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 :

Evolution Artificielle d'une population de robots humanoïdes

Par :

Melle ZERTAL Soumia

Soutenu le : … /…./2011

Devant le jury :

Président	**Dr Foudil CHERIF**	**Université de Biskra**
Rapporteur	**Pr NourEddine DJEDI**	**Université de Biskra**
Examinateurs	**Dr Med Chaouki BABAHENINI**	**Université de Biskra**
	Dr Zineddine BAARIR	**Université de Biskra**
	Dr AbdelWahab MOUSSAOUI	**Université de Sétif**

Remerciements

Je remercie, avant toute personne, mon encadreur Monsieur le Professeur ***NourEddine DJEDI****,*
pour ses précieux conseils et surtout pour sa disponibilité et sa grande aide.
Je remercie, également, Monsieur le Professeur ***Yves DUTHEN*** *et le Professeur* ***Jean Pierre JESSEL,***
pour leur accueil au sein de l'IRIT et plus spécialement Monsieur le Docteur ***Cédric SANZA***
et Monsieur le Docteur ***Stéphane SANCHEZ*** *,*
qui m'ont bien dirigé et m'ont aidé tout au long de mon premier séjour au sein de l'équipe VORTEX de l'IRIT à Toulouse (France).

Merci à mon père et à ma mère,
pour leur compréhension, leur patience et leur soutien.
Ils ont toujours été à la source de mes succès, pour cela je ne saurai jamais comment les remerciés
A mon frère et à ma sœur.
A toutes mes amies, Et mes camarades de ma promo.

Soumia ZERTAL

Table de matières

Liste de Figures

Liste de Tableaux

Chapitre 1

1. La simulation comportementale

1.1. Introduction:

La simulation comportementale est une partie de l'animation qui se rapproche des systèmes réels de par son principe de fonctionnement en assignant aux acteurs ou systèmes animés des comportements indépendants. Ces derniers ne seront alors plus régis par un système global gérant le mouvement de tous les acteurs mais par un mécanisme de décision local placé dans chaque individu. Chaque personnage de la simulation prendra donc les décisions comportementales concernant son mouvement au pas de temps suivant, selon son état et celui de l'environnement l'entourant à cet instant de la simulation. La simulation comportementale est donc un moyen de faire interagir de manière naturelle des acteurs en simulant leurs capacités dans un environnement. Terzopoulos [TT94] a été le premier à définir un modèle comportemental. Un découpage en trois éléments distincts: Capteurs, Module comportemental et Effecteur lui ont permis de réaliser des animations réalistes de bancs de poissons naviguant au milieu de vestiges de l'ancienne Rome.

Depuis, ce modèle a été repris de nombreuses fois [BT98, SDD99] et est quasiment devenu un standard. Chaque individu d'une simulation modifie son comportement en fonction de données fournies par des capteurs et retransmet ses actions à l'environnement à travers les effecteurs. L'utilisation exclusive de ce couple acteurs/effecteurs limite la connaissance et l'effet d'un individu à leurs seules capacités d'acquisition et d'action. La simulation comportementale est utilisée en réalité virtuelle pour permettre l'animation d'entités autonomes

La plateforme de simulation doit intégrer les différents modèles de contrôle du mouvement et des interactions d'objets dans une scène. Un système de simulation est constitué d'un certain nombre d'entités, dont l'état évolue au cours du temps, chaque entité possédant une fréquence propre de calcul. La plateforme doit donc permettre la simulation, sur une même échelle temporelle, de plusieurs entités dont la synchronisation et la communication sont gérées par un moyen temps réel. Les entités communiquent, entre elles, soit par des flots de données, soit par événements.

1.2. Modélisation des propriétés des humains virtuels

L'objectif majeur de la modélisation des comportements des acteurs est de construire des agents intelligents autonomes virtuels avec adaptation, perception et mémoire, capable d'agir librement et avec émotion, d'être conscient et imprévisible. (Figure 1.1)

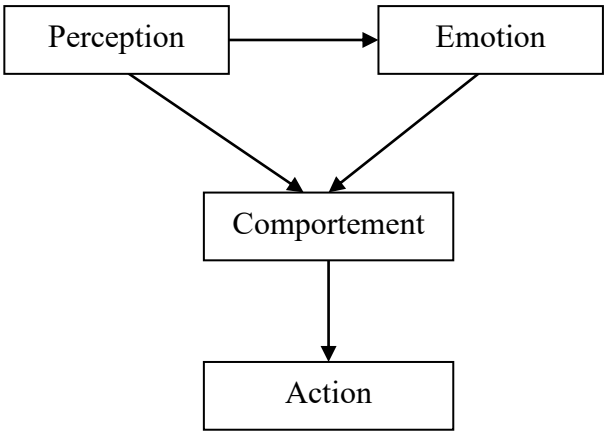


Figure 1.1 : Structure du modèle comportemental

1.2.1. La perception

La perception est définie comme la conscience des éléments dans l'environnement à travers des sensations physiques. Il est réalisé en équipant les agents avec des détecteurs visuels, tactiles et auditifs ainsi ils simulent le comportement quotidien humain (aspect visuel, mouvement, réaction, …). Le sous-système perceptuel le plus important est le système visuel. Une approche basée sur la vision [RMT90] est idéale pour le modelage d'une animation comportementale. Elle offre une approche universelle pour le passage d'information de l'environnement à l'acteur dans le contexte de recherche de chemin.

A un niveau plus haut, nous pouvons décomposer la perception comme suggérée par [BT98]. La perception d'un acteur peut être limitée aux objets et à d'autres acteurs dans le voisinage.

Mais cela limite le nombre de comportements possibles, parce que seules la présence et les caractéristiques d'un objet ou d'un acteur sont impliquées dans la sélection d'un comportement. Les actions des autres acteurs ne sont pas prises en considération.

Le module de perception produit trois types de perception: la perception de la présence d'objets et d'acteurs, la perception des actions d'acteurs et la perception d'acteurs exécutant des actions sur des objets.

1.2.2. L'émotion

L'émotion peut être définie comme l'aspect affectif de la conscience: Un état de sentiment, une réaction psychique et physique (comme la colère ou la crainte), subjectivement expérimenté comme un sentiment fort et physiologiquement qui augmente les changements préparant le corps pour une action vigoureuse immédiate.

Les acteurs doivent être capables de répondre, avec émotion à leur situation et agissant physiquement à cela. Les émotions visibles fournissent des designers avec un moyen direct pour affecter à l'utilisateur un état émotionnel propre à lui. Les acteurs sont donc équipés d'un modèle informatique simple de comportement émotionnel, qui est lié au comportement comme les expressions de visage qui peuvent être employées pour influencer leurs actions.

Une émotion est la réaction d'une personne à une perception. Celle-ci est amenée à répondre par une expression du visage, un geste, ou à choisir un comportement spécifique.

Une émotion arrive entre une perception et la réaction suivante. Deux personnes différentes peuvent avoir des réactions différentes à la même perception, selon la façon dont ils sont affectés par cette perception [OT90].

Les émotions sont causées par la réaction aux objets, les actions d'agents et les événements.

1.2.3. Le comportement

Le comportement est souvent défini comme une réponse d'un individu, d'un groupe ou d'une espèce, à son environnement. Il est habituellement présenté en langage naturel, comme ayant des implications sociales, physiologiques et psychologiques, pas toujours réductibles au mouvement d'un ou plusieurs muscles, ou à une réponse d'effecteurs. On peut également le

décrire comme la manière dont les hommes et les animaux agissent. En d'autres termes, modéliser un comportement pour une créature virtuelle, ne s'arrête pas à définir un ensemble de réactions à son environnement. Il s'agit surtout d'inclure le flux d'informations par lequel l'environnement influence la créature concernée, et la manière dont elle code et utilise ces informations.

L'approche comportementale[1] consiste à faire du comportement, la propriété qui détermine entièrement les actes d'un agent. Dans les modèles qui s'en inspirent[2], les comportements, qui sont décrits de manière hiérarchique, sont habituellement divisés en modules. Chaque module peut être à son tour décomposé en d'autres modules moins complexes, ou en actions élémentaires. Ces dernières s'exécutant de manière séquentielle ou concurrente. Un schéma classique de simulation basée sur de tels modèles se présente comme suit :

- Au début de l'animation, l'utilisateur rentre pour un acteur, une séquence de comportements (le script) qui sont stockés dans une pile. Le comportement en tête de la pile s'enclenche alors.
- Au terme du comportement courant - lorsque les conditions nécessaires à son déroulement ne sont plus vérifiées- le système dépile le comportement suivant et le déclenche. Ce procédé est répété jusqu'à ce que la pile soit vide.
- Les comportements décomposés en modules sont eux mêmes des piles, et s'exécutent de manière récursive lorsqu'ils sont sélectionnés par le système.

De nombreuses recherches sur l'architecture comportementale et sur les mécanismes de sélection sont effectuées.

Selon G. Hégron et B. Arnaldi [HA92], les systèmes d'animation comportementale se divisent en deux grandes familles qui distinguent les organismes qui agissent sur eux-mêmes de ceux qui agissent sur leur environnement.

1.2.3.1. Modèles de transformation internes

Ces modèles s'attachent à décrire de manière réaliste l'évolution interne d'entités vivantes (plantes, être humains…) en fonction de paramètre extérieurs. Il s'agit cependant de rendre compte des modifications de l'enveloppe externe de l'entité à animer (par exemple la croissance d'une plante).

[1] Souvent appelée behaviorisme - mot dérivé de l'anglais (behavior = comportement)
[2] Ils portent l'étiquette de modèles comportementaux

1.2.3.2. Modèles de plantes

Les modèles de croissance des plantes sont des systèmes de réécriture fondés sur un axiome et des règles de production appelés L-Systems. A. Lindenmayer [Lin68] définit un L-system comme étant un axiome constitué d'une chaîne de symboles paramétrés et temporisés, et de règles de production qui spécifient comment les symboles se transforment en fonction du temps. Les symboles sont associés à des primitives géométriques, ce qui permet de donner une représentation graphique au L-system.

La réécriture se fait en parallèle sur les différents éléments. Il y a deux manières de parvenir à une représentation graphique: d'une part, les symboles réécrits sont associés à une primitive graphique, et d'autre part, on peut insérer dans la chaîne de production des ordres équivalents à ceux que l'on donne à une tortue Logo. A noter que si la réécriture se fait en parallèle, l'interprétation graphique est complètement séquentielle.

Les règles de production des L-Systems sont des règles discrètes: dans [PHM93], P. Prusinkiewicz définit les dL-Systems en introduisant un temps continu à la place d'une séquence d'étapes de dérivation discrètes. Les paramètres w du système sont déterminés en continu par une équation différentielle ordinaire tant qu'ils restent dans un domaine D. Dès que w atteint une frontière de D, une règle de production qui dépend de la position sur la frontière, est appliquée. Ensuite, le système est à nouveau régi par l'équation différentielle. La figure 1.2 présente un exemple simple de L-system.

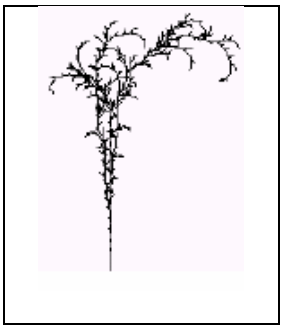


Figure 1.2 : Exemple de plante générée par un L-system.

En conclusion, les L-Systems sont utilisés à la fois en animation par ordinateur et en biologie pour simuler la croissance des végétaux. C'est aussi à la biologie (et notamment à la biomécanique humaine) que nous devons les travaux sur les modèles musculaires du corps humain que nous allons aborder maintenant.

1.2.3.3. Humanoïdes de synthèse

Du point de vue de l'animation, l'être humain se compose essentiellement d'un squelette, de muscles et d'une enveloppe extérieure, la peau. Pour animer un humanoïde, il faut définir un modèle d'un ou plusieurs de ces composants. La structure utilisée généralement est celle d'une structure osseuse (solide rigide articulé) sur laquelle est plaquée une peau déformable. Deux types de problèmes se posent donc: le mouvement du squelette (par exemple la reproduction de la marche humaine) et la déformation de la peau (par exemple, l'animation faciale).

1.2.3.3.1.1. Reproduction de la locomotion humaine

Le cas de la locomotion humaine est un cas particulier de la locomotion. En effet, contrairement à la nage d'un poisson ou au vol d'un oiseau, la locomotion humaine se distingue par les interactions avec un sol situé à altitude variable. Il n'est donc pas possible d'ignorer complètement les contraintes cinématiques. Différentes techniques sont utilisées:

- Dans le cadre du projet Jack [PB88, LPO+89] de l'université de Pennsylvanie, X. Zhao propose une méthode fondée sur la dynamique et la minimisation d'un critère énergétique. La minimisation de l'effort ou de l'énergie ne correspond pas toujours à une vérité biomécanique, à cause d'une corrélation très forte avec la morphologie du sujet. De plus, l'apprentissage de la marche, ou même la marche au pas des militaires sont des exemples de marche non économique.
- J. Hodgins [Hod96] utilise des techniques classiques de contrôle: il y a un contrôleur différent pour chaque phase de la marche, et ces contrôleurs sont reliés entre eux par des automates finis. Ces modèles sont extrêmement coûteux en temps de calcul.
- Dans le cadre de l'animation interactive, des approches purement cinématiques sont proposées [BMT90]. Plus récemment, R. Boulic [BM96] a intégré des notions de dynamiques en regroupant les articulations dans un arbre de distribution des masses. Ces approches sont intéressantes du point de vue du coût mais permettent difficilement la personnification des démarches obtenues. Une amélioration consiste à utiliser des mouvements acquis à l'aide de systèmes vidéo et à les rejouer [FGP+98]. Le coût mémoire est alors plus élevé mais la démarche nettement plus réaliste.
- F. Multon [Mul96] s'intéresse à la construction de modèles biomécaniques de la locomotion humaine. Le cycle de marche est décomposé en différentes phases reliées

par des automates. Pour chacun des cycles des trajectoires articulaires sont interpolées par de simples polynômes de degré 3. L'intérêt de l'approche réside dans le fait que les phases identifiées ne sont pas celles qui sont utilisées habituellement en biomécanique; trois phases sont utilisées pour l'extension du genou et une pour sa flexion, pour la marche comme pour la course. Cette méthode permet en outre l'identification simple de paramètres naturels comme la longueur ou la fréquence des pas.

1.2.3.3.1.2. Animation faciale :

L'animation faciale consiste à animer le visage d'un être humain. C'est une tâche complexe qui nécessite la modélisation, l'animation et le rendu de la géométrie du visage mais aussi d'éléments distinctifs du visage comme la peau, les cheveux, les lèvres ou la langue. P. Fua [FM97] classe les modèles d'animation faciale en deux catégories:

- les modèles à peu de degrés de liberté, qui se concentrent sur les mouvements de la tête. Ils utilisent des modèles simplistes et sont donc très pauvres graphiquement.
- Les modèles plus sophistiqués; il s'agit en premier lieu d'extraire la géométrie à partir d'images réelles [EBD+97] ou de la générer automatiquement [LTW95]. Ensuite, il faut définir des paramètres de déformation pour contrôler les mouvements du visage.

Trois types de modèles sont utilisés [DB97]: des modèles purement physiques, des modèles purement géométriques ou des modèles anatomiques.

1.2.3.4. Modèles de transformation externes :

Les modèles dits de transformation externes définissent le comportement extérieur d'un être, c'est à dire ses actions et réactions vis à vis de son environnement, que ce soit de façon individuelle ou collective.

Il existe plusieurs classifications des modèles de comportement externes. D. Zeltzer [Zel90] propose une classification fondée sur le niveau d'abstraction, tandis que S. Donikian [Don94] élabore la sienne selon le type de techniques utilisées. Cette classification paraît plus pertinente ici dans la mesure où les techniques employées sont plus faciles à classer. La plupart des approches de l'animation comportementale que nous allons voir seraient impossibles à classer dans une seule des catégories définies par Zeltzer.

1.2.3.4.1. Approche de R. Brooks

Brooks fait partie des pionniers ayant étudié les architectures dédiées à la génération de comportements. En 1986 il définit un modèle se fondant sur une décomposition en

comportements de base [Bro86]. Ces comportements sont représentés à l'aide d'automates à états finis temporisés fonctionnant de manière asynchrone. Les capteurs stockent les données dans des registres qui sont accessibles à tous les comportements. Ces comportements ont également entre eux un mode de communication basé sur des messages. Ils peuvent agir sur tous les effecteurs.

Le regroupement de ces comportements forme des niveaux de compétence. Ces systèmes sont hiérarchisés selon le principe de subsomption: les niveaux supérieurs manipulent les entrées et sorties des systèmes inférieurs, les contrôlant de manière indirecte. Pour cela, Brooks a introduit deux opérateurs: un opérateur d'inhibition et un opérateur de suppression.

L'opérateur de suppression agit sur les entrées des niveaux inférieurs en bloquant l'accès à certains capteurs pour les remplacer par ses propres données. Grâce à cette méthode, il contrôle l'information sur laquelle travaille le niveau inférieur. Le deuxième opérateur, l'inhibition, travaille sur les sorties des niveaux inférieurs en empêchant la diffusion de messages sur la sortie du comportement. Ces deux mécanismes sont qualifiés par le terme anglais "subsumption".

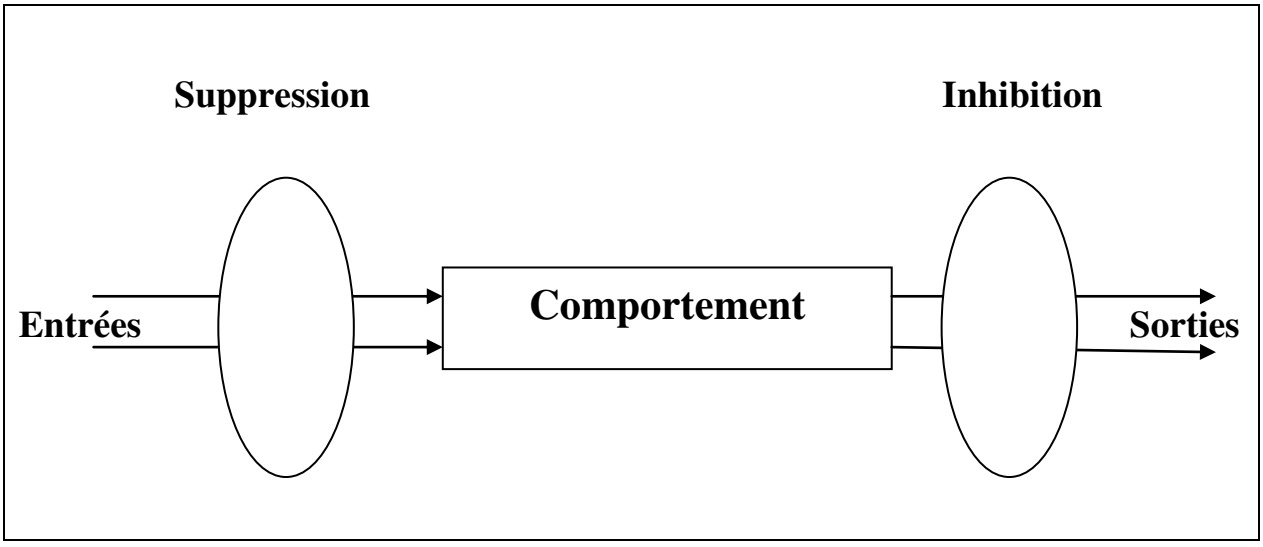


Figure 1.3 : Opérateurs de « subsumption »

Brooks décompose le problème du contrôle d'un robot mobile en niveaux de compétence dont chacun définit un comportement pour le robot mobile, par exemple : avoid objets, wander, explore… (Figure 1.4). Un niveau supérieur implique un comportement plus spécifique. Le principe de l'architecture de subsomption est qu'on peut ajouter des niveaux

supérieurs dans les niveaux actuels sans devoir modifier ces derniers. Les niveaux supérieurs peuvent manipuler les entrées et sorties des niveaux inférieurs (Figure 1.5).

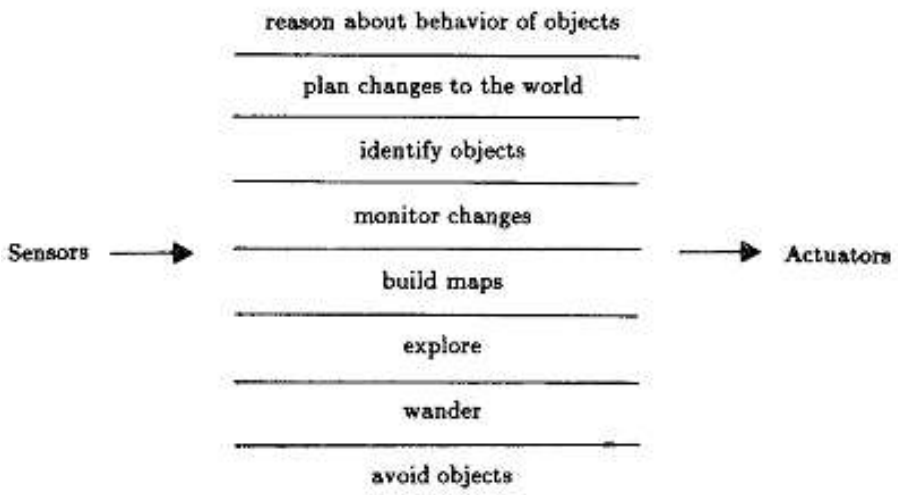


Figure 1.4 : Décomposition du contrôle d'un robot mobile en comportements

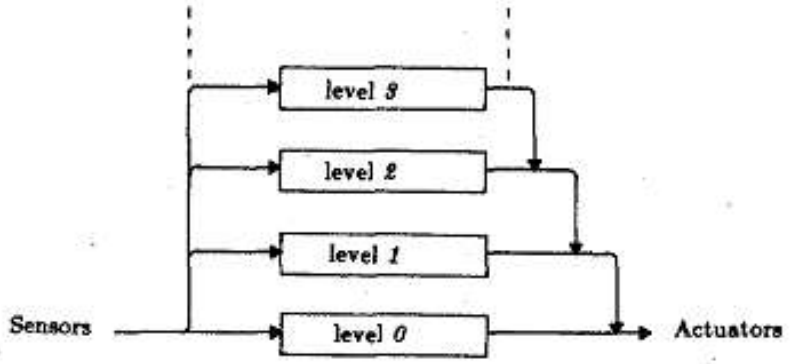


Figure 1.5 : Architecture de subsomption. Le contrôle est décomposé en niveaux de compétence. Un niveau supérieur subsume le rôle des niveaux inférieurs s'il veut prendre le contrôle

Un niveau de compétence est construit par un ensemble de comportements de base. Ils se communiquent par des messages. Quand un niveau supérieur veut prendre le contrôle et subsume un niveau inférieur, il utilise deux opérateurs de subsomption : suppression et inhibition. L'opérateur de suppression supprime les messages entrants des niveaux inférieurs et les remplace par ses propres données. L'opérateur d'inhibition bloque les messages sortants des niveaux inférieurs. Les deux opérateurs sont effectifs pendant un certain temps (Figure 1.6). La figure 1.7 montre un exemple d'architecture de subsomption pour le contrôle du robot mobile avec trois niveaux de compétence (avoid objects, wander et explore).

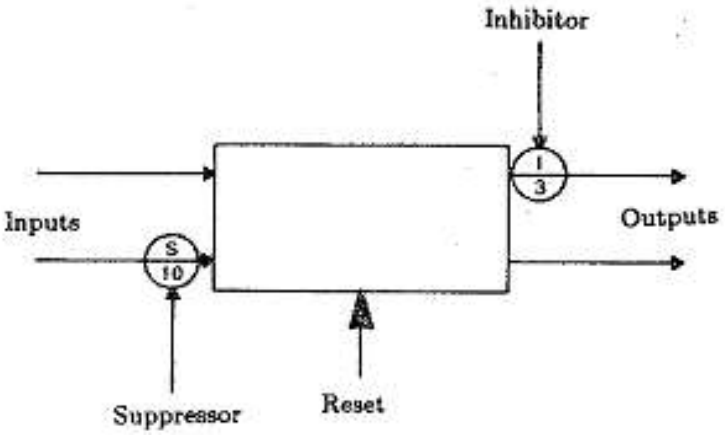


Figure 1.6: Opérateurs de subsomption. Le temps est noté dans le cercle

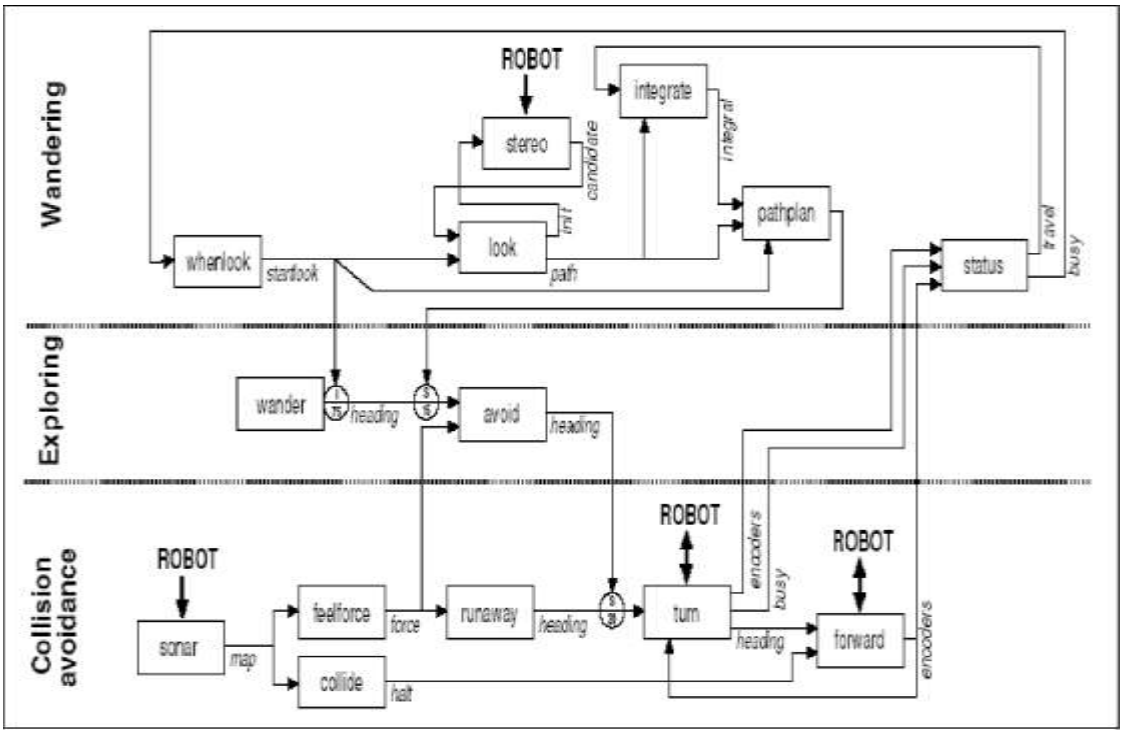


Figure 1.7: Exemple d'architecture de subsomption avec trois niveaux de compétence

Brooks a réussi à utiliser l'architecture de subsomption pour contrôler les comportements des robots mobiles virtuels et physiques tels qu'errer en évitant des obstacles ou atteindre une cible.

Mais, l'utilisation d'une telle architecture est alors une tâche très complexe car il faut définir manuellement les valeurs de temps effectif pour chaque opérateur de subsomption, l'interaction entre les comportements de base dans un niveau de compétence et l'interconnexion entre les niveaux de compétence. De ce fait, il est alors difficile de reconfigurer cette architecture pour faire des tâches différentes. [Tra07]

1.2.3.4.2. Les Boids de C. Reynolds

Dès 1986, C. Reynolds s'est attaché à construire un modèle comportemental permettant d'animer un ensemble d'individus à l'aide de la description du comportement individuel de chaque animal appartenant au groupe [Rey87]. Le terme «boids» représente le nom générique de ces entités. Ce modèle est constitué de trois comportements de base décrivant le déplacement d'un boid en fonction des positions et vitesses des individus à proximité de lui.
La cohésion permet de garder une formation. La séparation permet d'éviter les agrégats, et l'alignement est utilisé pour obtenir un déplacement uniforme des individus. En sommant les vecteurs résultants de ces opérations, Reynolds obtient le vecteur de déplacement désiré.

Trois règles de comportement sont définies: éviter les collisions, harmoniser sa vitesse à celle de ses voisins et rester proche de ses voisins. La figure 1.8 présente un exemple de nuée d'oiseaux dont le comportement respecte les règles précédentes.

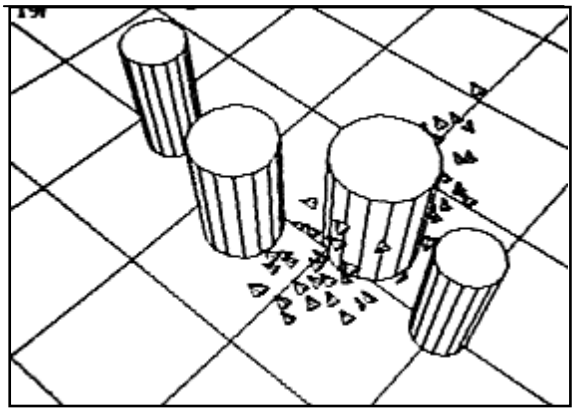


Figure 1.8 : Règles de comportement: une nuée d'oiseaux évitant un obstacle

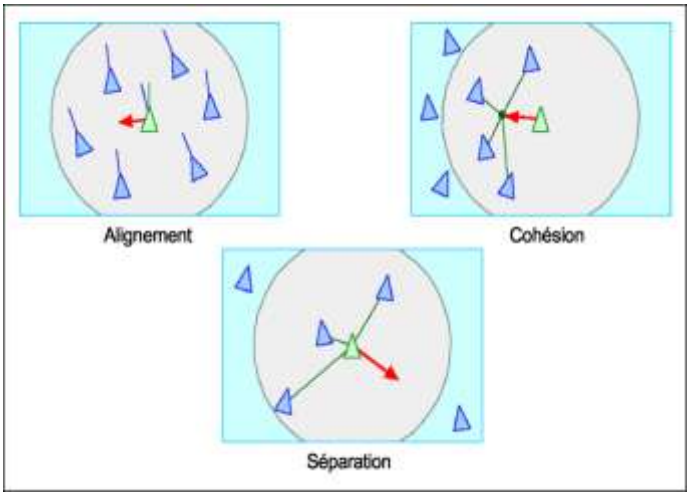


Figure 1.9 : Comportements primaires des boids.

Cette technique parait relativement simple mais permet, par émergence, d'obtenir des comportements complexes. Cette méthode sera, par la suite, utilisée avec succès dans l'industrie cinématographique pour animer des bancs de poisson, ou des hordes de chauvesouris. Les applications de ce modèle restent limitées à des animations de groupes d'individus.

En 1999, C. Reynolds étend son modèle pour contrôler le déplacement d'entités autonomes [Rey99]. Il propose alors une série de techniques afin d'obtenir des comportements de proie/prédateur, de suiveur dans un groupe, ou plus simplement d'évitement d'obstacles.

Ces comportements s'insèrent dans une structure hiérarchique globale se scindant en trois parties: un module de sélection de l'action qui, en fonction des buts et stratégies, va définir les comportements primaires à appliquer. Ceux-ci seront ensuite transformés en ordres d'animation.

La principale difficulté réside dans la construction de ces comportements primaires, souvent dépendants du support sur lequel ils sont appliqués.

1.2.3.4.3. Approche basée comportement: Mataric

J. Mataric [Mat99] s'appuie sur une collection de comportements pour contrôler l'animation de ses entités. Un comportement est un processus ou des lois de contrôle qui résolvent ou satisfont un objectif. Par exemple l'évitement d'obstacle satisfait l'objectif de ne pas avoir de collisions.

Chaque comportement reçoit une information des capteurs de l'entité et éventuellement une information provenant d'autres comportements. A l'aide, éventuellement de données propres, il envoie alors une réponse aux effecteurs de l'entité ou à d'autres comportements. L'architecture forme alors un réseau structuré.

Cette architecture utilise une représentation uniforme de l'échelle de temps. L'exécution de ces comportements est parallèle et distribuée. De plus, chaque comportement doit avoir un temps de réponse très court dans le but de satisfaire à des exigences de temps réel.

L'ajout de comportements se fait de manière incrémental: à partir de comportements de réactifs comme l'évitement d'obstacles, il est possible d'introduire des comportements plus complexes de suivi de trajectoire, d'exploration… des comportements sont donc ajoutés jusqu'à ce que leurs interactions conduisent au résultat escompté.

Ces modèles ont une représentation du monde distribué sur différents comportements augmentant les performances.

Ces modèles sont, en général très performants dans des environnements dynamiques lorsqu'ils impliquent plusieurs robots. L'aspect réactif entraîne une limitation au niveau de la complexité des actions à réaliser puisqu'il faut rester à des échelles de temps quasi-réactives.

1.2.3.4.4. Terzopoulos et la simulation comportementale

Terzopoulos a initié le modèle général utilisé en simulation comportementale [TT94]. Un environnement virtuel est simulé pour créer des comportements de recherche de nourriture, de prédation et d'accouplement. Ce système est composé de trois sous systèmes responsables de la perception, de la sélection du comportement et du contrôle moteur des entités. Des senseurs virtuels fournissent des informations sur la dynamique de l'environnement. Le système comportemental établit un lien entre la perception et le système moteur en déterminant les paramètres de contrôle des actions en fonction des perceptions couplées avec un générateur d'intention.

Le système moteur comprend un modèle dynamique de l'entité accompagné d'un ensemble de contrôleurs dédiés à la production d'un mouvement spécifique. Il décompose un ordre de haut niveau (tourner à gauche par exemple) en activations des contrôles moteurs nécessaires pour réaliser cette action (muscles).

Le résultat présente les évolutions de poissons, dans des fonds sous-marin, établies à partir de contractions musculaires, conséquence de la perception de l'environnement.

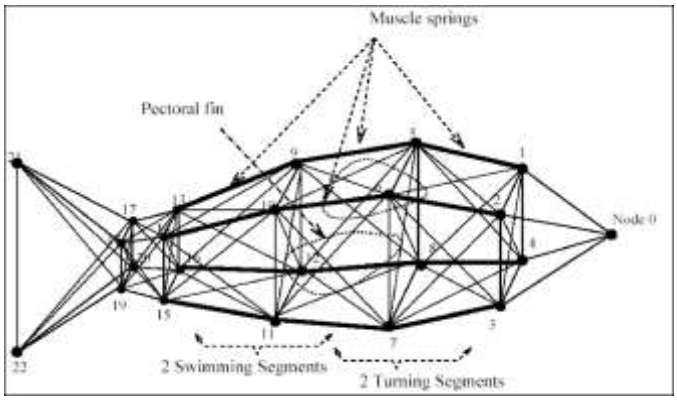


Figure 1.10 : Modèle dynamique masse/ressort d'un poisson de Terzopoulos

1.2.3.4.5. Approche basée théorie des schèmes de R. Arkin

Les théories des schèmes et des réseaux neuronaux correspondent aux deux principaux modèles informatiques utilisés pour représenter le fonctionnement du cerveau. Le schème est défini comme étant l'unité de base du comportement: il encapsule à la fois des connaissances lui permettant de réagir et un processus correspondant à la façon dont il doit réagir. Le comportement global émerge du contrôle des activités concurrentes des différents schèmes.

