **Cartes de cheminement probabilistes** les points qui seront les sommets du graphe sont générés aléatoirement. Deux points sont ensuite reliés si il existe un chemin en en ligne droite n'intersecté pas d'obstacle entre eux. Dans le cas générale, cette opération peut s'avérer très coûteuse mais permet de prendre en compte des mouvements très complexes.[1]

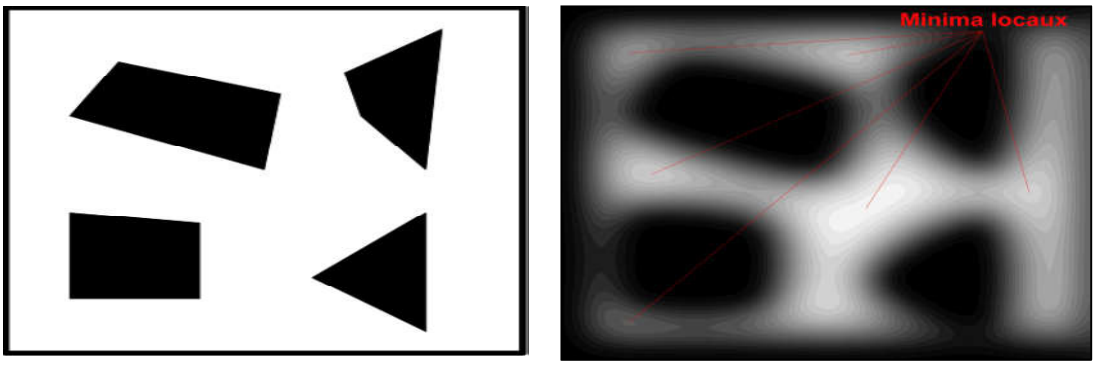


(a) Graphe de visibilité **(b)**Diagramme de *Vorono* **(c)** Cartes de cheminement probabiliste

Figure 1.3 : Environnement représenté par le graphe de visibilité et Diagramme de *Voronoi*, Cartes de cheminement probabilistes

3.3. Champs de potentiel

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. 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 1.4). La navigation d'une entité est alors simulée par une descente de gradient depuis sa position dans l'environnement.



(a) Environnement d'origine **(b)** Champs de potentiels

Figure 1.4: 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 1.4 (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) [11], 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'autres méthodes existent à base de tirage aléatoire de direction à suivre [12], 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 de hors 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 [13], 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.

3.4. Représentation multi-couches

Dans [20], les auteurs proposent de représenter les environnements ayant des structures topologiques complexes par un modèle multicouche subdivisé en trois couches (Figure 1.5)

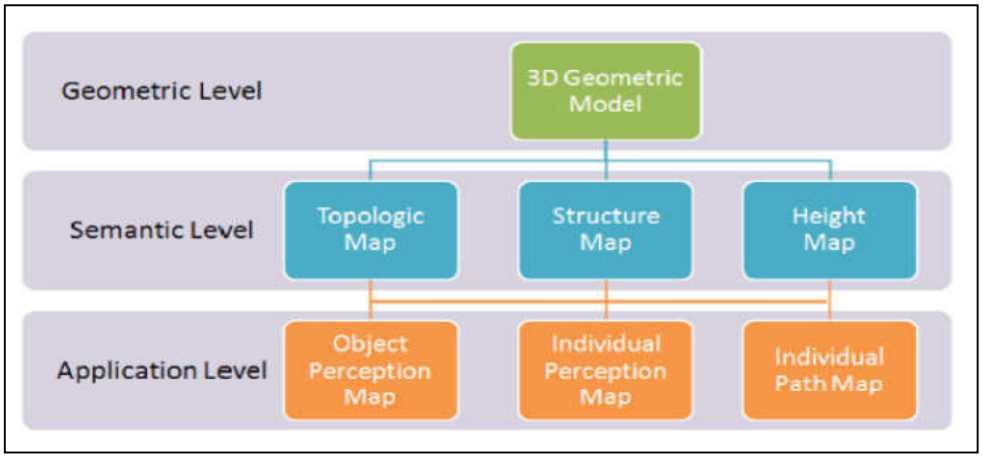


Figure 1.5 : Trois niveaux hiérarchiques «géométrique, sémantique, application».

a- Niveau géométrique

Ce niveau se base sur un modèle géométrique en **3D** de l'environnement, lequel, est utilisé pour l'affichage et l'extraction des informations sémantiques.

b- Niveau sémantique

La couche sémantique est représentée par :

- des cartes de structures, où tous les objets placés dans la même région sont inclus dans un bloc. Tous les blocs ainsi que leurs objets sont organisés en cartes de structures,
- des cartes topologiques qui sont utilisés pour stoker et récupérer les relations entre les régions,
- des cartes de hauteurs qui sont utilisées pour stocker l'information concernant les élévations de la surface.

c- Niveau application : Afin de permettre une interaction entre les piétons et l'environnement, l'information récupérée au niveau sémantique permet de générer des cartes de haut niveau telles que :

- la carte de perception des objets dans laquelle les blocs sont divisés en sous régions en utilisant une grille de cellules uniformes ; le but est de limiter la recherche des objets à une liste de cellules au lieu de tout le bloc,
- la carte de perception individuelle, où, comme pour la carte de perception des objets, une grille de cellules uniformes est associée à chaque bloc afin de stoker les informations concernant tous les individus se trouvant dans le bloc,
- le plan du parcours individuel, où chaque agent a un objectif initial. La carte topologique à base de blocs est utilisée pour générer une carte de voies individuelles.

Dans cette partie, nous avons présenté différentes manières d'extraire des informations de la géométrie de l'environnement, les structures de données obtenues vont nous permettre d'accélérer les traitements requis pour gérer les déplacements humanoïdes qui devront s'effectuer en temps interactif. Concernant les subdivisions spatiales, les décompositions en grilles présentent l'avantage d'être rapide à construire et facile à parcourir, dans les cas des grilles hiérarchiques, elles permettent un discrétisation de l'espace aussi fine que l'on souhaite au prix de l'espace en mémoire. Les décompositions exactes permettent de ne pas perdre

d'information relative à l'espace de navigation. Les cartes de cheminements donnent de l'information directement utilisable pour planifier les chemins mais supposent que l'humanoïde n'interagira pas avec des éléments de l'environnement considérés comme des obstacles.

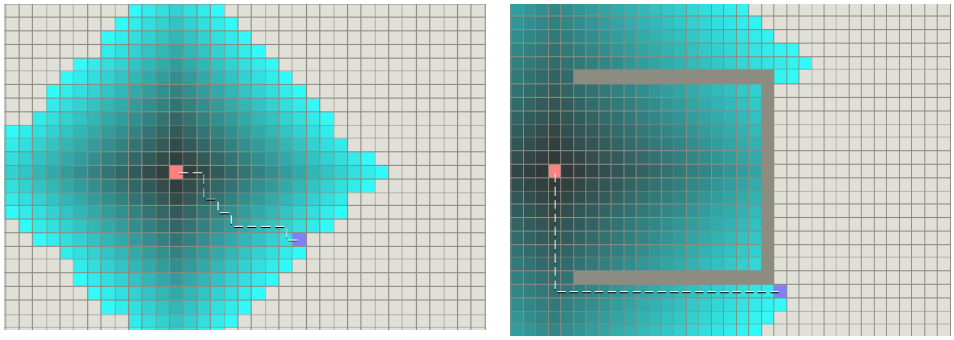
En résumé, les méthodes de représentation de l'environnement se classent en quatre catégories :

La méthode de représentation			Points forts	Limites
Explicite	Décomposition en cellules	Représentations approximatives Grilles, Quadtree	Rapide à construire et facile à parcourir	La résolution de la grille ne s'adapte pas aux détails de l'environnement. Les obstacles doivent être alignés sur les axes sinon on obtient une décomposition en escalier.
		Représentations exactes Cellules convexes	Exprimer toute l'information relative à l'espace de navigation.	les environnements à géométrie complexe, le nombre de triangles obtenus augmente très rapidement.
Implicite	Cartes de cheminement	Graphe de visibilité	Donnent des informations directement utilisables pour planifier le chemin	bien adaptées à un processus de planification de chemin général
		Diagramme de Voronoï généralisé		
		Cartes de cheminement probabilistes		
	Champs de potentiels		Simples et efficaces	le problème des minima locaux
	Représentation multi-couches	Niveau géométrique	Représentation des environnements ayant des structures topologiques complexes	Utilisation beaucoup d'informations et plusieurs cartes
		Niveau sémantique		
		Niveau application		

Tableau1.1 : Etude comparative entre les méthodes de représentation de l'environnement.

4. Algorithme de planification

Diverses formes d'algorithme de calcul sur les graphes peuvent être utilisées pour effectuer un calcul de plus court chemin. Parmi celles-ci l'algorithme de dijkstra permet de trouver l'ensemble des meilleurs chemins issus d'un sommet et permettant d'atteindre les autre sommets du graphe. Schématiquement, cet algorithme stocke pour chaque nœud exploré sa distance par rapport à l'origine et son prédécesseur dans le chemin ayant permis l'obtention de cette distance. L'algorithme commence par le nœud d'origine en explorant successivement tous les successeurs tout en privilégiant ceux étant les plus proches de l'origine. Cet algorithme travaille uniquement sur la structure du graphe sans utiliser d'informations supplémentaires. De fait, pour trouver le plus court chemin d'un point a à un point b, sachant que la longueur réelle du chemin de a à b est d, il va explorer tous les nœuds d'une distance inférieure à d avant de pouvoir fournir un chemin.

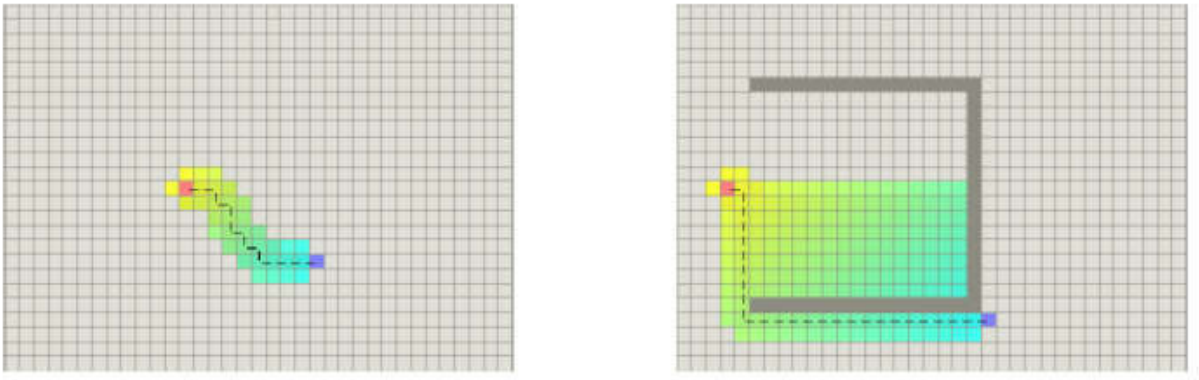


(a) Environnement non contraint (b) Environnement avec obstacle concave

Figure 1.6: Fonctionnement de l'algorithme de Dijkstra dans des cas contraints ou non .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.

Dans la mesure où dans les problèmes de planification de chemin, il est possible d'exploiter des propriétés sur l'environnement, l'algorithme A* lui est souvent préféré. La démarche est légèrement différente, chaque nœud exploré se voit attribuer une valeur correspondant à l'estimation du coût total du chemin passant par ce nœud. Cette valeur est calculée en fonction de la distance réelle par rapport à l'origine plus une estimation (fournie par une heuristique) de sa distance au but. La valeur $v(n_k)$ associée au nœud n_k a alors la valeur suivante : $v(n_k)=d(n_k,o)+h(n_k,b)$ où $d(n_k,o)$ représente la

distance réellement calculée entre l'origine et le sommet $n_k$et $h(n_k,b)$ la distance estimée (heuristique) du but . L'algorithme s'initialise avec le nœud de départ et stocke dans une queue triée l'ensemble des nœuds successeurs .L'algorithme fonctionne ensuite en gardant une queue triée de nœuds (suivant l'ordre décroissant de la distance estimée au but), que l'on peut qualifier d'ouverts car ils n'ont pas encore été explorés, explorant systématiquement le nœud promettant d'obtenir le plus court chemin. La puissance de convergence de cet algorithme dépend de la qualité de l'heuristique h. dans le cas de la recherche du plus court chemin dans un environnement, cette heuristique est le plus souvent une estimation de la distance. Un certain nombre de variantes existent par rapport à l'algorithme A* . parmi celles-ci l'algorithme ABC (A* with bounded cost) est proposé par logan[7]. Il s'agit d'une généralisation de l'algorithme A* permettant d'ajouter des contraintes molles à respecter (contraintes de limitation de temps et d'énergie par exemple) lors de la planification.



