UNIVERSITE SAAD DAHLEB DE BLIDA

FACULTE DES SCIENCES

Département d'informatique

Mémoire présenté par :
TOUBAL SEGHIER Amina
TOUBAL SEGHIER Ismail

En vue d'obtenir le Diplôme de Master

Domaine : Mathématique et Informatique

Filière : Mathématique et Informatique

Spécialité : Informatique

Option : Ingénierie de logiciel

Sujet :

Création d'un Framework pour la simulation d'un trafic de véhicules en 3D

Soutenue le : 20 septembre 2012 devant le jury composé de :

Président : Bennoar Djamal
Promoteur : Mr SIDOUMOU Mohamed Rédha
Examinateur : Cherif Zahar
Examinateur : Bachia

2011-2012

بسم الله الر حمن الرحيم

و صّلى الله و سلما على نبيّنا محمّد و على آله وأصحابه أجمعين.

REMERCIEMENTS

Au préalable, nous tenons à remercier Allah le Clément le Miséricordieux de nous avoir ouvert les portes du savoir, de nous avoir aidé spirituellement dans les moments difficiles, et de nous avoir permis d'être ce que nous sommes devenus.

Nous tenons à remercier vivement notre promoteur Mr SIDOUMOU qui nous a accordé son entière confiance, sa disponibilité, sa sagesse et sa patience pour nous avoir guider quant à la conception et la réalisation de ce mémoire, ainsi que ses conseils et ses orientations prodigués tout au long de cette année.

Nous adressons aussi nos chaleureux remerciements aux membres du jury.

Nous remercions également les enseignants ayant assurés avec abnégation nos années d'études dans cette filière.

Enfin, nos remerciements s'adressent pour nos familles pour leurs encouragements, conseils et leurs soutiens inconditionnels tout au long des différentes étapes de nos études.

Je tiens à remercier énormément :

Mes très chers parents pour les sacrifices et les encouragements qu'ils n'ont cessé de me conférer dans la joie ou dans la douleur. Qu'Allah puisse les garder éternellement. Merci du fond du cœur.

Mes sœurs : Mahdia, Hasna et Nachida, mes frères : Abdelhadi et Mohammed el hachemi, ma belle sœur : Chahrazed, mes beaux frères : Sofiane et Khire Eddine, mes neveux Cheims Eddine, Mustapha, Salah Eddine, Mohammed, Abdelkhalak dit « Missoum », Rayane, Khaled et Adam, ma nièce : Kawthar Sanaa ainsi que toute la famille.

Mon frère et collaborateur ISMAIL qui m'a aidé pour la réalisation de ce mémoire et projet.

Toutes mes amies et tout ceux qui m'ont aidé, pour leur soutien moral et leur transmets ma reconnaissance éternelle.

Mlle TOUBAL SEGHIER Amina

Je tiens à dédier ce travail à :

Mes parents qu'Allah les protège,

Et à tous mes amis de la promotion.

Mr. TOUBAL SEGHIER Ismail

"If you can see your path laid out in front of you
step by step, you know it's not your path. Your
own path you make with every step you take.
That's why it's your path."

- Joseph Campbell -

Résumé

Notre comportement humain est l'objet de l'étude par des chercheurs scientifiques pour cela ils ont développé des modèles basés sur la personnalité, faire analyser ces modèles permet d'évaluer, d'améliorer et d'étendre ces modèles pour définir des scénarios plus complexes.

Notre travail consiste à simuler graphiquement un trafic de véhicules, en traduisant des données numériques en des données visuelles, et représenter graphiquement l'évolution du monde 3D, en créant un Framework qui sera réutilisable et qui permet de se connecter à une intelligence artificielle.

Mots-Clés : Framework, comportement humain, intelligence artificielle, véhicule, simulation, trafic, détection de collision.

ملخص

سلوكنا الإنساني هو موضوع دراسة من قبل الباحثين لذا قاموا بتطوير عدة نماذج قائمة على أساس شخصية، القيام بتحليل هذه النماذج يمكن من تقييم وتحسين السلوك البشري لتجنب تكرار سيناريو سلبي.

مهمتنا تتمثل في محاكاة هذه النماذج بيانيا، وخلق هيكل إطار من شأنه أن يكون قابلا لإعادة الاستخدام، ويمكن من ربط الذكاء الاصطناعي اليه.

الكلمات الرئيسية :

الإطار، والسلوك البشري، والذكاء الاصطناعي، و المحاكاة، وحركة المرور.
والمحاكاة السيارة، وحركة المرور، وكشف التصادم

Abstract

Our human behavior is the subject of study by scientists for this, they have developed models based on personality, to analyze these models to assess, improve and extend these models to define more complex scenarios.

Our job is to graphically simulate vehicular traffic, translating numerical data into visual data, and graphically represent the evolution of the 3D world, creating a Framework that will be reusable and can connect to an artificial intelligence.