En 1998, R.Arkin propose une approche basée sur les schèmes moteurs, appliquée à la navigation de robots autonomes [Ark98]. La particularité de cette approche est la représentation des schèmes moteurs selon un format unique: le vecteur d'action. Ce vecteur est généré selon des méthodes à base de champs de potentiels considérant les cibles comme des attracteurs et les obstacles comme étant répulsifs.

La coordination entre les schèmes moteurs se fait alors simplement en additionnant l'ensemble des vecteurs calculés par les schèmes actifs, après multiplication de chaque vecteur par un poids dynamique associé à chaque schème. Il en résulte un unique vecteur d'actions correspondant à la direction que le robot doit suivre. Ainsi, chaque unité comportementale participe à l'émergence du comportement global.

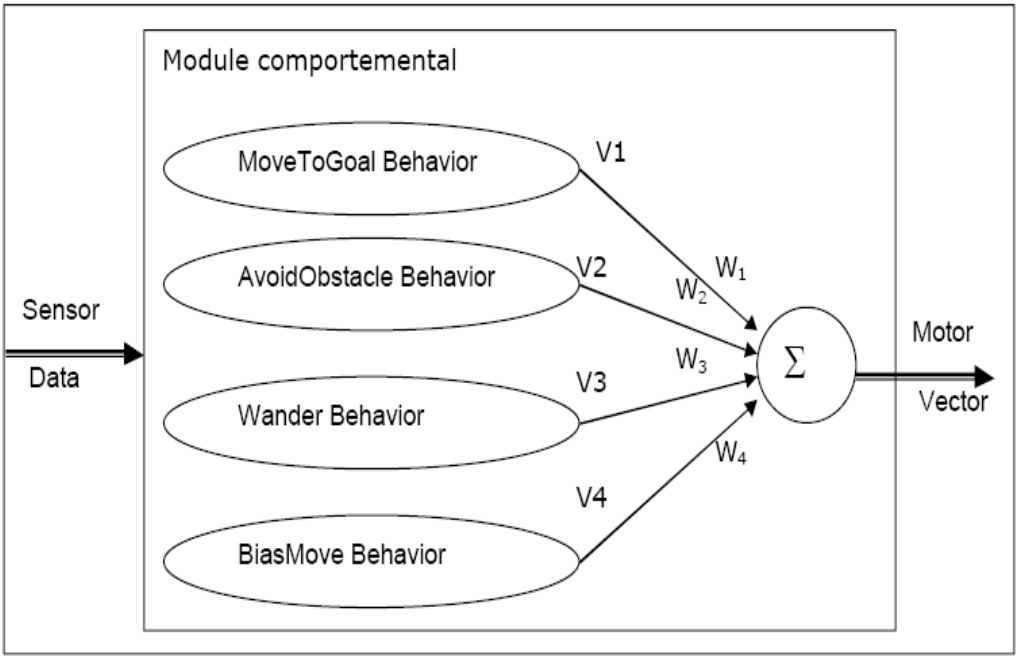


Figure 1.11 : Contrôle comportemental à base de schèmes moteurs

Un schème moteur génère un vecteur d'action à partir des informations fournies par les schèmes de perception. Selon les informations perçues dans l'environnement, chaque schème peut créer des stimuli utiles au schème moteur auquel il est attaché.

Le principal inconvénient de cette méthode est lié aux minimums locaux même si de nombreuses techniques permettent de limiter les risques de se retrouver dans une solution non optimale. Cette problématique est fortement liée à des problèmes de déplacement de robots autonomes et peut difficilement s'appliquer à d'autres domaines.

1.2.3.4.6. Approche ascendante de P. MAES

L'architecture ascendante de P. Maes [Mae89] s'appuie sur une sélection de l'action qualifiée d'ascendante. A partir de comportements de base d'un agent (s'approcher d'une source, boire…), on établit un réseau non hiérarchique de nœuds d'actions. Un seul nœud peut être exécuté simultanément sachant que, lors de l'activation, plusieurs nœuds peuvent être sélectionnés. Le choix se porte alors sur le nœud de niveau le plus élevé, ce niveau décroissant automatiquement à chaque pas de réaction de l'agent.

Un nœud perçoit l'environnement de l'agent sous forme de pré-conditions correspondant à la présence ou à l'absence de certaines caractéristiques de l'environnement au moment de leur évaluation. Un nœud est éligible seulement si toutes ses pré-conditions sont satisfaites, alors que tous les nœuds participent au choix du nœud à exécuter. Chaque nœud calcule son niveau d'activation en fonction des motivations de l'agent et de sa connexion avec d'autres nœuds. Ces connexions permettent de propager une excitation ou une inhibition, c'est-à-dire d'augmenter ou de réduire le niveau d'activation d'autres nœuds.

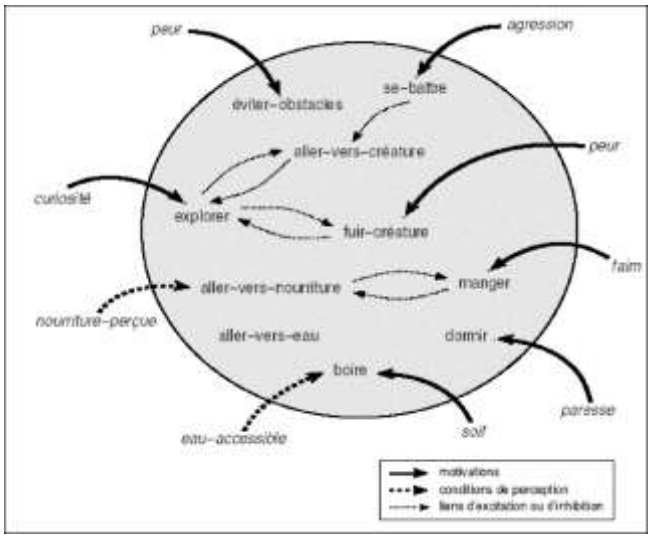


Figure 1.12 : Exemple d'architecture ascendante de P. MAES

Ce mécanisme est intéressant lorsqu'il faut interrompre un comportement en situation d'urgence ou pour profiter d'opportunités. Néanmoins, il ne gère pas les comportements conflictuels en particulier, car aucun mécanisme ne permet de combiner les actions des comportements.

1.2.3.4.7. Humains virtuels de N. Badler

Le système Jack [BPW93] s'appuie sur la planification de tâche et la simulation biomécanique pour animer des humains virtuels.

Ce modèle se décompose en deux parties: la première partie utilise un réseau d'automates qui contrôlent des comportements de haut niveau. La deuxième partie se concentre sur les capacités motrices qui manipulent la géométrie d'un agent. Cette couche de bas niveau est gérée par une boucle réactive appelée SCA (Sense Control Action).

B Reich [Rei97] a étendu ce modèle en se concentrant sur la locomotion et en décomposant la boucle réactive en un moteur d'animation et un ensemble de comportements de bas niveaux appelés comportements instantanés. A chaque comportement est associée une fonction d'évaluation qui est utilisée pour déterminer l'action qui sera choisie. Ainsi à chaque pas de la simulation, le système évalue pour chaque état possible de la simulation au pas de temps suivant, une valeur correspondant à l'efficacité de l'action choisie. Pour cela, il suffit de sommer les valeurs d'évaluation de chacun des comportements instantanés.

On retrouve deux types de comportements instantanés: Les comportements de niveau 0 correspondant aux comportements réflexes (évitement d'obstacle) et des fonctions permettant de satisfaire les contraintes physiques (inertie, gravité). Les comportements de niveau 1 regroupent les comportements permettant de résoudre les buts que l'entité cherche à atteindre (essentiellement des comportements d'attraction).

Les comportements de niveau 0 sont toujours actifs alors que les comportements de niveau 1 sont activés par les comportements de plus haut niveau appelé PaTNets (Parallel Transition Networks).

Ces comportements de haut niveau se basent sur des automates exécutés en parallèles. Chaque automate représente une action que l'humain virtuel peut exécuter en complément de sa locomotion. Il possède aussi un espace permettant de mémoriser des valeurs dans le but de le guider dans ses actions. Ce modèle a été mis en œuvre dans une simulation d'humain virtuel appelé Jack. (Figure 1.13).

Figure 1.13 : Jack dans une Ford Fiesta

1.2.3.4.8. Architecture de sélection de l'action de B. Blumberg

Dans le but de faciliter la compréhension des modèles comportementaux, B. Blumberg a choisi de séparer comportements et compétences motrices [BDI+02]. Dans ce cas, un comportement correspond à un système régit par les objectifs qu'il cherche à atteindre. Il s'appuie pour cela sur son système sensoriel. Chaque comportement peut générer une séquence d'actions. Cette séquence sera transcrite pour modifier les caractéristiques géométriques de l'entité par un module appelé contrôleur. On associe donc à une compétence motrice un contrôleur.

Une des particularités de ce modèle est de ne gérer qu'un comportement à la fois. Cependant, les résultats des différents modules comportementaux influent sur le résultat final. La sélection du comportement s'effectue à l'aide d'une architecture connexionniste. Chaque nœud représente un comportement de base qui est relié au système perceptif et éventuellement à d'autres nœuds afin de lier les comportements entre eux. Un système d'évaluation en fonction des flux d'entrée permet d'évaluer chacun de ces nœuds. La dernière étape consiste alors à sélectionner celui ayant la valeur la plus élevée.

Le niveau perceptif s'appuie sur la notion de filtres qui permettent d'extraire les événements et les objets pertinents des informations brutes en provenance de l'environnement.

La compétence motrice est définie par le comportement actif. Les autres comportements ne proposent alors que des actions secondaires à appliquer si la première tâche s'est terminée avant la fin du temps imparti. Ils peuvent aussi proposer des méta-actions précisant la méthode pour exécuter l'action principale.

A tous ces éléments viennent se greffer deux mécanismes: la fatigue et l'inhibition permettant de trouver un équilibre entre des comportements persistants et des comportements trop opportunistes. Ainsi, la fatigue est un paramètre réduisant l'influence des comportements activés dans le but de pouvoir activer les comportements les moins prioritaires. D'un autre côté, l'inhibition va influencer les liens entre les comportements de manière à en favoriser certains au détriment d'autres.

Enfin, Blumberg a regroupé les comportements en groupes. Chaque groupe comprend tous les comportements pouvant mener à la résolution d'une problématique. Ainsi le groupe «se déplacer d'un endroit à un autre» comprend les comportements «déplacement dans différentes directions» et «Évitement d'obstacles». Certains comportements peuvent appartenir à plusieurs groupes. Dans ce cas son évaluation entraîne, soit une influence pour la sélection de l'action à l'intérieur du groupe, soit un événement sur un contrôleur.

Cette architecture complexe a permis de réaliser des animats au comportement efficace tel que le chien. Elle permet de lier efficacement simulation comportementale et animation mais reste néanmoins difficile à mettre en œuvre.

1.2.3.4.9. Approche orientée émotion de P. Becheiraz

P. Becheiraz a fait le choix de créer une structure comportementale pour animer des humains virtuels utilisant les émotions afin de leur rajouter de la crédibilité [BT98]. En complément des modules perception/comportement/action, il ajoute un module d'émotion. Le module comportemental utilise alors ce nouveau module d'émotion ainsi que la perception pour sélectionner un comportement. (Voir Figure 1.1)

Un potentiel d'émotion est calculé pour un groupe d'éléments engendrant une émotion. Si le potentiel est au dessus d'un certain seuil, sa valeur est utilisée pour calculer l'intensité de l'émotion. L'ajout d'une mémoire d'émotion permet d'obtenir des acteurs aux comportements variés car possédants des expériences différentes.

Enfin, les comportements se décomposent en une hiérarchie de comportements élémentaires pouvant être traités de manière séquentielle, ou concurrentielle. Des liens d'inhibition donnent la possibilité d'éviter les conflits lorsque deux comportements mutuellement exclusifs doivent être exécutés en parallèle.

Le module d'action autorise l'exécution parallèle et séquentielle des actions permettant une grande variété d'expression mélangeant gestes et animations faciales.

1.2.3.4.10. Modèle HPTS de S. Donikian

Dans le but de fournir un modèle comportemental pour des environnements virtuels, S. Donikian a développé une architecture basée sur une hiérarchie de modules constitués d'automates parallèles: HTPS (Hierarchical Parallel Transition System) [Don01]. Elle fut mise en œuvre dans plusieurs exemples de simulation de conducteurs autonomes.

Donikian se base sur la composition de plusieurs sous-automates qui vont travailler sur la sélection de l'action. Un automate est composé d'entrées, de sorties, de paramètres de contrôle, d'une boîte aux lettres, et éventuellement d'autres automates.

Les paramètres de contrôle permettent d'influencer le comportement de l'automate. Ils sont gérés par l'automate lui-même ou par un automate extérieur.

Une fonction d'intégration permet de gérer, trier ou synthétiser l'ensemble des réponses fournies par les sous parties. On retrouve parmi les types de fonctions d'intégration, 5 types de fonctions.

Le premier type est une fonction classique manipulant des opérateurs de calcul ou de comparaison sur des types de base (entiers, réels et booléens). Le deuxième type de fonction se base sur une certaine saturation ou sur un seuillage pour réaliser une fonction de filtrage. L'utilisation de priorités permet de définir un troisième type de fonction dans le cas de réponses concurrentes. Un opérateur temporaire de retard permet d'ordonnancer les données.
Enfin une fonction d'intégration peut être composée de plusieurs des fonctions précédemment décrites.
Dans le cas d'un automate ne comportant aucune sous-partie, le résultat ne sera que fonction des entrées, de son état interne et des paramètres de contrôle. Finalement, la boîte aux lettres permet de communiquer avec d'autres automates (qu'ils soient supérieurs d'un point de vue hiérarchique ou de simples sous parties.). Les messages correspondent, soit à des requêtes pour obtenir le statut d'un automate (actif, inactif, suspendu), soit à des ordres pour contrôler l'activité d'un automate (lancement, suspension, reprise, terminaison).

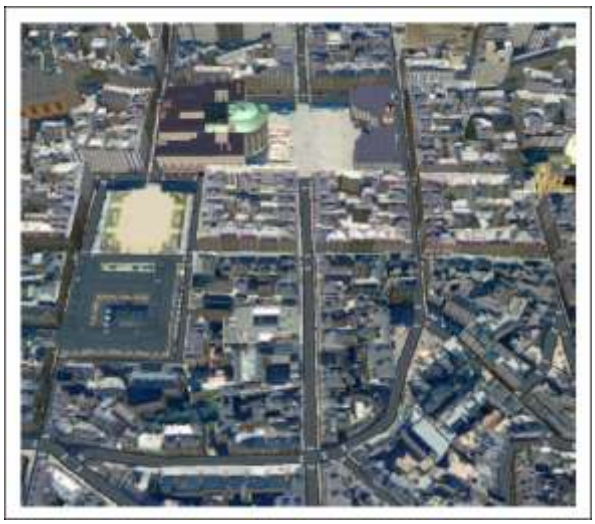


Figure 1.14 : Ville de Rennes virtuelle peuplée par des agents basés sur l'architecture HPTS

Le principal avantage de cette méthode réside dans sa modularité: en effet, un automate permet notamment de décrire aussi bien un module comportemental, que des capteurs ou des effecteurs. De plus, l'aspect distribué permet de répartir les automates en fonction de la puissance de calcul disponible. Cependant, leur utilisation est fortement liée à la plateforme de simulation GASP.

1.2.3.4.11. Approche basé classifieurs: C. Sanza

Dans le but de fournir un système comportemental adaptatif, Sanza [SDD99] s'est appuyé sur un système de classifieurs (aCS) pour modéliser le comportement de ses entités. Il se base donc sur une base de règles pour définir le comportement de ses entités La sélection de la règle, ou du classifieur, à appliquer s'effectue en deux phases:

La première étape présélectionne les classifieurs dont la partie condition correspond à l'état de l'environnement. La seconde permet de choisir parmi tous les candidats potentiels celui qui devrait fournir la meilleure réponse à l'état courant. Ainsi, en pondérant la force de chacune des règles avec sa spécificité (c'est-à-dire la précision avec laquelle la partie condition du classifieur décrit son environnement) et en ajoutant un bruit on obtient une valeur représentative de la règle. Une sélection stochastique permet alors de déterminer la règle choisie. Il ne reste plus qu'à activer l'effecteur déterminé par la partie action du classifieur.

En plus de la mécanique de sélection de l'action, ce modèle rajoute un élément de coordination entre entités: La communication, et par conséquent l'échange de règles, permet de profiter de l'expérience de plusieurs entités simultanément afin d'obtenir un système plus robuste.

Il ajoute également la possibilité de créer un dialogue entité/utilisateur au moyen d'un générateur syntaxique.

L'aspect adaptatif et collectif du modèle a permis la réalisation d'une simulation (une séquence d'un match de football) dans laquelle chaque entité est contrôlée par ce système. Les résultats sont pertinents et accompagné d'une forte émergence de comportements collectifs. Cependant, ce système présente l'inconvénient de n'activer qu'un effecteur à la fois, limitant ainsi les applications du modèle. De plus, il n'offre pas la possibilité d'intégrer d'autres méthodes de contrôle des entités dans le but de pouvoir comparer les résultats face à des situations prédéterminées.

1.2.3.4.12. Architectures basées sur les réseaux de neurones

Un réseau de neurones artificiel est constitué par l'interconnexion de cellules élémentaires: les neurones formels. Le neurone formel, à l'instar de son homonyme biologique, a pour unique tâche de transmettre un influx nerveux sous certaines conditions définies par une fonction de transfert. Le principe de fonctionnement du neurone formel de McCulloch et Pitts est le suivant [MP43]: Les potentiels présents sur chaque entrée du neurone sont additionnés puis une fonction d'activation décide de propager le signal ou non sur son unique sortie. C'est l'organisation du réseau de neurones qui permet à l'ensemble d'exprimer tout son potentiel. En simulation comportementale, les architectures sont essentiellement basées sur le modèle de la perception multicouche de Marvin Minsky et Seymour Papert [Rey99]. Dans ce dernier, les neurones sont organisés en couches: une couche d'entrée, une couche de sortie et une couche cachée lui permettant de résoudre les problèmes linéairement séparables. La modification des poids (représentant leur force d'activation) de chaque nœud peut se faire dynamiquement à l'aide d'un système d'apprentissage (une méthode d'apprentissage par renforcement ou, plus rarement, un algorithme génétique).

Bien qu'intéressante par ses capacités réactives importantes, l'approche stimulus-réponse des réseaux de neurones est d'un niveau d'abstraction faible et, ne permet donc, généralement que la modélisation de comportements instinctifs (généralement liés aux mouvements d'une entité virtuelle ou à son déplacement dans l'environnement). De plus, cette technique manque de souplesse d'adaptation, la modification de l'environnement ou d'une réaction à un stimulus nécessitant un nouveau calcul complet des paramètres du réseau. Enfin, les réseaux obtenus par adaptation dynamique à l'environnement sont souvent complexes et, par conséquent,

relativement opaques si on souhaite analyser en profondeur les mécanismes du comportement généré.

Les *Sensor Actuator Network (SAN)*

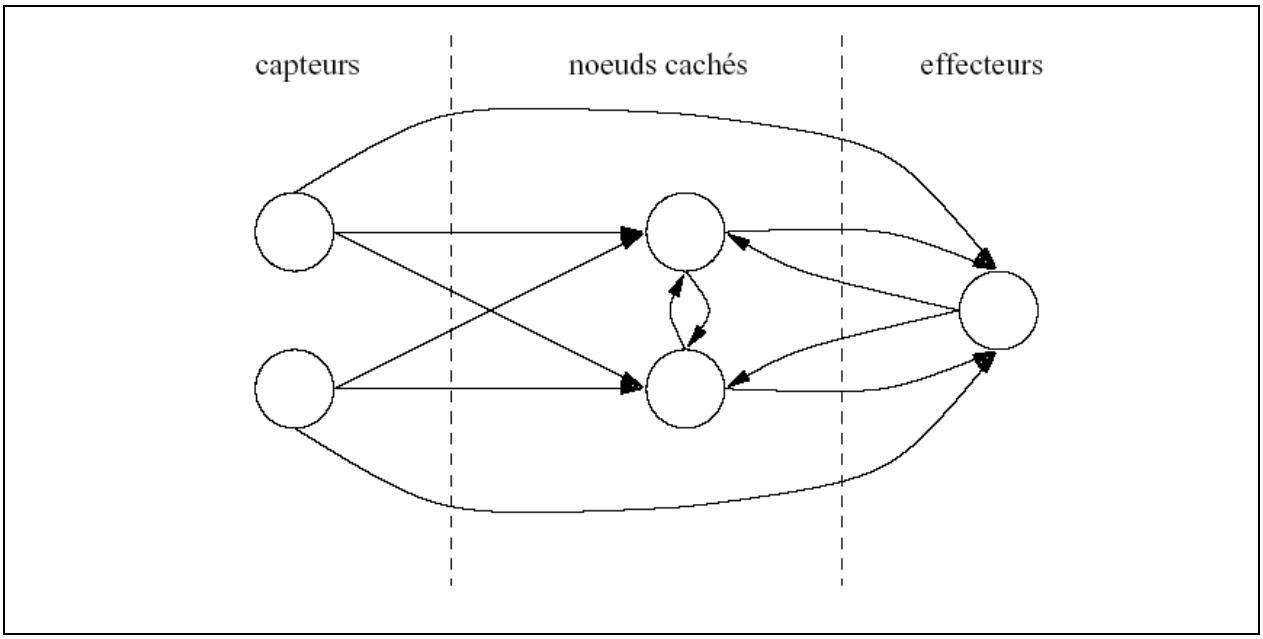


Figure 1.15 : Sensor Actuator Network

Van de Panne utilise des réseaux de neurones, baptisés Sensor Actuator Network pour animer des créatures virtuelles sur des terrains accidentés. L'utilisateur spécifie la configuration de la créature (effecteurs et capteurs). Les capteurs sont connectés à la fois à la couche cachée et aux effecteurs. Le poids d'activation des différents nœuds est calculé à l'aide d'un algorithme stochastique. Les résultats obtenus sont des créatures capables de se déplacer en fonction des mouvements générés par les activations successives des effecteurs en fonction de l'environnement (certains rampent, d'autres sautent, marchent, …).

1.2.4. L'action

Basé sur l'information perceptuelle, le mécanisme comportemental d'un acteur détermine les actions à exécuter. Les actions peuvent avoir plusieurs degrés de complexité. Un acteur peut se développer dans son environnement ou bien agir réciproquement avec l'environnement ou encore communiquer avec d'autres acteurs.

Les actions sont exécutées en employant une architecture de mouvement commune. Le module d'action gère l'exécution des actions employées par un comportement en animant un modèle d'homme générique basé sur une hiérarchie de nœud. Il permet l'exécution simultanée

ou séquentielle d'actions en gérant des transitions lisses entre des actions finales et des actions amorçantes [BMT95]. Une boucle comportementale conduit l'animation, son rôle est de mettre à jour l'état du monde virtuel. Le but est de déclencher est d'arriver à la solution.

A chaque itération, le temps est incrémenté, le monde virtuel est mis à jour avec en particulier une mise à jour de l'état de chaque objet et acteur. Dans le cas d'un acteur, la perception est d'abord exécutée, après ses émotions sont produites avant que son comportement et ses actions ne soient exécutés.

1.2.5. La Mémoire

La mémoire est d'habitude définie comme le pouvoir ou le processus de reproduction ou de rappel de ce qui a été appris et conservé, particulièrement par les mécanismes associatifs.

La mémoire est aussi le dépôt pour des informations apprises et à conserver, générées à partir de l'activité d'un organisme ou d'une expérience.

La mise en œuvre de la mémoire pour un acteur n'est pas très complexe, comme la mémoire est déjà un concept clef en informatique. Par exemple, dans [NT97], les auteurs proposent une troisième mémoire visuelle globale qui permet à un acteur de retenir l'environnement, de le percevoir et de s'adapter à ses changements.

1.3. Domaines d'application de la simulation comportementale

Les domaines d'application de la simulation comportementale sont variés et s'étendent des domaines ludiques tels que les jeux vidéo aux domaines de la médecine et des soins des phobies.

1.3.1. Jeux vidéo

Avec l'évolution de la puissance de calcul des ordinateurs, allant de paire avec la puissance des cartes graphiques, une bonne qualité de rendu graphique peut désormais être obtenue en utilisant relativement peu de temps processeur. Cette évolution permet aux concepteurs de jeux vidéo de se concentrer sur le comportement des personnages avec lesquels les joueurs peuvent interagir. Deux exemples notables sont les jeux de créatures de la société Cyberlife, et Black and White de la société Lionhead, qui proposent des mondes peuplés de créatures autonomes. Ces créatures peuvent prendre des décisions seules, l'intervention du joueur, qui peut cependant, dans certaines circonstances influer sur leur comportement est de tenter de leur apprendre des choses. Le développement de la simulation comportementale a fait

naître une nouvelle tendance: la fiction interactive [CCM02]. L'idée est, ici, de fournir les grandes lignes d'un scénario, tout en laissant le joueur libre d'interagir à son gré avec des entités semi-autonomes. Ces entités suivent la ligne directrice du scénario mais peuvent prendre des décisions pour interagir «intelligemment» avec le joueur. Cela donne une grande sensation de liberté, tout en pouvant générer des situations imprévues faces aux actions de l'utilisateur.

Dans le domaine de jeu, outre la qualité des animations et des graphiques, la crédibilité et la cohérence du comportement sont primordiales pour le réalisme et l'implication du joueur.

1.3.2. Films et effets spéciaux

La simulation comportementale a fait son apparition dans le monde de la cinématographie et des effets spéciaux depuis déjà un certain temps. La tendance actuelle est d'accroître son utilisation, à travers le développement d'outils de modélisation du comportement pour les logiciels de production d'animation en 3D. Cette tendance s'explique par un besoin de productivité accru, à travers des scènes de plus en plus complexes mais allant de paire avec des exigences de qualité croissantes. L'un des meilleurs exemples actuels est le deuxième volet de l'adaptation cinématographique du *seigneur des anneaux*. Pour la grande bataille du *gouffre de Helm*, un outil spécifique: MASSIVE, a été utilisé pour reproduire les comportements de foule et modéliser la composante décisionnelle des entités. Dans le même domaine, divers outils sont désormais disponibles, parmi lesquels AI-Implant de la société BIOGRAPHICS, qui peut être utilisé sous la forme de module pour 3D studio Max et Maya, ou bien encore les modules RTK-Crowd et RTK-Behavior de SOFTIMAGE. En termes de conception, ces outils offrent de grands avantages:

- Le travail du concepteur est simplifié par la gestion automatisée des comportements de groupe;
- Les modèles décisionnels permettent de générer automatiquement des animations adaptées au contexte;
- Les scènes paraissent plus réelles car elles sont plus variées.

Grâce à l'automatisation, ces techniques permettent de décrire, de manière implicite, des scènes d'une complexité inégalable grâce à l'aide d'outils dédiés.

1.3.3. Validité ergonomique des sites

La création de lieux publics pose des problèmes d'ordre ergonomique, autrement dit, relatifs à la qualité de leur utilisation. Des problèmes peuvent se poser quant à la bonne navigation à l'intérieur des lieux, la lisibilité des divers panneaux de direction, ou lors de situations de panique [HFV00]. L'animation comportementale, à travers des modèles se basant sur l'analyse du comportement humain, peut être utilisée pour effectuer des validations sur des maquettes virtuelles. Les éventuels problèmes peuvent alors être détectés et corrigés avant la construction des divers aménagements. Dans ce cadre, son utilisation offre des atouts qui sont d'ordres sécuritaire et ergonomique mais aussi d'ordre économique.

1.3.4. Mise en situation

L'interaction avec des agents autonomes, à travers la réalité virtuelle, permet de mettre un être humain en situation dans le cadre d'un scénario. Ces capacités peuvent être utilisées dans un cadre pédagogique où l'interaction avec des humanoïdes intelligents aura pour conséquences d'augmenter la rapidité d'apprentissage par l'intermédiaire de moyens d'interaction plus proches de la réalité [LTG+00].

Un autre domaine d'application concerne l'armée où la définition du comportement d'humanoïdes permet de tester des scénarios catastrophes ou de mettre les soldats en situation [TRG+03]. Dans le domaine hospitalier, l'animation comportementale, couplée à la réalité virtuelle, peut être utilisée dans le cadre de soins aux personnes phobiques [KV03]. L'idée consiste à plonger la personne, par l'utilisation de la réalité virtuelle, dans un milieu qui déclenche sa phobie. Dans le cadre du soin de l'agoraphobie, par exemple, les simulations de foules s'avèrent utiles, pour peupler les environnements virtuels et fournir au patient un milieu réaliste.

1.4. Conclusion

Ces domaines d'applications variés, justifient les recherches effectuées en simulation comportementale, et particulièrement le besoin d'établir un lien avec les études effectuées sur le comportement humain. Les modèles proposés, pour permettre d'obtenir un réalisme suffisant, doivent généralement prendre en compte les caractéristiques propres au comportement humain.

Nous avons présenté au niveau de ce chapitre la modélisation des propriétés des humains virtuels ainsi nous avons cité la plupart des modèles comportementaux utilisés. Dans le chapitre suivant, nous allons présenter un comportement très utilisé au niveau de la simulation comportementale : c'est le comportement *de recherche de chemin.*

Chapitre 2

2. Le pathfinding

2.1. Introduction

Qu'est ce que le pathfinding ?

Pour déplacer un personnage d'un point à un autre point dans une scène (2D ou 3D), on peut faire appel à deux grandes méthodes: le personnage peut être déplacé avec les touches directionnelles du clavier, et dans le cas d'un jeu 2D, il avancera case par case ou pixel par pixel; autrement, il peut indiquer le point où il veut se rendre et le déplacement se fait alors automatiquement. Dans le second cas, le moteur devra calculer le chemin optimal entre le point de départ et le point d'arrivée. C'est le Pathfinding (ou recherche de chemin). Le Pathfinding est aussi fréquemment utilisé dans la prévision et la gestion des déplacements des ennemis [Amia]**.**

Nous nous sommes intéressés à établir une étude informatique large, des algorithmes de plus courts chemins, basée sur les algorithmes les plus récents. Nous nous proposons ainsi de proposer des algorithmes nouveaux motivés par les résultats empiriques. Ces algorithmes nous ont menés à des résultats théoriques intéressants confortés par les observations empiriques. Notre étude informatique est basée sur plusieurs classes de problèmes naturels qui identifient les forces et les faiblesses d'algorithmes divers.

Ces classes de problèmes et la mise en œuvre d'algorithmes palliatifs, forment un environnement pour évaluer l'exécution des algorithmes de plus courts chemins. L'interaction

entre l'évaluation expérimentale du comportement des algorithmes et l'analyse théorique de l'exécution de ces algorithmes joue un rôle important dans notre recherche.

Nous décrivons des algorithmes pour la détermination de plus courts chemins et des distances dans des graphes plats qui exploitent la topologie particulière de l'entrée graphique. Une particularité importante de nos algorithmes réside dans le fait qu'ils peuvent travailler dans un environnement dynamique, où le coût de n'importe quelle entité peut être modifié, l'entité pouvant, également, être supprimée. Pour un objet simple, le mouvement semble facile tout en étant novateur.

Aspect novateur

Considérons la situation suivante:

Le but de notre acteur est de parvenir au sommet. Ne détectant aucun obstacle dans le secteur, il parcourt alors son chemin vers la cible (en rose). Près du sommet, il détecte un obstacle et change la direction. Il trouve alors sa voie le long de l'obstacle de forme « U » jusqu'à aboutir à la cible. Au contraire, un acteur averti aurait considéré un plus grand secteur (en bleu clair), mais n'aurait trouvé aucun chemin plus court (bleu), car ne détectant la cible celle-ci étant dans le champ de l'obstacle concave formé (figure 2.1).

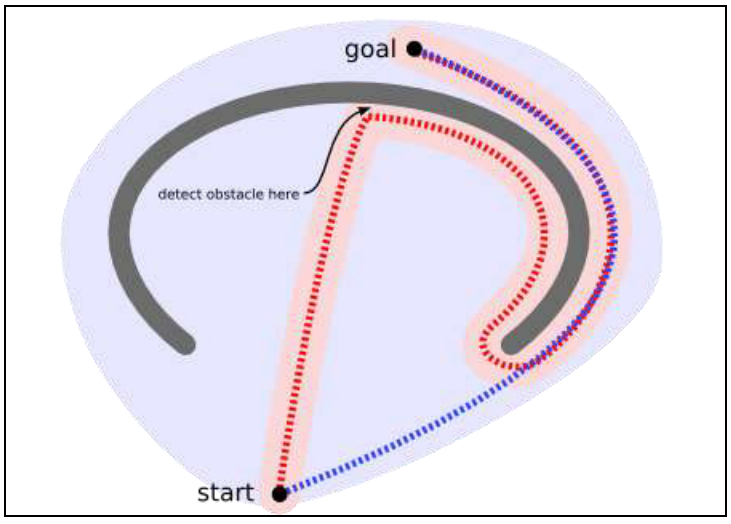


Figure 2.1 : Le mouvement pour un objet simple.

On peut cependant exploiter un algorithme de recherche pour travailler autour des pièges (figure 2.2). Le choix pourrait être l'évitement de création d'obstacles concaves, ou bien le marquage de la coque convexe comme étant dangereuse, l'acteur ne tentera une pénétration à l'intérieur de cette zone que s'il s'est avéré que le but est à l'intérieur.

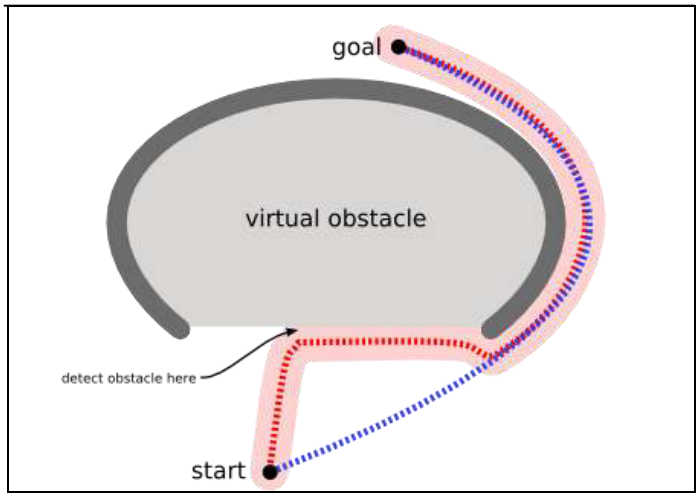


Figure 2.2 : Le mouvement pour l'évitement des obstacles

2.2. Les divers algorithmes de Pathfinding

Les algorithmes novateurs de recherche manuelle de chemins travaillent sur des graphiques, dans le sens mathématique. Un jeu de sommets avec bords: une carte de jeu carrelée peut être considérée comme un graphique dont chaque tuile représentant un sommet, les bords considérés comme des tuiles adjacents les uns des autres (figure 2.3).

Les algorithmes les plus novateurs de l'intelligence artificielle (algorithmes de recherche), sont conçus pour des graphiques arbitraires plutôt que des jeux basés sur réseau. La difficulté majeure réside dans le fait qu'il existe des aspects du comportement humain qui étaient naguère difficilement transposable pour une prise en charge informatique dans le but de l'animation d'acteurs virtuels. Nous nous intéressons dans ce qui suit à décrire un certain nombre de contributions et tentatives pour la prise en charge du problème de recherche de chemins.