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

Figure 1.7 : Fonctionnement de l'algorithme de A*

L'algorithme IDA* (iteratives-deepening A*) utilise une approche différente en effectuant une recherche en profondeur d'abord, supervisée par une heuristique [8]. Dans un premier temps une borne de longueur de chemin B est initialisée avec la distance estimée au but. L'exploration commence par le nœud d'origine du chemin et explore successivement tous les successeurs, ainsi que les successeurs des successeurs…, de ce nœud. cette recherche s'arrête dans deux cas :soit le chemin est trouvé, dans ce cas il s'agit du chemin optimal, soit un nœud est trouvé tel que sa distance réelle depuis l'origine sommée à sa distance estimée au but soit supérieure à B.

dans le cas où aucun chemin n'est trouvé durant une exploration, la borne B est remise à jour avec la plus petite estimation de la distance au but trouvée au cours des explorations. Certaines améliorations en terme de temps d'exécution peuvent être obtenues en utilisant un cache de taille fixe des estimations déjà calculées pour les nœuds précédemment explorés [6]. Ces estimations permettent un élagage plus rapide de l'arbre de recherche. [2].

En règle générale, l'algorithme de dijkstra est peu utilisé dans le cadre de la recherche de chemin, car il s'agit d'une domaine dans lequel il est souvent possible d'utiliser une heuristique qui réduit grandement le coût de la recherche. le choix entre l'algorithme A* et l'algorithme IDA* est plus difficile. L'algorithme IDA* est préféré à l'algorithme A* dans le cas de domaines de taille exponentielle [1] car la taille de la liste triée conservée par l'algorithme A* devient elle-même exponentielle [6]. Cependant, dans le cadre de la planification de chemin, cette considération n'a pas vraiment de signification si l'on considère que l'on dispose déjà du graphe en mémoire. D'un point de vue théorique, ces deux algorithmes possèdent des complexités équivalentes mais l'algorithme IDA* nécessite moins de mémoire. [2].

5. Navigation réactive

L'une des aptitudes que possède l'être humain et bon nombre d'animaux est la capacité à se déplacer au sein de leur environnement. Ce déplacement plus communément appelé navigation chez l'homme est régi par des codes, tels que le fait de prendre en compte géométriquement son environnement. Cette prise en compte permet ainsi à l'homme de s'adapter aux différentes contraintes que lui impose son environnement. L'humanoïde doit donc posséder cette faculté d'analyse et de décision vis à vis du but qui lui a été fixé. L'aspect extérieur de sa navigation doit également paraitre plausible aux yeux de tous.

La navigation a pour but de considérer les entités simulées de façon indépendante. Certains algorithmes de navigation vont plus se baser sur l'aspect réglé du mouvement, autrement dit, ils utilisent un système à base de règles. D'autres utilisent des modèles plus orientés vers la notion de force comme le modèle HiDAC [21], modèles tendant à représenter les humanoïdes plus sous forme d'animation de particules que de mouvement

humain. Enfin, d'autres algorithmes sont basés sur la géométrie prenant en compte la direction et la vitesse des entités.

5.1.Modèle à base de règles

Craig Reynolds [22], avec son modèle the boids model of flocks, introduit la notion de simulation microscopique à base de règles. Ici, le déplacement de chaque individu est régit par des règles de comportement de la forme « si condition alors action ». Reynolds propose trois règles (Figure 1.8) :

– Séparation afin d'éviter d'éventuelles collisions avec ses voisins.

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

– Cohésion afin de rester proche de ses voisins.

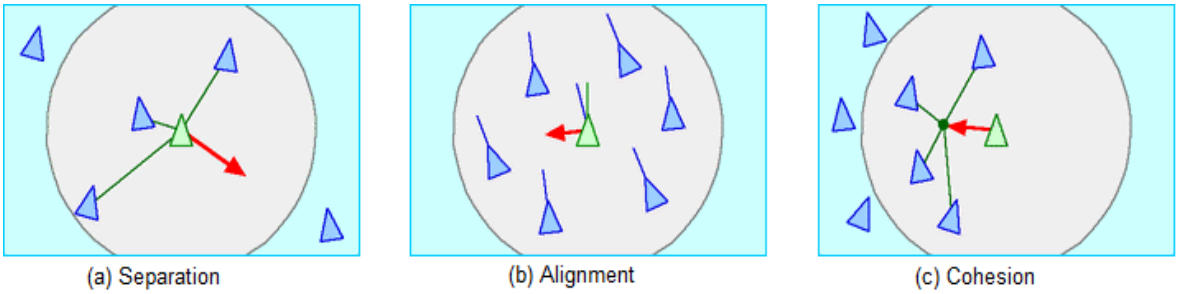


Figure 1.8 : Règles comportementales du modèle Flocks of Boids

Craig Reynolds [23], a étendu son modèle pour la navigation des agents autonomes.

Il a ajouté d'autres règles comme:

- Pursue and Evade (Poursuivre et Echapper)
- Wander (Déambuler)
- Arrival (Arriver)
- Obstacle Avoidance (Evitement d'obstacle)
- Leader following (suivi de leader)
- Path Following (suivi de chemin)

La navigation se fait par la combinaison de ces règles de comportement de base.

5.2.Modèle à base de particules

Ce type de modèle est l'un des plus utilisés dans les modèles de situation de panique. C'est un modèle qui fait l'analogie entre le déplacement d'individus en forte densité n'ayant quasiment pas de liberté pour se déplacer et l'écoulement de particules entre des compartiments. Dans ce type de modélisation, on peut voir apparaitre(figure 1.9) un phénomène d'agglutination quand les entités modélisées se dirigent vers une issue unique. Au final, le débit y devient quasi nul. Les piétons sont ainsi assimilables à des particules soumises à des forces venant de l'environnement (autres piétons et obstacles). Pour se représenter plus concrètement ce type de modèle, il suffit de s'imaginer un sac rempli de billes percé au fond. Au début, quelques billes tombent à cause de la gravité puis s'arrêtent car elles sont bloquées par la force qu'elles s'exercent mutuellement.

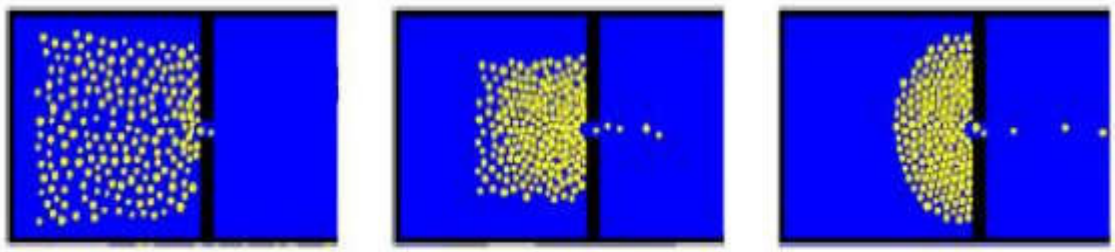


Figure 1.9: Phénomène d'agglutination dans le modèle de particules de *D.Helbing* [24]

5.3.Modèle prédictif

Feurtey [25] propose une approche prédictive de l'évitement de collision. Pour cela, il représente, dans unrepère (x, y, t), l'environnement au sein duquel les entités naviguent. Dans ce type de repère, le cône (Figure 1.10)représente l'ensemble des déplacements possibles pour une entité donnée. La pente de ce cône représente donc la vitesse maximale que peut atteindre cette entité et le sommet sa position actuelle. Une collision est représentéedans le cône sous la forme d'un segment, dans le cas où la trajectoire d'une entité voisine l'intersecte.

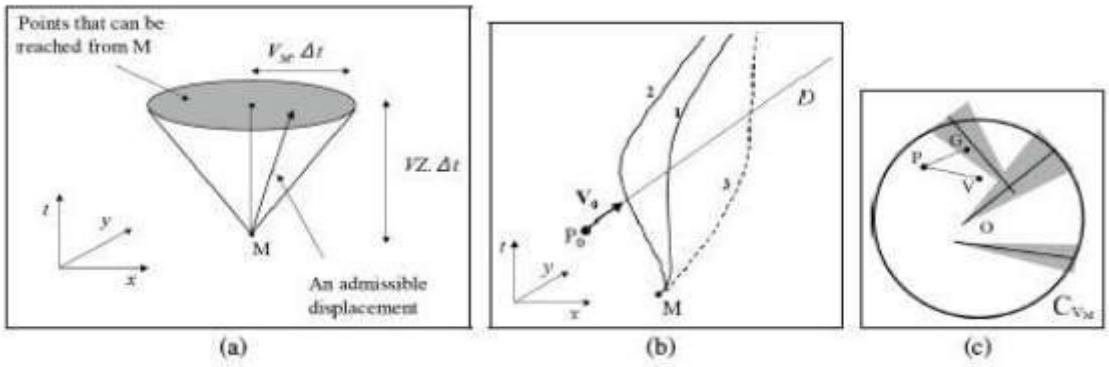


Figure 1.10 : Modèle prédictif de navigation de *Feurtey*.

Feurtey est parti du principe qu'une personne souhaitant éviter une collision applique trois règles : préservation de sa direction, de sa vitesse et du temps nécessaire au déplacement [26].

F.Lamarche et *S.Donikian* [27] proposent un modèle prédictif à base de règles d'évitement et d'optimisation. La subdivision spatiale utilisée est une triangulation de *Delaunay* filtrée. De cette subdivision est extraite une planification de chemin. Celle-ci est analysée afin de trouver une vitesse idéale à adopter. Cette vitesse est ensuite filtrée par l'espace personnel, qui est une règle sociale définissant une distance minimale entre la navigation des autres humanoïdes et entre les obstacles. Ensuite, afin d'éviter des collisions, la vitesse est modifiée. Un module de sécurité veille tout de même à ce que la vitesse prenne en compte l'inertie de l'humanoïde.

6. Environnement informé :

Un environnement informé présente des informations concernant un endroit, des localisations d'objets ou de lieux. L'intérêt de disposer d'informations portées par les objets de l'environnement virtuel a été montré. L'objectif est d'enrichir l'environnement par des annotations sur sa sémantique et de doter les objets d'informations permettant aux agents d'interagir avec eux (positions d'interaction, gestes, actions...), ainsi que de propriétés sémantiques manipulées par les actions des objets tout en ayant des comportements simples.

Cette idée renforce l'autonomie des agents et facilitent la planification de leurs actions.

6.1. Les Objets sémantiques

L'ensemble des travaux présentés ci-après proposent d'enrichir la description des objets par de l'information sémantique afin de faciliter les interactions entre les agents et leur environnement.

6.1.1. Smart Object

Les Environnements virtuels informés utilisent les **Smarts Objects** [14] pour interagir avec les entités de l'environnement. Kallmann & Thalmann proposent l'idée d'inclure dans la description de l'objet toutes les informations nécessaires pour décrire la façon d'interagir avec lui. Ainsi, nous pouvons distinguer des informations sur : (i) les propriétés de l'objet (sémantiques et physiques), (ii) comment interagir avec l'objet (actions, positions, gestes), (iii) le comportement de l'objet en réponse à une action et (iv) le comportement des agents pour réaliser l'interaction. Par exemple, la poignée d'une porte sert pour ouvrir ou fermer la porte, cette partie de la porte a une indication sur l'endroit où l'utilisateur doit mettre la main afin d'interagir avec cette partie. Cette indication définie la forme de la main et l'emplacement prévu afin de procéder à une action. Malheureusement, les smarts objects, tel qu'ils sont présentés par Kallmann, sont strictement destinés à l'animation et ne permettent pas de faire du raisonnement de haut-niveau tel que pour allouer aux agents la capacité d'adapter leurs comportements aux conséquences de leurs actions.

Les Smarts Objects ont été principalement utilisés pour l'animation du comportement des agents. Abaci et al. [17] ont proposé d'étendre cette solution en permettant aux agents de faire un raisonnement de haut-niveau sur la description sémantique des objets (obtenue des smarts objects) afin de planifier leur actions en tenant compte de leurs conséquences.

Les informations sémantiques ajoutées aux smarts objects sont décrites d'une manière formelle. Elles sont représentées par des règles en utilisant la logique du premier ordre.

Les interactions sont décrites par les conditions nécessaires pour leur application et leurs effets sur l'état de l'agent et/ou de l'objet, si l'action est effectuée.