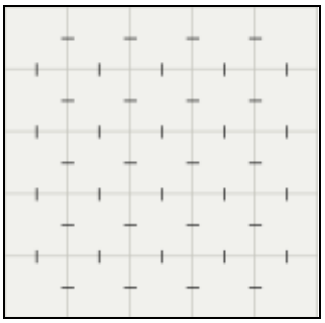


Figure 2.3 : Le jeu de vertices

2.2.1. La solution du fainéant

Une des premières solutions consiste à tracer une ligne droite entre les points de départ et d'arrivée et de demander à l'acteur ou l'agent de suivre cette ligne. S'il rencontre un obstacle, il le contourne par la droite ou par la gauche.

Ceci fonctionne plutôt bien pour des obstacles convexes mais présente beaucoup de faiblesses dans les cas des obstacles concaves. Il est impensable de l'utiliser tout seul dans tout jeu qui se respecte. Néanmoins, cet algorithme, couplé à un ou plusieurs autres, peut s'avérer très utile.

2.2.2. L'algorithme de Dijkstra

Les travaux de l'algorithme de ***Dijkstra*** consistent en un parcours des nœuds dans un graphe et ce en commençant avec le point de départ de l'objet. L'algorithme de ***Dijkstra*** calcule le chemin le plus court entre deux nœuds d'un réseau. L'idée de base est de maintenir pour chaque ensemble de nœuds P le chemin le plus court ayant été trouvé. Chaque nœud extérieur à P doit être atteint à partir d'un nœud déjà dans P. Il s'étend ainsi à l'extérieur du point de départ avant qu'il n'accède le but.

Le principe est relativement similaire:

- L'acteur part du point de départ puis calcule le poids de ses points adjacents;
- Il examine le point dont le poids est le plus faible;
- Il calcule, ensuite, le poids de ses points adjacents. Leurs poids sont éventuellement mis à jour (s'il est plus faible);
- Ensuite, il passe au point dont le poids est le plus faible parmi l'ensemble des points.
- Etc…….

L'implémentation d'un tel algorithme s'effectue en utilisant une structure de file. Quand un point est placé dans la file, il l'est en fonction de son poids calculé. Le nouveau point à examiner est alors celui dont le poids et le plus faible, qui se trouve en tête de la file. Cette fois-ci, le poids de la cellule est pris en compte, mais, comme l'algorithme précédent, les directions sont équivalentes.

Avantage

L'algorithme de Dijkstra garantit de trouver le plus court chemin du point de départ vers le point d'arrivée, tant qu'aucun des bords n'a un coût négatif. Dans le diagramme suivant, le carreau rose est le point de départ, le carreau bleu représente le but et l'exposition

de secteurs de sarcelle (d'élimination) les secteurs que l'algorithme de Dijkstra a parcouru. Les secteurs de sarcelle les plus légers sont les plus éloignés du point de départ et forment ainsi "la frontière" de l'exploration (Figure 2.4).

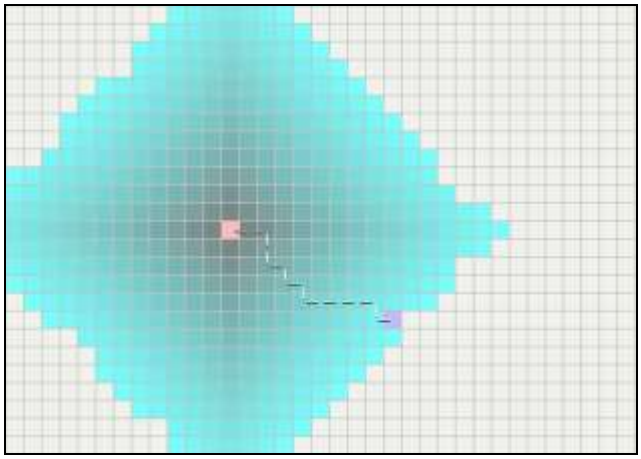


Figure 2.4 : L'algorithme de Dijkstra

2.2.3. La recherche du meilleur premier (BFS Breadth First Search)

L'algorithme travaille d'une façon semblable, sauf qu'il doit faire quelques évaluations (appelées heuristique). Au lieu de la sélection du sommet le plus proche au point de départ, il choisit le sommet le plus proche au but.

Inconvénient

BFS ne garantit pas de trouver le plus court chemin.

Avantage

Cet algorithme converge beaucoup plus rapidement que celui proposé par Dijkstra. Ceci est dû au fait qu'il emploie la fonction heuristique pour guider sa voie vers le but très rapidement.

Par exemple, si le but est au sud de la position de départ, l'algorithme BFS aura tendance à se concentrer sur des chemins orientés vers le sud. Dans le diagramme suivant, le jaune représente ces nœuds avec une haute valeur heuristique (coût élevé pour atteindre le but). Le noir représente, par contre, des nœuds avec une valeur heuristique basse (coût bas pour arriver au but). Il montre que l'algorithme BFS peut trouver des chemins très rapidement comparativement à l'algorithme de Dijkstra.

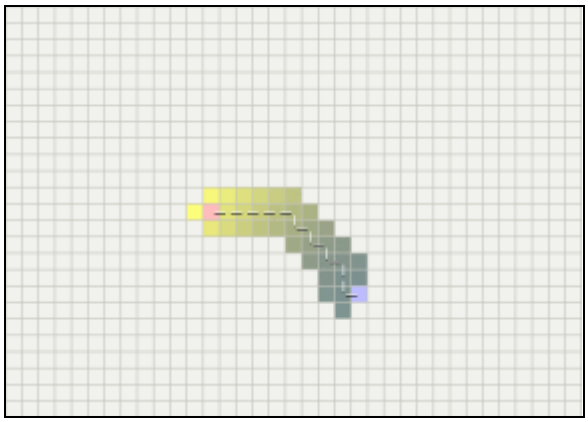


Figure 2.5 : La Recherche premier mieux (BFS)

Cependant, ces deux exemples illustrent le cas le plus simple. Quand la carte ne présente aucun obstacle, et le chemin le plus court est vraiment une ligne droite. Considérons l'obstacle concave comme décrit dans la section précédente. L'algorithme de Dijkstra travaille plus difficilement, mais garantit de trouver le plus court chemin (figure 2.6).

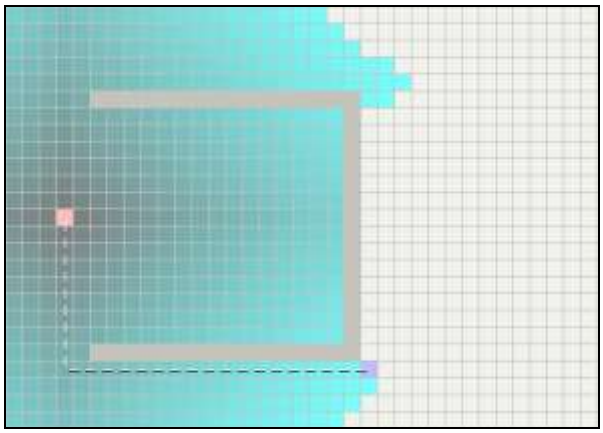


Figure 2.6 : Dijkstra avec des obstacles concaves

L'algorithme BFS nécessite, d'autre part, moins d'effort mais son chemin n'est pas visiblement le meilleur (figure 2.7).

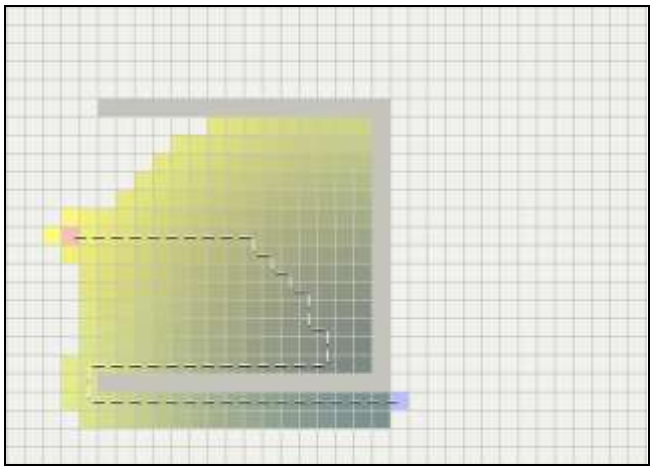


Figure 2.7 : BFS avec des obstacles concaves

Le trouble consiste en ce que BFS est passionné, et essaye de se déplacer vers le but même si ce n'est pas le bon chemin. Puisqu'il considère uniquement le coût lui permettant d'arriver au but, et ignore le coût du chemin parcouru jusqu'ici, il continue à avancer, même si le coût du chemin sur lequel il avance est devenu très élevé.

Il ne serait pas agréable de combiner le mieux d'entre toute les deux ?

L'algorithme A* a été développé en 1968 [Amia], pour combiner des approches heuristiques comme l'algorithme BFS, et des approches formelles comme l'algorithme de Dijsktra. C'est un peu commun dans des approches heuristiques, comme l'algorithme BFS donne d'habitude une façon approximative de résoudre un problème, sans garantir que l'obtention de la meilleure réponse. Cependant, l'algorithme A* est construit sur le sommet de l'heuristique, et bien que l'heuristique elle-même ne donne pas une garantie, l'algorithme A* peut garantir le chemin le plus court.

2.2.4. L'algorithme A*

Assez flexible et pouvant être employé dans un grand choix de contextes, l'algorithme A* a toujours été un choix des plus populaires [Amia].

De même que pour d'autres algorithmes de recherche graphique, l'algorithme A* permet d'effectuer des recherches virtuellement et ce dans un secteur énorme du domaine de recherche. Il est semblable à l'algorithme de Dijkstra, lequel pouvant être employé pour trouver le plus court chemin. Il est également proche de l'algorithme BFS, lequel peut employer une heuristique pour pouvoir se guider. Dans les cas les plus simples, il offre des vitesses de convergence proches de celles associées à l'algorithme BFS

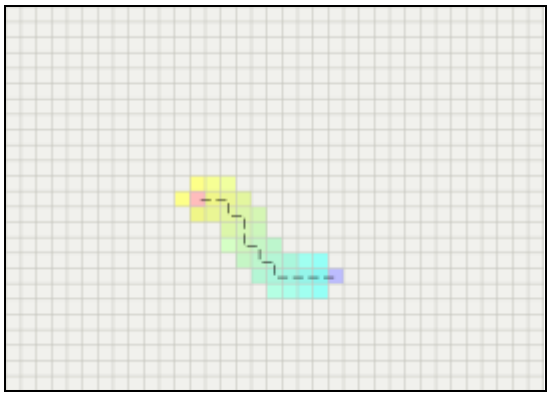


Figure 2.8 : La recherche graphique dans un secteur sans obstacles due plus cours chemin

Dans l'exemple avec un obstacle concave, A* trouve un chemin aussi bon que celui trouvé par l'algorithme de Dijkstra:

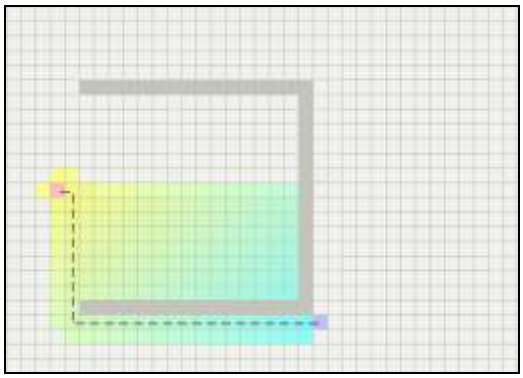


Figure 2.9 : l'algorithme A* avec un obstacle concave

Le secret lié au succès de l'algorithme A*, consiste dans le fait qu'il combine des morceaux d'information, les même que celles utilisées par l'algorithme de Dijkstra (favorisant le sommet qui est proche du point de départ), ainsi que l'information exploitée par l'algorithme BFS (favorisant le sommet qui est proche du but).

Dans la terminologie standard employée dans un algorithme A*:

g (n): représente le coût du chemin du point de départ à n'importe quel sommet n.

h (n): représente le coût heuristique estimé du sommet n au but.

Dans les diagrammes suscités, h (en jaune) représente les sommets qui sont loin du but, et g (bleu) représente les sommets loin du point de départ.

L'algorithme A* équilibre les deux tout en se déplaçant du point de départ au but. A chaque pas de la boucle principale, il examine le sommet n qui possède le coût le plus bas.

$$f(n) = g(n) + h(n).$$

2.2.4.1. Utilisation de la méthode heuristique *A**

L'heuristique peut être employée pour contrôler le comportement de l'algorithme A* [Amib].

- À une extrême, si (h(n) = 0), alors seul g(n) joue un rôle, l'algorithme A* conjugué à l'algorithme de Dijkstra, garantit l'existence du chemin le plus court.
- Si h(n) est toujours inférieur (ou égal) au coût de déplacement de n vers le but, on garantit donc que l'algorithme A* permet de trouver, lentement, le chemin le plus court.
- Si h(n) est exactement égal au coût de déplacement de n vers le but, alors l'algorithme A* suivra uniquement et très vite le meilleur chemin, et n'étendra jamais son domaine de recherche. Dans ce cas l'algorithme A* présente un comportement parfait.
- Si h(n) est parfois supérieur au coût de déplacement de n vers le but, dans ce cas on ne garantit pas que l'algorithme A* puisse trouver le plus court chemin, mais il peut converger plus rapidement.
- À l'autre extrême, si h(n) est très supérieur à g(n), dans ce cas seul h(n) joue un rôle dans la convergence de l'algorithme A* par rapport à l'algorithme BFS.

On a, donc, une situation intéressante dans ce qu'on peut décider, et dans la façon dont on peut exploiter l'algorithme A*. Dans ce cas, on peut prétendre, l'obtention des plus courts chemins et ce d'une manière rapide. Si on est trop bas, on continue donc à obtenir les chemins les plus courts, mais d'une manière moins rapide. Si on est trop haut, on abandonne donc la recherche des chemins les plus courts, mais l'algorithme A* convergera plus rapidement.

Dans un jeu, cette propriété de l'algorithme A* peut être très utile. Par exemple, il peut exister des situations où il n'importe pas de trouver le meilleur chemin et qu'un bon chemin soit suffisant. Pour changer l'équilibre entre g(n) et h(n), on peut procéder au changement de chacune des deux quantités.

2.2.4.2. Vitesse ou exactitude ?

La capacité de l'algorithme A* de varier son comportement est basée sur l'heuristique, et les fonctions de coût peuvent être très utiles dans la conception des jeux. La différence entre la vitesse et l'exactitude, peut être exploitée pour concevoir des jeux plus rapides. Pour la plupart des jeux, on n'a pas vraiment besoin du meilleur chemin entre deux points, on a juste besoin d'un chemin permettant de rallier les deux points en étant assez proche.

2.2.4.3. Échelle

L'algorithme A* calcule f(n) = g(n) + h(n). Pour ajouter les deux valeurs, ils doivent être à la même échelle.

2.2.4.4. Heuristique exacte

Si l'heuristique est exactement égal à la distance, le long du chemin optimal, on verra que l'algorithme A* étend très peu de nœuds, comme dans le diagramme montré dans la section suivante. Ce qui arrive à l'intérieur de l'algorithme A* est qu'il calcule f(n) = g(n) + h(n) au niveau de chaque nœud. Quand h(n) correspond exactement à g(n), la valeur de f(n) ne change pas tout au long du chemin. Tous les nœuds qui ne sont pas sur le juste chemin, auront une valeur f plus importante. [Amib]

2.2.4.5. Heuristique exacte pré calculée

Une façon de construire une heuristique exacte, est de pré-calculer la longueur du plus court chemin entre chaque paire de points. Ce n'est pas faisable pour la plupart des cartes de jeu, cependant, il y a des méthodes qui permettent de se rapprocher de cette heuristique:

Il y'a lieu d'ajouter alors, à l'heuristique h la fonction qui permet de calculer le coût pour aller de n'importe quel emplacement jusqu'à la voie voisine. L'heuristique finale devient:

$$H(n) = h'(n, w1) + \text{distance}(w1, w2), h'(w2, but).$$

On peut également opter pour une heuristique meilleure mais plus coûteuse, en évaluant toutes les paires w1, w2 qui proches du nœud et du but, respectivement.

Heuristique linéaire exacte.

Dans une circonstance spéciale, on peut opter pour une heuristique exacte sans pré-calcul. Si on a une situation sans obstacles ni de terrain lent, le plus court chemin du point de départ vers le but doit être en fait une ligne droite. En définitive, cette heuristique correspond à une heuristique exacte.

2.2.4.6. Heuristique pour des cartes de réseaux

Sur un réseau, il y a des fonctions heuristiques bien connues pouvant être employées. [Amib]

- **Distance de Manhattan**

La norme utilisée dans l'heuristique est la distance du Manhattan. Le coût pour le déplacement d'un espace vers un espace adjacent est D. Donc, l'heuristique dans notre jeu doit être:

$$h(n) = D * (abs (x_n - x_{but}) + abs (y_n - y_{but})).$$

L'échelle devant être utilisée est celle qui correspond à la fonction du coût.

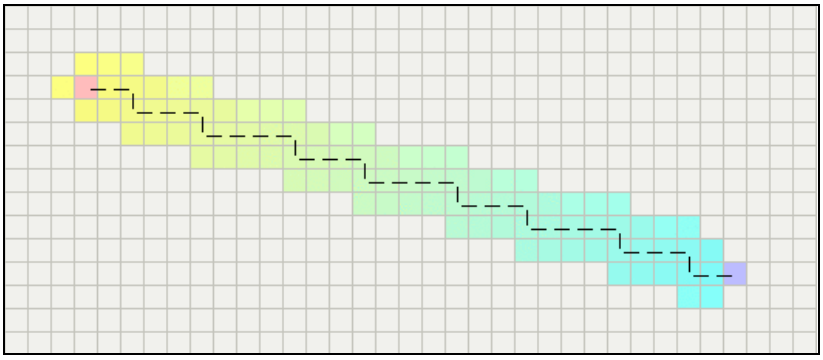


Figure 2.10 : La distance du Manhattan

- **Distance diagonale**

Si sur notre carte nous permettons le mouvement diagonal nous avons besoin d'un différent heuristique. La distance du Manhattan pour (4 est, 4 nord) sera 8*D. Cependant, nous pourrons simplement nous déplacer (4 nord-est) au lieu de cela, donc l'heuristique doit être 4*D. Cette fonction manipule des diagonales, assurant qu'et directement et le mouvement diagonal coûte D:

$$h(n) = D * max(abs (x_n - x_{but}) + abs (y_n - y_{but})).$$

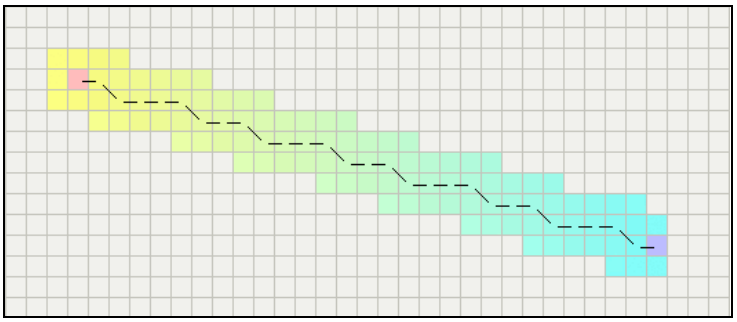


Figure 2.11 : Distance diagonale

Si le coût en diagonale du mouvement n'est pas D, mais une quantité semblable à $D_2 = \sqrt{2} * D$, ladite heuristique ne sera pas juste pour nous. Nous voudrions, au lieu de cela, quelque chose de plus sophistiqué:

h_diagonale (n) = minute (abs (x_n - x_{but}) + abs (y_n -y_{but}))

h_straight (n) = (abs (x_n - x_{but}) + abs (y_n -y_{but}))

h (n) = D_2 * h_diagonale (n) + D * (h_straight (n) – 2 * h_diagonale (n)))

Ici, nous calculons:

h_diagonale (n) = le numéro de pas que nous pouvons prendre le long d'une diagonale.

h_straight (n) = la distance du Manhattan et combiner ensuite deux en considérant tous les pas diagonaux pour coûter à D_2 et ensuite tout le maintien directement marche (notez que c'est le numéro de droit intervient la distance du Manhattan, moins deux directement pas pour chaque pas diagonal que nous avons pris au lieu de cela) coûtent D.

- **La distance Euclidienne**

Si les unités peuvent se déplacer suivant un angle (au lieu des directions du réseau), on donc probablement employer une distance de ligne droite: [Amic]

$$h(n) = D * \sqrt{(x_n - x_{but})^2 + (y_n - y_{but})^2}$$

Cependant, si c'est le cas, on peut avoir donc des difficultés pour l'utilisation directe de l'algorithme A* parce que la fonction du coût g ne correspondra plus à la fonction heuristique h.

De plus, pour cette méthode, la distance Euclidienne est plus courte que la distance de Manhattan ou la distance diagonale. On obtient toujours le chemin les plus courts, cependant l'algorithme A* prendra plus de temps à l'obtenir.

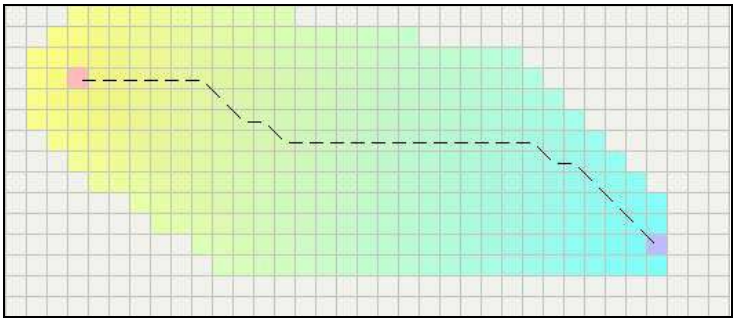


Figure 2.12 : Distance Euclidienne

- **Liens de casse**

Les liens dans l'heuristique peuvent parfois mener à une exécution inadéquate. Quand plusieurs chemins ont la même valeur de f, ils sont tous explorés bien que l'on a réellement besoin d'explorer l'un d'entre eux:

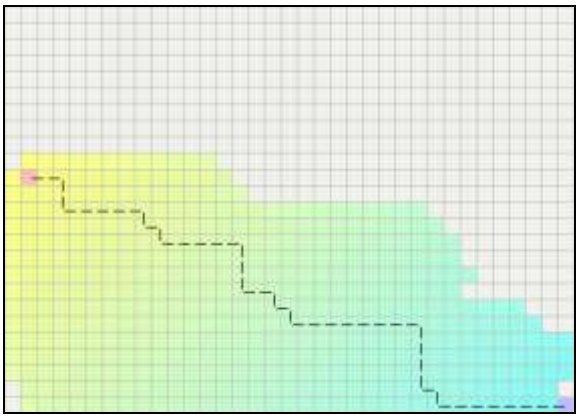


Figure 2.13 : Liens de casse dans les valeurs de f

Pour résoudre ce problème, nous pouvons ajouter un petit interrupteur de lien au niveau de l'heuristique. Cet interrupteur de lien doit être déterminé en rapport avec le sommet et doit faire de telle sorte que les valeurs de **f** soient différentes. C'est en fait une exploration des valeurs possibles de **f**.

Une façon de casser des liens est de pousser légèrement le coude l'échelle de la fonction h. Si l'échelle est peu élevée, donc f augmentera comme au fur et à mesure que l'on se déplace vers le but. Malheureusement, cela signifie que l'algorithme A* préférera étendre les sommets proches du point de départ au lieu d'étendre les sommets proches du but. Ainsi, au lieu d'augmenter l'échelle de la fonction h même légèrement (0.1 %, à titre d'exemple), l'algorithme A* préférera étendre les sommets près du but.

Heuristique * = (1.0 + p).

Le facteur p doit être choisi tel que: p < (coût minimum pour un pas) / (la longueur maximale de chemin attendue).

En garantissant que l'on ne s'attend pas à ce que les chemins soient d'une longueur supérieur à 1000 pas, on peut choisir p = 1/1000. Le résultat de ce coup de coude, cassant lien est une nouvelle variante de l'algorithme A* qui explore une partie moindre du champ de recherche par rapport aux variantes précédentes (Figure 4.14):

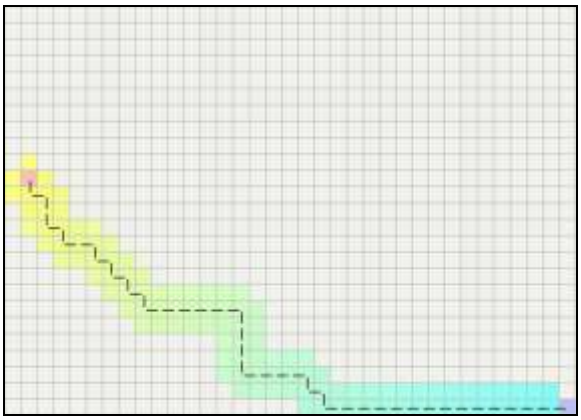


Figure 2.14 : Graduation cassant lien supplémentaire à heuristique

Lorsqu'il il y a des obstacles bien sûr, il doit toujours procéder à une exploration pour trouver une voie autour entre ces obstacles. Cependant, notons qu'une fois l'obstacle détourné, l'algorithme A* explore un champ de possibilités plus restreint (Figure 2.15).

Une façon différente de casser des liens est de préférer les chemins qui sont le long de la ligne droite du point de départ vers le but:

$$Dx_1 = x_{current} - x_{goal}$$
$$Dy_1 = y_{current} - y_{goal}$$
$$Dx_2 = x_{start} - x_{goal}$$
$$Dy_2 = y_{start} - y_{goal}$$
$$Croix = abs\ (dx_1*dy_2 - dx_2*dy_1)$$
$$Heuristique + = cross*0.001$$

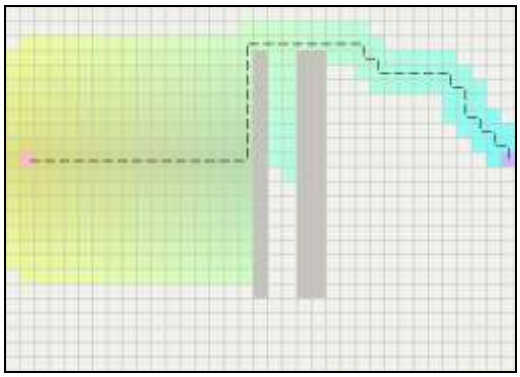


Figure 2.15 : La graduation cassant lien supplémentaire heuristique travaille gentiment avec des obstacles

Ce code calcule le produit vectoriel mutuel, entre le début au vecteur de but, et le point actuel au vecteur de but. Quand ces vecteurs ne s'alignent pas, le produit mutuel sera plus grand. Le résultat est que, ce code donnera une préférence légère à un chemin, qui se trouve le long de la ligne droite liant le point de départ et le but. Ainsi, lorsqu'il n'y a aucun obstacle,

l'algorithme A*, non seulement explore moins d'espace de recherche, mais génère des chemins plus réalistes (Figure 2.16).

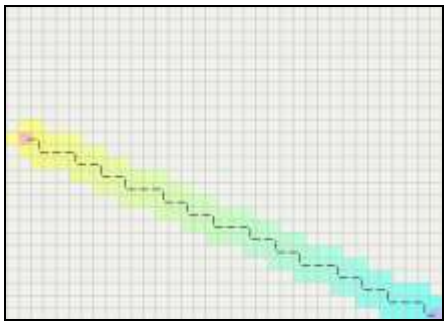


Figure 2.16 : Le produit mutuel cassant lien supplémentaire à .heuristique, produit de jolis chemins

Cependant, du fait que cet interrupteur de lien emprunte les chemins qui suivent la ligne droite reliant le point de départ au but, des artefacts peuvent survenir (discrètes) lorsque l'acteur rencontre des obstacles (Figure 2.17) (notez que le chemin est toujours optimal; cela semble juste étrange).

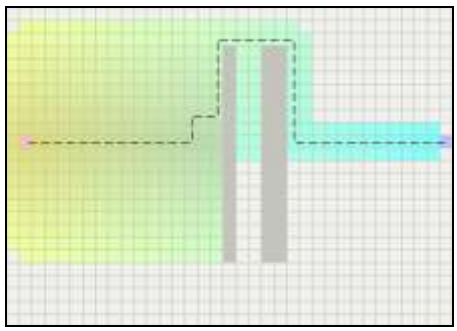


Figure 2.17 : Produit mutuel cassant un lien supplémentaire à heuristique avec obstacles (artefact)

Pour un mode interactif explorer l'amélioration de cet interrupteur de lien, (voir l'applette de James Macgill A*). L'utilisation "Claire" purifie la carte et choisit deux points à l'opposé des coins de la carte. Quand méthode classique de l'algorithme A* est utilisée, l'effet des liens est mis en évidence. D'un autre côté, lorsqu'on utilise la méthode "Fudge" (ou d'évitement), on constate l'effet du produit mutuel supplémentaire à l'heuristique.

Il existe également une autre façon de casser les liens, elle consiste à construire soigneusement la file d'attente des priorités de l'algorithme A* pour que de nouvelles insertions avec une valeur de f spécifique soient toujours classées comme étant les meilleures (inférieur) que les vieilles insertions avec la même valeur de f.

- **Recherche d'un secteur**

Si nous voulons chercher une tâche (un endroit) près d'un but quelconque, au lieu d'un espace particulier, nous pourrions construire h'(x) une heuristique qui est le minimum de h1 (x), h2 (x), h3 (x)... où h1, h2, h3 est l'heuristique à chacune des tâches (endroits) voisines.

Cependant, une voie plus rapide consiste à laisser l'algorithme A* chercher le centre du secteur de but. Une fois que l'on obtient un espace voisin du jeu OUVERT, nous pouvons arrêter et construire un chemin.

2.3. Recherche de chemin

Le déplacement d'humanoïdes à l'intérieur d'un environnement virtuel est, le plus souvent, lié à la volonté d'atteindre une position particulière. Il s'agit du problème traité par la recherche de chemins: comment trouver depuis une position donnée allant vers une autre position voulue, en évitant les obstacles statiques de l'environnement. Pour générer un chemin plausible, l'approche la plus adaptée est de rechercher un chemin minimisant certains critères comme la distance, le coût énergétique,… Même si cette recherche de chemin optimal n'est pas forcement corrélée avec la navigation humaine [WM03], elle permet de générer des chemins plausibles évitant des détours excessifs qui s'avèrent beaucoup moins réalistes. Deux grands types d'approches peuvent être distingués:

- Les approches à base de graphe;
- Et les approches à base de champs de potentiel.

2.3.1. Calcul sur les graphes

L'utilisation de l'algorithme des graphes pour le calcul d'un chemin nécessite l'utilisation d'une discrétisation de l'espace car un nœud du graphe est assimilé à un point de l'environnement. Les arcs représentent alors une notion d'accessibilité entre deux points et sont values par une estimation de l'effort à fournir pour relier ces points (il s'agit le plus souvent de la distance). La recherche d'un chemin va donc se résumer à la recherche d'une suite de nœuds, reliés par des arcs dont la somme des valeurs associées minimise le coût global du chemin.

2.3.1.1. Discrétisation de l'espace et construction du graphe

Avant de pouvoir appliquer les algorithmes de calcul sur les graphes, il faut disposer d'une représentation de l'environnement sous cette forme. Les cartes de cheminement [Lam03] fournissent directement cette information: chaque point clef de l'environnement devient un nœud alors que chaque arc représente la relation d'accessibilité entre ces nœuds. Dans le cas d'une discrétisation de l'espace sous la forme de cellules, les informations de cellules et de connexité sont utilisées pour générer une carte de cheminement. Traditionnellement, les points clefs sont choisis soit comme étant le centre de la cellule soit comme appartenant aux bords franchissables de cette même cellule. Les arcs du graphe sont alors construits en fonction des relations de connexité entre les cellules. La figure 2.18 montre les deux formes de graphes de cheminement qui peuvent être construites en utilisant comme base une triangulation de Delaunay contrainte; la figure de gauche utilise les centres de gravité des triangles comme points clefs; la figure de droite utilise le milieu des segments non contraints reliant deux triangles.

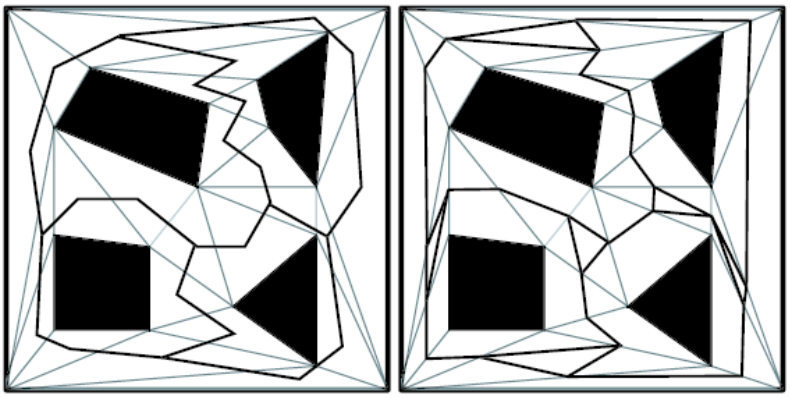


Figure 2.18 : Exemple de cartes de cheminement générées à partir d'une discrétisation par triangulation de Delaunay contrainte

Une autre approche, utilisée dans le cadre d'environnement intérieur, consiste à travailler sur les graphes d'adjacence des cellules (extraites de la subdivision spatiale) pour caractériser des espaces (pièces, couloirs, portes,…) [Lau89]. Chaque espace étant un agrégat de cellules, le graphe topologique des espaces est plus abstrait est donc moins complexe que le précédent. Cela permet de définir une stratégie de navigation hiérarchique s'appuyant sur le graphe topologique et raffinant les calculs, à posteriori, à l'intérieur de chaque espace.

2.3.1.2. Les algorithmes utilisés

Diverses formes d'algorithmes de calcul sur les graphes peuvent être utilisées pour effectuer le calcul du 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 autres 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 qui sont les plus
proches de l'origine. Cet algorithme travaille uniquement sur la structure du graphe sans utiliser d'informations supplémentaires. De ce 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 dont la distance est inférieure à D avant de pouvoir fournir un chemin.

Dans la mesure où 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 est évaluée par l'expression suivante:

$v(n_k) = d(n_k,o) + h(n_k, b)$

où $d(n_k,o)$ représente la distance réelle 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 file 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. L'algorithme explore ainsi 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[LA98]. Il s'agit d'une génération 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.

L'algorithme IDA* (Iterative-Deepening A*) utilise une approche différente en effectuant une recherche en profondeur d'abord, supervisée par une heuristique [Kor85]. 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 cette distance estimée au but soit supérieur 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 termes de temps d'exécution peuvent être retenues en utilisant un cache de taille fixe des estimations déjà calculées pour les nœuds précédemment explorés [RM94]. Ces estimations permettent un étalage plus rapide de l'arbre de recherche.

En règle générale, l'algorithme de Dijkstra est peu utilisé dans le cadre de la recherche de chemin car il s'agit d'un 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. En général, l'algorithme IDA* est préféré à l'algorithme A* dans le cas de domaines de taille exponentielle car la taille de la liste triée conservée par l'algorithme A* devient elle-même exponentielle[RM94]. Cependant, dans le cadre de la planification de chemins, cette considération n'a pas vraiment de signification si 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 d'espace mémoire.

2.3.2. Méthodes à champs de potentiel

Contrairement aux approches utilisant les graphes, les méthodes à base de champs de potentiel ne travaillent pas forcément sur une représentation discrète de l'environnement. Les obstacles sont vus comme des émetteurs de forces répulsives, alors que le but est un attracteur [Lat91, Rei97]. Un potentiel est donc défini pour chaque point de l'environnement comme la somme des potentiels de répulsion liés aux obstacles avec le potentiel d'attraction lié au but. L'entité étant localisée dans l'environnent, la direction à prendre pour converger vers son but est alors opposée au gradient de potentiel défini en ce point.

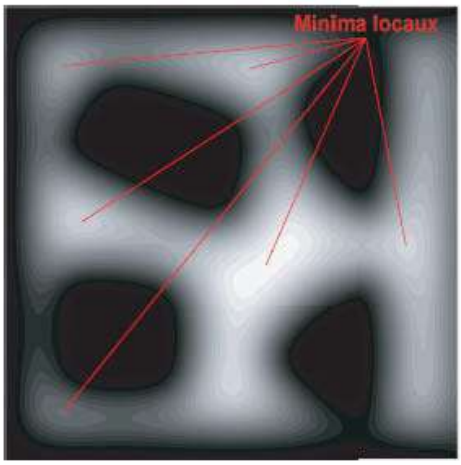


Figure 2.19 : Une carte de champ de potentiel, les parties noires représentent les obstacles, les parties plus claires les zones de navigation. Les dégradés de couleurs représentent pour leur part la valeur du potentiel associé au point de l'environnement. Cette image montre six minima locaux caractérisés par des couleurs plus claires.

Ces méthodes s'avèrent simples et efficaces mais posent le problème de chemin local (figure 2.19). La méthode de navigation associée pousse l'entité à se déplacer vers le minimum local le plus proche qui n'est pas forcément le minimum de la surface et en conséquence qui ne représente pas le but. Pour pallier ce type de problème, des méthodes à base de marche aléatoire sont utilisées. Par exemple, dans RPP (Random Path Planner) [BL91], 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, à base de tirage aléatoire de direction à suivre, existent [CRR01].

2.4. Conclusion

Dans ce chapitre, nous avons présenté le principe du pathfinding, et les divers algorithmes mis à notre disposition pour le gérer, pour la découverte des plus courts chemins et des distances dans des graphes plats qui exploitent la topologie particulière de l'entrée graphique. Nous avons vu aussi, les différentes étapes, puis les points forts et les points faibles de chacun des algorithmes. L'algorithme A* prend les avantages de l'algorithme BFS et de l'algorithme Dijkstra. Il est ainsi semblable à l'algorithme de Dijkstra dans lequel il peut être employé pour trouver le plus court chemin, et également semblable à l'algorithme BFS dans lequel il peut employer un heuristique pour se guider. Nous avons, par ailleurs, présenté l'heuristique liée à l'algorithme A*, et enfin les différentes variantes de l'algorithme A*.

Les méthodes de recherche de chemin présentées dans ce chapitre, ne sont pas très adaptées à la simulation comportementale. Les comportements induits par les tirages aléatoires, s'ils sont acceptables pour des robots, ils ne le sont pas pour des humanoïdes car ils sont en dehors de la logique de la navigation humaine

Chapitre 3

Les niveaux de détails et la simulation comportementale:

3. Les niveaux de détails

3.1. Introduction :

La simulation de scènes en synthèse d'images est une technique maîtrisée, largement utilisée pour les effets spéciaux cinématographique et les jeux vidéo. La possibilité de rendre un mouvement réaliste constitue la partie essentielle de la plupart des environnements virtuels et des applications de simulation visuelle.

En effet, malgré le développement des machines graphiques, la visualisation en temps réel de modèles complexes reste une véritable problématique. Il existera toujours des scènes dont la complexité dépasse les capacités de calcul d'une machine aussi puissante soit-elle. Ceci est d'autant plus vrai que les besoins des utilisateurs évoluent autant – si ce n'est plus vite – que les progrès en informatique [Bel98].

Il faut pour cela mettre en œuvre une optimisation dont la multirésolution fait partie. La multirésolution ou la représentation d'un modèle avec des niveaux de détails plus ou moins fins selon les besoins de l'utilisateur et le type d'application. La méthode consiste à simplifier la géométrie des objets, à en produire plusieurs représentations puis à visualiser le modèle à l'aide d'un algorithme qui réponde aux critères de l'utilisateur. Nous présenterons, avec plus de détails, les enjeux et les difficultés de cette approche dans la première partie de ce chapitre.

La deuxième partie est consacrée à la présentation de l'exploitation de la technique de niveaux de détails dans la simulation comportementale.

3.2. Les niveaux de détails :

Réaliser un niveau de détail d'un élément consiste à en construire une version moins complexe géométriquement, c'est-à-dire comportant moins de facettes [Val99]. Il s'agit donc d'afficher des versions plus au moins dégradées de l'objet selon certains critères tel que la distance entre la caméra et l'objet ou la taille de l'objet à l'écran.

La complexité des environnements virtuels influence le temps de traitement informatique et d'affichage. Plus on ajoute de complexité visuelle à une scène, plus long sera le délai induit au niveau du rendu. Pour bien préciser l'utilité de l'aspect niveaux de détails dans le rendu des scènes en synthèse d'image on va commencer par la définition de la complexité dans une scène que ce soit statique ou dynamique.

3.2.1. La complexité dans une scène statique :

Une scène géométrique est définie par un ensemble de données comme les maillages polygonaux, les textures ou les sources de lumière. La complexité d'une scène dans l'absolu est fonction de la quantité de chacun des divers éléments la composant [Val99]. Lorsqu'on a une scène trop complexe, il devient nécessaire d'en simplifier certaines parties. Il faut rejeter les éléments non visibles ou cachés par d'autres éléments dans le processus de calcul d'image. En ce qui concerne les éléments visibles (au moins partiellement), la suppression est également envisageable mais à utiliser avec précaution car certains éléments peuvent être importants pour l'application. En revanche, la dégradation de l'élément, c.à.d. son remplacement par un élément plus simple (avec moins de détails) est très utile.

3.2.2. La complexité dans une scène dynamique:

La complexité d'une scène animée dépend du nombre d'entités animées, du type de mouvement qu'on veut avoir et du taux de réalisme qu'on souhaite obtenir.

Le principe le plus utilisé pour gérer la complexité en animation consiste à créer des NDDs de modèle d'animation, c'est-à-dire une famille de modèles d'animation avec des coûts de calcul différents et permettant d'animer une même partie de la scène ou une même entité, de telle sorte qu'on utilise une version dégradée pour les entités éloignées de la caméra qui ne nécessite pas beaucoup de détails, et une version plus détaillée pour les

entités les plus proche de la caméra pour qu'on puisse avoir un niveau de réalisme en conséquence.

3.2.3. Les types de LODs

3.2.3.1. Niveaux de détails discrets

C'est le cas le plus simple pour la représentation des LODs des maillages, Il consiste en une collection de maillages de différentes tailles. Chaque maillage représente un objet avec une résolution différente. Chacune de ces représentations est définie d'une façon indépendante des autres. Toutes les représentations sont générées dans une étape de prétraitement et sont stockées. Une représentation est choisie au besoin (au moment du traitement) selon certain critère de sélection [Con01].

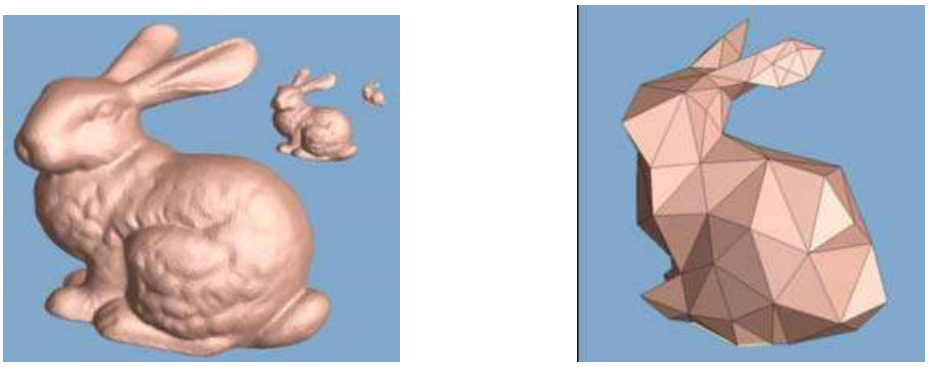


Figure 3.1: Niveaux de détails discrets

- ***Avantage :***
  - L'avantage le plus important des LODs statiques est la simplicité de la programmation. Il y a une séparation entre l'algorithme de simplification et celui du rendu, ce qui implique un programme simple contenant seulement la sélection du LOD le plus approprié.
- ***Inconvénient :***
  - Chaque représentation doit être stockée indépendamment, la taille de la mémoire nécessaire pour le stockage des représentations augmente avec le nombre de LODs stockés, en pratique il y a toujours des contraintes pour la taille requise, ce qui signifie qu'on ne peut pas stocker plusieurs LODs. [Ben 04].

3.2.3.2. Niveaux de détail continus

Dans le cas de LODs continus, la structure de données est créée de telle sorte que n'importe quel LOD peut être extrait au moment du traitement. Une telle structure de données est appelée maillage *multirésolution.* Selon la complexité de la structure de donnée, Il est possible d'obtenir un petit ou un grand nombre de maillage à différentes résolutions [Con01].

Le nombre de LODs qui peuvent être générés n'est pas fixé à priori. On peut citer quelques propriétés de la multirésolution continue :

- Elle doit être capable de générer la résolution désirée dans un temps très court (temps réel).
- La transition entre les différentes résolutions ne doit pas être perceptible.

A cause du nombre important des représentations possibles d'un modèle original, la création et le stockage de chacune d'elles sont impossibles. Les informations communes de ces représentations sont donc utilisées pour créer une seule représentation unifiée. A partir de cette dernière, le LOD désiré est extrait au moment de traitement selon les paramètres de vue.

- ***Avantages:***
  - Les LODs dynamiques exécutent quelques simplifications comme un prétraitement, mais reportent le calcul du reste du travail au système de visualisation au moment du traitement. Cela permet d'incorporer plus de critères de sélection de LODs.
  - Les LODs dynamiques ajustent les détails graduellement et avec augmentation, en réduisant le passe visuel.

- ***Inconvénient :***
  - Requière un espace mémoire et un temps de calcul supérieur à celui des LODs statiques.

3.2.3.3. Les LODs dépendants du point de vue

C'est une extension des LOD continus qui intègre un critère de simplification qui est dépendant du point de vue. Ainsi cette représentation est anisotropique: différentes zones du même objet sont visualisées à des niveaux de détail différents.

Les parties de l'objet proches du point de vue peuvent apparaitre à une résolution plus haute que les parties éloignées, cette méthode permet de réaliser des dégradations plus importantes qu'avec les méthodes précédentes, mais à un cout de calcul relativement élevé.

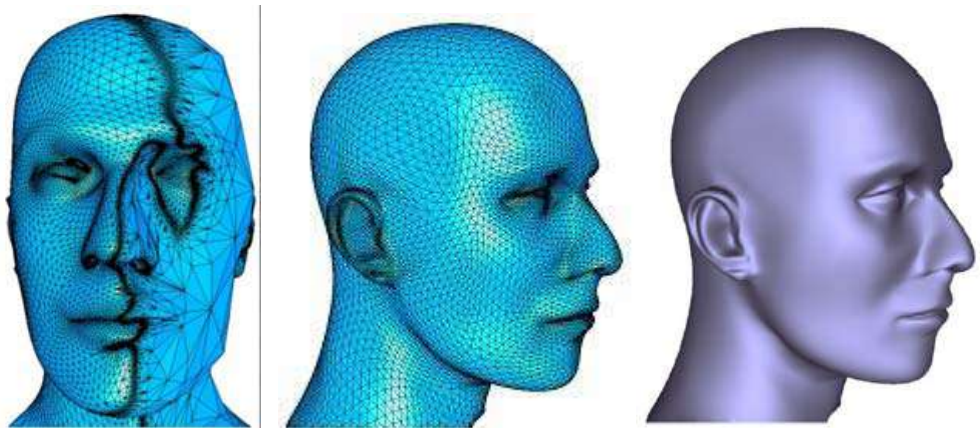


Figure 3.2 : Niveaux de détails dépendants du point de vue

En pratique :

En pratique les LOD discrets sont les plus utilisés. Les LOD continus n'apparaissent que dans quelques rares applications spécifiques à cause du coup de calcul de la géométrie en cours d'exécution [Lrc+ 03].

3.2.4. Création des niveaux de détails :

3.2.4.1. Objectifs algorithmiques

En traversant les NDDs des modèles du plus complet au plus dégradé, on constate [val99] :

- Une réduction de l'espace des trajectoires possibles.
- Une réduction des coûts de calcul.
- Une moins bonne prise en compte des contraintes extérieures.

Idéalement, les algorithmes de créations des niveaux de détails doivent posséder plusieurs qualités [Kru97]:

- Pouvoir créer une suite de représentations de plus en plus simplifiées.

- Pouvoir mesurer et contrôler de manière intuitive le degré de simplification. Cela passe par la définition d'une mesure de l'approximation.
- Tenir compte des parties significatives des objets pour les préserver dans les représentations simplifiées. Intuitivement, il s'agit de trouver les parties de l'objet que l'observateur percevra le plus facilement.
- Pouvoir faire varier le degré de simplification à travers l'objet pour, par exemple, simplifier davantage les parties peu importantes ou lointaines d'un objet.

- ***Difficultés :***

  D'après G. Debunne dans [Deb00], il y a deux problèmes principaux dans la mise en place des approches basées sur le principe de NDD :

  - Les différentes résolutions que l'on veut mélanger doivent avoir, dans une certaine mesure, le même comportement dynamique, c'est-à-dire approximer le même mouvement au cours du temps. Si tel n'était pas le cas, la simulation serait chaotique, son comportement dépendant des résolutions employées alors que c'est précisément ce que l'on veut cacher à l'utilisateur.
  - Les différentes résolutions doivent pouvoir cohabiter, c'est-à-dire que les transitions entre les niveaux ne doivent pas être visibles. Ces transitions sont d'une grande importance et de leur discrétion dépend le résultat. Cette notion est à différencier de la précédente, car même avec un comportement dynamique identique, deux résolutions ne peuvent pas donner un bon résultat si elles ne cohabitent pas. Deux mouvements, même similaires, doivent par exemple être en phase à l'endroit où l'on passe de l'un à l'autre. On devra aussi veiller à éviter les clignotements dus à une oscillation entre deux résolutions ainsi que les apparitions soudaines de détails qui ne faisaient pas partie de la résolution grossière précédente, ou tout autre indice visible dépendant de l'application.

3.2.4.2. Simplification orienté géométrie :

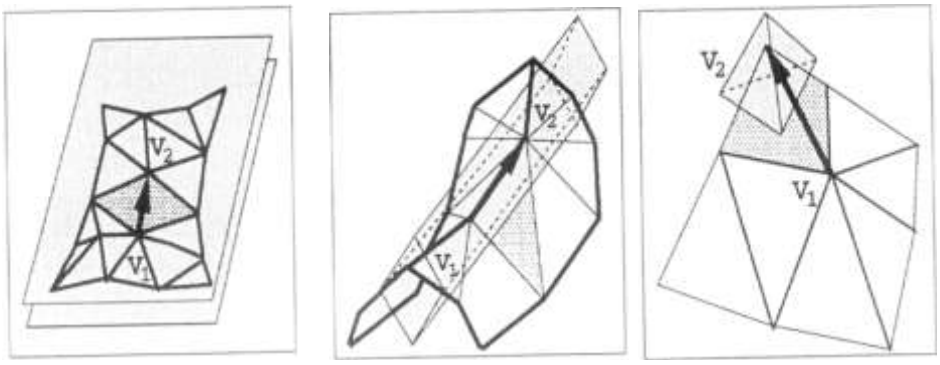
Pour simplifier les objets polygonaux il existe deux types de méthodes. Les premières cherchent à simplifier la représentation géométrique des objets, en réduisant le nombre de polygones par exemple. On parle alors d'algorithmes de simplification polygonale. Les deuxièmes utilisent une représentation totalement différente, par exemple une boite texturée par une image issue d'une représentation détaillée de l'objet. [Kru 97]

3.2.4.2.1. Simplification géométrique polygonale:

L'objectif d'un algorithme de simplification polygonale est de prendre un modèle détaillé avec un nombre important de polygones et de générer un modèle plus simple avec un nombre inférieur de polygones qui apparaît semblable au modèle d'origine, tout en conservant leurs importantes caractéristiques visuelles. L'avantage de la représentation simplifiée est qu'elle peut être rendue plus rapidement qu'un modèle original [Con01].

La simplification doit être faite de telle sorte que la forme générale du modèle soit préservée. Quelques particularités des algorithmes de simplification qui doivent être précisés sont [Kbg+97]:

- ***Faces coplanaires (planer area) :*** sont identifiées en examinant les normales des polygones adjacents, ces polygones peuvent être fusionnés pour former un polygone plus grand. C'est le type de simplification le plus facile. **Figure (3.3.a)**
- ***Arêtes alignées (Sharp edges) :*** sont trouvées en comparant les angles entre les normales des facettes adjacentes, elles peuvent être simplifiées en fusionnant les arêtes connectées qui sont presque colinéaires. **Figure (3.3.b)**
- ***Arêtes saillantes (Pointed edges) :*** doivent être préservée, elles peuvent être détectées en utilisant la courbure locale autour d'un sommet. **Figure (3.3.c)**



(a)Faces coplanaires (b) Arêtes alignées (c) Arêtes saillantes

Figure 3.3 : Formes caractéristiques

3.2.4.2.2. Opérateurs de simplification :

Les algorithmes qui entrent dans cette catégorie sont indépendants les uns des autres mais partagent cependant un certain nombre de caractéristiques. Astheimer et

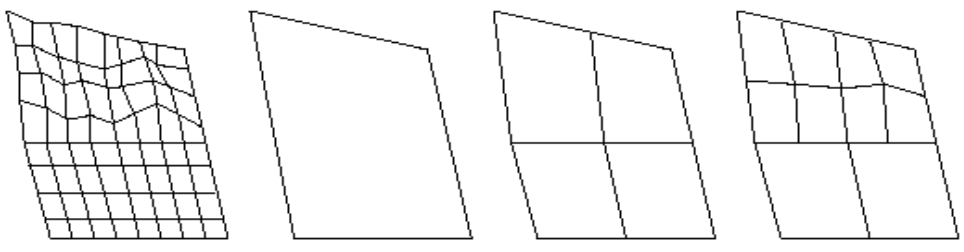
Pöche présentent dans [AP94] une liste, définie de manière intuitive, d'opérateurs de simplification :

- **Normalisation** : élimine les points et arêtes définis plusieurs fois.
- **Simplification des Sommets** : tous les points à l'intérieur d'un volume (cube d'une grille uniforme ou sphère autour d'un point) sont regroupés. Ainsi, les amas de points et les petites facettes sont combinés.
- **Simplification des arêtes :** tous les bords d'une longueur inférieure à une longueur donnée sont éliminés.
- **Simplification basée sur les angles** : les bords qui contiennent un angle inférieur à une certaine valeur sont éliminés.
- **Simplification basée sur la surface d'une facette** : les facettes dont la surface est inférieure à une certaine valeur sont éliminées. Les trous qui en résultent peuvent être bouchés par triangulation en utilisant, par exemple, le barycentre des points retirés.
- **Simplification basée sur la coplanarité d'un ensemble de facettes** : les facettes dont les normales sont presque parallèles sont éliminées (il faut aussi boucher le trou). Il faut noter qu'il est nécessaire de connaître les relations de voisinage en facettes.

En se basant sur ces opérateurs, il est possible de définir une classification des méthodes de simplification polygonale:

3.2.4.2.3. Subdivision Adaptative (Adaptative Subdivision) :

Cette méthode commence avec un modèle de base très simple et le subdivise récursivement en ajoutant des détails à certains endroits du modèle à chaque étape. L'algorithme s'arrête lorsque le modèle approxime le modèle original à un degré spécifié par l'utilisateur. La figure 3.4 montre un exemple de subdivision adaptative.



Modèle initial simplification originale 1er subdivision 2ème subdivision

Figure 3.4 : La subdivision adaptative

Cette méthode est assez peu utilisée car il n'est pas facile de construire la simplification initiale dans le cas général [Kru99].

3.2.4.2.4. Réduction Géométrique (Geometry Removal):

Cette méthode part du modèle d'origine et le simplifie en retirant des faces ou des sommets récursivement. L'algorithme s'arrête lorsqu'il ne peut plus retirer de géométrie et lorsque le modèle satisfait un degré d'approximation spécifié par l'utilisateur. La majorité de ces algorithmes ne retirent des faces ou des sommets que si cela ne cause pas de violation de la topologie. La figure 3.5 montre un exemple de réduction géométrique.

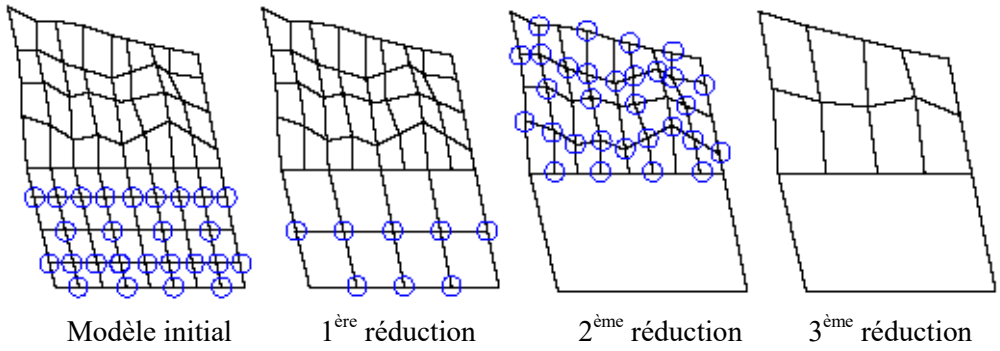


Figure 3.5 : La réduction géométrique

Cette technique est utilisée fréquemment dans les algorithmes de simplification polygonale les plus récents [Kru99].

3.2.4.2.5. Echantillonnage (Sampling):

Cette méthode fonctionne de manière opposée à la méthode précédente au sens où elle cherche à trouver les primitives à conserver plutôt que celles à retirer. Elle effectue un échantillonnage de la géométrie du modèle original, soit en choisissant arbitrairement un certain nombre de sommets, soit en englobant le modèle dans une grille tridimensionnelle, et en échantillonnant chaque boîte de la grille. L'algorithme essaye alors de créer un modèle simplifié qui soit proche des données échantillonnées. L'utilisateur peut contrôler le degré d'approximation en changeant le nombre de points ou la taille de la grille. La figure 3.6 montre un exemple d'échantillonnage.

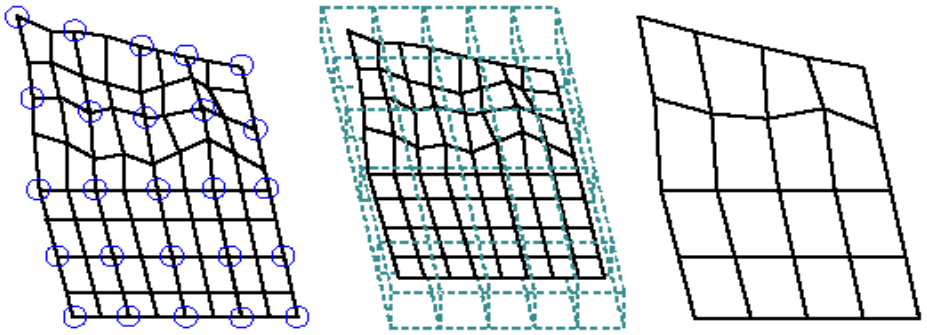


Figure 3.6 : L'échantillonnage

Cette méthode, historiquement assez importante, n'est pas fréquemment utilisée dans les algorithmes récents. Il est en effet plus difficile de trouver l'ensemble des sommets à préserver, plutôt que de tester chaque sommet pour savoir s'il peut être retiré [Kru99].

Pour se construire une idée plus approfondie sur les trois catégories de la méthode de simplification polygonale le lecteur se reportera à [Red97] et [Kru99] pour voir les différentes techniques publiées dans chacune de ces catégories.

Nous avons relativement détaillé cette technique (Simplification polygonale) parce que pratiquement tous les systèmes de réalité virtuelle utilisent les polygones pour le rendu de leurs scènes.

3.2.4.2.6. Simplification structurelle, polygone texture :

Ces méthodes changent la structure de représentation des objets. Par exemple il est possible de remplacer un objet polygonal par une boite englobante texturée à l'aide d'une image produite à partir d'une version détaillée de l'objet. Les LODs crées avec cette méthode sont appelés des imposteurs. Ces méthodes produisent souvent des imposteurs qui dépendent du point de vue et posent des problèmes pour les calculs d'éclairement dans le cas ou les projections d'ombres sont nécessaires (le résultat est correct dans le cas d'un éclairage ambiant). Certaines méthodes proposent de remplacer les objets par un polygone texturé à l'aide d'une image de l'objet original calculée avec un haut niveau de détails [Kru 97].

3.2.4.3. Simplification orientée scène :

La plupart des algorithmes orientés scène cherche à simplifier des régions de la scène plutôt que les objets eux-mêmes. Ainsi dans une phase de pré-calcul ils décomposent récursivement la scène en zones 3D (en utilisant un octree ou une grille 3D) pour obtenir une description hiérarchique, chaque niveau étant plus détaillé que le précédent. Ensuite à chaque nœud de la hiérarchie une représentation simplifiée de la sous hiérarchie est produite. Cette représentation sera utilisée si les critères de sélection sont satisfaits.

Toutes ces méthodes obligent à opérer au sein de l'algorithme de sélection des LODs un parcours d'arbre pour chaque image afin de déterminer le niveau de détails à utiliser. De plus, elles ne sont vraiment efficaces que sur des scènes très profondes sans occlusion due par exemple à des murs [Kru 97].

3.2.4.3.1. Boite colorée :

Cet algorithme est proposé par [Cdl+95] il décompose la scène à l'aide d'un octree. Il fait ensuite correspondre à chaque nœud un cube chaque face ayant une couleur représentative de la couleur de l'ensemble des facettes comprises dans le sous arbre. Cet arbre construit a priori est utilisé dynamiquement. En cela l'algorithme peut être vu comme simplifiant une région de la scène plutôt qu'un objet. L'algorithme de rendu parcourt l'arbre et détermine la taille dans l'espace image de la boite englobante à chaque nœud. Si cette taille est inférieure au pixel alors le cube est utilisé et les facettes du sous arbre sont ignorées. Si la taille est supérieure l'algorithme descend récursivement l'arbre.

Toutefois, cet algorithme fait apparaître des trous entre les cubes et il augmente la taille globale de l'objet. De plus le rendu est plus sombre qu'avec l'objet original.

3.2.4.3.2. Fusion hiérarchique :

Cet algorithme est proposé par [Lue 96]. Il décompose la scène en utilisant un octree. Pour chaque nœud, il fusionne l'ensemble des sommets dans le sous arbre en un sommet représentatif (estimé en fonction de la courbure locale). Ensuite, lors de l'affichage il parcourt l'arbre et détermine en fonction de la taille dans l'espace image quel nœud doit être utilisé. Cette méthode a l'avantage de pouvoir donner à différentes parties d'un objet des LODs différents. Ceci est particulièrement intéressant pour les grands objets.

3.2.4.3.3. Modèles d'illumination :

En utilisant des techniques d'illumination différentes pour l'éclairage et l'ombrage, on peut obtenir différents niveaux de détails pour les objets. Ceci signifie que nous pouvons utiliser par exemple moins de polygones et un algorithme d'illumination amélioré pour obtenir une représentation semblable à celle avec plus de polygones.

Néanmoins, certaines de ces techniques exigent un nombre important d'opérations de calcul pour être implémentées, ainsi leur utilisation n'est pas toujours appropriée. Quelques modèles d'éclairage incluant les modèles de Phong et de Gouraud produisent des résultats plus réalistes que ceux produits par le modèle de Lambert [Con01].

3.2.5. Classification:

L'art de l'utilisation des niveaux de détails consiste à trouver le juste équilibre entre le réalisme et la vitesse de calcul .Ainsi, nous cherchons à décrire l'importance d'un objet dans une scène afin de choisir la finesse de sa représentation [Fts+ 93] stipule que la gestion des différents LODs doit être faite de manière prédictive, basée sur la complexité de la scène plutôt que réactive, basée sur le temps de calcul de l'image précédente .Ainsi, il est possible de garantir un temps de rendu inferieur à une certaine limite. A cette fin les auteurs définissent un certain nombre de facteurs qui permettent de choisir un LOD parmi d'autres.

3.2.5.1. La sélection d'un NDD:

Différentes techniques sont utilisées pour choisir un niveau spécifique de détail. Ces techniques essayent d'éviter la transaction inutile entre les NDDs en réduisant les détails seulement quand ils ne sont pas perceptibles, offrant le bénéfice d'améliorer la réaction du système sans aucune perte dans le coût de transition entre les niveaux de détails.

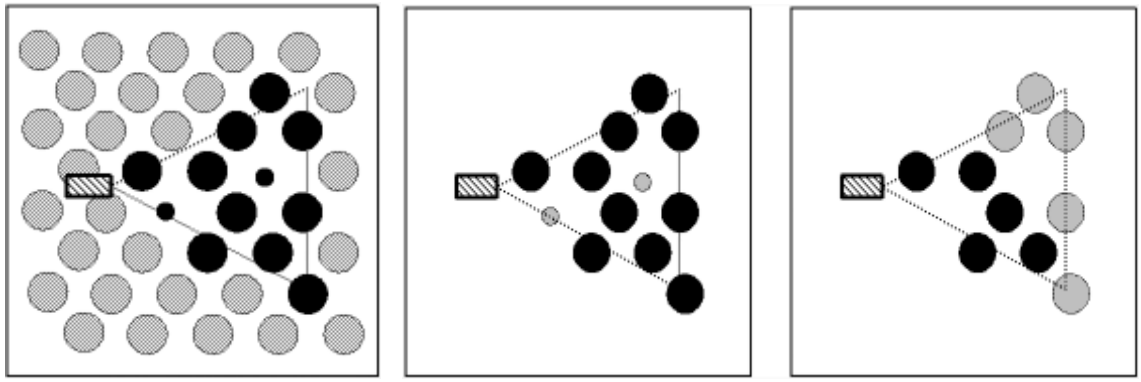
Z. Constantinescu dans [Con01] a classifié ces méthodes dans les groupes suivants :

- Le premier groupe se concentre à l'enlèvement des détails qui ne doivent pas être rendus par le matériel d'affichage graphique. Dans ce cas, nous avons différentes techniques de sélection, lesquelles se basant sur certains critères, elles éliminent quelques parties du modèle.

- Le deuxième groupe consiste à éliminer les détails qui ne peuvent pas être rendus par le matériel. Parmi les méthodes qui utilisent ce principe nous citons celles basées sur la distance (§3.2.5.1.2) et sur la taille de l'objet (§ 3.2.5.1.3).
- Dans le troisième groupe nous avons les algorithmes qui éliminent les détails qui ne peuvent être perçus par l'observateur. Ces méthodes sont celles basées sur la vision périphérique (§ 3.2.5.1.4), la profondeur du champ de vision (§ 3.2.5.1.5) et la vitesse (§ 3.2.5.1.6). Il existe aussi une autre méthode dont le but principal est de maintenir un taux d'affichage constant, indépendamment de la complexité du modèle.

3.2.5.1.1. Sélection (culling):

Il existe des situations où l'utilisateur ne voit jamais quelques parties de l'objet. Dans ces cas, ces parties ne sont pas représentées par le système graphique. Dans ce qui suit, quelque cas de ces situations sont présentées [Cab97].



a. sélection par pyramide de vision b. sélection de contribution c. sélection d'occlusion

Figure 3.7 : Techniques de sélection

- La méthode de sélection par pyramide de vision (view frustum culling) enlève tous les objets qui sont à l'extérieur de la pyramide de vision de l'observateur. Un tel cas est présenté dans la figure (3.7.a). Tous les objets qui sont à l'extérieur de la pyramide sont écartés (ils sont présentés en gris dans cette figure). Cela améliore la performance de la représentation graphique en éliminant le travail inutile dans les premières étapes du rendu.

- La deuxième méthode est la sélection de contribution (contribution culling). Dans cette méthode les objets dont la contribution visuelle est inférieure à un certain seuil sont écartés, même s'ils sont visibles. Les facteurs définissant la contribution sont généralement la taille de l'objet, la taille de la fenêtre d'affichage et la résolution de l'écran. Cette situation est présentée dans la figure (3.7.b) dont quelques petits objets (en gris) sont écartés, bien qu'ils sont à l'intérieur de la pyramide de vision. Cette méthode réduit l'exactitude fournie par la scène originale, mais elle peut réduire sensiblement sa complexité et spécialement quand nous avons plusieurs objets de petite taille.
- La technique de sélection la plus puissante est la sélection d'occlusion (occlusing culling). Dans ce cas (présenté dans la figure 3.7.c) la sélection est obtenue par la découverte et l'enlèvement des objets qui sont cachés ou occlus par d'autres objets. Il est aussi possible, d'écarter les objets qui sont partiellement occlus.

3.2.5.1.2. Distance :

La deuxième technique prend en considération la distance de l'objet par rapport à l'observateur. La mesure est basée sur la distance Euclidienne entre le point de vue et un point prédéterminé à l'intérieur de l'objet. Cette approche est basée sur la théorie suivante: "plus l'objet est loin du point de vue, moins de détails de ces composants sont visibles". Cela signifie que nous pouvons choisir une représentation moins complexe sans trop affecter l'exactitude de l'image [Con01].

- **Avantages :**
  - La simplicité : tout ce qui est exigé est de vérifier si une distance excède un seuil prédéterminé.
  - L'efficacité : cette technique est efficace car un seul calcul doit être effectué, c'est le calcul de la distance entre le point de vue et l'objet.

- **Inconvénients :**
  - L'inconvénient principal est lié à la manière de choisir un point du volume de l'objet. Si nous choisissons un point fixe dans l'objet, la distance réelle entre le point le plus proche de l'objet et le point de vue peut changer selon l'orientation de

l'objet. Pour les objets grands ou très proches, le choix de ce point n'est pas toujours facile.

- Le deuxième inconvénient peut apparaître lorsqu'on fait un changement d'échelle d'un objet, ou quand on change la résolution d'affichage, dans ce cas le seuil de la distance originale sera invalide et doit lui aussi changer.

Cette technique a été largement employée dans les simulateurs de vol et dans la visualisation de terrain.

(a) (b) (c) (d)

Le niveau du détail varie en fonction de la distance. Le modèle original (a) contient plus de 600 polygones (à l'exclusion de l'arme), tandis que (b), (c), et (d), le nombre de sommets a réduit à 75%, à 50%, et à 25%. © 1999-2001 Epic Games, Inc. de copyright.

Figure 3.8 : Le niveau de détail en fonction de la distance

3.2.5.1.3. Taille (Size) :

Une troisième technique profite de la capacité réduite de l'œil à percevoir les objets comme la diminution de la taille de ces objets. Elle prend en considération la taille des objets représentés, les objets les plus petits étant représentés avec moins de détails, tandis que les objets les plus grands sont représentés avec plus de détails [Con01]. Cette technique peut être considérée comme une autre façon d'implémentation de la distance dans les niveaux de détails, en effet, plus un objet est loin, plus sa taille apparente sera réduite.