6.1.2. Star Fish

Dans [15], les auteurs proposent des objets synoptiques **STARFISH** en suivent la même philosophie que celle des *Smarts Objects*. L'idée est de définir un ensemble minimal d'actions primitives de base qui sont utilisées pour construire des actions complexes. Ces actions sont associées à une interface interactive qui est la partie géométrique de l'objet concernée par l'action. Les agents n'auront plus qu'à interagir avec ces interfaces pour récupérer l'information relative à l'objet qu'ils veulent utiliser et adapter leur comportement en conséquence.

La combinaison des informations de haut-niveau ainsi que les données d'animation dans les smarts objects, permet de faire une planification plus efficace. Cette combinaison a l'avantage de simplifier le processus de conception. Ainsi, les opérateurs de planification doivent être conçus en même temps que les scripts d'interaction, ce qui contribue à assurer la cohérence entre la sémantique formelle de l'action et sa mise en oeuvre réelle. Les *Smarts object* sont utilisés pour l'animation ils sont très intéressants pour des applications comme les jeux vidéos où ils ont été utilisés dans le jeu The SimsTM [16]. [5]

7. Conclusion

La représentation de l'environnement joue un rôle important, d'une part, pour la planification de chemin et, d'autre part, pour la détection de collision. Nous nous sommes focalisés dans ce chapitre sur cette représentation de l'environnement virtuel en exposant les techniques permettant sa représentation et en résumant les différents algorithmes utilisés pour la planification de chemin et la navigation en environnement virtuel. Les environnements virtuels sémantiques (EVS) sont utilisés pour enrichir le domaine d'information par des données sémantiques qui ne peuvent pas être déduites de la géométrie de l'environnement

Chapitre 2

Conception de système

1. Introduction

Dans ce chapitre nous présentons notre modèle proposé pour la représentation d'environnement, notre principal objectif est de clarifier le cadre formel permettant de planifier un chemin qui permettant à une entité virtuelle de déplacer au sein de son environnement virtuel en prenant en compte la topologie de l'environnement et ses obstacles et permettre à l'entité virtuel d'accomplir un comportement de navigation crédible similaire à celui de l'humain réel.

Dans ce travail nous nous intéressons à l'ajout des informations à propos des objets constituant la scène à la représentation de l'environnement. La prise en compte de ces informations dans le comportement de navigation des humanoïdes permettra d'augmenter sa crédibilité. Pour formuler le problème de navigation de manière objective, nous proposons une méthode à la fois efficaces pour modéliser la topologie de l'environnement et ensuite nous choisi2ssons un algorithme afin de trouver un chemin adéquat, et faire doter l'entité virtuel des capacités qui sont nécessaires pour faire suivre le chemin trouvé en préservant le réalisme de simulation. Pour pouvoir construire tel système, on doit passer par plusieurs étapes qui sont définis par le cycle de développement d'un logiciel. La première étape dans le cycle de développement d'un logiciel est l'analyse des besoins. Son but global est de préciser les services et les objectifs principaux qui seront réalisé et rendus par le logiciel à l'utilisateur. Cette phase est suivie par une phase cruciale qui est la phase de conception. Dans n'importe quel système, la conception est l'étape la plus importante,

La phase de conception est réalisée par un processus itératifs pour déterminer les divers composants et modules du système et leurs interactions. Une bonne conception est la clé de développement d'un logiciel efficace. Un système bien conçu est facile à réaliser, à maintenir, facile à comprendre. La dernière phase consiste en une réalisation, lors de cette étape on réalise un ensemble d'unités de programme, écrites dans un langage de programmation précis.

2. Objectifs

La planification et la navigation en environnement virtuel nécessite la prise en compte de la topologie de l'environnement. Le processus de navigation est sans aucun doute l'un des comportements élémentaires les plus indispensables pour un individu. Ce processus consiste à simuler de façon indépendante les entités dans un environnement virtuel avec leur propre processus de décision, leurs adaptations avec les variations de leurs environnements, ainsi que les interactions liées au déplacement pédestre.

Les principaux objectifs auxquels doit répondre notre application sont :

- Représentation d'un environnement de déplacement par la méthode de triangulation de *Delaunay*, avec une configuration (espace navigable augmenter d'informations sur les objets, espace non navigable, point de départ, point destination),
- planification de chemin en utilisant l'algorithme A*,
- navigation de l'entité virtuelle,
- l'interaction avec les objets.
- le rendu en utilisant Ogre 3D.

3. Conception générale de notre système

Notre système est résumé par le schéma de la figure 2.1. La première étape consiste à représenter l'environnement sous forme d'un maillage triangulaire (triangulation de *Delaunay*)augmenté d'informations sur l'état des objets pour générer un mesh de navigation. Sur ce dernier repose l'élaboration de la planification du chemin en appliquant l'algorithme A*. Pour rendre le chemin obtenu plus réaliste, on are cours à l'utilisation de l'algorithme « funnel » qui se base sur la liste des triangles fournie par l'algorithme A*. L'étape suivante se résume en la réalisation de la navigation de l'humain virtuel et faire l'interaction avec les objets de la scène. Le rendu de la scène est réalisé par le moteur du rendu « Ogre 3 D».

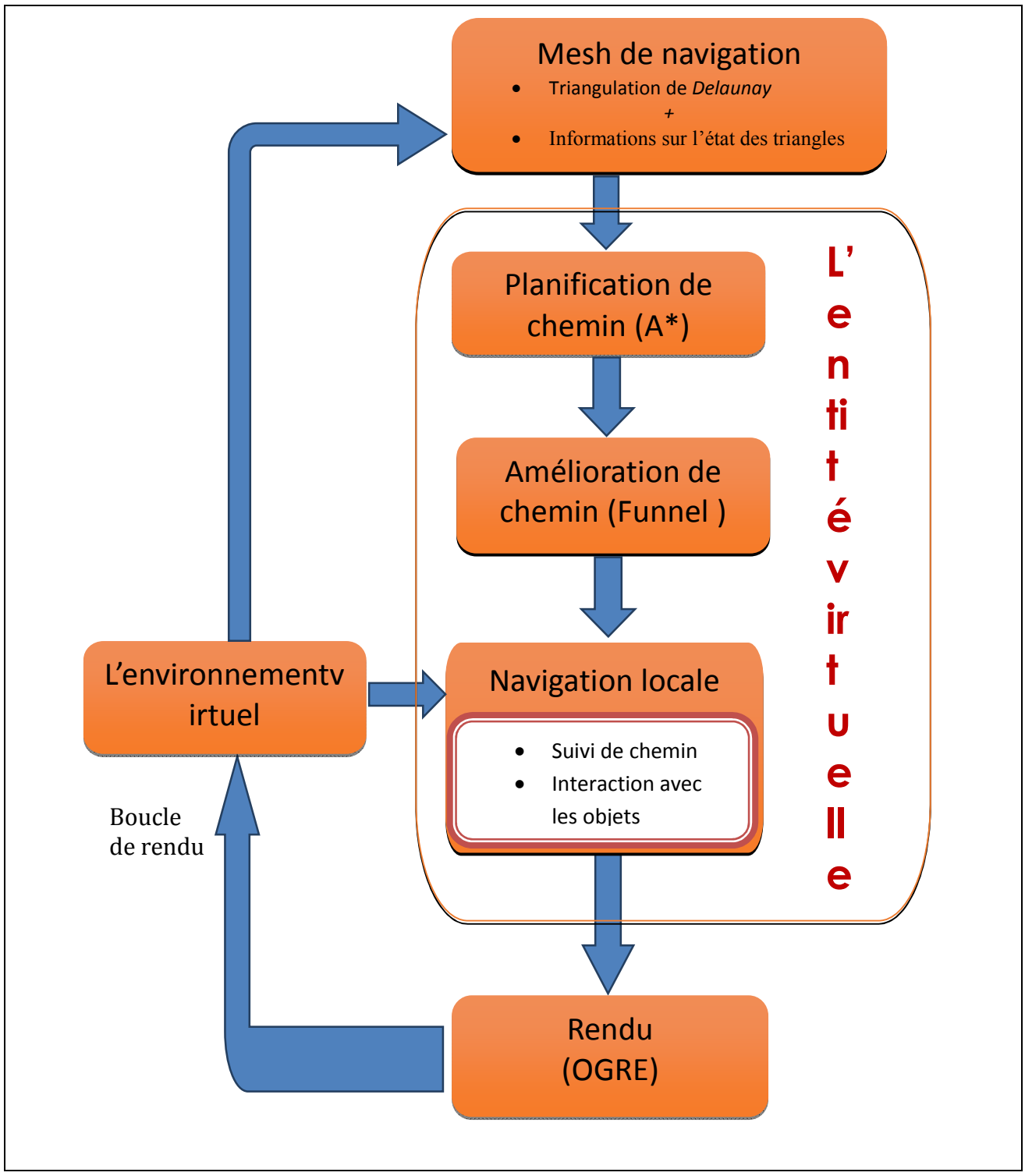


Figure 2.1 : Conception générale du système.

4. Conception détaillée de notre système

Notre système est composé des modules suivants.

4.1. Représentation de l'objet

Chaque objet de la scène est représenté par les propriétés suivantes :

Type : le type d'objet soit un obstacle, objet déplaçable ou bien navigable

Cout : cette priorité représente la difficulté imposée par l'objet sur la navigation de l'entité (objet déplaçable ou navigable).

Code : exprimer comment l'interaction se fera entre l'objet et l'entité (l'action à accomplir si l'entité rencontre l'objet)

Index : simplifier l'accès à l'objet

Une propriété supplémentaire pour certain objets :

Etat : l'état actuel de l'objet

Exemple :

La porte : état ouverte ou bien fermer.

Le feu de circulation : couleur rouge ou vert.

4.2. Représentation de l'environnement de navigation

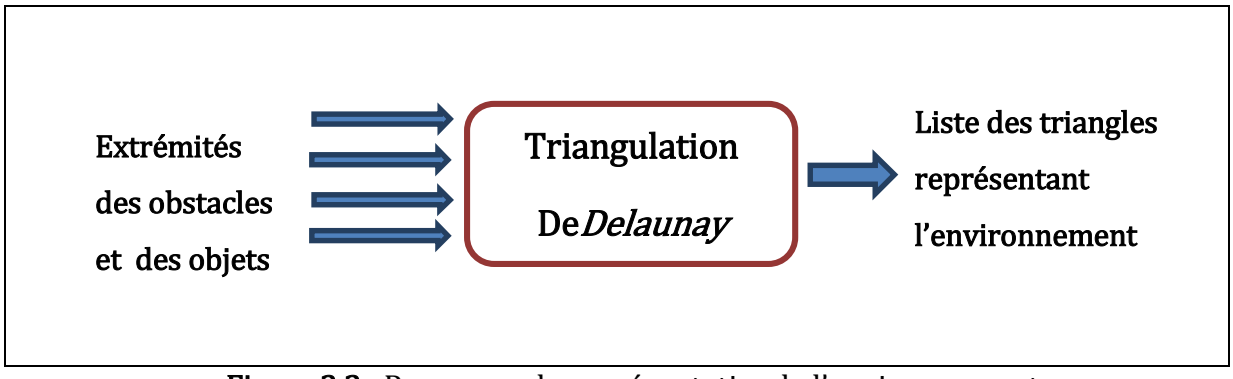


Figure 2.2 : Processus de représentationde l'environnement.

Pour la représentation de notre environnement, nous avons choisi une méthode qui reproduit exactement l'environnement d'origine tout en l'organisant de manière plus accessible. Cette méthode est décrite ci-dessous.

I. Triangulation de *Delaunay*

Il existe un très grand nombre de méthodes théoriques et pratiques pour générer une triangulation à partir d'un ensemble de points. Mais toutes ne sont pas exploitables. En effet, on cherche à construire un ensemble de triangles qui soit arrangé de manière assez uniforme et régulière [28].Par exemple, les triangles générés doivent être les moins allongés possibles et former un ensemble homogène. Dans le cas contraire, les calculs effectués par la méthode des éléments finis seraient moins représentatifs et les résultats moins proches de la réalité.