- ***Avantage :***
  - fournit une mesure pour déterminer la visibilité de particularités dans un objet, indépendamment de la résolution d'affichage ou de changement d'échelle de l'objet. Dans cette technique on n'a plus besoin de choisir un point pour le calcul de la distance.

- ***Inconvénient :***
  - Comparée à la méthode précédente, elle est plus coûteuse en temps de calculs, parce que nous devons d'abord faire une projection des objets dans l'espace de vision et calculer ensuite leurs tailles avec les coordonnées de vue. Cependant, il existe quelques implémentations très efficaces pour cette méthode dans la littérature.

3.2.5.1.4. Vision périphérique (Eccentricity) :

Une autre technique, pour choisir un NDD, profite de la capacité réduite de l'œil à percevoir les objets en dehors du centre du champ de vision (la périphérie). Cela suggère de diviser le champ de vision en deux secteurs: un correspondant au centre du champ de vision de l'œil pour les régions détaillées, et l'autre correspondant à la périphérie du champ de vision de l'œil pour les régions moins détaillées.

Des expériences ont montré que nous pouvons réduire la complexité visuelle dans la périphérie sans affecter la performance de la tâche de recherche visuelle. Cela suggère qu'un système de gestion de NDD utile puisse être mis en œuvre en employant une approche de dégradation périphérique. Cette technique de dégradation périphérique exige la connaissance de la direction actuelle de regard fixe de l'utilisateur. Pour cela, il faut tracer le mouvement de la tête et/ou des yeux. Il est aussi possible d'assumer que l'utilisateur regardera vers le centre d'affichage et que les objets sont donc dégradés par rapport à leur déplacement de ce point [Con01].

- **Avantage :**
  - Offre un outil efficace pour réduire le temps de calcul dans les systèmes de réalité virtuelle.
- **Inconvénient :**
  - La sélection d'un NDD optimal pour chaque objet dans la scène est difficile. Pour cela, plusieurs travaux sont consacrés à développer un mécanisme formel pour choisir le NDD le plus approprié.

Figure 3.9 : Les niveaux de détails dégradés en vision périphérique

3.2.5.1.5. Profondeur de champ (depth of field) :

La représentation d'un objet peut être choisie en se basant sur la profondeur du centre du champ de vision de l'observateur. Les objets dans les frontières ou derrière ce champ sont non concentrés. Ils sont donc représentés avec moins de détails, tandis que ceux dans le centre du champ de vision sont plus détaillés. [Con 01]

3.2.5.1.6. Vitesse (Velocity):

Comme le précise Reddy dans [Red94], un objet en mouvement est moins bien perçu par le système visuel humain qu'un objet fixe, il peut donc être simplifié. Si un objet tourne rapidement sur lui-même, les détails de sa surface seront moins perceptibles. On pourra alors choisir une représentation moins détaillée. Un autre cas est le mouvement par rapport à l'observateur. Si un objet passe rapidement dans le champ de vision de l'utilisateur, celui-ci le verra légèrement flou. On pourra ainsi également utiliser une représentation moins détaillée [Kru97].

Différentes techniques peuvent être utilisées pour le choix du niveau de détail, la proportion de la vitesse de l'objet à la taille de cet objet, par exemple [Con01].

3.2.5.1.7. Incidence

Dans certain cas, un objet peut avoir une forme particulière selon l'angle de vue. Par exemple, une porte ou un tube, vu de profil, se réduisent à des représentations très simples [Kru 97].

3.2.5.1.8. Taux d'affichage d'une image fixe :

Pour avoir une bonne interactivité, il faut maintenir un taux élevé et cohérent d'affichage de l'image. Ce taux sera maintenu indépendamment de la complexité de la scène.

Cette technique inclut, habituellement un planificateur dont le travail est d'analyser le système de chargement et d'assigner l'évaluation des NDDs à chaque objet. Il existe différentes approches pour cette technique: les systèmes réactifs, les systèmes prédictifs, les systèmes préemptifs [Con01].

- Le système réactif ajuste simplement le détail en se basant sur le taux d'affichage de l'image précédente. Si l'affichage de la dernière image a été achevé après une certaine limite, les détails seront réduits, autrement ils peuvent être augmentés.
- Le système prédictif estime la complexité de l'image et choisi le NDD de telle sorte qu'il n'excède pas une certaine limite.
- Dans le système préemptif, les objets du modèle sont rendus dans l'ordre de priorité. L'utilisateur met le taux d'affichage désiré, et si le temps permis pour une image s'est écoulé, l'affichage de cette image sera arrêté et on commence par l'image suivante. Ainsi, si le taux d'affichage est choisi et l'ordinateur n'est pas assez puissant, donc moins de détails du modèle seront perçus durant la navigation

3.2.5.1.9. Tâche :

Une des caractéristiques importantes de toute représentation est qu'elle doive être utile à une tâche. Il est alors envisageable d'adapter la représentation à la tâche en cours. Par exemple, en utilisant des niveaux de détails dégradés pour les objets qui n'interviennent pas directement dans la tâche. La dégradation des objets repose donc sur l'importance associée à l'objet considéré, notamment dans le cas de scènes interactives et dynamiques où il ne faut pas dégrader un objet important à la compréhension d'une action, par exemple.

3.2.5.1.10. Densité :

Il semble naturel de penser que si un objet est présenté au sein d'une scène complexe, ses détails seront moins perceptibles car l'œil humaine devra d'abord filtrer la grande quantité d'informations qui l'entoure. Nous pouvons ainsi suggérer que si la

densité d'objets est élevée, alors chacun de ces objets pourra être représenté avec moins de détails. La validité de cette hypothèse doit toutefois être confirmée par des expérimentations ergonomiques [Kru99].

3.2.6. Transition entre Niveaux de Détails :

Le basculement entre deux NDDs ou deux résolutions d'un même objet génère deux problèmes principaux :

- ***Problème de coût:*** un des objectifs d'utiliser les niveaux de détails est le gain dans le temps de calcul, mais si le basculement entre les NDDs est trop fréquent, ce gain sera perdu en calcul nécessaire de basculement. Pour pallier ce problème, il faut gérer deux seuils : un par sens de basculement. Les deux seuils étant assez éloignés, les phénomènes d'hystérésis dus à des allers-retours entre différents NDDs disparaissent. Cette technique permet donc de borner la fréquence de basculement.
- ***Problème de continuité:*** si deux images successives utilisent deux représentations différentes d'un objet, il est possible d'échanger immédiatement les représentations entre les deux images. Toutefois, cela peut provoquer des sauts perceptibles par l'utilisateur si les deux représentations sont utilisées alternativement dans plusieurs images successives. Pour résoudre ce problème, deux techniques peuvent être utilisées :
  - La première consiste à gérer les deux représentations en mémoire et à cacher progressivement la première pour afficher progressivement la deuxième en faisant varier la transparence des objets. Mais ceci est coûteux, puisque deux représentations du même objet sont présentes en mémoire et sont calculées à un même moment.
  - La seconde technique dite des géomorphes consiste à contraindre la construction des NDDs pour permettre une interpolation purement géométrique entre deux NDDs successifs.

3.2.7. Bilan :

A ce jour, il n'existe pas une méthode de simplification polygonale qui soit capable de simplifier tous les objets, quelque soit leurs caractéristiques. Par exemple, il existe une limitation importante concerne la définition du taux d'approximation désiré. Certains systèmes définissent un taux d'erreur qu'il est parfois difficile d'estimer.

D'autres, plus simple, permettent de choisir le nombre de polygones (ou de sommets) dans l'objet simplifié

Par contre, les algorithmes de simplification orienté scène sont assez primaires et posent certains problèmes .Par exemple, que faire si un objet se déplace dans une scène? De plus, ils ont tendance à produire des résultats assez peu satisfaisants du point de vue visuel, car ils peuvent produire des discontinuités entre les polygones et faire apparaitre des trous, ce qui restreint leur usage.

Mais, ils ont des avantages : ils s'intègrent bien dans les algorithmes de gestion de scène au sens large, comme l'élimination des objets non visibles (object culling).De plus, ils permettent de faire de la gestion de LODs progressive, en dégradant les détails sur les parties éloignées d'un objet [Kru 97].

3.2.8. Conclusion

Après la définition du principe de la multirésolution et les différentes techniques pour la génération des niveaux de détails et les critères de sélection de LODs ainsi que la présentation des problèmes générés par le basculement entre ces niveaux et les solutions proposées pour résoudre ces problèmes, nous consacrons la deuxième partie de ce chapitre à la présentation de l'utilisation de la technique de niveaux de détails dans la simulation comportementale et les différents travaux exploitant cette technique dans la simulation du comportement des humains virtuels .

4. Intégration de la technique de niveaux de détails dans la simulation comportementale

Un grand monde virtuel habité par des humains virtuels nécessite beaucoup de ressources informatiques. La simulation devrait exécuter un nombre très élevé d'événements en même temps ou dans un temps très proche, afin de réaliser une simulation crédible, rapide et en temps réel, quelque soit la taille du monde simulé ; et dans les limites des ressources informatiques utilisées.

Dans des jeux d'ordinateur, le problème est habituellement résolu en ne simulant pas les régions du monde qui ne sont pas observées par l'utilisateur (joueur). Cependant, ceci rend souvent l'environnement anormal ; nous pouvons voir quelques contradictions (par exemple, des monstres réapparaissent quand nous revenons à un certain endroit pour la deuxième fois).

En revanche, notre but était de maintenir la simulation aussi crédible que possible. En combinaison avec la condition d'utiliser un PC simple même pour des mondes virtuels très complexes, nous devons penser à une meilleure manière pour réduire la taille de la simulation. La technique de niveau de détail au niveau comportemental (LOD IA) est utilisée.

La technique de niveau de détail (LOD) est utilisée couramment dans les infographies pour la même raison – la réduction du coût informatique-. Les jeux d'ordinateur contiennent beaucoup d'objets mobiles qui consomment beaucoup de ressources informatiques.

L'idée de réduire la quantité d'instructions graphiques est basée sur le fait qu'il n'est pas important de rendre chaque aspect de la représentation visuelle du monde aux endroits ou l'utilisateur ne peut pas les voir correctement (par exemple, ils sont trop loin). Ces objets sont rendus selon leur position par rapport à l'œil de l'observateur.

L'utilisation de la technique de niveau de détail LOD au niveau comportemental consiste à effectuer un transfert du domaine des infographies des ordinateurs vers le domaine de l'intelligence artificielle. Cette dernière est basée sur une simple idée qui consiste en ce qui suit:

Il existe souvent peu de places, dans le monde artificiel à un moment donné de la simulation, ou le comportement des humains virtuels est important, celles de moindre importance n'ont pas besoin de simuler le comportement des humains virtuels avec précision, leurs simulation sera dégradée.

Si le comportement des humains virtuels est simulé partiellement, les demandes de la simulation en ressources informatiques peuvent être réduites significativement [Kkp+06].

4.1. Les travaux relatifs

Une idée très robuste pour l'emploi de la technique des LOD au niveau comportemental, en utilisant les machines à états définis hiérarchiques est présentée dans [Cha03], mais elle représente seulement un modèle qui ne semble pas avoir été encore exploré.

Sullivan et al. ont utilisé la technique de LOD pour le comportement conversationnel, à l'aide de l'outil de génération de comportement « BEAT » en utilisant la technique de passage de rôle [Scv+02].Les règles et les rôles peuvent être vus comme

des morceaux d'un programme posé au-dessus d'un humain virtuel de base. Si l'humain virtuel n'est pas visible, le rôle ne lui est pas passé, et par conséquent seulement le comportement de base est effectué. Contrairement à l'approche proposée, cette technique simplifie seulement le comportement conversationnel.

Figure3.10 : Une conversation normale dans un groupe

Une approche plus robuste pour la planification du temps machine pour chaque humain virtuel est présentée dans [WM00]. Cependant, comme les scripts comportementaux non planifiés ne sont pas exécutés, cette approche peut être insérée dans le domaine de la simulation « on/off ».

Un autre projet (IVE[1]: Intelligent Virtual Environnement) propose la simulation de grands mondes virtuels extensibles habités par des humains virtuels, en utilisant la technique de niveau de détail au niveau comportemental. Ce projet réduit la qualité de la simulation, en employant une simulation précise au niveau des localisations les plus importantes et une simulation approximative au niveau des localisations les moins importantes, en prenant en compte les ressources informatiques disponibles. Ce projet simplifie la simulation complète[2] [Kkp+06].

[1] Le code du projet, un scénario de test et la documentation peuvent être téléchargés à partir du site http : // mff.modry.cz/ive.
[2] Il simplifie le comportement et la topologie du monde artificiel.

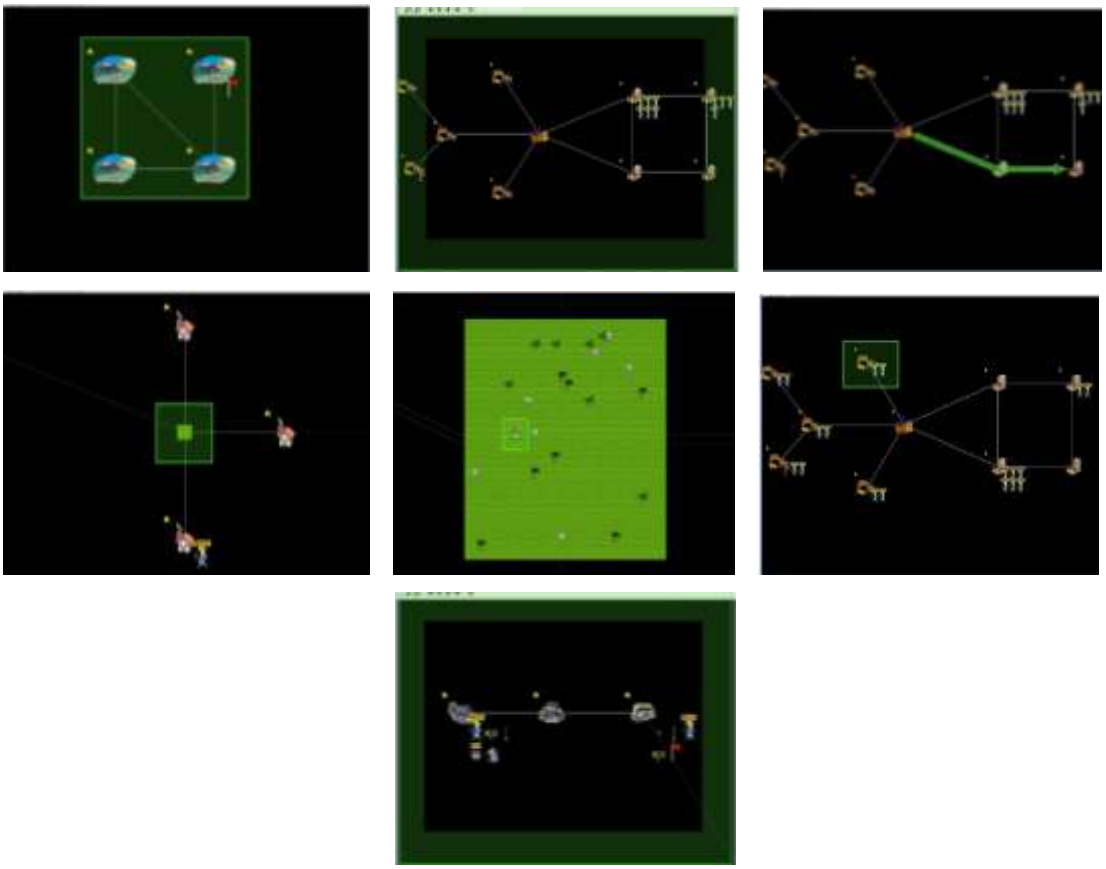


Figure 3.11 : Un scénario du projet IVE

Le tableau 3.1 montre une vue d'ensemble des travaux précédents utilisant la technique LOD IA, caractérisée par les critères suivants : le critère de détermination de LOD, le nombre de niveaux de détails, l'usage de LOD et les comportements simulé par l'IA.

Auteurs	**Le critère de détermination de LOD**	**Le nombre de LOD**	**L'usage de LOD**	**Les comportements simulés par IA**
Chenney et al. [Caf01]	Le potentiel de la visibilité	02	Mettre à jour le mouvement	La navigation, l'évitement de collision
O'Sullivan et al [Scv+02]	La distance	Non spécifié	La géométrie, les animations, l'évitement de collision, les gestes, les expressions faciales et la sélection d'action	La navigation, l'évitement de collision, le dialogue complexe entre les agents

Brockington [Bro02]	Distance	05	La recherche, la navigation, la sélection d'action dans un combat	La navigation, l'évitement de collision, les interactions de combat complexes
Niederberger and Gross [NG05]	La distance et la visibilité	21	La planification, l'évitement de collision, la recherche du chemin et les décisions de groupe	La navigation, l'évitement de collision et la planification du chemin
Pettré et al [Pcm+06]	La distance et la visibilité	05	La géométrie, la mise à jour de mouvement, l'évitement de collision, la navigation	La navigation, l'évitement de collision, la recherche de chemin
Brom et al. [Bsp07]	La distance simplifiée	04	La sélection d'action (avec l'arbre AND-OR), la simplification environnementale	La navigation, les interactions complexes avec les objets et les autres agents
Paris et al. [Pgo09]	La distance	03	La navigation et l'évitement de collision	La recherche de chemin, la navigation et l'évitement de collision
Lin et Pan [LP07]	distance	Non spécifié	La géométrie et les animations	La locomotion
Osborne and Dickinson [OD10]	La distance	Non spécifiée	La navigation, flockage, les décisions de groupe	La navigation
M.Wibner et al [Wka10]	La distance et la visibilité	08	La mise à jour du mouvement, l'évitement de collision, la navigation et l'exécution de l'action	La navigation, l'évitement de collision, les interactions basées sur les désires avec les agents et les objets intelligents, les dialogues

Tableau 3.1 : Comparaison des différentes approches de LOD IA

4.2. Conclusion:

Ce chapitre est divisé en deux axes principaux, la multirésolution et l'utilisation de la technique de niveau de détail dans la simulation comportementale.

Le chapitre suivant comportera une description détaillée de notre modèle proposé et une présentation des différents outils utilisés pour l'implémentation de notre modèle.

Chapitre 4

4. Le modèle proposé

4.1. Introduction

Les humains virtuels sont devenus fréquemment utilisés dans les domaines éducatifs et industriels. Les applications usant de ces techniques incluent les jeux vidéo, les fabulations virtuelles, l'industrie cinématographique, le divertissement, les simulations militaires et la modélisation comportementale. [Sps+06]

Par l'humain virtuel nous voulons dire un logiciel qui imite le comportement d'un humain dans un monde virtuel et qui est équipé d'un corps virtuel visualisé par une visionneuse graphique. [Blo+06]

Selon les recherches antérieures, il a été admis que le problème ne se limite pas dans la simulation d'un humain virtuel simple ayant un comportement significatif et crédible, mais il la transcende à la simulation d'un grand monde artificiel possédant un nombre important d'humains virtuels exécutant un comportement crédible qui de plus doit se faire en temps réel. Par ailleurs, il est également admis, que la simulation du comportement de tous les acteurs virtuels par un seul ordinateur, relève plutôt de l'impossible compte tenu des ressources informatique limitées.

C'est pour cette raison que la plupart des applications exhibent des humains virtuels dans de petits mondes artificiels (une chambre ou un village,…), ou exhibent une petite partie du comportement de l'être humain (saisir un objet ou marcher,…..). Par ailleurs, une application ou une technique exhibant une grande simulation serait extrêmement utile dans les domaines des jeux vidéo et dans les fabulations virtuelles. [Sps+06]

En résumé, les problèmes de la simulation de grands mondes virtuels concernent :

- La crédibilité dans le processus de simulation du comportement des humains virtuels.
- La simulation rapide et en temps réel, mais dans les limites des ressources informatiques disponibles.

Notre travail s'est fixé pour objectif la présentation d'un modèle permettant la prise en charge des deux problèmes cités ci-dessus.

Notre modèle présenté s'est inspiré de deux projets de recherche, dont :

- Le premier étant basé sur des techniques d'animation des niveaux de détails.
- Le second s'intéresse à une technique de niveau de détail dans le domaine d'intelligence artificielle LOD IA, pour la simulation de grands environnements virtuels et crédibles.

Ces deux techniques devant concourir à la création d'environnements virtuels peuplés par des humains virtuels.

Le premier volet de recherche est entrepris par le groupe de la synthèse d'image, qui se caractérise par le niveau de détail adaptatif du système d'animation humain (ALOHA). Son but est d'animer et de rendre des humains-virtuels en temps réel [Ps 00]. Le système ALOHA s'est basé sur les limitations du système visuel humain, pour utiliser la technique de niveau de détail LOD afin de calculer :

- Les modèles moins précis quand la perte de précision est peu susceptible d'être notée ;
- Les modèles plus précis susceptibles d'être le centre de l'attention de l'observateur. [MD+02]

Le deuxième volet de recherche touche la technique LOD IA incorporée dans le projet IVE (Intelligent Virtual Environment) qui vise la simulation rapide et crédible du comportement des humains virtuels dans un grand environnement virtuel, mais dans les limites des ressources informatiques disponibles.

Dans la recherche de la solution, nous avons décidé d'utiliser les concepts décrits dans la section suivante :

4.2. Les concepts de base

Dans cette section, nous décrivons les concepts de base du modèle proposé : la hiérarchie des localisations et des processus, les objets, la planification et les règles hiérarchiques if-cond-then et la technique « LOD IA ».

4.2.1. Terminologie

Nous avons également décidé de définir une terminologie plus appropriée pour les humains virtuels. Ces derniers se composent de deux parties :

- La première partie est le corps sans aucune intelligence, c'est lui qui effectue réellement les actions : c'est un *acteur*
- La deuxième partie est le cerveau qui donne l'intelligence aux acteurs. l'humain virtuel l'utilise en tant que sa source dans la prise des décisions : on l'appelle le *centre décisionnel.*
- La troisième partie est la *mémoire* de l'humain virtuel.

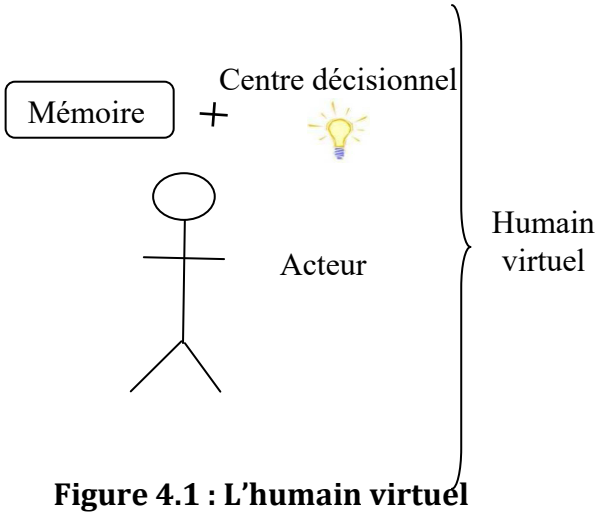


Figure 4.1 : L'humain virtuel

Et le triplet <centre décisionnel, mémoire, acteur> s'appelle un *humain virtuel*. Voir Figure 4.1

4.2.2. La hiérarchie des localisations

Le monde se décompose en des localisations organisées dans une structure hiérarchique arborescente. La racine de la hiérarchie représente le monde entier, ses enfants sont les parties de base du monde. Par exemple sur la figure 4.2 nous pouvons voir un monde qui se compose de deux villes celles-ci se composent à leur tour par d'autres endroits.

Les niveaux de la hiérarchie d'arbre correspondent aux valeurs de niveaux de détails (LOD_value), cela signifie, que nous avons seulement un monde, représente la vue la plus

approximative, ainsi nous pouvons avoir la valeur du niveau de détail du monde(LOD_value) est 1. En voyant les villes, cela nous donne la vue la plus détaillée du monde, sa valeur est donc plus grande- 2 et ainsi de suite.

La décomposition du monde en des niveaux est basée sur des critères prédéfinis par le concepteur (la distance de cette localisation par rapport à la position de l'observateur, la complexité de cette localisation, la périphérie,…….).

- *Si le critère est la distance par rapport à l'œil de l'observateur*

La localisation la plus proche à l'œil de l'observateur possède la valeur de niveau de détail plus grande (elle sera plus détaillée) que celle moins proche (qui sera moins détaillée).

- *Si le critère est la complexité de la localisation*

La localisation la moins complexe possède une valeur de niveau de détail plus grande que la plus complexe.

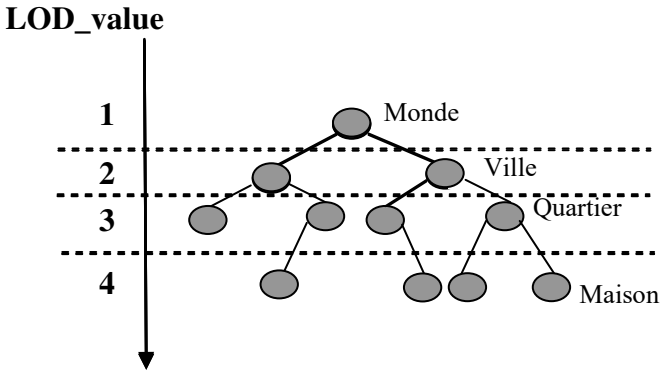


Figure 4.2 : La hiérarchie des localisations consistant en un monde et deux villes

4.2.3. Les objets

Nous pouvons distinguer deux types d'objets :

- Les acteurs : qui sont le corps des humains virtuels –sans le centre décisionnel et la mémoire-.
- Les objets statiques

Les objets sont organisés dans une structure hiérarchique arborescente ; ils peuvent contenir d'autres objets. En fait, le monde se compose d'un seul objet dans lequel d'autres objets sont situés. Cet objet parent peut également produire des objets qu'ils le composent. C'est utile quand nous ne simulons pas une partie du monde virtuel. Si la place où cet objet parent est localisé devient importante ainsi c'est ici ou nous voulons commencer à simuler, l'objet parent a toutes les informations sur des objets qu'il contient et il peut les générer.

Chaque objet est situé quelque part dans la hiérarchie des localisations. La valeur de LOD d'un objet est définie en correspondance de niveau de détails de la localisation dans laquelle l'objet est situé.

4.2.4. La planification et les règles hiérarchiques if-cond-then

La planification est un processus par lequel les humains virtuels prennent leurs décisions dans le monde virtuel afin d'acquérir leurs objectifs.

Dans le processus de planification rien n'est prévu à l'avance. L'humain virtuel décide à chaque fois que le monde change selon l'ensemble de règles qui lui ont été fournies.

Les règles ont leurs priorités et elles sont toutes traitées lorsque l'humain virtuel veut prendre une décision. La première règle qui correspond à l'état du monde (la règle est la première en fonction de sa priorité) est choisie et l'action correspondante est exécutée. Par exemple, l'humain virtuel peut avoir l'ensemble de règles suivantes : (à partir de la plus prioritaire):

1. **Si** le soleil brille, aller à la piscine.
2. **S**'il neige, faire du ski.
3. **Si** cela est vrai, rester à la maison.

Ensuite, à chaque fois que la décision doit être prise, les conditions des règles sont testées à partir du numéro 1 au numéro 3. Si l'une des conditions est vraie, l'action de la règle appropriée est exécutée et le reste des règles est ignoré. Ainsi, si le soleil brille et il neige en même temps, l'humain virtuel va toujours à la piscine et non pas au ski parce que la deuxième règle est moins prioritaire que la première. Si le soleil ne brille pas et il ne neige pas, la troisième règle est toujours exécutée parce que sa condition est toujours vraie.

Les humains virtuels prennent leur décision à partir de:

- Ses propres croyances qui représentent comment il perçoit le monde et comment il traduit ses perceptions à sa propre représentation.
- L'intention ce sont les buts de l'humain virtuel

Pour utiliser la technique de niveau de détail également à un niveau comportemental, les règles **if-cond-then** sont utilisées. Ces règles décrivent l'action dans le monde.

if conditions_sont_complies **do** sous_arbre_des_taches **then** résultat

- **Les conditions** : nous indiquent quand l'action peut être faite (par exemple, je dois avoir des ciseaux pour couper les cheveux),

- **Le sous-arbre des tâches** : sont d'autres règles au niveau inférieur ou actions atomiques qui doivent être exécutées pour atteindre l'objectif (par exemple : à couper la partie arrière, partie avant,…..). Le sous-arbre des taches peut être ignoré.
- **Le résultat** : est l'action à exécuter (couper les cheveux,….) [kkp+06]

4.2.5. La hiérarchie des processus

Un processus est la suite d'actions que l'humain virtuel exécute pour satisfaire son objectif.

Les processus sont organisés dans une structure hiérarchique étroitement liée à la hiérarchie des locations. (Voir Figure.4.3). Chaque processus peut être exécuté d'une façon atomique ou augmentée aux sous-processus. Chaque processus exécuté peut se trouver dans un de ces états :

- *Non-existant* : un super processus est exécuté. Ce processus ne sera pas visible.
- *Atomique* : le processus fonctionne d'une manière atomique. Et il effectue des changements au niveau de l'état du monde virtuel.
- *Augmenté* : le processus est augmenté aux sous-processus. Un tel processus n'effectue aucune action, il attend seulement que ses sous-processus effectuent tout le travail. Cependant, il peut devenir atomique par la suite en fonction des changements de la position de l'observateur par exemple.

Chaque processus situé, quelque part, dans la hiérarchie des processus possède une valeur de complexité (LOD_complex). Les valeurs de complexité des processus correspondent à leur profondeur dans la hiérarchie. (Voir figure.4.3)

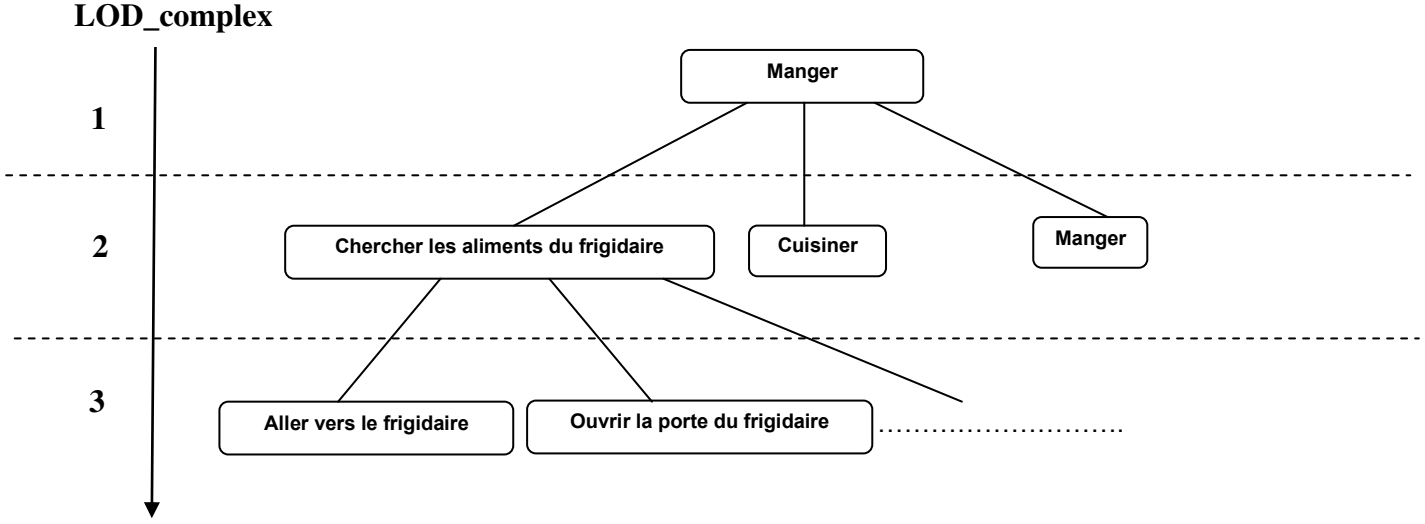


Figure4.3 :La hiérarchie des processus avec leurs valeurs de complexité

Les règles hiérarchiques if-cond-then sont utilisées pour décrire les processus. L'exécution d'une action est guidée par le concept de but. Les processus également n'augmentent pas directement aux sous-processus, mais plutôt aux sous-buts (voir la figure 4.4.a et figure 4.4.b). Dans ce concept, le centre décisionnel d'un humain virtuel obtient la liste de buts requis pour satisfaire le but du processus père et son travail est de trouver et exécuter les sous-processus appropriés.

Figure4.4.a : La hiérarchie des processus sans les buts

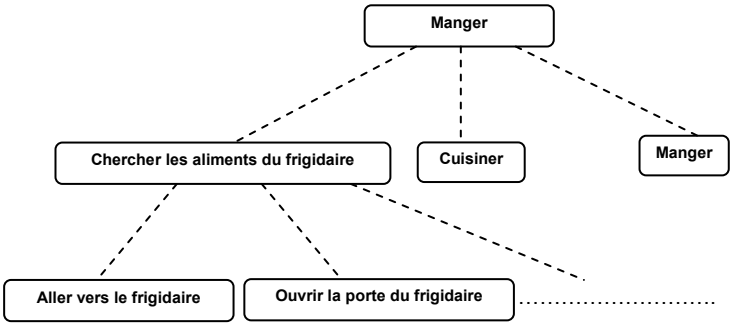
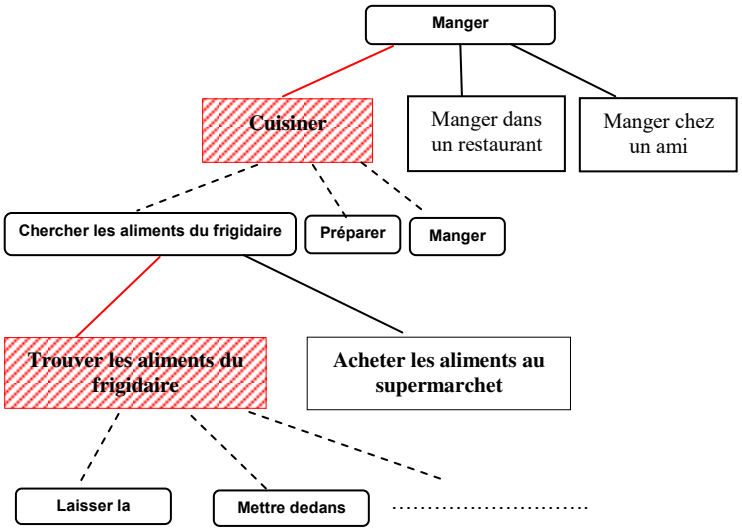


Figure 4.4.b : La hiérarchie des processus-objectif, lorsque le processus agit comme une mise en œuvre d'un but.



4.2.6. Niveau de détail comportemental (LOD IA)

L'utilisation de la technique de niveau de détail LOD au niveau comportemental consiste à effectuer un transfert du domaine des infographies des ordinateurs vers le domaine de l'intelligence artificielle. Cette dernière est basée sur une simple idée qui consiste en ce qui suit:

Il existe souvent peu de places, dans le monde artificiel à un moment donné de la simulation, ou le comportement des humains virtuels est important, celles de moindre importance n'ont pas besoin de simuler le comportement des humains virtuels avec précision, leurs simulation sera dégradée.

Si le comportement des humains virtuels est simulé partiellement, les demandes de la simulation en ressources informatiques peuvent être réduites significativement.

Par conséquent, les problèmes apparaissent au moment ou nous essayons d'identifier explicitement les places importantes(ou la simulation du comportement doit être précise), et les parties du monde moins importantes où la simulation du comportement sera approximative. Ainsi, ceci peut se résumer dans les questions suivantes :

1. Comment identifier les places importantes et celles moins importantes ?
2. Comment l'humain virtuel peut adapter son comportement selon l'importance de son endroit?