Le mathématicien russe *BorisDelaunay* (1890 – 1980) a énoncé pour la première fois la définition de la triangulation qui porte son nom en 1934 dans son œuvre «Sur la sphère vide» [29].La triangulation de *Delaunay* impose une contrainte : pour chaque triangle du maillage, son cercle circonscrit ne doit contenir aucun élément de l'ensemble des sommets comme le montre la figure suivante:

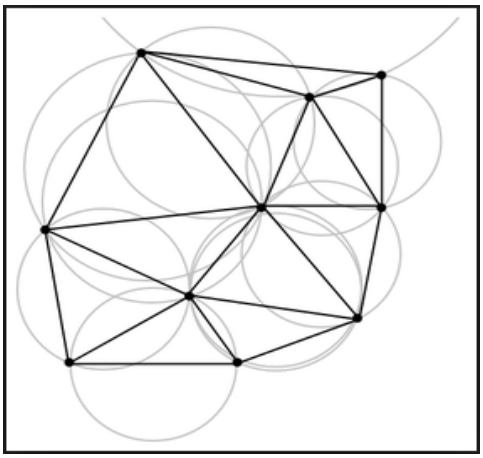


Figure 2.3: Représentation de la propriété de *Delaunay*

Cela implique une construction assez naturelle du maillage et « arrange » les triangles pour uniformiser leurs dimensions. Les seuls triangles plats qui peuvent advenir se situent aux frontières de l'ensemble de points considéré.

La triangulation de *Delaunay* possède des propriétés intéressantes pour construire un algorithme de génération de maillage [30]. On se restreint ici aux propriétés de la triangulation dans un espace à deux dimensions.

Soit n le nombre de points :

• la triangulation est unique s'il n'y a pas trois points alignés ou quatre points cocycliques,

• la triangulation contient au plus n simplexes,

• chaque sommet est connecté en moyenne à 6 triangles,

• l'union des simplexes de la triangulation forme l'enveloppe convexe de l'ensemble des points.

Il existe deux types d'algorithmes pour générer une triangulation : les algorithmes itératifs d'une part, et récursifs d'autre part. Dans les deux cas, on part d'un ensemble de points du plan et le résultat obtenu est un ensemble de triangles (la triangulation) [31].

- L'approche **itérative**tend à partir d'un ensemble constitué d'un unique triangle et à ajouter à cet ensemble les nouveaux triangles obtenus pour finir avec l'ensemble désiré.
- L'approche **récursive** adopte une stratégie « diviser pour régner ». Une division par dichotomie de l'ensemble de points est effectuée avant fusion des ensembles.

Dans notre application, nous avons opté pour la première approche.

Le processus de triangulation se résume endeux phases :

Phase 1: Extraction des informations géométriques à partir de l'environnement

Une fois les obstacles sont définis dans l'environnement, ils sont englobés dans des parallélépipèdes et les points d'extrémités des obstacles sont calculés en utilisant la fonction getBoundingBox ( ) qui est prédéfinie dans Ogre 3D. Cette fonction nous aide à extraire les quatre sommets (coordonnées) de la base de chaque parallélépipède englobant l'obstacle. Chaque sommet est ajouté pour réaliser la triangulation

Phase 2 : Exécution de l'algorithme

L'algorithme général se penche fortement sur une propriété importante des triangulations de *Delaunay*.Il n'y a pas d'autres points sur ou à l'intérieur du cercle circonscrit d'un triangle quelconque que les sommets de ce triangle. En d'autres termes, tous les cercles circonscrits sont vides [32].

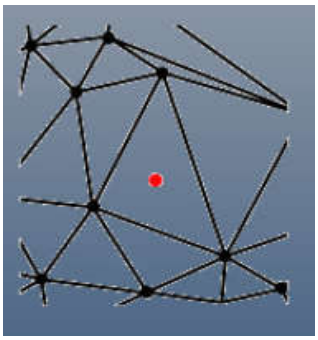
Pour notre application, on utilise l'algorithme suivant pour ajouter un sommet à une triangulation déjà existante.

Ajouter_sommet (sommet)

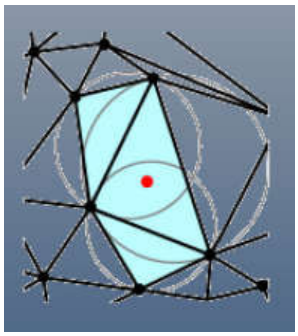
```
Début
    Pour (chaque triangle) faire
        Si(les sommetest à l'intérieur du cercle circonscrit de cetriangle)alors
            Enregistrer les arêtes du triangle enedge_buffer
            Effacer le triangle
        Fin Si ;
    Fin Pour ;
    Retirer toutes les arêtes qui sont en  double dans edge_buffer,
    garder uniquement les arêtes simples
    Pour (chaque arêteenedge_buffer)faire
    Former un nouveau triangle entre l'arêteet le sommet
    Mettre  à jour l'information d'adjacence
        Si (tous les sommets de ce triangle appartiennent au même obstacle) alors
            Si(obstacle statique)
                    Dénoter ce triangle comme non navigable
            Sinon      // objetdéplaçable ou bien navigable
                    Dénoter ce triangle comme  navigable
            Fin Si ;
        Sinon ;// espace libre
            Dénoter ce triangle comme  navigable
    Fin Si ;

    Fin Pour ;
Fin.
```

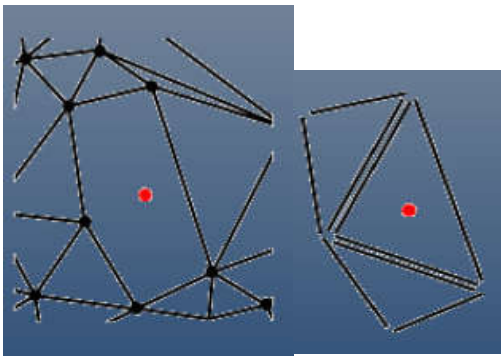
L'application de cet algorithme est expliquée comme suit :



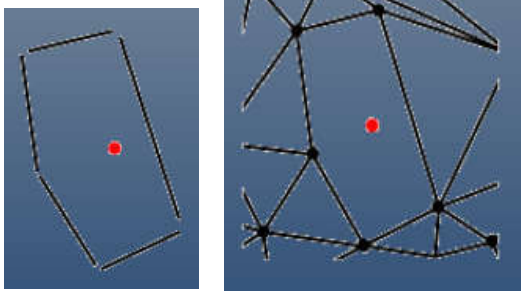
Pour insérer un nouveau sommet ...



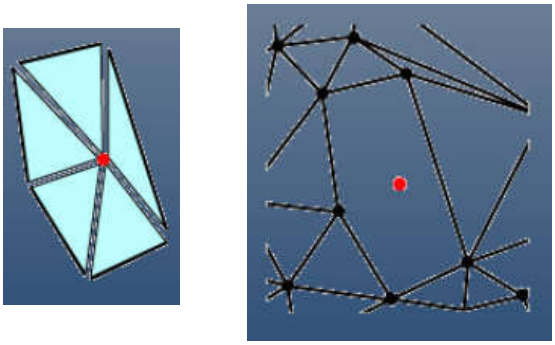
... D'abord, trouver les triangles dont les cercles circonscrits englobent le sommet.



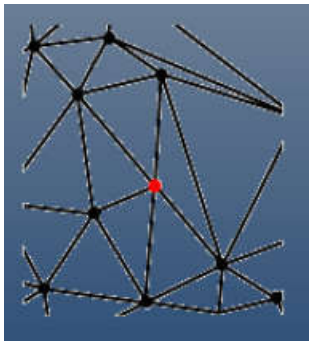
Enlever ces triangles, mais se rappeler leurs arêtes.



Retirer les arêtes en doubles et ne conserver que les arêtes simples.



Former de nouveaux triangles entre les arêtes restantes et le sommet inséré ...



Et, enfin, ajouter ces derniers a la liste des triangles.

Figure 2.4 : Processus d'insertion d'un sommet à une triangulation déjà existante [32].

Maintenant, il suffit de trouver un moyen de commencer l'ensemble du processus. D'une certaine manière, on doit créer une triangulation de *Delaunay*valide pour commencer avec. On ajoute ensuite successivement tous les sommets.

Cette triangulation initiale valide est facile à trouver. Il suffit juste de calculer un grand triangle "super triangle" qui englobe tous les sommets. Bien sûr, cela signifie que les triangles superflus seront formés, mais ils peuvent être enlevés après facilement.

L'algorithme complet s'écrit donc comme suit:

Triangulation ()

```
Début
    Créer  un supertriangle etl'ajouterà la triangulation
    Pour(chaque sommet)faire
        Ajouter sommet (sommet)
     Fin Pour ;
    Pour (chaque triangle)faire
        Si (unouplusieurs sommets appartenant au supertriangle) alors
            Supprimer ce triangle
        Fin Si ;
     Fin Pour ;
Fin.
```

Après avoir achevé l'opération de représentation de l'environnement par triangulation de *Delaunay*, dans laquelle nous avons décrit ses propriétés et utilisé la méthode incrémentale, nous allons aborder l'étape de planification de chemin qui exploite cette triangulation.

4.3. Planification de chemin

Il existe plusieurs algorithmes de recherche de chemin, parmi lesquels nous avons choisi l'algorithme A*.

4.3.1. Algorithme A*

La figure suivante illustre la tâche de la planification de chemin par l'algorithme A*.

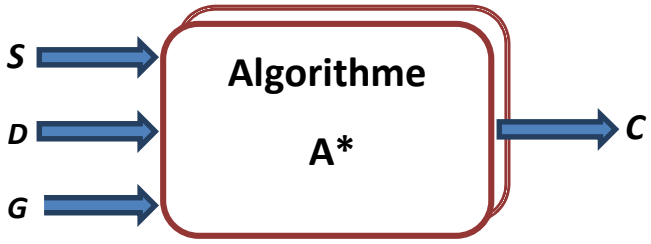


Figure 2.5 : Planification de chemin par A*.

Avec :

S (Source) : le point de départ.

D (Destination) : le point d'arrivé.

G (Graphe) : le graphe à parcourir.

C (Chemin) : le chemin optimal reliant le nœud de départ et le nœud de destination.

Algorithme A* :c'est un algorithme qui permet la recherche d'un chemin optimal entre un nœud de départ et un nœud d'arrivé. Il s'agit d'une extension de l'algorithme de *Dijkstra*. Il utilise une évaluation heuristique qui estime la distance entre un nœud quelconque du graphe et le nœud cible et visite ensuite les nœuds par ordre de cette évaluation heuristique.

La fonctiondistance peut être calculée comme suit:

Soit un nœud N de coordonnés (X, Y), et un nœud destination D de coordonnés (X', Y').

- Distance Euclidienne (théorie de Pythagore) :

  $H(N)=\sqrt{(X-X')^2+(Y-Y')^2}$

- Distance maximum :

  $H(N)=\max(|X-X'|, |Y-Y'|)$

- Distance de Manhattan :

  $H(N)= (|X-X'|+|Y-Y'|)$

On calcule donc un chemin optimal par A* de la façon suivante:

1. créer deux listes vides Ouverte et Fermée,
2. insérer dans la liste Ouverte le nœud de départ (S),
3. tant que la liste Ouverte n'est pas vide et ne contient pas le nœud destination :

a. chercher dans la liste Ouverte le nœud dont le coût estimé (F) est minimal,

b. ajouter le nœud à la liste Fermée,

c. supprimer le nœud de la liste Ouverte,

d. pour chaque nœud voisin V du nœud actuel N :

i. calculer G' comme la somme de coût G de N et du coût associé au nœud adjacent V+ le cout de l'objet(si l'objet n'est pas un obstacle).

ii. si (V est dans la liste ouverte) :
    si(le coût G' de V est inférieur à G)
        Mettre à jour le Nœud V (cout G, F, Parent).

iii. sinon
    ajouter V à la liste ouverte.

Fin si

4. si le nœud d'arrivé n'est pas dans Fermée, terminer l'algorithme sur un échec : il n'existe pas de chemin depuis le nœud de départ vers le nœud d'arrivé.
5. sinon, reconstruire le chemin en suivant l'information parent dans la liste Fermée.