L'approche de la technique LOD au niveau comportemental proposée est étroitement liée à la hiérarchie des processus (voir section 4.2.5) et l'utilisation des règles **if-cond-then** (voir section 4.2.4).

4.3. L'architecture proposée

Une vue élevé de l'architecture proposée nous indique quatre modules importants (voir figure.4.5.) : la base de règles, le séparateur LOD, le contrôleur géométrique, le contrôleur d'animation et le contrôleur de comportement.

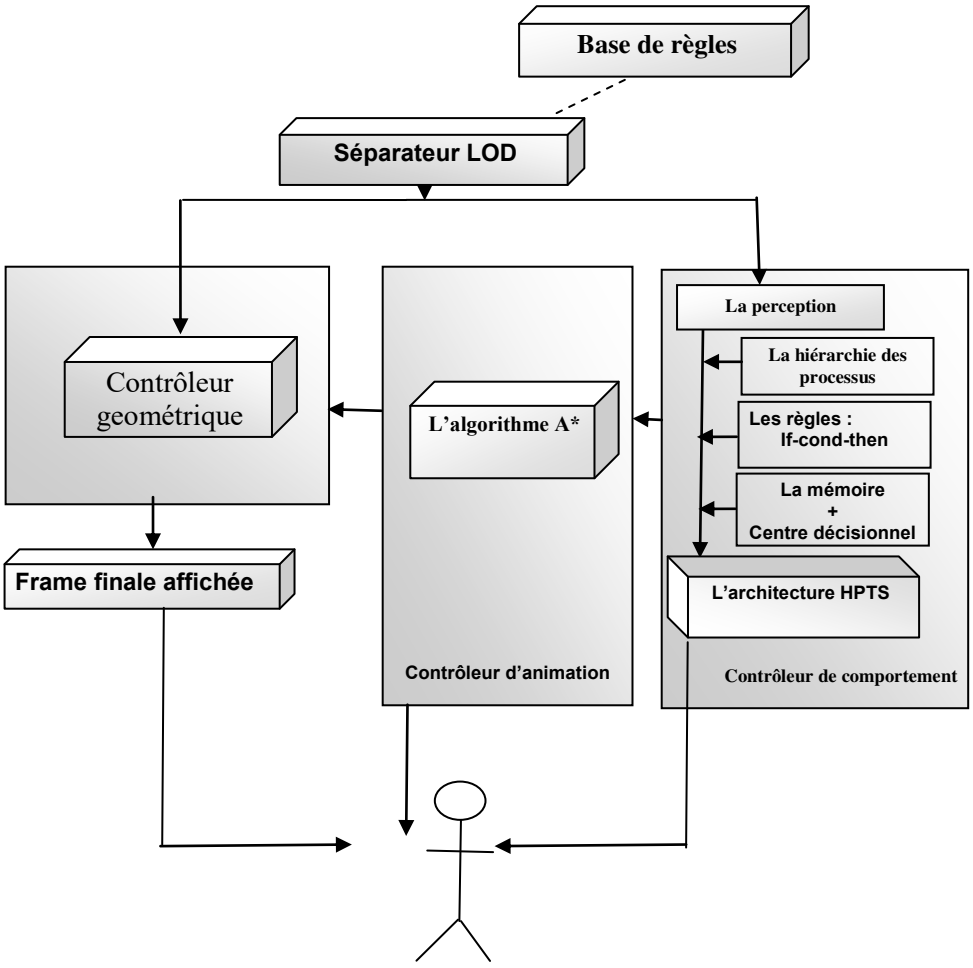


Figure.4.5 : Une vue élevée de l'architecture proposée

4.3.1. La base de règles

La base de règles est fondamentalement un ensemble de règles qui contrôle le niveau de détail pour chaque objet (statique ou un humain virtuel) qui se trouve dans le monde virtuel. L'ensemble de règles peut être réglé par le concepteur pour être en conformité avec les capacités du matériel informatique utilisé. Les règles sont spécifiées pour contrôler le niveau de détail géométrique, le mouvement et le comportement de chaque objet, basé sur des critères simples à savoir : la vitesse, la position dans le plan d'observation, la distance du spectateur,….etc. (voir section 3.2.5.1)

4.3.2. Le séparateur LOD

Ce module est le cœur de l'architecture proposée. Le séparateur LOD ne se limite pas à la définition de la structure hiérarchique et les différents niveaux de détails du monde virtuel, mais il canalise également les paramètres qui guident les contrôleurs géométriques et de

comportement. La communication entre le séparateur LOD et les deux contrôleurs se produit sur la base d'un maître / esclave.

Après avoir extraire les attributs des objets nécessaires et définir leur valeurs de niveaux de détails à l'aide des directives de la base des règles, il envoie les résultats aux contrôleurs. En outre, ce module agit comme une couche d'abstraction entre le contrôle du monde virtuel et sa composition, ce qui aboutit à un système plus extensible.

4.3.3. Le contrôleur de comportement

Notre contrôleur de comportement a la capacité de contrôler des comportements compliqués.

C'est le devoir du contrôleur de comportement pour stocker l'information observée dans la mémoire associée à chaque humain virtuel, pour mettre à jour la connaissance, il utilise ainsi l'ensemble de règle hiérarchiques et il emploie la technique de niveau de détail au niveau comportemental (LOD IA) afin d'établir le rôle approprié pour chaque humain virtuel en utilisant l'architecture HPTS (Voir section 1.2.3.2.10).

4.3.3.1. Mécanisme de sélection d'un comportement selon une intention

Une fois un humain virtuel a commis une intention (un but), il doit agir afin d'accomplir cette intention.

Ces intentions peuvent être hiérarchiquement imbriquées. Plus spécifiquement, afin d'accomplir une intention, on peut avoir des intentions secondaires.

L'algorithme de sélection, d'une action selon une intention, est exécuté par les centres décisionnels des acteurs. L'algorithme du choix d'action selon une intention alloue les intentions des humains virtuels .À un instant donné, une intention est active. Quand l'intention active est atteinte par un processus, une nouvelle intention devient active

Algorithme1 : La sélection d'une action-intention

Cet algorithme est exécuté par le centre décisionnel d'un humain virtuel

En entrée : Un acteur dans une localisation donnée et son centre décisionnel avec un ensemble des intentions

1) I ← ensemble des intentions
2) I_a ← l'intention active (à partir de I)
3) A ← action à exécuter pour satisfaire I_a
4) Passer A à l'architecture subsomption afin d'établir le rôle de l'humain virtuel
5) Lorsque l'exécution de A se termine, allez à l'étape (1)

4.3.3.2. Mécanisme hiérarchique de sélection d'un comportement selon une intention

Afin qu'un humain virtuel atteint son intention, il doit exécuter un processus qui est une suite des actions.

Chaque action est couplée avec un ensemble de paires <sub_Intention,advise> :

- Sub_intention est une intention secondaire que l'humain virtuel doit la commettre afin d'accomplir une partie de l'action.
- Un advise est une règle que l'humain virtuel devrait respecter pour activer les sub_Intention de l'action dans un certain ordre afin que l'intention originale soit satisfaire.

En conclusion, l'action sera exécutée par l'exécution de quelques autres actions qui accomplissent les sub_intention.

Par exemple, les sub_intention de l'action « Arroser un jardin », peuvent être :

(a) sub_intention1 : Trouver « ce qui a les moyens d'exécuter l'arrosage » (un bidon par exemple),

(b) sub_intention2 : Remplir,

(c) sub_intention3 : Trouver « ce qui a les moyens d'être arroser » (un jardin par exemple),

(d) sub_intention4 : Nettoyer « celui qui a les moyens d'exécuter l'arrosage ».

En respectant advise, (c) sera exécuté dans une boucle, pour chaque jardin une fois.

Voir l'algorithme 2 ci-dessous :

Algorithme2 : La sélection hiérarchique d'une action-intention

Cet algorithme est exécuté par le centre décisionnel d'un humain virtuel
En entrée : Un acteur dans une localisation donnée et son centre décisionnel avec un ensemble d'intentions

1) I ← ensemble des intentions et des sub_Intention
2) I_a ← l'intention active (à partir de I)
3) A ← action à exécuter pour satisfaire I_a (à partir de l'algorithme 1)
4) If A est une action atomique, then
   Passer A à l'architecture subsomption
   Lorsque l'exécution de A se termine, allez à l'étape (1)
5) else I← I U A [sub_Iintention] ; allez à l'étape (1)

Dans l'étape (2) de l'algorithme 2, le centre décisionnel choisit l'intention active sur la base de tous les advise des sub_Intention.

4.3.3.3. Niveau de détail d'intelligence artificielle (LOD IA)

Pendant la simulation, des valeurs de complexité de niveau de détail (LOD_complex) sont associées aux niveaux de la hiérarchie des processus par le concepteur. Par exemple, la complexité de LOD d'arroser un jardin peut être 3, alors que l'arrosage d'un lit individuel du jardin peut être 4. Puis, au cours de la simulation, à chaque instant, les valeurs de LOD sont assignées à tous les endroits. En outre, l'étape (4) de l'algorithme 2 est raffinée comme suit :

if A est une action atomique, ou A.LOD_complexe = location.LOD_value, then …..

Cela signifie que les sub_Intention de l'action A ne seront pas transmises au centre décisionnel, si la valeur de LOD est si basse. Au lieu de cela, l'action A sera exécutée comme si elle est atomique.

4.3.4. Le contrôleur d'animation

Le contrôleur d'animation est le responsable qui manipule n'importe quelle animation à n'importe quel niveau de détail spécifié par le séparateur de LOD, pour fournir un cadre extensible. C'est son devoir d'enregistrer les demandes d'animation, pour stocker les informations nécessaires, de mettre à jour les animations en cours et de gérer le stockage des vecteurs d'état pour les objets.

4.3.4.1. L'algorithme A*

Pour faire déplacer les objets d'un point à un autre, dans une scène bidimensionnelle ou tridimensionnelle, le système doit calculer le chemin optimal entre deux points. Dans notre recherche de chemin heuristique nous avons utilisés l'algorithme A* [NS76].

L'algorithme standard de recherche pour le problème de chemin le plus court dans un graphe est l'algorithme A*. C'est une recherche en largeur dirigée qui combine les avantages des autres méthodes, citées précédemment dans le chapitre 02, et qui utilise une fonction de la forme:

$f(n) = g(n) + h(n)$ selon les étapes suivantes:

1. Ajouter le point de départ à une "liste ouverte";
2. Répéter les instructions suivantes:
    a. Rechercher la cellule ayant le plus faible coût ***f*** dans la "liste ouverte", qui devient la cellule en cours;
    b. Eliminer la cellule en cours de la "liste ouverte" et l'ajouter à la "liste fermée";
    c. Pour les 8 cellules adjacentes à la cellule en cours, faire:
        - Si on ne peut pas traverser la cellule, on l'ignore;
        - Si elle n'est pas dans la "liste ouverte", on l'y ajoute. La cellule en cours devient le parent de cette cellule. On calcule les coûts ***f, g*** et ***h*** de cette cellule.
        - Si elle est déjà dans la "liste ouverte", on teste si le chemin passant par la cellule en cours est le meilleur, en comparant les coûts ***g***;

Un coût ***g*** inférieur: signifie un meilleur chemin;

- Si c'est le cas, on change le parent de la cellule pour devenir la cellule en cours, en on recalcule les coûts ***f*** et ***g***. Si on conserve une

"liste ouverte" triée par coût ***f,*** la liste doit être retirée à ce moment là;

d. On s'arrête lorsque:

- La cellule de destination est ajoutée à la "liste ouverte";
- On ne trouve pas la cellule de destination et la "liste ouverte" est vide;

3. Enregistrez le chemin. En partant de la cellule de destination, remontez d'une cellule à son parent jusqu'à atteindre la cellule de départ.

Une vue d'ensemble du système d'animation tout en évitant les collisions est décrite dans la figure 4.6

```
Pour chaque agent faire
Appliquer l'algorithme A* pour chercher itinéraire
Fin pour
Pour chaque pas d'animation comportementale faire
    Pour chaque agent faire
        Prédiction de collision
        Si pas de collision alors
            L'agent suit son chemin
        Sinon
            Appliquer l'algorithme d'évitement de collision
        Fin si
    Fin pour
Fin pour
```

Figure 4.6 : Algorithme général du système.

4.3.5. Le contrôleur géométrique

Le contrôleur géométrique encapsule toutes les informations nécessaires pour décrire l'aspect externe de l'objet dans le cadre actuel.

A travers de la communication produite entre les deux contrôleurs (géométrique et de mouvement), un vecteur d'état qui décrit les orientations et les positions des objets est transmis à partir du contrôleur de mouvement vers le contrôleur géométrique. Ce vecteur est ensuite

enveloppé dans les surfaces extérieures, selon le niveau de détail précisé par le séparateur LOD, au niveau du contrôleur géométrique.

4.4. L'environnement choisi

L'implémentation de notre modèle proposé nécessitera la disponibilité d'un environnement physique réaliste (3D) parce qu'il nous permet de réaliser un monde virtuel proche du monde réel qui de plus devra permettre à notre humain virtuel de réaliser des mouvements réalistes proches des mouvements de l'humain réel.

Il y'a lieu, enfin de conclure que dès lors l'étape du choix de l'environnement réalisée, il sera question de trouver le moyen ou bien l'outil qui assure la simulation de toutes les caractéristiques nécessaires pour cet environnement, c'est le moteur physique.

4.4.1. Les moteurs physiques

Plusieurs travaux cherchent à faire une simulation réelles des comportements des humains virtuels dans des environnements physiques réalistes et en leur faisant reproduire des mouvements réalistes également. De ce fait, le seul moyen qui leur permet de garantir ce réalisme réside dans la simulation où l'on a à reproduire les lois de la physique. La simulation crédible du comportement des humains virtuels sur un environnement 3D incluant la physique était alors un travail relativement complexe à faire. Actuellement et depuis les années 2000, nous disposons des outils (moteurs physiques libre) et du matériel (puissance des ordinateurs) nécessaires pour une bonne simulation d'humains virtuels quelque soit leur formes ou les mouvements qu'on voudra leur faire reproduire.

Parmi les moteurs physiques qui sont à l'usage public (ou libre) nous pouvons citer, Breve[1], ODE[2], Newton Dynamics, OGRE[3]et bien d'autres, ceux là étant les plus répandus.

Notre choix c'est retourné vers le moteur physique le plus populaire qui est ODE (Open Dynamics Engine).

4.4.2. Open Dynamics Engine

Open Dynamics Engine (ODE) est une bibliothèque logicielle open source se plaçant dans la catégorie des moteurs physiques. Elle sert à simuler l'interaction physique de corps rigides. Elle est actuellement utilisée dans les jeux vidéo, les outils 3D et les outils de simulation.

[1] www.spiderland.org/
[2] http://ode.org/.
[3] http://www.ogre3d.org/.

ODE est disponible sur plusieurs plateformes (Linux, Mac OS ou Windows) et utilise une interface de programmation en C pour une plus grande compatibilité, bien qu'en interne, le code source soit écrit en C++. Cette bibliothèque possède plusieurs types de jointures et intègre un détecteur de collision avec friction. Le moteur utilise plusieurs intégrateurs en fonction de la précision et de la robustesse de la simulation désirées.

Plusieurs primitives sont disponibles et le moteur peut gérer les surfaces constituées de triangles. [ODE, 2006]

Nous utiliserons les 2 aspects d'ODE, à savoir le moteur physique et la gestion des collisions. ODE donnera un aspect « réel » à l'application.

4.4.2.1. Corps rigide

Un corps rigide simulé avec ODE possède les propriétés suivantes :

Les propriétés qui changent avec le temps sont :

- Position du vecteur (x, y, z) du point de référence du corps, généralement le point de référence doit correspondre au centre de la masse du corps.
- La vitesse linéaire correspondante au point de référence, qui est un vecteur (v_x, v_y, v_z).
- L'orientation d'un corps rigide, représentée par un Quaternion (q_s, q_x, q_y, q_z) ou une matrice 3*3 de rotation.
- Vecteur de vitesse angulaire (w_x, w_y, w_z) qui décrit comment l'orientation change
- avec le temps.

D'autres propriétés sont constantes dans le temps:

- La masse du corps.
- La position du centre de masse (centre d'inertie). Dans les simulations actuelles le centre de la masse et le point de référence doivent coïncider.
- La matrice d'inertie est une matrice 3*3 qui décrit la manière dont la masse du corps est divisée autour du centre de masse.

4.4.2.2. Intégration du temps

Le processus d'une simulation d'un corps rigide nécessite l'intégration d'une contrainte très importante qui est le temps. Chaque pas (step) d'intégration avance le temps courant d'une valeur « step size » donnée qui fait ajuster l'état de tout le corps par la nouvelle valeur du temps.

4.4.2.3. Application des forces

Pour chaque pas de simulation, l'utilisateur peut appeler une fonction qui applique une force au corps rigide. Cette force est ajoutée à une accumulation de forces qui sera remise à zéro après chaque pas d'intégration.

4.4.2.4. Application de jointures

Il existe une relation qui est appliquée entre deux corps (primitives) qui peut avoir certaines positions et orientations par rapport aux autres. Cette relation est appelée contrainte ou jointure. Il existe différents types de jointures pouvant relier les parties d'un corps rigide (primitives géométriques):

- Ball and socket (voir figure 4.7.(1))
- Hinge (voir figure 4.7(2)) ;
- Slider (voir figure 4.7(3)) ;
- Universal (voir figure 4.7 (4)) ;
- Hing-2 (voir figure 4.7(5)) ;
- Fixed ;
- Contact (voir figure 4.7(6))

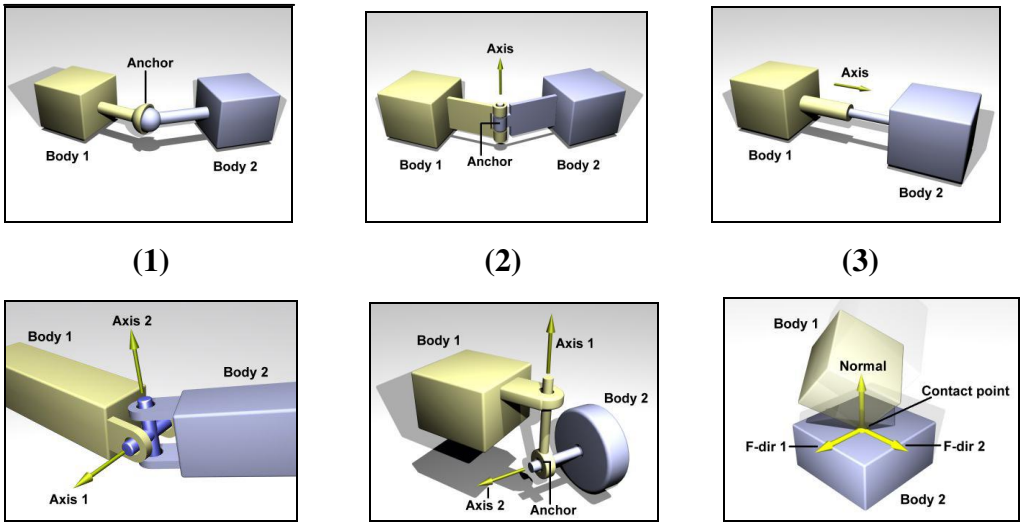


Figure 4.7: Les types de joints

Nous donnons ci-après une explication détaillée sur le fonctionnement de chacune des jointures utilisées au niveau de notre modèle proposé dans le chapitre suivant :

- **Hinge Joint**

La jointure de ce type, contraint le mouvement des deux parties reliées par cette jointure et cela pour les retourner autour d'un axe prédéterminé. Le joint de type hinge tient compte d'un degré de liberté seulement, et c'est la jointure la plus simple. Au niveau de notre simulation des humains virtuels, nous utilisons seulement 2 senseurs :

- angle hinge courant,
- le taux de l'angle hinge courant.

Notant h(t) l'angle courant entre 2 parties du corps à l'instant t, et r(t) le taux de l'angle hinge qui est défini comme la dérivée de l'angle hinge en fonction de temps :

$$r(t) = \frac{d}{dt} h(t)$$

La valeur retournée pour l'angle hinge sera entre - PI et PI, cette jointure possède deux senseurs qui peuvent fournir l'angle courant et la vitesse angulaire du corps, la jointure possède également un effecteur qui affecte la vitesse désirée en tant que son entrée.

- **Ball & socket joint**

Ce type de jointure est le plus complexe des jointures existantes dans ODE, cette jointure tient compte de 3 degrés de liberté autour des 3 axes. De plus, cette jointure fournit 2 senseurs de rétroaction pour chaque axe. Ces deux senseurs fournissent les données suivantes :

- L'angle Ball & socket pour l'axe donné,
- Le taux d'angle de Ball & socket pour l'axe donné.

Le taux de l'angle pour un axe est calculé en prenant la dérivée en ce qui concerne la période de l'angle courant sur l'axe particulier. En raison des 3 degrés de liberté, les 2 senseurs peuvent fournir jusqu'à 6 informations au RNN.

- **Contact joint**

Le joint de contact empêche le corps 1 et le corps 2 de s'interpénétrer au point de contact.

Il fait ceci en permettant seulement aux corps d'avoir une vitesse " sortante " dans la direction de la normale de contact.

Des joints de contact sont dynamiquement créés et supprimés en réponse à la détection de collision. Les joints de contact simulent le frottement au contact en appliquant des forces dans les 2 directions de frottement (friction) qui sont perpendiculaires à la normale [Smi02].

Les joints de contact ont typiquement une vie liée au pas de temps. Ils sont créés et supprimés en réponse à la détection de collisions.

- **Fixed joint**

Un joint fixe maintient une position et une orientation relatives fixes entre deux corps, ou entre un corps et l'environnement statique. L'utilisation de cette jointure n'est presque jamais une bonne idée dans la pratique, exceptée dans le cas où on procède à des corrections. Si on a besoin de deux corps pour être collés ensemble, il est préférable de les représenter comme un simple corps.

4.4.2.5. La détection de collision

ODE possède deux composants principaux: un moteur de simulation de la dynamique et un moteur de détection de collisions. Le moteur de détection de collisions fournit à l'utilisateur des informations sur la forme de chaque corps. A chaque pas de temps, le moteur de détection de collisions détermine le corps qui présente une collision potentielle avec les autres et fournit ensuite à l'utilisateur l'information sur le point de contact. L'utilisateur crée alternativement des joints de contact entre les corps.

- **Geoms**

Les objets géométriques (primitives géométriques, Sphere, Box, Capped cylinder, Ray, Triangle mesh, Simple space) abrégés par Geoms sont les éléments fondamentaux dans le système. Un Geom peut être une primitive géométrique ou un groupe de primitives géométriques qui représente un type spécifique de Geom appelé Space. Space est également similaire au concept World sauf qu'il est appliqué à la collision et non pas à la dynamique.

- **World**

L'objet du monde est un récipient pour les corps rigides et les joints. Les objets appartenant à différents mondes ne peuvent pas agir l'un sur l'autre, par exemple les corps rigides de deux mondes différents ne peuvent pas se heurter. Tous les objets dans un monde existent au même point dans le temps, ainsi une raison d'employer les mondes séparés doit simuler des systèmes à différents taux. La plupart des applications nécessiteront l'utilisation seulement d'un seul monde.

- **Space**

Un espace est un Geom ne pouvant être placé et qui peut contenir d'autres Geoms. Il est semblable au concept de corps rigide du "monde", sauf qu'il s'applique à la collision et non pas à la dynamique.

4.4.2.6. Usage de la carte graphique

ODE est un moteur logiciel, il est complètement indépendant de n'importe quel librairie graphique (i.e. une librairie associée à une carte graphique comme: PhysiX[4]). Cependant, les réalisations exploitant ODE utilisent OpenGL. Cependant, et selon les besoins des utilisateurs de ce moteur, les efforts nécessaires à l'exploitation d'une librairie avec une simulation basé ODE, sera exclusivement à la charge de l'utilisateur.[Nes09]

4.4.3. Librairies graphiques

4.4.3.1. OpenGL(Open Graphics Library)

OpenGL est une API (Application Programming Interface) multi plate-forme pour le développement d'applications gérant des images 2D et 3D. Elle a été développée en 1989 par Silicon Graphics, puis portée sur d'autres architectures en 1993. Depuis 1992, sa spécification est surveillée par l'OpenGL Architecture Review Board (ARB).

Son interface regroupe environ 150 fonctions qui peuvent être utilisées pour afficher des scènes tridimensionnelles complexes en décrivant des objets (caméras, lampes, modèles 3D…) et les opérations que l'on peut effectuer pour les manipuler.

Elle à été implémentée sur la plupart des plates-formes informatiques actuelles (Linux et Unix, Windows, MacOs, BeOS...) et est disponible pour de nombreux langages de programmation (C, C++, Java, Fortran, Ada...).

Du fait de son ouverture, de sa souplesse d'utilisation et de sa disponibilité sur toutes ces plates-formes, elle est utilisée par la majorité des applications scientifiques, industrielles ou artistiques 3D et certaines applications 2D vectorielles. [Nes09]

4.5. Conclusion

Dans ce chapitre, nous avons essayé d'expliquer notre modèle proposé qui consiste en l'utilisation de la technique de niveaux de détails à un niveau comportemental « la technique LOD IA », afin que la simulation des comportements des humains virtuels soit crédible, rapide et en temps réel, mais dans les limites des ressources informatiques disponibles. Pour ce faire,

[4] *La carte physiX d'Ageia (Nvidia), carte dédiée aux calculs physiques.*

nous avons commencé par la définition des différents concepts de base utilisées dans le modèle proposé, et ensuite nous avons défini les cinq modules importants constituant l'architecture proposée, qui sont les suivants :

- La base de règles qui est un ensemble de règles utilisées pour définir les différents niveaux de détails selon des critères déterminés par le concepteur.
- Le séparateur LOD est le responsable de la définition des différents niveaux de détails du monde virtuel.
- Le contrôleur de comportement est le responsable de la définition du comportement approprié pour chaque humain virtuel selon son niveau de détail en utilisant les règles hiérarchiques et la technique LOD IA.
- Le contrôleur d'animation est utilisé pour manipuler n'importe quelle animation à n'importe quel niveau de détail défini par le séparateur de LOD.
- Le contrôleur géométrique décrit l'aspect externe de l'objet selon son niveau de détail.

Ensuite, nous avons présenté les outils utilisés pour réaliser notre implémentation. Dans le chapitre suivant, nous allons discuter et donner les détails de l'implémentation de notre modèle ainsi que les résultats obtenus tout en les analysant.

Chapitre 5

5. Implémentation et Résultats:

5.1. Introduction

Ce chapitre a pour but de valider expérimentalement, à travers une simulation, le modèle proposé dans le chapitre précédent. Nous disposons d'un système centré sur une caméra représentant l'œil de l'observateur, tel que cette dernière puisse modifier sa position (en se rapprochant ou bien s'éloignant). Pour chaque humain virtuel de la scène, et en fonction de sa position par rapport à la caméra, le système reproduira avec plus ou moins de précision, le comportement approprié à chaque humain virtuel. Le comportement simulé au niveau de notre système, consiste en la construction d'une maison.

5.2. Le langage de programmation

Techniquement, le modèle proposé permettant de simuler le comportement des humains virtuels selon leurs niveaux de détails, ce modèle est réalisé par le biais d'une application implémentée dans le langage de programmation C++.

Nous avons choisi l'environnement de programmation Visuel C++ pour, d'abord la qualité offertes par le C++, telle que la programmation orientée objet et pour l'interface qui pourra nous fournir et surtout car il offre une compatibilité et des capacités d'interfaçage avec le moteur physique « ODE 0.11[1] ». Le moteur ODE étant destinée, dans notre travail, à la réalisation des mouvements physique et du graphisme.

[1] La dernière version du moteur ODE obtenue en 2009.

5.3. Le moteur physique de l'application

L'intégration d'un moteur physique permet à une application de reproduire des aspects physiques réels, on parle alors de simulation.

Le développement d'un « moteur physique » de l'application permet de concentrer tous les appels à ODE en un module. L'utilisation de ce dernier ne requiert donc pas la maîtrise du moteur ODE. Il s'agit, en fait, de réunir dans une classe (ODEInterface) l'ensemble des primitives nécessaires au fonctionnement du moteur ODE.

Ainsi, les objets de l'environnement peuvent être matérialisés par des primitives géométriques existantes sous ODE (GeomBox sous ODE), tous ces objets étant positionnés dans l'environnement de simulation. Il faut, par conséquent, ajouter un plan statique (GeomGround sous ODE) qui constitue le sol de l'environnement. L'espace de collision s'applique sur ces objets, ce qui signifie qu'un corps rigide (la créature où un objet quelconque) ne pourrait en aucun cas les traverser.

Un mouvement physique ne pouvant être continu numériquement, il faut échantillonner une simulation. C'est cette idée qui introduit la notion de pas de simulation : faire un pas de simulation revient à avancer dans le temps la simulation, il faut pour cela définir la taille du pas (généralement 0.05 secondes environ). Plus la taille du pas est petite, plus la simulation sera précise, mais le temps d'exécution sera plus long. Dans notre cas de simulation nous avons choisi un pas de 0,05 secondes.

5.4. L'architecture globale de l'implémentation

Le schéma ci-dessous montre les relations entre les différentes techniques faisant partie du système global implémenté afin d'accomplir le rôle recherché dans ce travail, ce rôle étant la simulation des comportements des humains virtuels, tel que ces comportements possèdent des précisions différentes, selon leurs position par rapport à l'œil de l'observateur, dans un environnement physique réaliste.

Comme nous pouvons le remarquer dans le schéma, notre système est constitué de cinq modules importants :

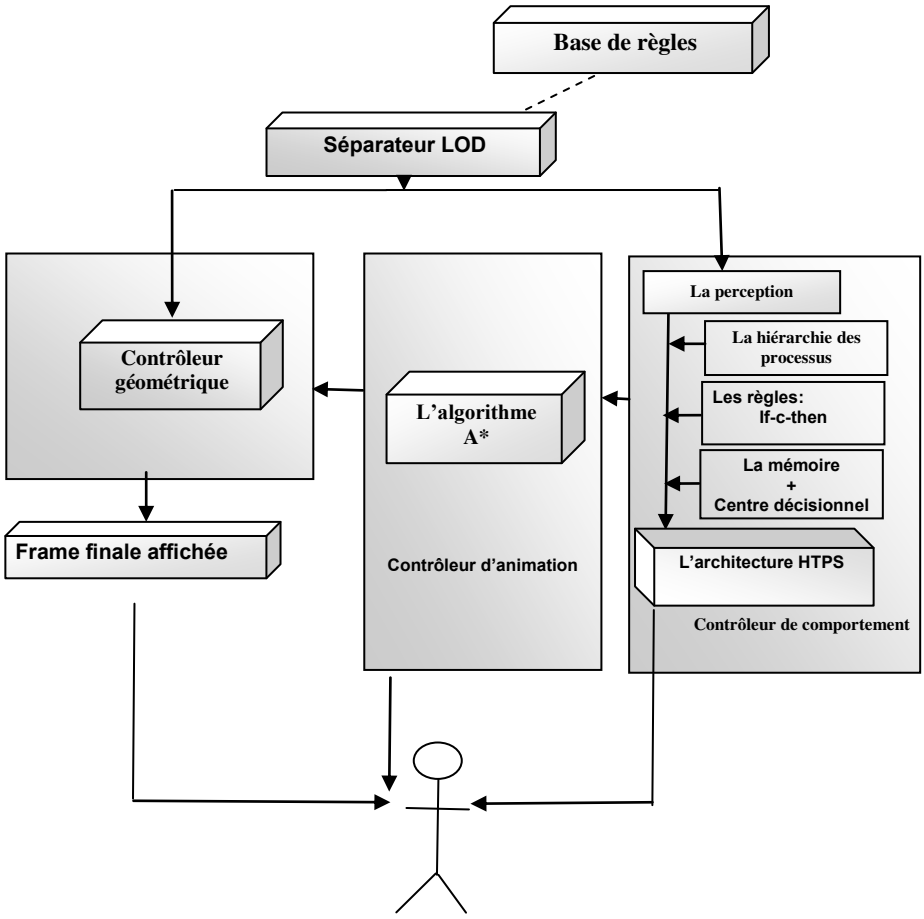


Figure.5.1 : Une vue élevée de l'architecture proposée

5.4.1. La base de règles

L'évaluation de la position d'un humain virtuel par rapport à l'œil de l'observateur peut tenir compte de plusieurs facteurs (voir section 3.2.5). Nous nous sommes limités, dans le cadre de notre travail, à la distance par rapport à l'œil de l'observateur : (Voir Figure 5.2).