On peut simplifier l'algorithme A* par le schéma suivant :

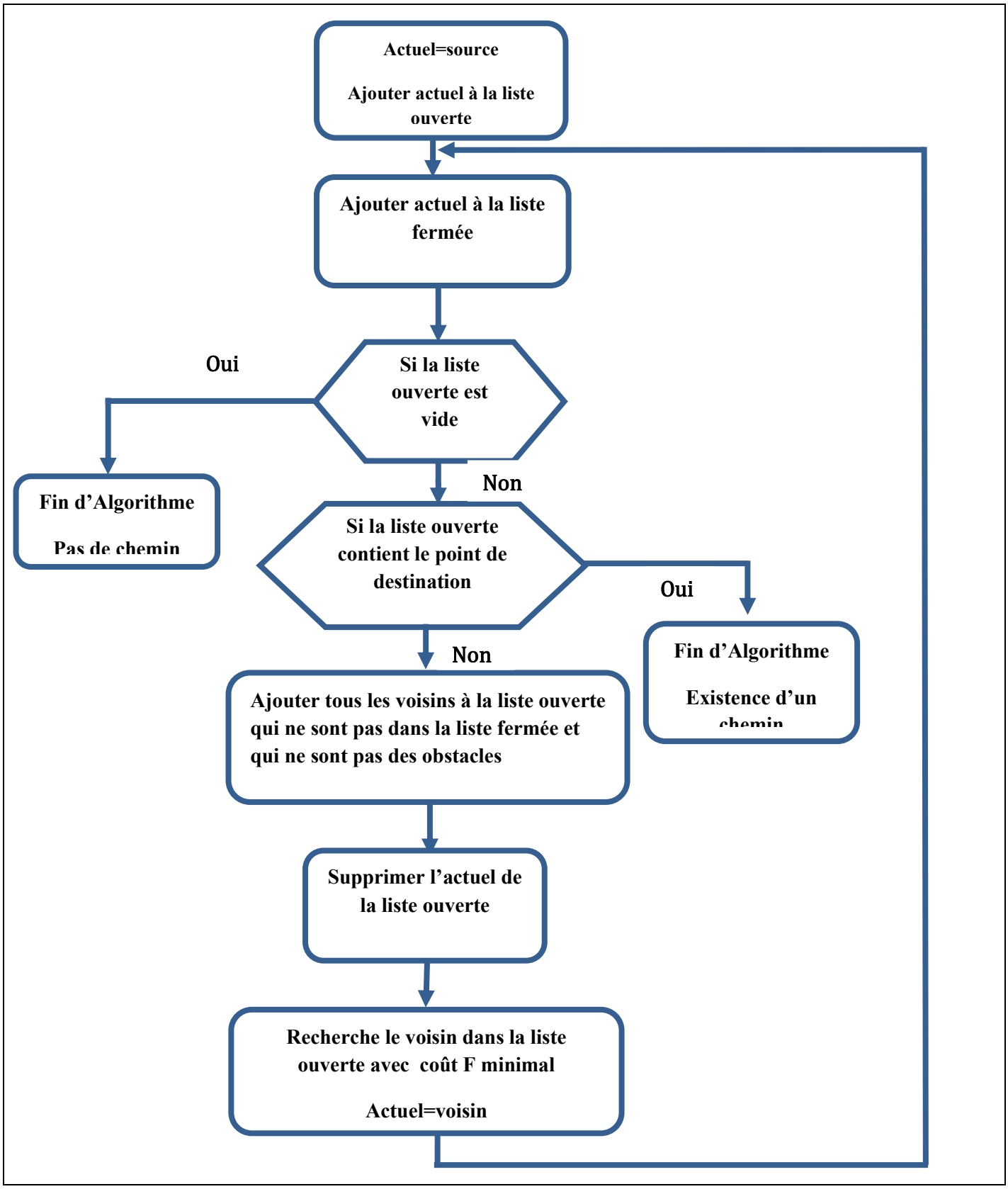


Figure 2.6 : Planification de chemin par A*

4.4. Navigation

Cette figure illustre la tâche de navigation.

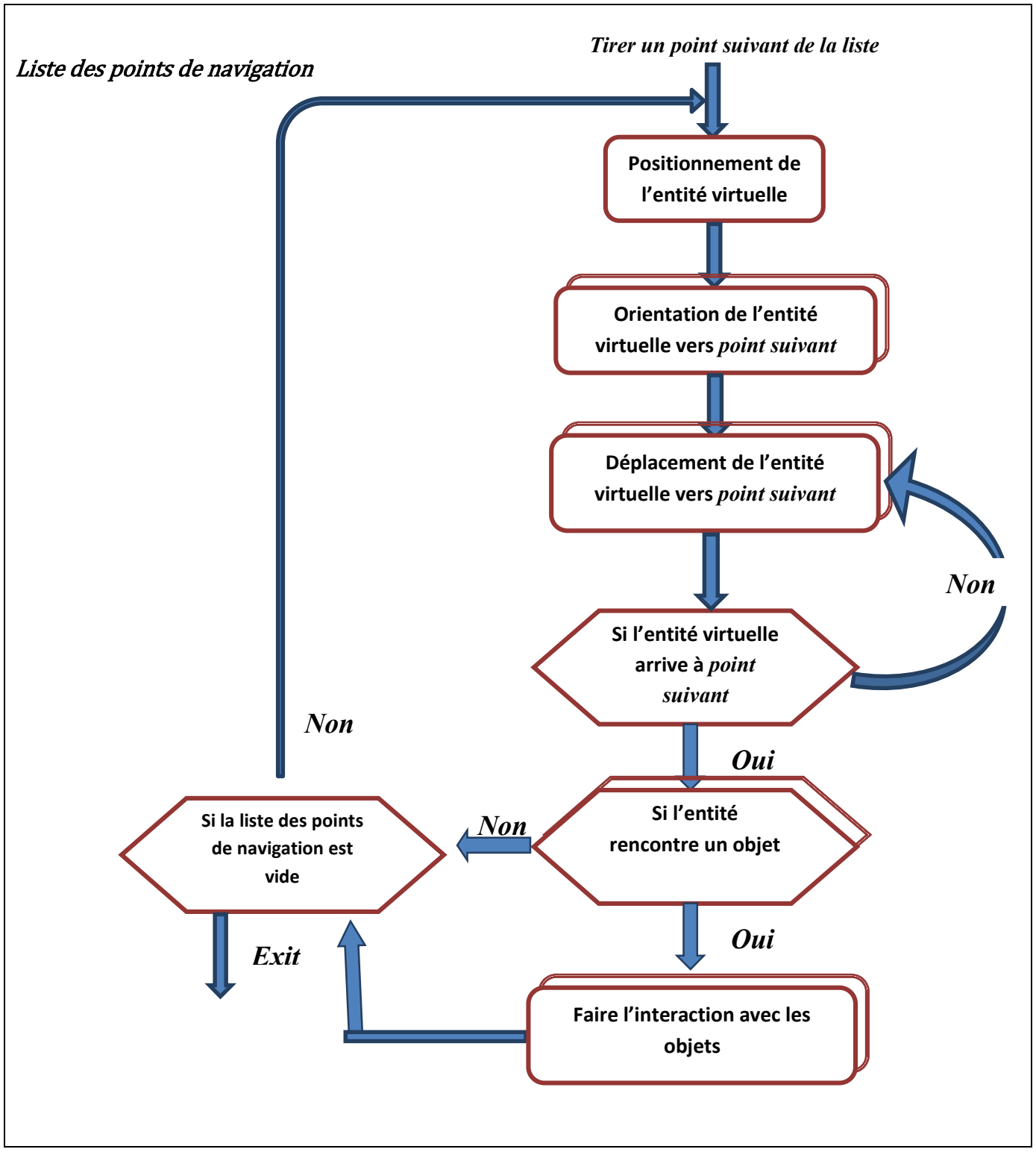


Figure2.9 : la tâche de navigation.

Positionnement : permetde positionner l'entité virtuelle dans les coordonnés des points de navigation.

Orientation: permet de calculer la direction de l'entité virtuelle pour aller au point suivant, le vecteur de direction $\vec{D}$ peut être calculé comme suit :

Soit la position de l'entité virtuelle ayant comme coordonnés(X, Y) et les coordonnées du point de destination (X', Y') :

$\vec{D}=\overrightarrow{XX'}$ donc $Dx=$ (X'- X) et $Dy=$ (Y'- Y)

Déplacement : On a ***: d*=v* Δt**, tel que :

d : le déplacement de l'entité.

v : la vitesse de l'entité virtuelle.

Δt: le temps écoulé depuis le dernier frame.

4.5. l'interaction avec les objets

La **figure 2.10** illustre le comportement de l'entité virtuelle lorsque cette dernière rencontre un objet sur son chemin de navigation.

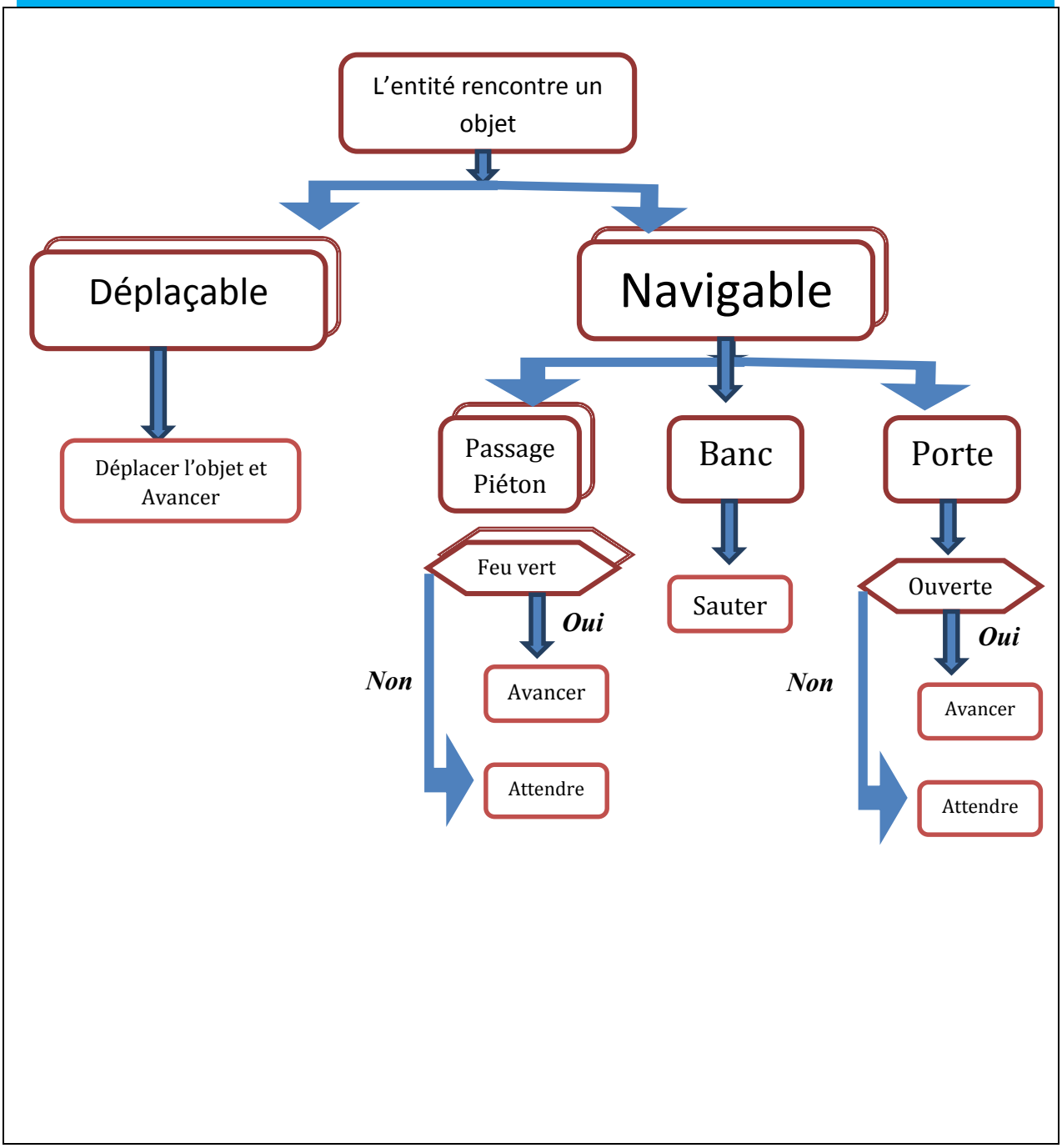


Figure2.10 : **l'interaction avec les objets**

5. Conclusion

Dans ce chapitre, on a globalement défini les objectifs assignés à notre système, présenté la conception générale de ce dernier, puis détaillé les tâches de chacun de ses modules. Dans le chapitre suivant, nous allons présenter la phase de réalisation de cette conception ainsi que les outils utilisés pour implémenter notre système.

Chapitre 3

Implémentation et résultat

1. Introduction

Dans ce chapitre, nous allons décrire la mise en œuvre des différentes étapes de notre système conçu dans le chapitre précédent. Nous allons commencer par justifier l'environnement de développement utilisé, puis présenter les structures de donnée et les fonctions utilisées. Nous allons enfin présenter quelques résultats expérimentaux de notre travail.

2. Outils de développement

2.1. Langage de programmation

Après étude de nos besoins, nous avons choisi l'environnement de programmation Qt et programmé notre application en utilisant le langage C++. Notre choix est motivé par les faits suivants :

- Le langage de programmation C++ est un langage très puissant.
- Le langage C++ offre la possibilité de programmer avec les classes et les objets (orienté objet).

2.2. Moteur de rendu

Ogre 3D est un moteur graphique Open Source (d'où son nom : Object-OrientedGraphicsRenderingEngine, qui veut dire moteur de rendu graphique orienté objets). Un moteur graphique est la solution pour afficher en temps réel des formes 3D. C'est la base de toute application graphique (jeu vidéo, simulateurs, réalité virtuelle…).

Voici quelques notions indispensables dans Ogre :

- **Scene Manager**

SceneManagerest un élément d'Ogre responsable d'organiser et de rendre les objets dans une scène. Tout ce qui apparaît dans une scène est contrôlé par le **SceneManager**. Quand un objet, est placé sur la scène, la classe **SceneManager** s'occupe de garder en mémoire leurs locations.

- **Entity**

Une entité est la représentation dans la scène d'un modèle 3D, aussi appelé mesh . Le mesh est l'ensemble des polygones élémentaires que l'on crée dans un logiciel de modélisation et qui constitue le modèle 3D complet. Tous ces polygones sont reliés entre eux par leurs sommets. Plus il y a de vertices et donc de polygones, plus le mesh est précis. Généralement, le mesh possède aussi une ou plusieurs textures (une image plaquée sur les polygones) qui lui donnent un aspect plus réaliste qu'une couleur unie lors du rendu.

On peut créer une entité comme suit :

```
Entity *head= mSceneMgr->createEntity("Tete", "ogrehead.mesh");
```

tel que le premier paramètre de la fonction createEntityest le nom de l'entité et le deuxième est le nom du fichier qu'on veut charger. Ce fichier à l'extention**.mesh**et contient les modèles reconnus par Ogre.

- **SceneNode :**

Un SceneNode est un objet invisible auquel on va pouvoir attacher un nombre indéfini d'entités, lesquelles deviennent solidaires de ce nœud et subissent donc les mêmes transformations que lui. C'est donc une sorte de conteneur qui contient les informations de positionnement de chacune des entités de la scène qui lui sont rattachées.

- **Camera:**

La caméra est l'élément qui définit la position de notre point de vue dans la scène, dans quelle direction on regarde, mais aussi jusqu'à quelle distance il est possible de voir s'afficher les objets éloignés.

La fonction suivante permet de créer une caméra :

```
mCamera = mSceneMgr->createCamera("Ma Camera");
```

Le déplacement de la camera se fait par la fonction **setPosition()** en déterminant les coordonnés x, y et z :

```
mCamera->setPosition(Vector3(x, y, z));
```

lookAt( ), comme son nom l'indique, permet de déterminer le point de la scène observé par notre caméra.

```
mCamera->lookAt(Vector3(0.0, 100.0, 0.0));
```

3. Structures de données

3.1. Représentation du terrain

Notre scène est représentée par un maillage de navigation triangulaire.

```
StructSVertex          //  structure d'un sommet
{
        X, Y, Z : double ;        //coordonnées du sommet

        Obs : entier;   //  identifiant de l'obstacle
};

StructSedge            // structure d'un segment
{
        V1, V2 : entier ;         // indices des extrémitésdu segment

};

StructSTriangle                 //  structure d'un triangle
{
        V1, V2, V3 : entier ;   //  les indices des  3 sommets du triangle

        E1, E2, E3 : entier ;    //les indices des segments formant le triangle

        ETAT : entier ;          //    Navigable ou Non
};

StructSEdges
{
        V1, V2, T1, T2 : entier ;     //segments communs entre 2 triangles adjacents

};
```

Les sommets, les segments et les triangles générés sont sauvegardés dans des tableaux de structure approprie à chaque type.

3.2. Fonction A*

a) Représentation d'un nœud

On a utilisé la classe Nœud pour représenter l'élément de base de notre graphe de recherche (liste ouverte et liste fermée).

```
Class Nœud {
    X, Y, cout G, cout H, cout F, état : entier ;
    Parent : pointeur vers Nœud ;}
```

b) Représentation des listes des nœuds

La fonction A* utilise trois listes essentielles :

- une liste ouverte qui contient les nœuds à explorer,
- une liste fermée qui contient les nœuds choisis,
- une dernière liste qui contient les nœuds du chemin final.

On a implémenté ces trois listes comme étant des tableaux dynamiques en utilisant la class **vector** de la bibliothèque standard de C++. Cette classe possède plusieurs fonctions qui facilitent la manipulation sur les tableaux dynamiques.

push(): Ajouter un élément.

top() : Consulter le dernier élément ajouté .

pop() : Supprimer le dernier élément ajouté.

sort() : Pour trier le contenu de la liste.

c)Représentation des objets :

```
typedef enum Type {obstacle=0,obstacle_mobile=1, obstacle_navigable=2} TYPE;
```

On a utilisé la classe objet pour réaliser les objets de la scène

```
Class objet {
    Nom : String ;
    Type : TYPE ;
     code, index, cout, état :entier ;
    Scale, position :Ogre :: vector3;
}
```

d) Représentation des entités :

```
Class entité {
Nom : String ;
Index :entier ;
Scale, position :vector3;
```

Ogre::Vector3 GetDestination() ;//retourner les cordonner de point de destination

SetDestination(Ogre::Vector3 v) ; //spécifier les cordonner de point de destination

}

e)Fonction de calcul la valeur de F :

```
Pour chaque Nœud inséré dans la liste ouverte faire
        Calculer F(la somme de l'heuristique H et la distance parcouru G).
        Si (le Nœud représente un objet déplaçable ou navigable)
            Augmenter  F du Cout de l'objet
        Fin si
Fin pour
```

f) Fonction de l'interaction de l'entité avec les objets :

```
Pour (chaque triangle du Chemin) faire
        Lire l'index de l'objet
    Si (Espace libre) alors
        Avancer
    finsi
    Si (Objet  Navigable) alors
        Si (l'objet est une porte) alors
            Si (Ouverte) alors
                Avancer
            Sinon
                Attendre
            Fin si
        finsi
        Si  (l'objet est un banc) alors
            Sauter
```

```
        Fin si
        Si l'objet est un Feu de circulation alors
            Si (Rouge) alors
                Attendre
            finsi
            Si (Vert) alors
                Avancer
            finsi
        finsi
    finsi
    Si (Objet Déplaçable) alors
        Déplacer L'objet
    Finsi
fin
```

4. Résultats expérimentaux

Pour nos simulations, nous avons utilisé cette interface graphique pour simplifier l'interaction avec la fenêtre de rendu La figure 3.2 : résume le management des objets (3 cas : Object management contient (1) liste des objets utilise pour modéliser la scène, (2) les propriétés « le nom, le type, le cout, la position, la rotation, le changement des chailles » et (3) deux boutons pour affiché et appliqué les propriétés, Navigation (4) Delaunay triangulation : représenter l'environnement par un ensemble de triangle. Path finding : représenter le chemin de navigation. Start : lancer l'animation. Stop : arrêter l'animation. (5) la fenêtre de rendu utiliser pour modéliser la scène 3D

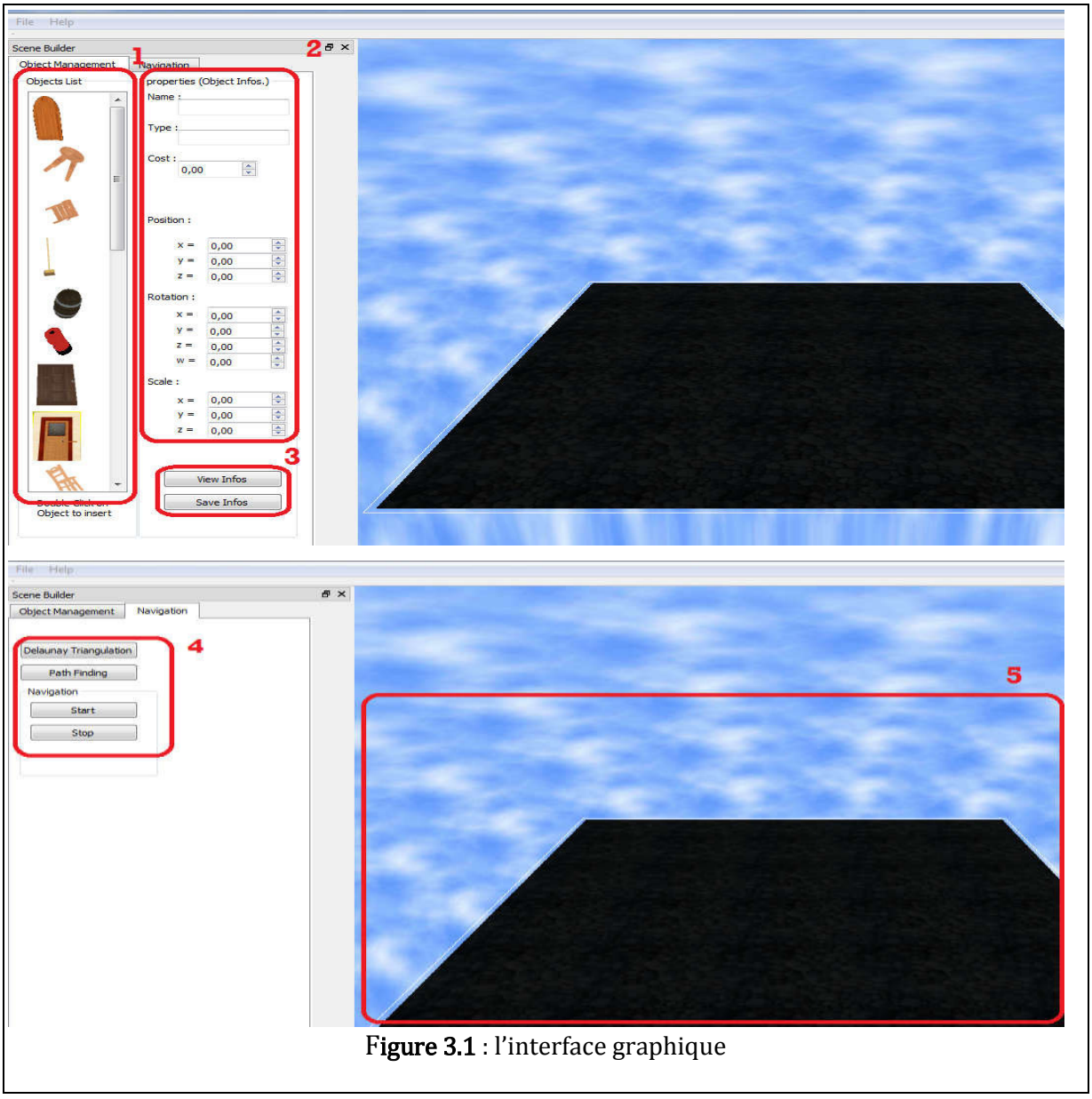


Figure 3.1 : l'interface graphique

nous avons utilisé la même configuration pour notre scene : les points de départ et de déstination sont gardés fixes. Chaque obstacle dans la scene est représenté par quatre points.