- La distance par rapport à l'œil de l'observateur est calculée de manière classique – la distance euclidienne - La distance euclidienne entre le point $A(x_a,y_a,z_a)$ et le point $B(x_b,y_b,z_b)$ est calculée de la manière suivante :

$$AB=BA= \sqrt{(x_b-x_a)^2+(y_b-y_a)^2+(z_b-z_a)^2}$$

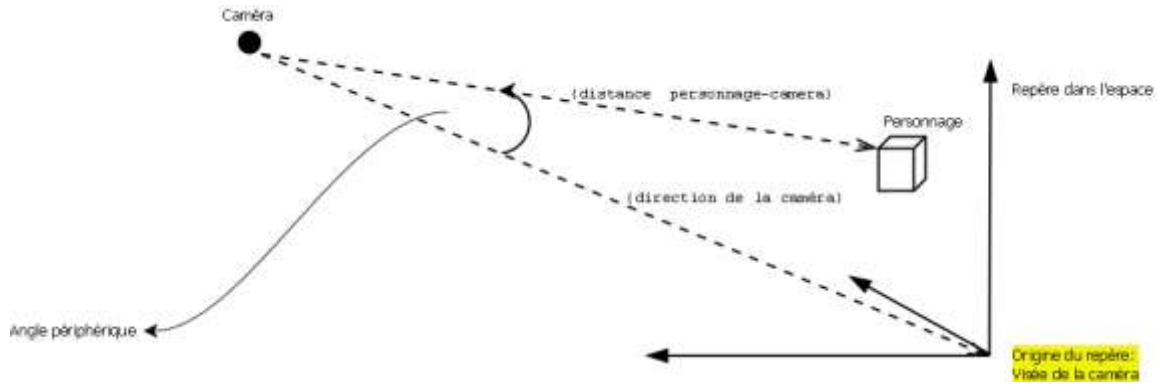


Figure 5.2 : Position par rapport à l'œil de l'observateur

5.4.2. Le séparateur LOD

A partir des données définies dans la base de règle, nous avons déduit trois niveaux de complexité liés à la position de l'œil de l'observateur :

- Niveau 1 : Le comportement de l'humain virtuel est très précis du fait que nous voyons son comportement d'une manière précise (la manière dont il construit la maison est brique par brique). (LOD_value=3)
- Niveau 2 : La visibilité du comportement de l'humain virtuel est moins précise (nous voyons la construction de la maison se faire ligne par ligne). (LOD_value=2)
- Niveau 3 : le comportement de l'humain virtuel est non visible. (LOD_value=1)

Nous définissons par la suite deux variables d_1 et d_2 de la manière suivante (voir Figure5.3) :

- d_1 : Tout humain virtuel situé dans le champ de vision à une distance inférieure ou égale à d_1 possède une position de niveau 1 (LOD_value = 3) ;
- d_2: Tout humain virtuel situé dans le champ de vision à une distance comprise entre d_1 et d_2 possède une position de niveau 2 (LOD_value = 2) ;
- Tout humain virtuel situé dans le champ de vision à une distance supérieur à d_2 possède une position de niveau 3 (LOD_value = 1) ;

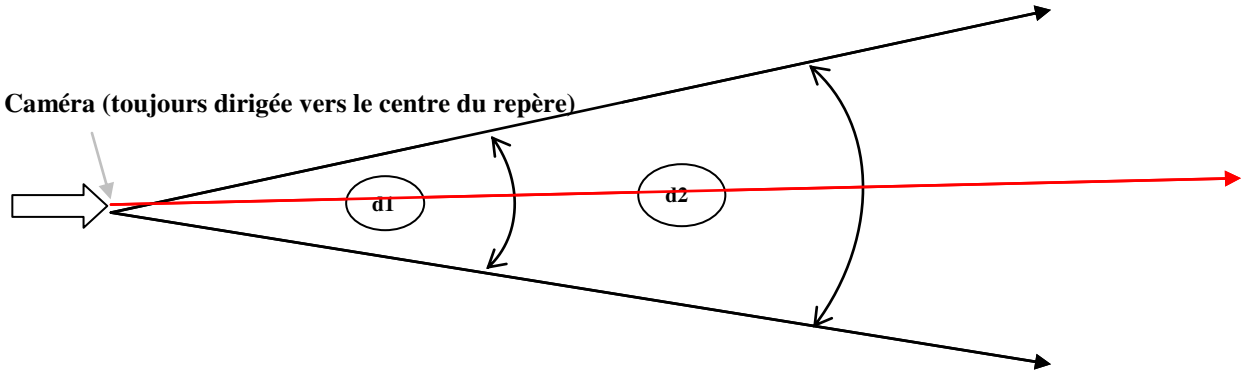


Figure.5.3 : Définition des niveaux de détails

5.4.3. Le contrôleur de comportement

Le comportement recensé est le comportement de construction d'une maison:

- A un niveau très proche de l'œil de l'observateur, la construction de la maison est effectuée brique par brique.
- A un niveau moins proche, la construction de la maison est effectuée ligne[2] par ligne.
- A un niveau très loin, le comportement est invisible

Le comportement de la construction consiste en les actions élémentaires suivantes :

- Déplacer le bras
- Eviter l'obstacle
- Prendre un objet[3]
- Déplacer l'objet
- Eviter l'obstacle
- Mettre l'objet

Nous pouvons présenter ce comportement par l'automate de déplacement d'un objet en utilisant l'architecture HPTS, présenté dans la figure suivante :

[2] La ligne est composée de trois briques et chaque mur se compose de trois lignes
[3] Dans notre simulation, un objet est représenté par une brique

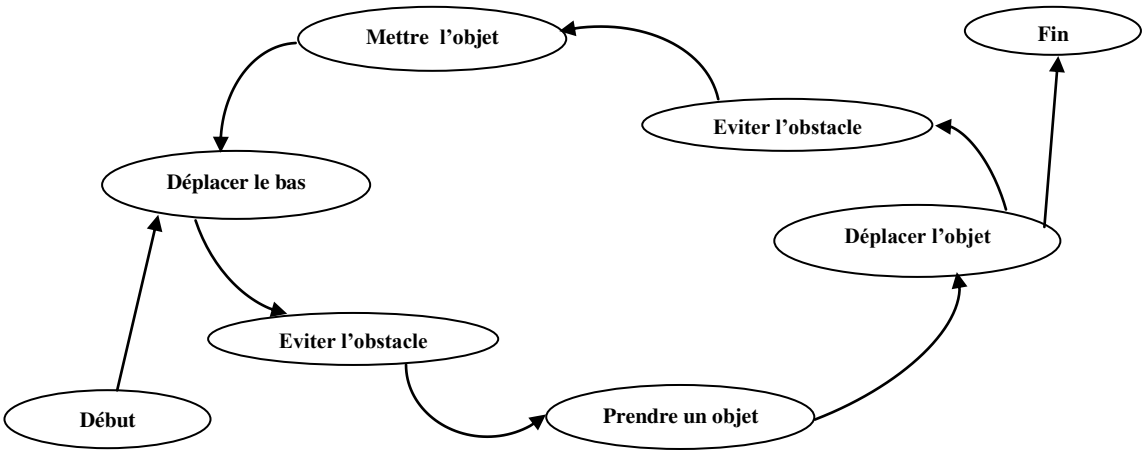


Figure 5.4 : L'architecture HPTS utilisée

5.4.4. Contrôleur de mouvement

D'après l'automate présenté, le mouvement que nous devons simuler est le déplacement d'un bras ainsi qu'un déplacement d'une brique. Le niveau de précision de la simulation de ce mouvement est limité par la position du bras par rapport à l'œil de l'observateur, nous pouvons définir les niveaux suivants :

- Niveau 1 : où le bras est très proche de l'œil de l'observateur. La simulation du mouvement est réalisée en précision.
- Niveau 2 : le mouvement est moins proche de l'œil de l'observateur et dans ce cas le mouvement sera simulé avec moins de précision.
- Niveau 3 : le comportement est invisible, par conséquent le mouvement est également invisible.

Un déplacement continu pour chaque humain virtuel, couterait beaucoup de ressources informatiques. Il faudrait alors effectuer des micro-translations répétitives (de l'ordre d'une translation tous les centièmes de seconde) pour que l'utilisateur ait l'illusion voulue. Ces opérations surchargent le processeur et ralentissent par conséquent la vitesse de la simulation. Notre idée est de réduire la fréquence et d'augmenter le pas de translation au fur et à mesure que la caméra s'éloigne de l'humain virtuel (ou vice versa) tout en maintenant la vitesse de déplacement - pour ne pas changer les données de la simulation. (Voir tableau 5.1)

Niveau	(fréquence, pas)
1	($f_1=f_{max}$, $step_1=step_{min}$)
i (i>1)	(f_i, $step_i$)

Tableau.5.1 : Tableau des paramètres de la fonction de déplacement.

$\forall$ i > 1, $f_i < f_{max}$ et $f_{i-1} < f_i$
$\forall$ i > 1, $step_i > step_{min}$ et $step_{i-1} > step_i$

f_{max} et $step_{min}$ sont les valeurs qui produisent le déplacement le plus crédible à l'œil de l'observateur.

5.4.5. Contrôleur géométrique

Comme nous avons cité précédemment, nous avons déduit 03 niveaux de complexité liés à la position de l'œil de l'observateur:

– Niveau 1 : Le personnage est entièrement visible avec précision.

– Niveau 2 : Le personnage est partiellement visible.

– Niveau 3 : Le personnage est invisible.

Remarque : Nous avons limité l'application de la technique de LOD des données à la représentation de la géométrie des humains virtuels seulement parce que c'est l'élément géométrique existant le plus complexe et qui nécessite beaucoup de ressources informatiques, par contre les autres éléments géométriques (la table et les cubes) ont une représentation unifiée pour toutes les niveaux à cause de leurs simplicité géométrique.

5.4.5.1. Modèle proposé de l'humain virtuel à un niveau très précis

A un niveau très précis, le modèle de la représentation d'un humain virtuel utilisé est vu en tant qu'une représentation d'un modèle d'humain virtuel générique. Il a été créé à partir de primitives de corps et de joints disponibles sur ODE. Le modèle d'humain virtuel se compose de 12 primitives géométriques de corps rigide, et 13 joints : 8 Hinge joint, 2 Ball and socket joint, 2 contact joint et 1 fixed joint. Ces joints reliant les primitives du corps dans une structure articulée de corps-rigide.

Les primitives géométriques utilisées sont : 8 Capped cylinders, 3 parallélépipèdes et une sphère. La structure de l'humain virtuel est définie en utilisant les chaînes multiples, à partir de ses pieds avec chaque lien décrit en termes de liens précédents. Cette composition a comme conséquence un modèle d'humain virtuel possédant 15 degrés de liberté (DOF). La

structure du corps rigide de notre humain virtuel qui est simulé, avec l'organisation des liens entre les différentes primitives, est représentée dans la figure 5.5 et dans le tableau 5.2

Comme motionné dans le tableau 5.2, les types de jointures qui sont utilisées dans notre modèle d'humanoïde sont : hinge joint, ball & socket joint, Fixed joint et contact joint[4].

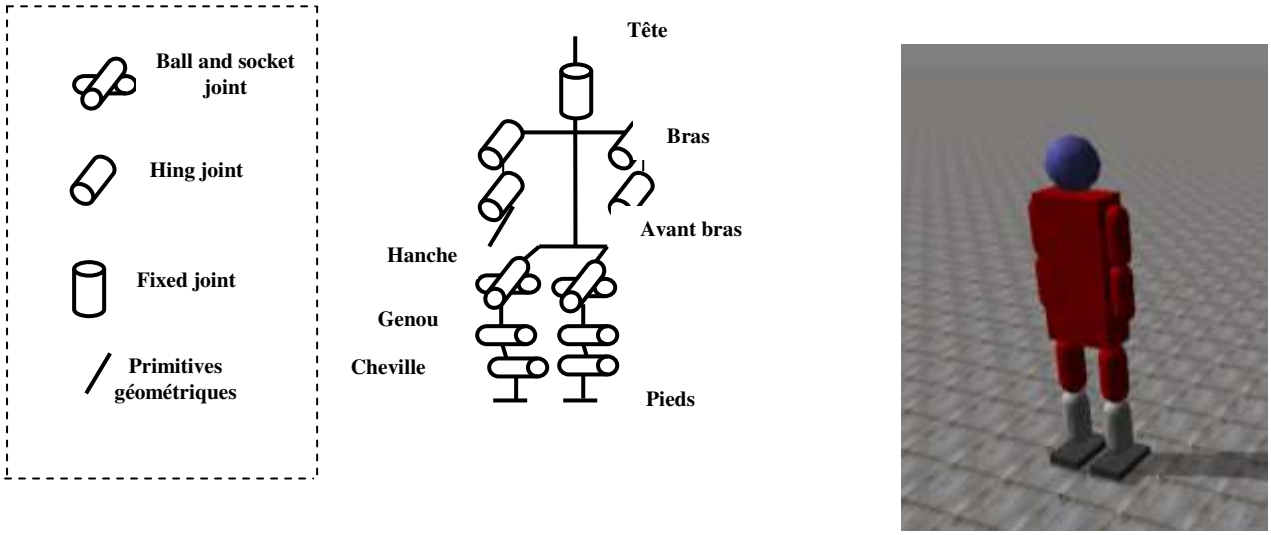


Figure 5.5: Modèle de l'humain virtuel simulé par ODE à un niveau très précis.

Parties de corps	Jointure	Primitive géométrique
Tête	Fixed joint	Sphère
Bras	Hing joint + Contact joint	Capped cylinder
Hanche	Ball and socket joint	Parallélépipède
Genou	Hing joint	Capped cylinder
Cheville	Hing joint	Capped cylinder
Pieds	/	Parallélépipèdes

Tableau 5.2: Paramètres de simulation de l'humain virtuel à un niveau très précis.

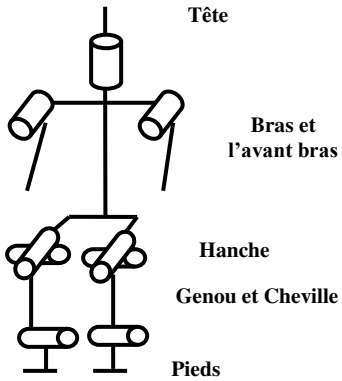
5.4.5.2. Modèle proposé de l'humain virtuel à un niveau moins précis

A un niveau moins précis, le modèle d'humain virtuel se compose de 08 primitives géométriques de corps rigide, et 09 joints : 04 Hinge joint, 2 Ball and socket joint, 2 contact joint et 1 fixed joint. Ces joints reliant les primitives du corps dans une structure articulée de corps-rigide.

[4] *Hinge*, *Ball and socket*, *contact* et *fixed* joints sont les noms des types de jointures existantes dans ODE.

Les primitives géométriques utilisées sont : 04 Capped cylinders, 3 parallélépipèdes et une sphère. La structure de l'humain virtuel est définie en utilisant les chaînes multiples, à partir de ses pieds avec chaque lien décrit en termes de liens précédents. La structure du corps rigide de notre humain virtuel qui est simulé, avec l'organisation des liens entre les différentes primitives, est représentée dans la figure 5.6 et dans le tableau 5.3

Comme motionné dans le tableau 5.3, les types de jointures qui sont utilisées dans notre modèle d'humanoïde sont : hinge joint, ball & socket joint, Fixed joint et contact joint[5].



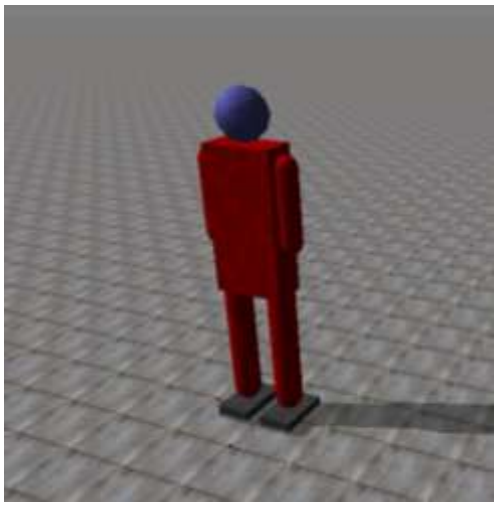


Figure 5.6: Modèle de l'humain virtuel simulé par ODE à un niveau moins précis.

Parties de corps	Jointure	Primitive géométrique
Tête	Fixed joint	Sphère
Bras et l'avant bras	Hing joint + Contact joint	Capped cylinder
Hanche	Ball and socket joint	Parallélépipède
Genou et cheville	Hing joint	Capped cylinder
Pieds	/	Parallélépipède

Tableau 5.3: Paramètres de simulation de l'humain virtuel à un niveau moins précis.

5.5. Algorithmes utilisés

L'application réalisée, dans le cadre de notre projet, est basée sur deux concepts très importants. En premier étant la simulation graphique tridimensionnelle et physique des humains virtuels, de l'environnement et les objets statiques (i.e. utilisation d'un simulateur

[5] *Hinge*, *Ball and socket*, *contact* et *fixed* joints sont les noms des types de jointures existantes dans ODE.

physique qui est ODE) alors que le second aspect est celui de l'implémentation de l'architecture proposée, détaillée ci-dessus, qui se compose de cinq modules importants :

Le séparateur_LOD, le contrôleur de comportement, le contrôleur de mouvement et le contrôleur géométrique et la base de règles.

Tous ces modules sont des fonctions appartenant chacune à une classe spécifique. Ces fonctions sont exploitées par une fonction principale, cette fonction est : simLoop[6] .

5.5.1. Structures de données

La hiérarchie des processus est implémentée sous la forme d'une structure de données composée de trois champs (voir la structure de données1) :

- Le nom de l'action qui peut être : Construire une maison, construire une ligne ou construire une brique.
- L'état de l'action qui peut être atomique ou non atomique.
- Les sous_intentions[7] de l'action par exemple la sous_intention de l'action « construire une maison » est « construire une ligne » et la sous_intention de l'action construire une ligne est construire une brique.

Structure de données 1 : La hiérarchie des processus

```
typedef struct hierarchie_processus hierarchie_processus;
struct hierarchie_processus
{
      const char nom[100];
      const char etat[100];
      const char sous_intentions[100];
};
hierarchie_processus action[3]={
  "construire une maison","non atomique","construire une ligne",
  "construire une ligne","non atomique","construire une brique",
  "construire une brique","atomique",""};
```

Les intentions des actions sont implémentées sous la forme d'une structure de données composée d'un seul champ qui est le nom de l'intention, tel que nous avons initialisés l'ensemble de nos intentions par l'intention « construire une maison » seulement (voir la structure de données 2).

[6] La fonction de la boucle de simulation qui appartient à la classe mère, cette fonction permet de gérer la boucle en prenant en charge les différents modules de notre architecture.

[7] Une intention de l'action est le but ou l'objectif de cette action

Structure de données 2 : Les intentions des actions

```
typedef struct intention intention;
struct intention
{
        char nom[100];
};
intention Intention[3]={"construire une maison","","",};
```

5.5.2. Fonctions

Les valeurs de niveaux de détails des objets[8] (LOD_value) sont définies à l'aide de la fonction 1[9] et le comportement approprié pour chaque humain virtuel selon son niveau de détail est définie à l'aide de la fonction2.

Fonction 1 : Le séparateur LOD

```
// le séparateur LOD
static int definir_LOD_value(dGeomID table_geom)
{
 const dReal *position =dGeomGetPosition(table_geom);
double distance =
distanceeuclidienne(position[0],position[1],position[2],xyz[0],xyz[1],xyz[2]
);
if(distance >=0 && distance<d1)   return 3;
if(distance >=d1 && distance <d2) return 2;
if (distance>=d2) return 1;
}
```

Remarque 1 : d_1 et d_2 sont des valeurs définies par l'utilisateur.

Remarque 2 : la « *distanceeuclidienne* » est une fonction programmée par l'utilisateur sert à calculer la distance euclidienne entre deux points.

[8] Un objet peut être un humain virtuel ou un objet statique comme un cube ou une table
[9] La fonction 1 sert à définir la valeur de niveau de détail pour un objet

Fonction 2 : Le contrôleur de comportement

```
static void controleur_comportement(dGeomID body_geom)
{
      while(strcmp(Intention[j].nom,"") !=0)
      {
       for(i=0;i<3;i++)
        {
         if(strcmp(action[i].nom,Intention[j].nom) == 0) {

           if((strcmp(action[i].etat,"atomique")==0) ||
(definir_LOD_complex(action[i].nom)==definir_LOD_value(body_geom)))
                      {
            if (strcmp(action[i].nom,"construire une maison")==0)
      {Appel à la fonction de construction une maison à un niveau 3;}
             if (strcmp(action[i].nom,"construire une ligne")==0)
      {Appel à la fonction de construction une maison à un niveau 2 ;}


            if (strcmp(action[i].nom,"construire une brique")==0)
      {Appel à la fnction de construction une maison à un niveau 1;}
                    }
                        i=3;
                        j=j+1;
                       strcpy(Intention[j].nom,"");
            }
           else
           {
                 j=j+1;
                 strcpy(Intention[j].nom,action[i].sous_intentions);
            } }
```

Remarque 3 : Les deux fonctions « strcmp » et « strcpy » sont des fonctions prédéfinies dans le langage de programmation C++, tel que :

- La fonction « strcmp » est utilisée pour comparer deux chaines de caractères.
- La fonction « strcpy » est utilisée pour copier une chaine de caractère dans une autre chaine de caractère.

Remarque 4 : Les deux fonctions « definir_LOD_complex » et « definir_LOD_value » sont deux fonctions programmées par le concepteur, tel que :

- La fonction « definir_LOD_value » est utilisée pour définir la valeur de niveau de détail d'un objet.
- La fonction « definir_LOD_complex » est utilisée pour définir la valeur de la complexité d'un processus.

Remarque 5 : Les fonctions présentées ci-dessus ont été utilisées pour la simulation du comportement de trois humains virtuels.

5.5.3. Exemple d'une simulation sous ODE

La troisième fonction présentée dans ce chapitre est celle de la modélisation d'un objet de l'environnement qui est représenté physiquement, cet objet étant une table carrée possédant quatre supports.

Nous avons choisi de représenter cette fonction dans le but d'illustrer le fonctionnement de l'environnement ODE ainsi que la manière dont nous pouvons l'exploiter pour simuler un objet.

Fonction 3 : La construction d'un objet 3D

```
static void creertable
(dBodyID table_bodies[1000],dGeomID table_geom[1000],int TB,int TG,double d)
{
    // le haut de la table
      table_bodies[TB] = dBodyCreate (world);
      dBodySetPosition (table_bodies[TB],0,1.8 - d ,2.5);
      dMassSetBox (&m,1,6,3,0.2);
      dMassAdjust (&m,BMASS);
      dBodySetMass (table_bodies[TB],&m);
      table_geom[TG] = dCreateBox (space,6,3,0.2);
      dGeomSetBody (table_geom[TG],table_bodies[TB]);

      TB +=1;
      TG +=1;

      // pied1 de la table
      table_bodies[TB] = dBodyCreate (world);
      dBodySetPosition (table_bodies[TB],2.2,2.5 - d ,1.25);
      dMassSetBox (&m,1,0.2,0.2,2.5);
      dMassAdjust (&m,BMASS);
      dBodySetMass (table_bodies[TB],&m);
      table_geom[TG] = dCreateBox (space,0.2,0.2,2.5);
      dGeomSetBody (table_geom[TG],table_bodies[TB]);

      TB +=1;
      TG +=1;

      // pied2 de la table
      table_bodies[TB] = dBodyCreate (world);
      dBodySetPosition (table_bodies[TB],2.2,0.7 - d ,1.25);
      dMassSetBox (&m,1,0.2,0.2,2.5);
      dMassAdjust (&m,BMASS);
      dBodySetMass (table_bodies[TB],&m);
      table_geom[TG] = dCreateBox (space,0.2,0.2,2.5);
      dGeomSetBody (table_geom[TG],table_bodies[TB]);
```

```
TB +=1;
TG +=1;

// pied3 de la table
table_bodies[TB] = dBodyCreate (world);
dBodySetPosition (table_bodies[TB],-1.8,0.7 - d ,1.25);
dMassSetBox (&m,1,0.2,0.2,2.5);
dMassAdjust (&m,BMASS);
dBodySetMass (table_bodies[TB],&m);
table_geom[TG] = dCreateBox (space,0.2,0.2,2.5);
dGeomSetBody (table_geom[TG],table_bodies[TB]);

TB +=1;
TG +=1;

// pied4 de la table
table_bodies[TB] = dBodyCreate (world);
dBodySetPosition (table_bodies[TB],-1.8,2.5 - d ,1.25);
dMassSetBox (&m,1,0.2,0.2,2.5);
dMassAdjust (&m,BMASS);
dBodySetMass (table_bodies[TB],&m);
table_geom[TG] = dCreateBox (space,0.2,0.2,2.5);
dGeomSetBody (table_geom[TG],table_bodies[TB]);

TB +=1;
TG +=1;}
```

5.6. Expérimentations

Le but de notre application est d'exploiter la technique de niveaux de détails dans la simulation crédible et en temps réel du comportement des humains virtuels dans un environnement physique réaliste. Pour ce faire, nous avons utilisé la technique de LOD IA.

Il serait intéressant, non seulement d'observer les humains virtuels en pleine action, mais aussi, de mesurer les performances de l'exécution de notre application avec et sans la mise en place de la technique de LOD IA. L'objectif est d'arriver à montrer au moins les trois propriétés suivantes :

- Il est possible d'observer la précision du comportement des humains virtuels selon leurs position par rapport à l'œil de l'observateur (très précis, moins précis et invisible).
- La capacité de notre système à changer la précision du comportement de nos humains virtuels une fois que la caméra change de position : à partir d'un comportement très précis, nous pouvons avoir un comportement moins précis et à partir d'un comportement invisible, nous pouvons avoir un comportement moins précis.

- Tous les comportements, quelque soient leurs niveaux de précision, doivent commencer en même temps et doivent se terminer en même temps, même si un changement au niveau de la position de la caméra est effectué.

Les tests ont été développés sur un poste muni de deux processeurs et équipé d'une carte graphique moyenne.

5.6.1. Mise en œuvre de la simulation

Les expérimentations reposent sur une simulation d'un environnement physique réaliste tridimensionnel qui contient certains objets statiques (des tables simples et des cubes) et des humains virtuels. Notre environnement simulé est centré sur une caméra représentant l'œil de l'observateur, tel que cette dernière puisse modifier sa position (en se rapprochant ou bien en s'éloignant). Pour chaque humain virtuel de la scène, et en fonction de sa position par rapport à la caméra, le système reproduira avec plus ou moins de précision, le comportement approprié à chaque humain virtuel.

Le comportement simulé au niveau de notre système, consiste en la construction d'une maison simple. Comme nous l'avons cité précédemment, nous avons trois niveaux de complexité liés à la position de l'œil de l'observateur :

- Niveau 1 : Le comportement de l'humain virtuel est très précis du fait que nous voyons son comportement d'une manière précise, le comportement de la construction de la maison est effectué brique par brique ;
- Niveau 2 : La visibilité du comportement de l'humain virtuel est moins précise. Le comportement de la construction est effectué ligne par ligne ;
- Niveau 3: le comportement de l'humain virtuel est invisible.

5.6.1.1. L'environnement virtuel

L'environnement virtuel de la simulation est un espace plat simple ne possédant aucune dénivellation. Il s'agit donc d'une surface non bornée sur laquelle nous avons positionné des objets statiques (des cubes superposés sur des tables) et des humains virtuels avec des positions fixes et ceci pour étudier le comportement de construction d'une maison selon plusieurs niveaux de détails. Cet environnement possède toutes les caractéristiques physiques nécessaires pour réaliser une simulation relativement réaliste.

5.6.1.2. Objets statiques et humains virtuels

Du moment que l'environnement simulé possède les caractéristiques physiques nécessaires à la procédure de la simulation, les humains virtuels doivent commencer le comportement de construction approprié, selon leur position par rapport à l'œil de l'observateur.

Les humains virtuels effectuent leurs constructions à l'aide des cubes qui représentent des briques, ces derniers sont superposés sur des tables (voir Figure 5.7).

Les humains virtuels, comme mentionnés précédemment, selon leurs positions par rapport à l'œil de l'observateur possèdent une représentation plus ou moins précise (Voir figure.5.8).

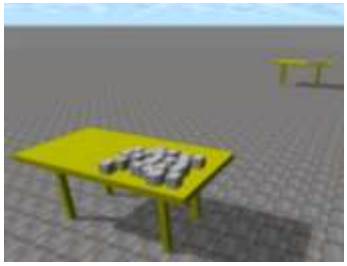


Figure 5.7 :L'environnement simulé avec des tables et des cubes en trois niveaux de détails

Figure 5.8 :L'environnement simulé avec des humains virtuels en trois niveaux de détails

Les figures ci-dessus nous montrent deux humains virtuels seulement, parce que le troisième humain virtuel est invisible .Comme il est illustré dans la figure 5.8, nous avons :

- Un humain virtuel simulé avec une précision parce qu'il est très proche de la caméra, il est situé en premier niveau.
- Le deuxième humain virtuel est simulé avec plus ou moins précision parce qu'il est situé en deuxième niveau de détail.
- Le troisième humain virtuel n'apparait pas parce qu'il est très loin de l'œil de l'observateur, il est situé en troisième niveau de détail.

Au niveau de la figure 5.7, nous avons trois tables mais il n'apparait que deux tables, parce que la troisième table est invisible, elle se situe en troisième niveau de détail.

5.6.2. Conditions initiales de l'application

L'application est réalisée sous certaines conditions qui sont nécessaires au déroulement de la simulation. Ainsi, nous avons définis tous les paramètres ainsi que les valeurs initiales avant le commencement de la simulation.

Nous avons simulé le comportement de construction d'une maison par des humains virtuels, chaque humain virtuel est muni d'un ensemble de cubes superposés sur une table.

Le nombre de cubes est fixé à 24 cubes alors que la maison simulée sera composée de quatre murs, chaque mur étant constitué de trois lignes et chaque ligne composée par trois briques[10] .

Les niveaux de détails ont été définis sur la base des distances d_1 et d_2 tels que :

- Chaque objet[11] se trouve à une distance inférieure ou égale à d_1, est au niveau 1 ;
- Chaque objet se trouve à une distance comprise entre d_1 et d_2, est au niveau 2 ;
- Chaque objet se trouve à une distance supérieure à d_2, est au niveau 3.

Les distances d_1 et d_2 ont été fixées respectivement à 30 et 60.

Le pas de déplacement des bras des humains virtuels a été fixé, selon le cas:

- Pour un niveau très précis à 1/30 pixels.
- Pour un niveau moins précis 1/10 pixels.

Par ailleurs, la caméra, peut être rapprochée ou éloignée selon le besoin.

Le comportement de construction d'une maison est simulé avec plus au moins précision, selon le niveau de détail de chaque humain virtuel :

- Si l'humain virtuel se trouve au niveau 1 : le comportement est effectué d'une manière très précise (brique par brique) ;
- Si l'humain virtuel se trouve au niveau 2 : le comportement est effectué d'une manière moins précise (ligne par ligne) ;
- Si l'humain virtuel se trouve au niveau 3 : le comportement sera invisible.

5.7. Résultats obtenus

Pour ce qui est des expérimentations, nous nous sommes attelés à évaluer notre système par rapport aux buts que nous nous sommes assignés précédemment.

Au niveau de notre système, le nombre d'humains virtuels est fixé à 3, 6 puis 10 humains. Nous avons ainsi procédé au test de l'application pour chacun des cas précédents.

5.7.1. Cas où le nombre d'humains virtuels est fixé à 03

A chaque niveau de détail, nous trouvons un humain virtuel équipé avec des cubes superposés sur une table (Voir figure.5.9 -5.10 -5.11 -5.12) :

[10] Une brique est représentée par un cube
[11] Un objet peut être un humain virtuel, un cube ou une table

- Le niveau1 : le premier humain virtuel se trouve à la position (x=0, y=0). Son comportement est simulé avec précision, il construit la maison brique par brique
- Le niveau 2 : le deuxième humain virtuel situé à la position (x=0, y=30). Son comportement est simulé avec moins de détail, il construit la maison ligne par ligne.
- Le niveau 3 : le troisième humain virtuel se trouve à la position (x=0, y=60). Son comportement est invisible.

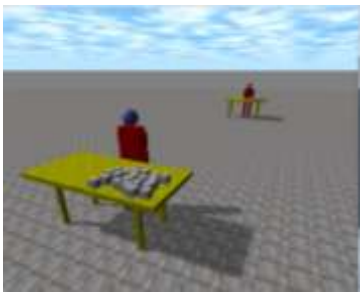


Figure 5.9. Le début de la construction

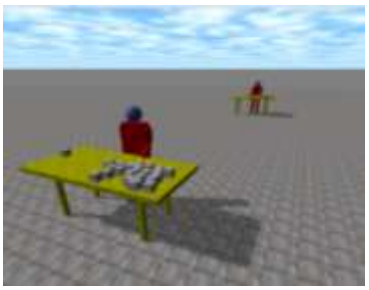


Figure 5.10 : Les humains virtuels ayant commencé la construction

Figure 5.11 : Les humains virtuels ayant construit le premier mur.

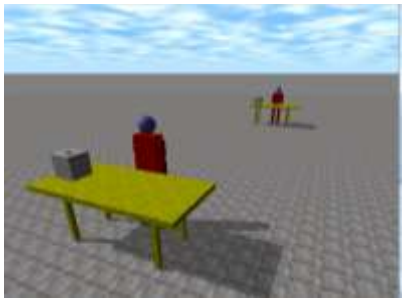
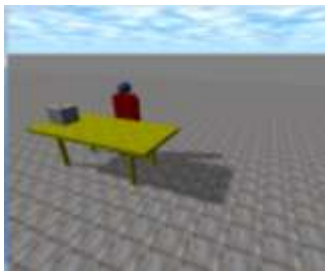


Figure 5.12 : Les humains virtuels ayant terminé la construction

Selon les figures ci-dessus, nous pouvons constater que le troisième humain virtuel n'apparait pas, parce qu'il appartient au troisième (le dernier) niveau de détail par contre le comportement des deux premiers humains virtuels apparaît avec plus ou moins de précision selon leurs niveau de LOD.

Un changement[12] au niveau de la position de la caméra, nous a permis de voir le comportement du troisième humain virtuel avec précision[13] parce qu'il est devenu très proche de la caméra (Voir figure.5.13).

[12] Nous avons effectué un rapprochement de la position de la caméra, en appuyant sur la touche A
[13] Il est situé en premier niveau de détail



Figue 5.13 : Changement de la position de la caméra est effectué, tel que l'humain virtuel, qui était invisible est devenu plus précis.

5.5.1. Cas où le nombre d'humains virtuels est fixé à 06

Dans ce cas, le nombre d'humains virtuels est fixé à 6 (Voir figure.5.14 -5.15 -5.16) :

- Pour le niveau 1, nous avons un humain virtuel situé à la position (x=0, y=5).
- Pour le niveau 2, nous avons quatre humains virtuels situés respectivement aux positions (x=2, y=30), (x=-35, y=35), (x=-20, y=40) et (x=-8, y=45).
- Pour le niveau 3, nous avons un humain virtuel situé à la position (x=-8, y=60).

Figure 5.14 : Humains virtuels ayant commencé la construction

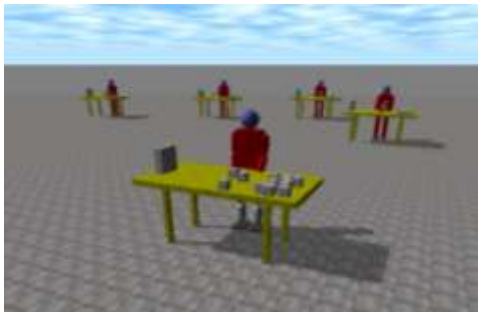


Figure 5.15 : Humains virtuels ayant construit le premier mur.

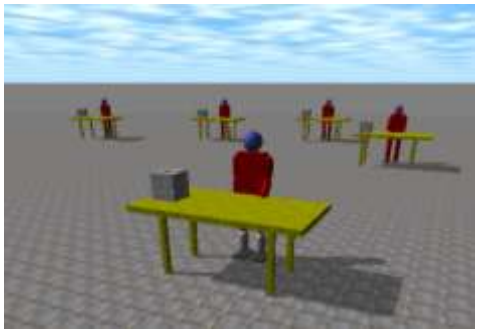
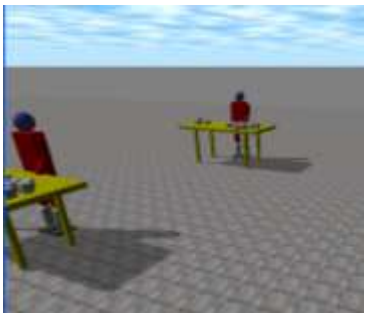


Figure 5.16 : Humains virtuels ayant terminé la construction.

Selon les figures ci-dessus, nous pouvons constater que le dernier (le sixième) humain virtuel n'apparait pas, parce qu'il appartient au troisième (le dernier) niveau de détail par contre le comportement des cinq premiers humains virtuels apparaît avec plus ou moins de précisions selon leurs niveau de détails.



Figue 5.17 : Changement de la position de la caméra est effectué, tel que l'humain virtuel, qui a été invisible est devenu très précis

Un rapprochement au niveau de la position de la caméra, nous a permis de voir le comportement du sixième humain virtuel avec précision (Voir figure.5.17).

5.5.2. Cas où le nombre d'humains virtuels est fixé à 10

Dans ce cas, le nombre d'humains virtuels est fixé à 10 (Voir figure.5.18 -5.19) :

- Dans le niveau 1, nous avons deux humains virtuels situés respectivement aux positions (x=0, y=5) et (x=-10, y=15).
- Dans le niveau 2, nous avons cinq humains virtuels situés respectivement aux positions (x=2, y=30), (x=-35, y=35), (x=-20, y=40), (x=-8, y=45) et (x=-30, y=50).
- Dans le niveau 3, nous trouverons trois humains virtuels situés respectivement aux positions (x=-8, y=60), (x=-20, y=75) et (x=-30, y=90).