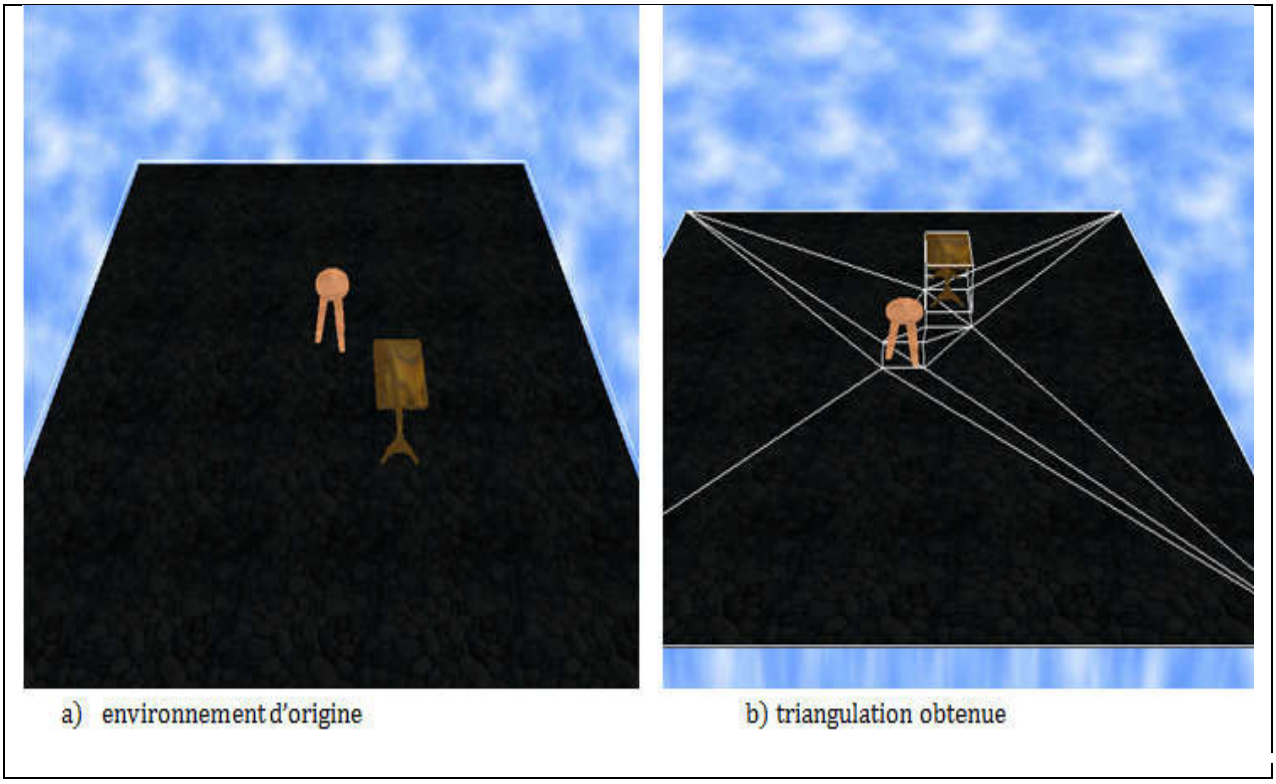


Figure 3.2 : Cas de 2 obstacles.

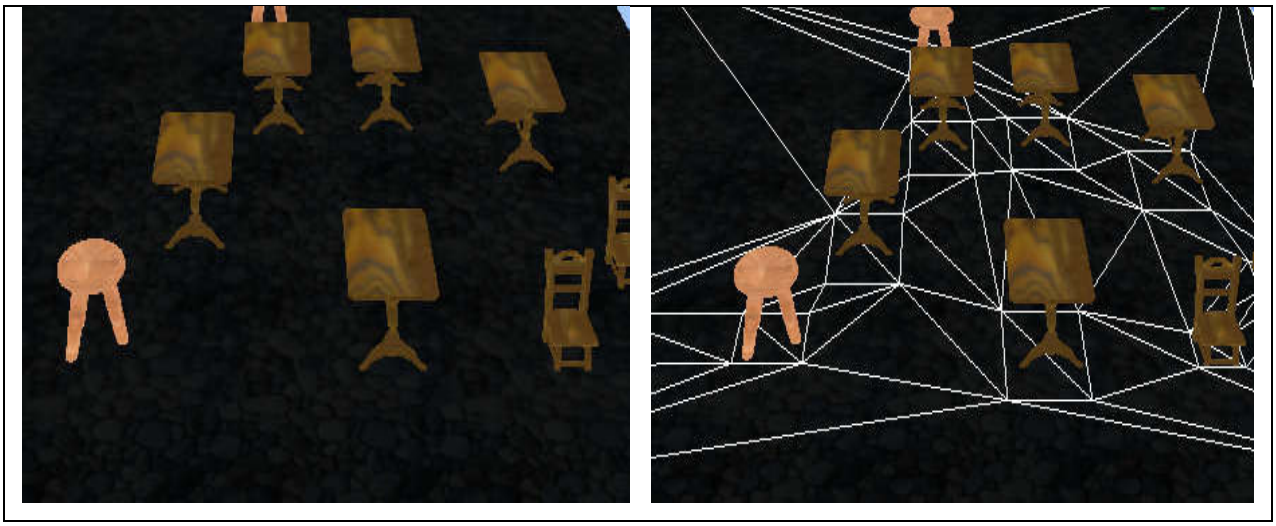


Figure 3.3 : Cas de plusieurs obstacles.

Après avoir fixé le nombre d'obstacles et réalisé la triangulation associée, nous avons, pour la recherche du plus cours chemin, utilisé l'algorithme A*

Après l'exécution de la planification de chemin en utilisant l'algorithme A*, on obtient la liste des triangles sur lesquels l'entité peut passer pour atteindre son but. Pour effectuer cette tâche, on a 2 cas de figures:

- l'entité passe par les centres de gravité des triangles,
- ou bien l'entité passe par les milieux des portails formés par ces triangles.

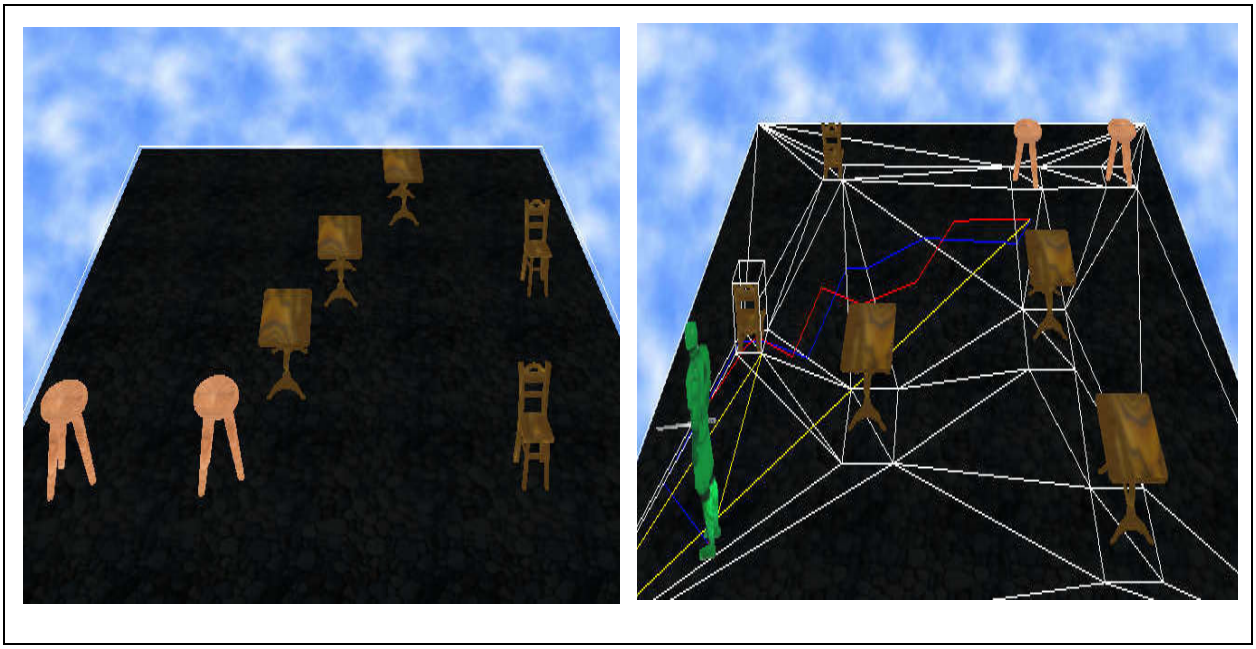


Figure 3.4 : Environnement d'origine et chemins planifiés par A*

La figure 3.5 résume quelques résultats obtenus en utilisant l'algorithme A* (2 cas : centres de gravités (en rouge) et milieux des portails (en vert)) et l'algorithme funnel (en jaune). On en déduit que ce dernier fournit des résultats plus réalistes que l'algorithme A*.

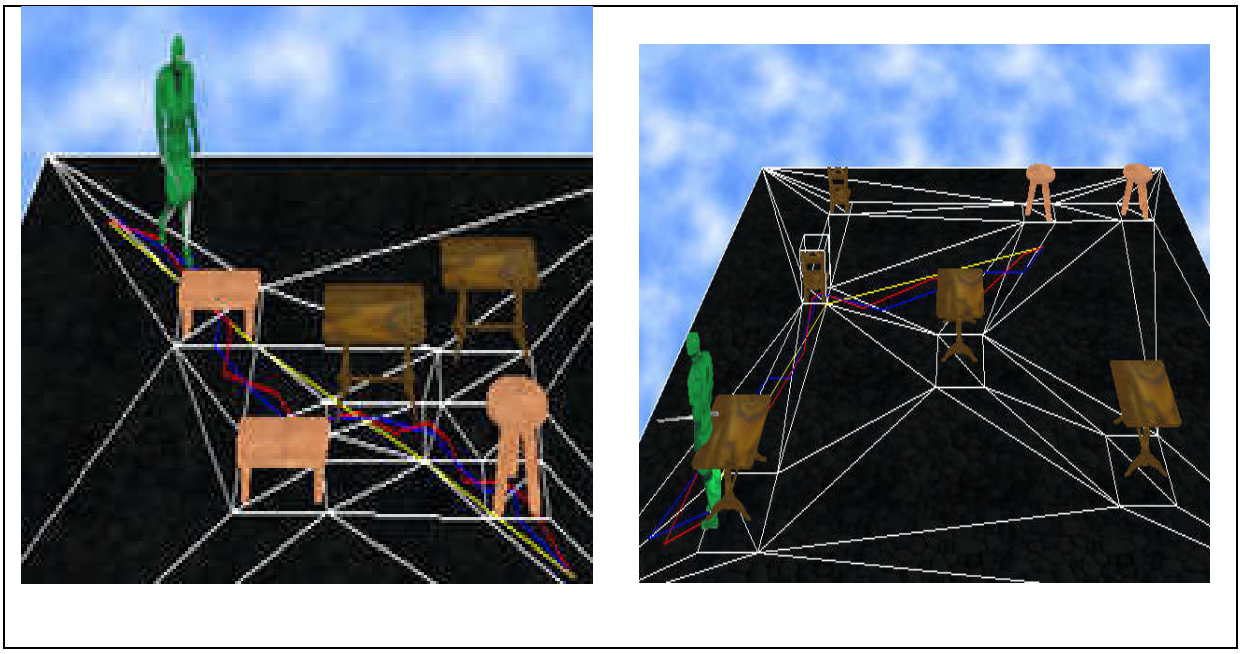


Figure 3.5 : Résultats obtenus en utilisant l'algorithme A* (2 cas : centres de gravités et milieux des portails) et l'algorithme funnel.

La figure 3.6 résume le résultat obtenu en utilisant un objet déplaçable (le 1^{er} cas exprime la scène avant que l'entité arrive à l'emplacement de la chaise et le 2^{eme} cas exprime la scène après que l'entité arrive à l'emplacement de la chaise)

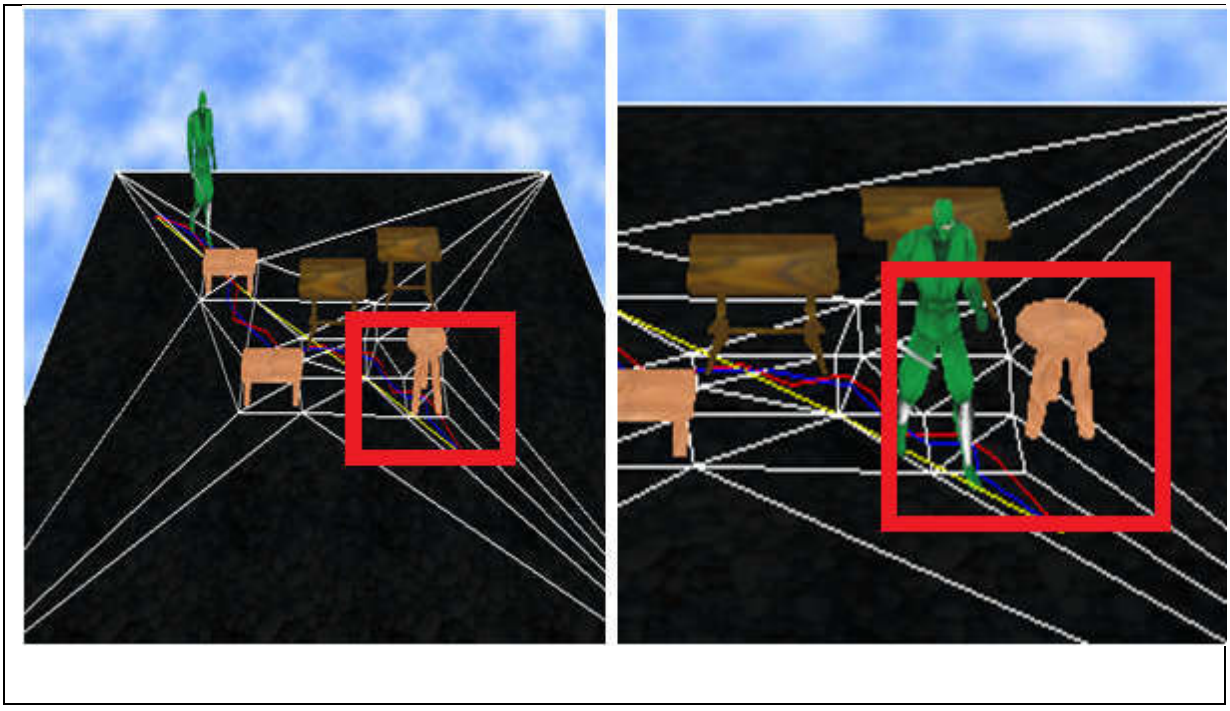


Figure 3.6 : Résultats obtenus en utilisant un objet déplaçable

La figure 3.7 résume le résultat obtenu en utilisant un objet navigable (le 1^{er} cas exprimé la scène avant que l'entité arrivé à l'emplacement de la table et le 2^{eme} cas exprimé la scène après que l'entité arrivé à l'emplacement de la table)