Figure 5.18 : Les humains virtuels ayant commencé la construction

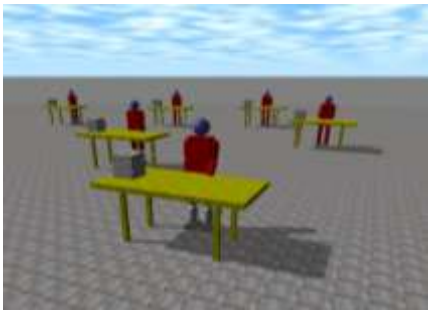
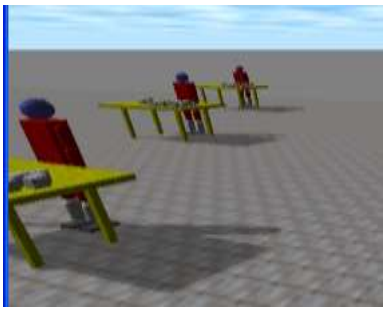


Figure 5.19 : Les humains virtuels ayant terminé la construction

Le comportement des trois derniers humains virtuels est invisible parce qu'ils se situent en troisième niveau de détail, par contre le comportement des autres humains est simulé avec plus ou moins de précisions selon leurs de LOD.



Figue 5.20 : Changement de la position de la caméra est effectué, tel que les trois humains virtuels, qui ont été invisibles sont devenus très précis.

Un rapprochement au niveau de la position de la caméra, nous a permis de voir le comportement des trois derniers humains virtuels[14] (Voir figure.5.20).

5.6. Elaboration du test

Nous avons effectué un test comparatif. La technique de LOD IA n'a pas été mise en place, et l'application a été exécutée avec les paramètres suivants :

- Déplacement : fréquence maximale et pas minimal.
- Comportement : précis (la construction est effectuée brique par brique).
- Représentation graphique pour les humains virtuels (Le modèle d'humain virtuel se compose de 12 primitives géométriques de corps rigide, et 13 joints).

Niveau	Déplacement (fréquence, pas)
1	(1/56s, 0.1/3)
2	(1/56s, 0.1/3)
3	(1/56s, 0.1/3)

Tableau.5.4 : Tableau des Paramètres de déplacement, s=seconde et l'unité de mesure est le pixel

Après avoir collecté les couples de données (nombre de polygones, temps d'exécution), (voir Tableau.5.5 -5.6) associées à chaque série de test (3, 6, et 10 Humains virtuels), pour chaque série, on dessine les histogrammes correspondants(voir Figure 5.21 -5.22) et on déduit un certain nombre de conclusions.

5.6.1. Cas sans application de la technique de LOD IA

Le tableau 5.5 résume l'ensemble des résultats obtenus pour 3, 6 et 10 humains virtuels sans l'application de la technique de LOD IA.

[14] Les humains virtuels n'ont pas commencé encore la construction.

Nombre de personnage	Nombre de polygones	Temps de calcul
03	123	19.35s
06	246	38.70s
10	410	67s

Tableau. 5.5 : Tableau récapitulatif (la technique de LOD IA a été supprimée)

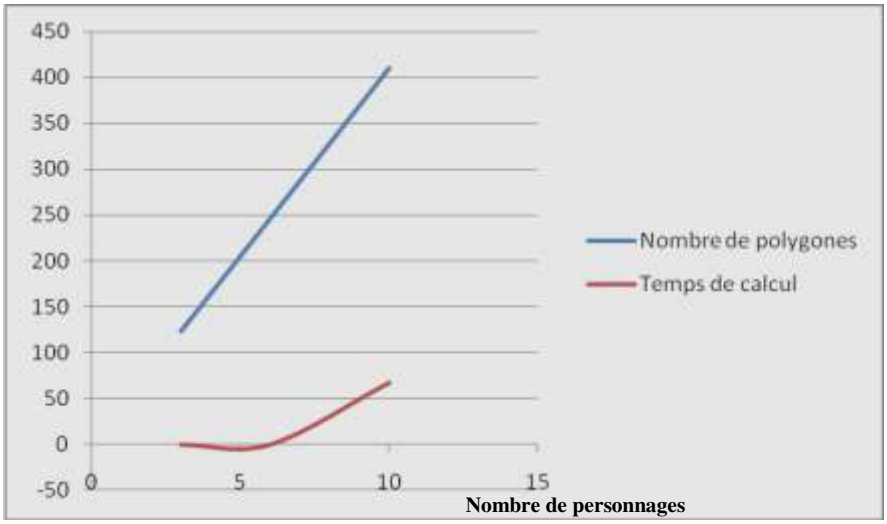


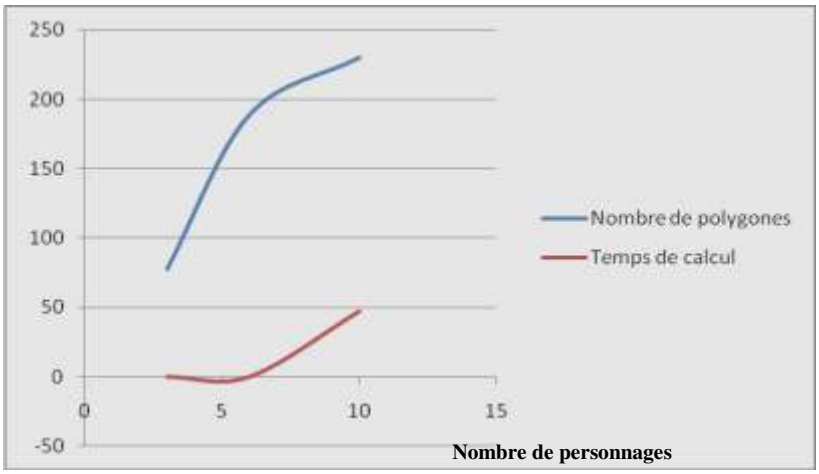
Figure 5.21 : Graphe du temps de calcul et du nombre de polygones (sans l'application de la technique de LOD IA)

5.6.2. Cas avec application de la technique de LOD IA

Le tableau 5.6 résume l'ensemble des résultats obtenus pour 3, 6 et 10 humains virtuels avec l'application de la technique de LOD IA.

Nombre de personnage	Nombre de polygones				Temps de calcul
	Niveau 1	Niveau 2	Niveau 3	Nombre total	
03	41	37	00	78	12.35s
06	41	148	00	189	24.70s
10	82	148	00	230	47s

Tableau. 5.6 : Tableau récapitulatif (la technique de LOD IA a été utilisée)



Figue 5.22 : Graphe du temps de calcul et du nombre de polygones (avec l'application de la technique de LOD IA)

5.6.3. Evaluation des résultats

Sans l'application de la technique de LOD IA (voir Figure 5.21):

- Pour le nombre de personnages 3, les performances est 123 pour le nombre de polygones et 19.35s pour le temps d'exécution.
- Pour le nombre de personnages 6, les performances est 246 pour le nombre de polygones et 38.70s pour le temps d'exécution.
- Pour le nombre de personnages 10, les performances est 410 pour le nombre de polygones et 67s pour le temps d'exécution.

Avec l'application de la technique de LOD IA (voir Figure 5.22) :

- Pour le nombre de personnages 3, les performances est 78 pour le nombre de polygones et 12.35s pour le temps d'exécution.
- Pour le nombre de personnages 6, les performances est 189 pour le nombre de polygones et 24.70s pour le temps d'exécution.
- Pour le nombre de personnages 10, les performances est 230 pour le nombre de polygones et 47s pour le temps d'exécution.

D'après les résultats, nous constatons que les performances dans le cas de l'application de la technique de LOD IA sont toujours meilleurs que les performances sans l'utilisation de la technique de LOD IA quelque soit le nombre de personnage :

- Pour trois personnes : nous avons une différence de 45 polygones et une différence de 7s en temps d'exécution.
- Pour six personnes : nous avons une différence de 57 polygones et une différence de 14s en temps d'exécution.

- Pour dix personnes : nous avons une différence de 180 polygones et une différence de 20s en temps d'exécution.

Conclusion

La mise en place de la technique de LOD IA, nous a permis de réaliser une simulation crédible ,d'un grand monde virtuel ,dans les limites des ressources informatiques disponibles.

5.7. Délimitation des niveaux : choix des distances

Le but de ce test est de déterminer la bonne séparation des niveaux, en d'autres termes, trouver les bonnes valeurs pour les variables d_1 et d_2. La meilleure façon de procéder, est de tester plusieurs valeurs, et de retenir celles qui satisfont le mieux l'utilisateur (exemple de la figure 5.23). Nous pouvons prévoir des modifications en temps réel pour donner plus de flexibilité à l'utilisateur.

Le niveau 1 : Le choix de la distance d_1 est déterminant, car si elle trop grande, on risque d'avoir beaucoup trop de personnages en niveau 1. L'application serait surchargée de calcul, et ralentirait. Si elle est trop petite, trop peu de personnages seront visibles et cela risquerait de compromettre l'observation.

Les autres niveaux : Les autres distances doivent être choisies de telles sorte qu'en modifiant la position de la caméra, le changement de la forme des humains virtuels ne soit pas trop voyant et aussi assurer la fluidité du basculement entre les niveaux de la simulation progressive du comportement[15]. De plus, comme nous l'avons signalé dans le paragraphe précédent, il faut avoir à l'esprit, la répartition de la complexité des calculs. Il est préférable, par analogie à l'œil humain, de les choisir de telle sorte qu'il y ait un écart croissant entre elles : $d_2 - d_1 < d_3 - d_2 < d_4 - d_3$ ….. avec $d_1 < d_2 < d_3 < d_4$.

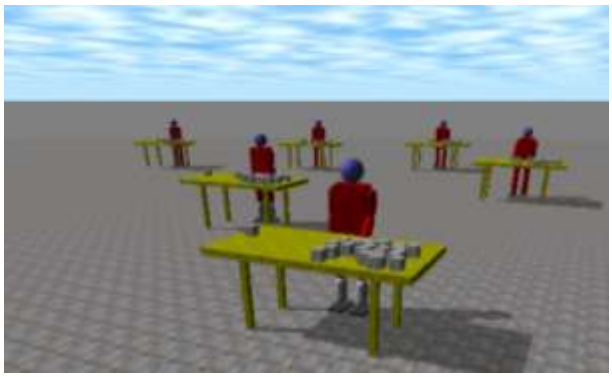


Figure 5.23 – Délimitation des niveaux : d1 = 30 et d2 = 60

[15] La fluidité du basculement entre le comportement de construction de la maison brique par brique et la construction de la maison ligne par ligne.

5.8. Comparaison avec les travaux antérieurs

Notre travail se focalise sur la simulation de grands mondes artificiels, en incorporant la technique de niveau de détail (habituellement utilisée pour réduire la complexité des données d'une scène), au niveau comportemental « LOD IA ».

Cette technique : « LOD IA » consiste à réduire les demandes de la simulation en ressources informatiques en diminuant graduellement la qualité de la simulation des comportements des humains virtuels qui se trouvent dans des places moins importantes, tout en gardant une simulation crédible. Notre travail proposé repose sur une architecture inspirée du modèle ALOHA qui offre un banc d'essai graphiquement sophistiqué pour de nouvelles technologies d'agents simulés.

La technique de LOD a été employée dans les jeux d'ordinateur, par la non-simulation des régions du monde qui ne sont pas visibles par l'utilisateur. Cependant, ceci rend souvent l'environnement incorrect et peut générer des contradictions.

L'utilisation de la technique des LOD IA consiste à effectuer un transfert du domaine des infographies des ordinateurs vers le domaine de l'intelligence artificielle [Kkp+06]. Cette dernière est basée sur une simple idée qui consiste en ce qui suit:

Il existe souvent peu de lieux, dans le monde artificiel, à un moment donné de la simulation, où le comportement des humains virtuels est important. Dans ces localités, le comportement doit être simulé avec précision, les localisations les moins importantes n'ont pas besoin de simuler le comportement des humains virtuels avec précision, leur simulation sera dégradée.

Si le comportement des humains virtuels est simulé partiellement, les exigences de la simulation en matière de ressources informatiques peuvent être réduites significativement.

Le tableau 5.7 montre une vue d'ensemble des travaux précédents utilisant la technique LOD IA, caractérisée par les critères suivants : le critère de détermination de LOD, le nombre de niveaux de détails, l'usage de LOD et les comportements simulés par l'IA.

Auteurs	Le critère de détermination de LOD	Le nombre de LOD	L'usage de LOD	Les comportements simulés par IA
Chenney et al. [Caf01]	Le potentiel de la visibilité	02	Mettre à jour le mouvement	La navigation, l'évitement de collision
O'Sullivan et al [Scv+02]	La distance	Non spécifié	La géométrie, les animations, l'évitement de collision, les gestes, les expressions faciales et la sélection d'action	La navigation, l'évitement de collision, le dialogue complexe entre les agents
Brockington [Bro02]	Distance	05	La recherche, la navigation, la sélection d'action dans un combat	La navigation, l'évitement de collision, les interactions de combat complexes
Niederberger and Gross [NG05]	La distance et la visibilité	21	La planification, l'évitement de collision, la recherche du chemin et les décisions de groupe	La navigation, l'évitement de collision et la planification du chemin
Pettré et al [Pcm+06]	La distance et la visibilité	05	La géométrie, la mise à jour de mouvement, l'évitement de collision, la navigation	La navigation, l'évitement de collision, la recherche de chemin
Brom et al. [Bsp07]	La distance simplifiée	04	La sélection d'action (avec l'arbre AND-OR), la simplification environnementale	La navigation, les interactions complexes avec les objets et les autres agents
Paris et al. [Pgo09]	La distance	03	La navigation et l'évitement de collision	La recherche de chemin, la navigation et l'évitement de collision
Lin et Pan [LP07]	distance	Non spécifié	La géométrie et les animations	La locomotion
Osborne and Dickinson [OD10]	La distance	Non spécifiée	La navigation, flockage, les décisions de groupe	La navigation
M.Wibner et al [Wka10]	La distance et la visibilité	08	La mise à jour du mouvement, l'évitement de collision, la navigation et l'exécution de l'action	La navigation, l'évitement de collision, les interactions basées sur les désires avec les agents et les objets intelligents, les dialogues
Notre approche S.Zertal et al. [Zds 11]	La distance	03	La géométrie, les animations et la mise à jour du comportement selon le niveau de LOD	L'évitement de collision, la recherche du chemin et le comportement de construction d'une maison.

Tableau 5.7 : Comparaison des différentes approches de LOD IA

Ainsi, de cela, nous pouvons dire que la nouveauté apportée par notre travail consiste à pouvoir effectuer une simulation crédible du comportement des humains virtuels, en réduisant les demandes de la simulation en ressources informatiques.

Donc les points forts de nos résultats sont :

- La crédibilité dans le processus de simulation du comportement des humains virtuels.
- La simulation rapide, en temps réel et dans les limites des ressources informatiques disponibles.
- la simulation d'un grand monde artificiel possédant un nombre important d'humains virtuels exécutant un comportement précis.
- L'extensibilité de notre système, nous pouvons ajouter d'autres humains virtuels exécutant des comportements complexes comme le comportement de préparation du diner par exemple

Les points que nous pourrons améliorer à propos des autres travaux ou comme nouvelle perspectives sont :

- L'application de la technique de LOD au niveau de la géométrie, ce qui permettrai de gagner davantage dans l'optimisation, en utilisant à titre d'exemple le maillage triangulaire.
- L'utilisation des techniques d'animation comme la dynamique, la cinématique et la cinématique inverse pour simuler les mouvements et les animations des humains virtuels avec plus ou moins de précision. Par exemple, pour simuler un mouvement très précis, nous pourrons exploiter la dynamique et pour un mouvement moins précis nous pourrons nous contenter de la cinématique.
- La prise en compte de la complexité et la diversité du comportement au niveau de notre application.
- L'utilisation d'autres critères de détermination des LODs comme la périphérie à titre d'exemple.
- Le contrôle du comportement de nos humains virtuels peut être amélioré en intégrant un algorithme d'apprentissage.
- La simulation des comportements non-hiérarchiques qui ne peuvent pas être divisés en d'autres comportements.

5.9. Conclusion

Ce chapitre a été consacré à la description de quelques détails de l'implémentation de notre architecture proposée dans le chapitre précédent qui consiste à simuler le comportement des humains virtuels selon plusieurs niveaux de détails en appliquant la technique de « LOD IA ». Nous avons essayé de spécifier les différentes étapes d'implémentation tout en décrivant la structure de données et les algorithmes utilisés. La présentation des résultats et l'élaboration des tests nous a permis de faire une évaluation globale sur les performances du modèle proposé.

Bibliographie

[Ami] a)http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.htmI Introduction au pathfinding
b)http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html Heuristiques
c)http://theory.stanford.edu/~amitp/GameProgramming/ImplementationNotes.html Implémentation

[AP94] Peter Astheimer et Maria-luise Pöche : *Level of detail Generation Its Applications in virtual reality*. In Proc. Of VRST'94, pages 299-312, 1994.

[Ark98] Arkin (R.C.): *Behaviour-based robotics* - MIT Press, Cambridge, MA. 1998.

[Bdi+02] Blumberg (B.), Downie (M.), Ivanov (Y.), Berlin (M.), Johnson (M.P.) and Tomlinson (W.): *Integrated Learning for Interactive Synthetic Characters* - Proceedings of the 29th Conference on Computer Graphics and Interactive Techniques (SIGGRAPH-02), ACM Transactions on Graphics, Vol. 21, 3, pp.417-426, ACM Press, July 21-25 2002.

[Bel98] Salim Belblida : *Modélisation et visualisation par Niveaux de Détails de scènes architecturales complexes*. Thèse de doctorat, Institut National Polytechnique de Lorraine, 1998.

[Ben04] Benameur Sabrina : *Implémentation D'une méthode D'intégration De La Multirésolution Dans Un Système Masse-Ressort : Application A L'Animation De Tissu* .Mémoire de Magister .Option : Intelligence Artificielle et Image 2004.

[Blo+06] Cyril Brom1, Jiří Lukavský2, Ondřej Šerý1, Tomáš Poch1, Pavel Šafrata1: *Affordances and level-of-detail AI for virtual humans.* This work is partially

supported by the Program “Information Society”; the project 1ET100300517, This work was presented at Game Set and Match 2 Conference, Delft, The Netherlands.

[Bmt90] Boulic (R.), Magnenat Thalmann (N.) et Thalmann (D.): *A global human walking model with real-time kinematic personification* - Visual Computer, vol. 6(6), pp. 344-358, 1990.

[Bmt95] Boulic (R.), Mas (R.) et Thalmann (D.): *Position control of the center of mass for articulated figures in multiple support,* - Dans: Computer Animation and Simulation'95, éd. Par Springer-Verlag (New-York), pp. 130-144. - Maastricht, Netherlands, 1995.

[BM96] Boulic (R.) et Mas (R.): *Hierarchical kinematic behaviors for complex articulated figures* - Dans: Advanced Interactive Animation, éd. Par Thalmann (Daniel) et Magnenat-Thalmann (Nadia), chap. 3. Prentice-Hall Europe, 1996

[Bpw93] Badler (N.), Phillips (C.) et Webber (B.): Simulating Humans. - Computer Graphics Animation and Control. Oxford University Press, New York,NY,1993

[Bsp07] Brom, C., ˇSery, O., Poch, T.: *Simulation Level of Detail for Virtual Humans*. In: Pelachaud, C., Martin, J.-C., Andr´e, E., Chollet, G., Karpouzis, K., Pel´e, D. (eds.) IVA 2007. LNCS (LNAI), vol. 4722, pp. 1–14. Springer, Heidelberg (2007)

[BT98] Becheiraz (P.) et Thalmann, (D.) : *A Behavioral Animation System for Autonomous Actors personified by Emotions* – Dans: Proceedings of the First Workshop on Embodied Conversational Characters (WECC '98), Lake Tahoe, California. 1998.

[Bro86] Brooks (R.A.): *A robust layered control system for a mobile robot* – IEEE Journal of Robotics and Automation, pp 14-23, 1986.

[Bro02] Brockington, M.: *Level-Of-Detail AI for a Large Role-Playing Game*. In: AI Game Programming Wisdom, pp. 419–425. Charles River Media (2002)

[Cab97] Brian Cabral. OpenGl Optimizer 1.0: The power of silicon Graphics’ Next-Generation Visualization Technology, Developer News 1997 http://student.cosy.sbg.ac.at/~maus/lod/sourcen/lokaleSeiten/opengloptimize-MJ97.html

[Caf01] Chenney, S., Arikan, O., Forsyth, D.A.: *Proxy Simulations For Efficient Dynamics.* In: Proceedings of Eurographics (2001)

[Ccm02] Cavazza (M.), Charles (F.) et Mead (S. J.): *Planning Characters' Behaviour in interactive storytelling*. Dans the Journal of the Visualization and Computer Animation. Vol. 13, pp. 121-131, 2002.

[Cdl+95] Chamberlain, B., DeRose, T., Lichinski, D., Salesin, D., et Snyder, J. (1995): *Fast rendering of complex environments using a spatial hierarchy*. Technical Report UW-CSE-95-05-02, University of Washington

[Cha03] A. J. Champandard: *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviours*, New Riders Publishing, USA, 2003.

[CH97] Deborah Carlson and Jessica K. Hodgins: *Simulation levels of detail for real-time animation*. In Procee-dings of Graphic Interface, pages 1–8, 1997.

[Con01] Zoran Constantinescu: *levels of detail: Overviews*. First NTNU CSGSC, Norwegian University of Science and Technology, Mai 2001. http://csgsc.idi.ntnu.no/2001/pages/papers/zoran.pdf

[DB97] Dubreuil (N.) et Bechmann (D.): Facial *animation.- Dans: Computer Animation'97*, pp. 98-109. Genève, Suisse, 1997.

[Deb00] Gilles Debunne: *Animation multirésolution d'objets déformables en temps réel application à la simulation chirurgicale*. Thèse de doctorat. Institut Nationale Polytechnique de Grenoble. Décembre 2000.

[Don94] Donikian (S.) : *Les modèles comportementaux pour la génération du mouvement d'objets dans une scène*. - Revue Internationale de CFAO et d'Infographie, pp. 847-871, vol. 9, n 6, 1994

[Don01] Donikian (S.) - *HPTS: a behaviour modelling language for autonomous agents* – Dans: Proceedings of the Fifth International Conference on Autonomous agents, pp. 401-408, ACM Press, May 2001.

[Ebd+97] Essa (I.), Bassu (S.), Darrell (T.) et Pentland (A.):*Modeling, tracking and interactive animation of faces and heads using input from video*.- Dans: Computer Animation'97, pp. 68-79. Genève, Suisse, 1997.

[Fgp+98] Fua (P.), Gruen (A.), Plaenkers (R.), D'apuzzo (N.) et Thalmann (Daniel): *Human body modeling and motion analysis from video sequences* - Dans: International Symposium on Real Time Imaging and Dynamic Analysis. – Hakodate, Japon, 1998.

[FM97] Fua (P.) et Miccio (C.): *Fitting sophisticated facial animation models to image data* – Dans: Optical 3-D .Measurement Techniques Conference. –Zurich, Suisse, 1997.

[Fts+93] Funkhouser, T. A et Séquin, C. H. (1993) : *Adaptive display algorithm for interactive frame rates duriong visualization of complex virtual environments*. In Kajiya, J. T, editor, SIGGRAPH 93, volume 27, pages 247-254.

[Gcb95] John P.Gralieri, Jonathan Crabtree et Norman I. Badler. *Production and playback of human figure motion for visual simulation.* ACM Transactions on modeling and computer simulation. Juliet 1995.

[Gmp+00] Giang T, Mooney R, Peters C, O'Sullivan C: *ALOHA: Adaptive Level Of Detail for Human Animation. Towards a new Framework*, Eurographics 2000, Short Papers Programme.

[HA92] Gérard Hégron et Bruno Arnaldi : *Computer Animation: Motion and deformable control*. Cambridge, Grande-Bretagne, Eurographics Technical Series, Septembre 1992.

[HA98] Hègron (G.) et Arnaldi (B.) : *Computer Animation: Motion and Deformation Control.* - Cambridge, Grande-Bretagne, - Eurographics'92 Tutorial Notes, Eurographics Technical Series, Septembre 1998.

[Hfv00] Helbing (D.), Farkas (I.) et Vicsek (T.): *Simulating Dynamical Features of Escape Panic* - Dans: Nature, vol. 407, pp. 487-490, 2000.

[Hod96] Hodgins (J.) : *Three dimensional human running* - Dans: IEEE Conference on Robotics and Automation ICRA'96. – 1996

[Kbg+97] Mike KRUS, Patrick Bourdot, Françoise Guisnel et Guillaume THIBAULT : *Levels of Detail and Polygonal Simplification*. ACM Crossroads, (3.4), 1997. http://camis.kaist.ac.kr/~sskim/Research/LOD.htm

[Kkp+06] Kubr, Kulhánek, Poch, Šafrata, Šerý, Šulc *: Intelligent Virtual Environment* Software project, MFF UK. http://mff.modry.cz/ive/public/download.php

[Kle03] Jon Klein. Breve: *A 3d environment for the simulation of decentralized systems and artificial life*. InICAL2003: Proceedings of the eighth international conference on Artificial life, pages 329–334, Cambridge, MA, USA, 2003. MIT Press.

[Kru97] Mike Krus: *Maillage Polygonaux et Niveaux de Détails*. Etude bibliographique. Otes et Documents LIMSI N°: 97 – 10, mai 1997. http://mkrus.free.fr/Papers/LODStr.ps.gz

[Kru99] Michael KRUS. *Connexion et facettisation: Gestion Adaptative de Scènes Virtuelles Application à la Navigation dans des Installations Industrielles*. Thèse de Doctorat, Université de Paris XI, juin 1999. http://www.limsi.fr/Individu/krus/Papers/memoire.pdf.gz

[KV03] Klinger (E.) et Viaud-Delmon (I.) : *Le traité de la réalité virtuelle* - 2ème édition, Chap. Réalité virtuelle et pshychatrie, pp. 297-324, - Ecole des mines de Paris – Les Presses, Vol 2. , 2003.

[LP07] Lin, Z., Pan, Z.: *LoD-Based Locomotion Engine for Game Characters*. In: Hui, K.-c., Pan, Z., Chung, R.C.-k., Wang, C.C.L., Jin, X., G¨obel, S., Li, E.C.-L. (eds.) EDUTAINMENT 2007. LNCS, vol. 4469, pp. 214–224. Springer, Heidelberg (2007)

[Lrc+ 03] D. Luebke, M. Reddy, J.D. Cohen,A. Varshney, B. Watson and R. Huebner (eds), Morgan Kaufmann: *Level of Detail for 3D Graphics*, 2003.

[Ltg+00] Lester (J.), Towns (S. G.), Callaway (C.B.), Voeman (J.L.) et FitzGerald (P.J.) : *Embodies conversational agents* - Chap. Deictic and Emotive Communication in Animated Pedagogical Agents, pp. 123-154, - The MIT Press, 2000.

[Ltw95] Lee (Y.), Terzopoulos (D.) et Waters (K.). : *Realistic modeling for facial animation*- Dans: SIGGRAPH'95, éd par Cook (Robert), pp. 55-62. – Los Angeles,USA, 1995.

[Luc97] Michel Lucas : *Synthèse d'image.* Techniques de l'ingénieur E5530. 1997

[Lue 96] Luebke, D. (1996) : *Hierarchiacl structures of dynamic polygonal simplification*. Technical Report TR 96-2006, Department of Computer Science, University of North Carolina, Chapel Hill,North Carolina.

[Mae89] Maes (P.): *How to do the right thing* - Dans: Connection Science Journal, Vol.1 (3) pp. 291-323, 1989.

[Mat99] Mataric (J.): *Behavior Based Robotic*- Dans: the MIT Encyclopedia of Cognitive Sciences, Robert A. Wilson and Frank C. Keil, eds., MIT Press, pp. 74-77, Avril 1999.

[Mdc+02] Brian MacNamee, Simon Dobbyn, Padraig Cunningham, Carol O'Sullivan .*Men Behaving Appropriately: Integrating the Role Passing Technique into the ALOHA System*. Proceedings of the AISB'02,2002.(remplacer 15 par 2)

[Mor98] Moreau (G.). : *Modélisation du comportement pour la simulation interactive: application au trafic routier multimodal*.- Rennes, Thèse de doctorat, Université de Rennes I, Novembre 1998.

[MP43] Warren S. McCulloch & Walter Pitts: *A logical calculus of ideas immanent in nervous activity*. Bulletin of mathematical biophysics 5:115– 133. Reprinted in McCulloch, W. S., Embodiments of mind. Cambridge, MA: MIT Press. (1943).

[Mul96] Multon (F.) : *Animation d'humanoïdes synthétiques par un modèle biomécanique* - Dans: 4ème Journées de l'Association Française Graphique. – Dijon, novembre 1996.

[NG05] Niederberger, C., Gross, M.: *Level-of-detail for cognitive real-time characters*. The Visual Computer: Int. Journal of Computer Graphics 21(3), 188–202 (2005).

[NS76] Newell (A.) et Simon (H.A.) : *Computer Science as Empirical Enquiry* – Dans: Communications of the ACM, Vol. 19, pp. 113-126, 1976.

[NT97] Noser (H.) et Thalmann (D.): *Sensor based synthetic actors in a tennis game simulation*. - Dans: Computer Graphics International'97. pp. 189-198. - Hasselt, Belgium, Juin 1997.

[OD10] Osborne, D., Dickinson, P.: *Improving Games AI Performance using Grouped Hierarchical Level of Detail*. In: Proc. of the 36th Annual Convention of the Society for the Study of Artificial Intelligence and Simulation of Behaviour (2010)

[Ode06] Russell Smith: *Open Dynamics Engine* V0.5 user guide, 2006.

[OT90] Ortony (A.) et Turner (T.J.): *What's basic about basic emotions?* - Psychological Review, 97(3). pp. 315-331, 1990.

[Oua09] N.Ouannes : *La phylogenèse pour la création de créatures artificielles*. Thèse de magister en informatique, option : synthèse d'image et vie artificielle, 2009

[Par98] Rick Parent: *Computer animation : Algorithmes and techniques,* publier par Morgan Kaufmann. 1998

[PB88] Phillips (C.) et Badler (N. I.). - Jack: *a toolkit for manipulating articulated figures*- Dans: ACM /SIGGRAPH Symposium on user interface software,- Banff, Canada, October 1988

[Pcm+06] Pettré, J., de Heras Ciechomski, P., Maim, J., Yersin, B., Laumond, J.P., Thalmann, D.: *Real-time navigating crowds: scalable simulation and rendering*. Comput. Animat. Virtual Worlds 17(3-4), 445–455 (2006)

[Pgo09] Paris, S., Gerdelan, A., O'Sullivan, C.: *CA-LOD: Collision Avoidance Level of Detail for Scalable, Controllable Crowds*. In: Egges, A. (ed.) MIG 2009. LNCS,vol. 5884, pp. 13–28. Springer, Heidelberg (2009)

[Phm93] Prusinkiewics (P.), Hammel (M. S.) et Mijolsness (E.): *Animation of plant development.* Dans: SIGGRAPH'93, éd. Par Kajiya (James T.). pp. 351-360, - Anaheim, USA, 1993.

[Red94] Martin Reddy : *Reducing Lags in Virtual Reality Systems using Motion Sensitive Levels of Detail*. Proceedings of the 2nd UK VR-SIG Conference.

[Red97] Martin Reddy : *Perceptually Modulated Level of Detail for Virtual Environments.* PHD Theses, University of Edinburgh, 1997
http://www.icsa.informatics.ed.ac.uk/reports/csg-theses/CST-134-97.ps.gz

[Rei97] Reich (B.D.): *An architecture for behavioural locomotion* - Thèse de Doctorat, Departement of Computer and Information Science, University o Pennsylvania, 1997.

[Rey87] Reynolds (C.W.): *Flocks, herds and schools: a distributed behavioural model* – Dans: SIGGRAPH'87, Vol. 21(4) of Computer Graphics, pp 25-34, ACM Press, Anaheim (USA), 1987.

[Rey99] Reynolds (C. W.): *Steering Behaviors For Autonomous Characters*. – GDC 99. pp. 763-782 A. Yu (Ed.) Miller Freeman, San Fransisco, 1999.

[Rmt90] Renault (O.), Magnenat-Thalmann (N.) et Thalmann (D.): *A vision based approach to behavioural animation.* - Journal of Visualization and Computer Animation, pp. 18-21vol. 1, n 1, 1990

[Scv+02] C. O'Sullivan, J. Cassell, H. Vilhjálmsson, J. Dingliana., S. Dobbyn, B. McNamee, C. Peters and T. Giang: *Levels of Detail for Crowds and Groups* . submitted to COMPUTER GRAPHICS Forum (9/2002). Volume xx (200y), Number z, pp. 1–8.

[Sdd99] Sanza (C.), Destruel (C). et Duthen (Y.) : *Autonomous Actors in an Interactive Real-Time Environment* – Dans: ICVC'99, International Conference on Visual Computing, Goa, Inde, Février 1999.

[Smi02] Russell Smith: *How to make new joints in ODE.* Copyright © 2002 February 24, 2002

[Sps+06] Ondřej Šerý, Tomáš Poch, Pavel Šafrata, and Cyril Brom.: *Level-Of-Detail in Behavior of Virtual Humans,* Accepted for publication in proceedings of Current Trends in Theory and Practice of Computer Science, SOFSEM 2006, LNCS 3831, 565-574

[Tha98] Daniel Thalmann : *Infographie*. Ecole polytechnique fédérale de lausanne. Mars 1998.

[Tra07] Trung Hau TRAN : *Approches évolutionnaires pour le comportement adaptatif d'entités autonomes*, Thèse de doctorat 2007.

[Trg+03] Traun (D.), Rickel (J.), Gratch (J.) et Marsella (S.): *Negotiation over tasks in hybrid human-agent teams for simulation-based training* - Dans: Proceedings of the Second International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS'03). pp. 441-456. – Melbourne, Australia, Juillet 2003.

[TT94] Tu (X.) et Terzopoulos (D.): *Artificial Fishes: physics, locomotion, perception, behaviour.* – Dans: SIGGRAPH 94 Conference Proceedings, pp 43-50, Orlando, FL, USA, 1994.

[Val99] Bernard Valton: *Gestion de la complexité de scènes animées et interactives : contributions à la conception et à la représentation*. PhD thesis, Université de Rennes 1, 1999. ftp://ftp.irisa.fr/techreports/theses/1999/valton.ps.gz

[Wka10] Michael Wißner, Felix Kistler, and Elisabeth André: *Level of Detail AI for Virtual Characters in Games and Simulation.* R. Boulic, Y. Chrysantou, and T. Komura (Eds.): MIG 2010, LNCS 6459, pp. 206–217, 2010

[WM00] Wright, I., and Marschall, J.: More AI in Less Processor Time: Egocentric, AI. In:Gamasutra on line 2000. http://www.gamasutra.com/features/20000619/wright_01.htm

[Woo02] Wooldridge, M.: *An Introduction to MultiAgent Systems*. John Wiley & Sons (2002).

[Zds11] S.Zertal, N.Djedi, C.Sanza, S. Sanchez, Y. Duthen, « Exploitation des niveaux de détails dans la simulation des comportements d'humains virtuels », 1st International Conference on Information Systems and Technologies « ICIST 2011 », Tebessa .Algérie

[Zel90] Zeltzer (D.): *Making Them Move: mechanics, control and animation of articulated figures.* - Chap. Task-level graphical simulation: Abstraction, representation, and control, pp. 3-33. 1990.