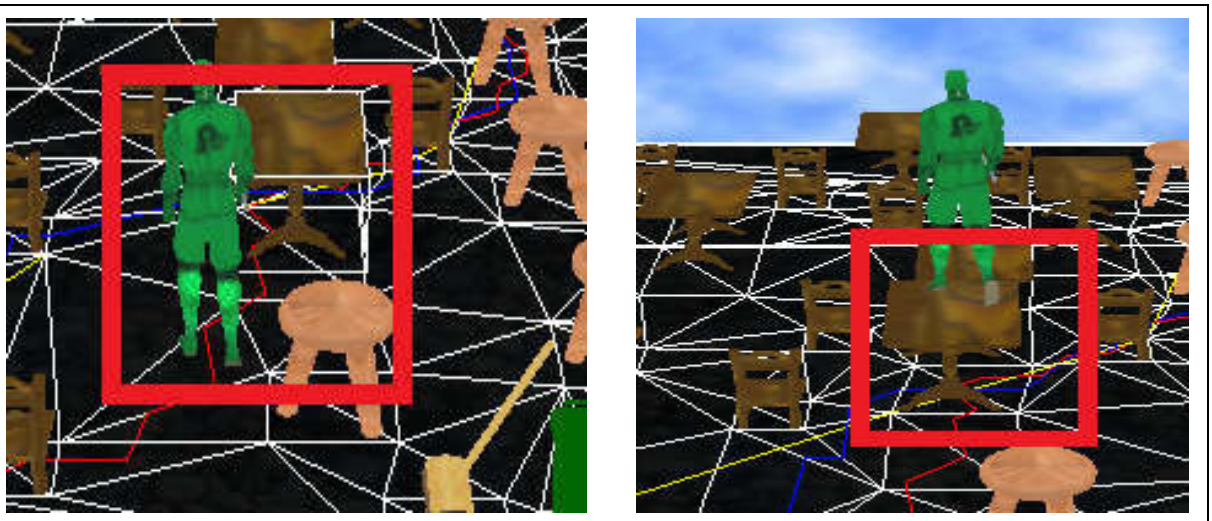


Figure 3.7: Résultats obtenus en utilisant un objet navigable (l'entité fait un saut la dessus)

La figure 3.8 : résume le résultat obtenu en utilisant un objet mobile le 1^{er} cas exprimé la scène avant l'entité arrivé à l'emplacement de la porte et le 2^{eme} cas exprimé la scène après l'entité arrivé à l'emplacement de la porte

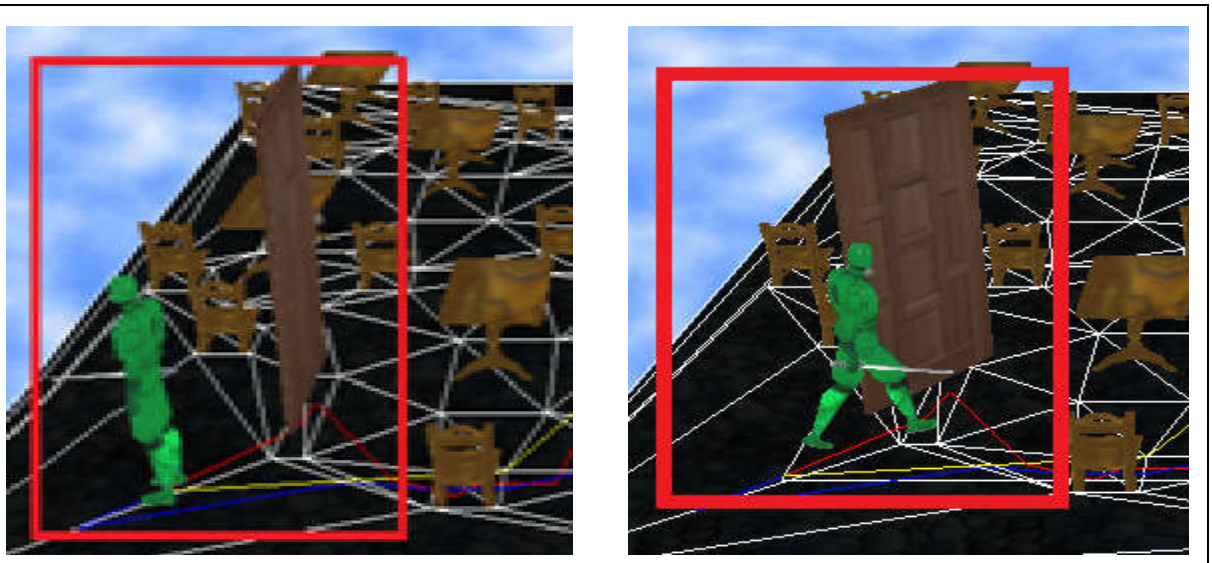


Figure 3.8 : Résultats obtenus en utilisant la porte comme un objet mobile

5. Conclusion

Dans ce chapitre, nous avons présentés le choix de l'environnement ainsi que le langage de programmation utilisés pour implémenter notre système par la suite nous avons exposés les différentes structures de données utilisées. Et enfin nous avons présentés les différents résultats expérimentaux de notre système.

Conclusion générale

Conclusion et Perspectives

Grâce aux progrès de la synthèse d'images, la modélisation des mondes en trois dimensions et l'automatisation de leur peuplement par des humanoïdes ouvrent de nombreux champs d'applications. Les réalisateurs utilisent largement ces outils pour leurs effets spéciaux. Les jeux vidéo sont de plus en plus réalistes et les architectes peuvent valider la conception de leurs maquettes. Le choix d'une représentation appropriée de l'environnement joue un rôle très important pour le processus de peuplement.

Nous avons traité au sein de ce travail la problématique de navigation en environnements virtuels. Cette problématique est renforcée dans notre cadre d'application par la nécessité de tenir compte des besoins d'interaction des humains virtuels avec l'environnement ou ils évoluent, afin de remédier aux inconvénients des méthodes basés uniquement sur la représentation géométrique.

La solution que nous avons réalisée se base sur :

- Représenter l'environnement de navigation par la triangulation de Delaunay.
- Augmenter cette représentation par des informations relatives aux objets de la scène
- Planifier le chemin avec l'algorithme A*et le raffiner en utilisant l'algorithme funnel.
- Permettre à l'entité virtuelle de naviguer vers sa destination.

L'ajout de l'information dans la représentation de l'environnement permet de caractériser certaines zones selon le type d'objets qui l'occupent. Les objets peuvent être de type : obstacle, déplaçable ou navigable. Ces zones facilitent la tâche de navigation. L'humanoïde en connaissant le type de la zone où il se trouve il peut changer de comportement.

Les résultats obtenus sont satisfaisants du point de vue temps de calcul et qualité de simulation obtenue. Néanmoins, notre projet pourra être amélioré par l'ajout d'autre fonctionnalités comme :

- Utiliser une fonction de lissage de chemin obtenu.
- Prendre en considération le volume de l'entité durant le processus de planification.
- Prendre en considération d'autres entités mobiles dans la scène.
- Utiliser des règles de comportement pour améliorer le processus de navigation.
- Adapter la solution pour le cas d'environnements complexes.

Table de figure

Table de figure

Liste des tableaux

Table de matière

Table de matière

Table de matière

Bibliographie

Bibliographie

[1] Clodéric Mars. Navigation piétonnière en environnement informé pour des humanoïdes virtuels. IRISA7,2,2006

[2] Fabrice Lamarche. Humanoïdes virtuels, réaction et cognition : une architecture pour leur autonomie. Université rennes 1, 2003.

[5] Kenza Harkouken Saiah Etude et définition de mécanismes sémantiques dans les environnements virtuels pour améliorer la crédibilité comportementale des agents : utilisation d'ontologies de services

[6] Reinefeldv(A.) et Marsland (T. A.). Enhanced iterative-deepening search. IEEE Transactions on pattern Analysis and Machine Intelligence, vol. 16, n 7, 1994, pp.701 -710.

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

[8] Korpf (R. E.). Depth-first iterative-deepening : An optimal admissible tree search. Artificial intelligence, vol. 27, n1, 1985, pp. 97-109.

[11] J. Barraquand et J.-C. Latombe,"*Robot motion planning: a distributed representation approach*", International Journal of Robotics Research, pages 628–649, 1991.

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

[13] C. Gloor, P. Stucki, et K. Nagel,"*Hybrid techniques for pedestrian simulations. 4th Swiss Transport Research Conference*", Monte Verità, Ascona, 2004.

[14] Kallmann, M. and Thalmann, D. (1999). Direct 3d interaction with smart objects. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '99)*, pages 124–130.

[15] Badawi, M. and Donikian, S. (2004). Autonomous agents interacting with their virtual environment through synoptic objects. In *Procedings of The 17th Conference on Computer Animation and Social Agents*.

[16] Forbus, K. D. and Wright, W. (2001). Some notes on programming objects in the simsTM. In *Northwestern University*.

[17] Abaci, T., Cíger, J., and Thalmann, D. (2005). Action semantics in smart objects workshop paper. In *Workshop towards Semantic Virtual Environments*.

[18] GAGUI ALI. Navigation réactive d'humain virtuel par Planification locale

[19] A. Bouguetitiche, "*Modélisation topologique et sémantique de l'environnement*", mémoire de magister en informatique, option synthèse d'image et vie artificielle, Université de Mohamed Khider Biskra., 2011.

[20]H. Jiang, W. Xu, T. Mao, C. Li, S. Xia, and Z. Wang,"*A semantic environment model for crowd simulation in multilayered complex environment*". Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology, VRST '09, pages 191_198, New York, NY, USA, 2009. ACM.

Bibliographie

[21] N Pelechano, J M Allbeck, and N I Badler,"*Controlling individual agents in highdensity crowd simulation*", Association for Computing Machinery, 2007.

[22] Reynolds CW. Flocks, Herds, and Schools: "*A Distributed Behavioral Model*",Computer Graphics 21. 1987;4:25 – 34.

[23]Craig W. Reynolds,"*Steering Behaviors For Autonomous Characters*": Game developers conference, 1999.

[24] D. Helbing, I. Farkas, and T. Vicsek,"*Simulating dynamical features of escape panic*", Nature, 407:487–490, 2000, article f. lamarch.

[25] Franck Feurtey, "Simulating *the collision avoidance behavior of pedestrians*", department of Electronic Engineering, 2000.

[26] Soteris Stylianou and Yiorgos Chrysanthou,"*Crow self-organisation, streaming an short path smoothing*", journal of WSCG - Science Press, 14, 2006.

[27] Fabrice Lamarche and Stéphane Donikian,"*Crowd of virtual humans: a new approach for real time navigation in complex and structured environments*",Eurographics, 2004.

[28] Frey P. et George P, " *Le maillage facile*", Hermès Science Publications Paris, Paris, 2003.

[29] B. Delaunay, " *Sur la sphère vide* ", Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, 7:793800, 1934.

[30] Description des propriétés de la triangulation de Delaunay :

<http://fr.wikipedia.org/wiki/Triangulation_de_Delaunay>

[31] J. Coatelen et K. Falk, "*Implémentation de la triangulation de Delaunay en C++*",Projet ISIMA 2e année F4.

[32] "*Delaunay Triangles*", Posted by Sjaak Priester on December 28th, 2004:

<http://www.codeguru.com/cpp/cpp/algorithms/general/article.php/c8901/Delaunay-Triangles.htm>.