Keywords: Framework, Human behavior, artificial intelligence, car, simulation, traffic, collision detection.

Sommaire

Liste des figures

Liste des Tableaux

Liste des acronymes

2D	:	deux dimensions
3D	:	trois dimensions
OpenGL	:	Open Graphic library
Direct3D	:	bibliothèque de Microsoft
GL	:	OpenGL library
GLU	:	OpenGL utility library
GLUT	:	OpenGL Utility Toolkit
GLEW	:	OpenGL Extension Wrangler Library
IA	:	intelligence artificielle
API	:	Application programming interface
Assimp	:	Open asset import library
WebGL	:	Web graphic library

Introduction générale

Contexte général

Aujourd'hui, l'informatique pénètre tous les domaines de notre vie, elle fait partie intégrante dans toutes nos activités, toutes les études et les recherches sont marquées de son sceau. Elle joue un rôle très important en évolution des études et les recherches au rythme des progrès de l'informatique et des besoins du milieu.

La simulation est devenue incontournable et indispensable dans les domaines d'activités professionnelles où la réalité est dangereuse, coûteuse, difficile à gérer et inaccessible. L'un des domaines où la simulation est indispensable est le domaine des recherches sur le comportement humain.

La simulation du comportement humain est très importante afin que nous comprenions et avec un souci de l'améliorer. Pour cela les scientifiques ont développé un ensemble de modèle basé sur la personnalité, les émotions et un certain nombre de paramètres. Ces algorithmes et modèles doivent être simulés graphiquement afin de les évaluer et faire sortir une analyse par des personnes qui ne font pas partie du domaine tel que les industriels.

La simulation du trafic de véhicules est une des plus complexes projets, elle est étudiée et utilisée pour plusieurs buts comme celui de contrôler et d'observer la circulation et donner des solutions pour résoudre quelque problème par exemple : accident, collision, etc.

Notre sujet : ***Création d'un Framework pour la simulation d'un trafic des véhicules en 3D***

Objectif

Le but de notre projet est de créer un Framework qui permet de simuler graphiquement un trafic de véhicules dans une ville en 3D. Ce framework doit pouvoir se connecter à une intelligence artificielle externe afin d'évaluer cette dernière, donc ce framework doit être très adaptable, réutilisable, portable et offrir une vue à l'intelligence artificielle connectée au monde 3D.

Nous devons proposer une architecture d'un Framework qui lui assure le bon fonctionnement.

Organisation du mémoire

Ce mémoire est organisé en 5 chapitres, qui sont :

➢ Le premier chapitre présente la géométrie et les notions mathématiques qui sont des notions de manipulation de géométrie en 2D ou 3D, où nous citons le système de coordonnées sphériques, la rotation 2D et 3D, la translation qui permet le glissement d'un objet, le changement de repère, la distance entre 2 points. Par la suite nous présenterons la bibliothèque de développement 3D OpenGL.

➢ Le deuxième chapitre nous donnerons un aperçu pour connaitre à quoi servirai l'OpenGL, son historique et les différentes fonctions de bases avec lesquelles nous avons construit notre framework.

➢ Dans le troisième chapitre nous allons présenter l'intelligence artificielle qui sera intégré aux véhicules, en citant le problème de cheminement dans un graphe, en présentant l'algorithme de résolution du problème de plus court chemin et une technique permet la résolution du problème de collision.

➢ En passant à la modélisation dans le quatrième chapitre, nous proposons une architecture à notre Framework, et nous présentons le langage de modélisation utilisé, afin de passer à la réalisation du système.

➢ Dans le cinquième chapitre, en passant à l'étude technique du Framework, nous citerons les différents outils utilisés pour la réalisation de notre système, ensuite nous présenterons les interfaces qui nous montrent la simulation du trafic de véhicules.

➢ Enfin la conclusion générale qui conclura notre mémoire, où nous citons nos perspectives.

Chapitre 1
Géométrie et Notions Mathématiques

Introduction :

La géométrie est la partie mathématique qui étudie les figures dans le plan et dans l'espace. Elle a été développée dans la Grèce antique où Euclide y définit quelques bases importantes (la base fondamentale de cette science).

Dans la géométrie, on peut aussi faire des transformations sur des figures planes ou des figures dans l'espace. On peut faire une symétrie centrale, axiale, une translation [1].

La géométrie euclidienne commence avec les Éléments d'Euclide, qui sont à la fois une somme des connaissances géométriques de l'époque et une tentative de formalisation mathématique de ces connaissances. Les notions de droite, de plan, de longueur, d'aire y sont exposées et forment le support des cours de géométrie élémentaire. La conception de la géométrie est intimement liée à la vision de l'espace physique ambiant au sens classique du terme.

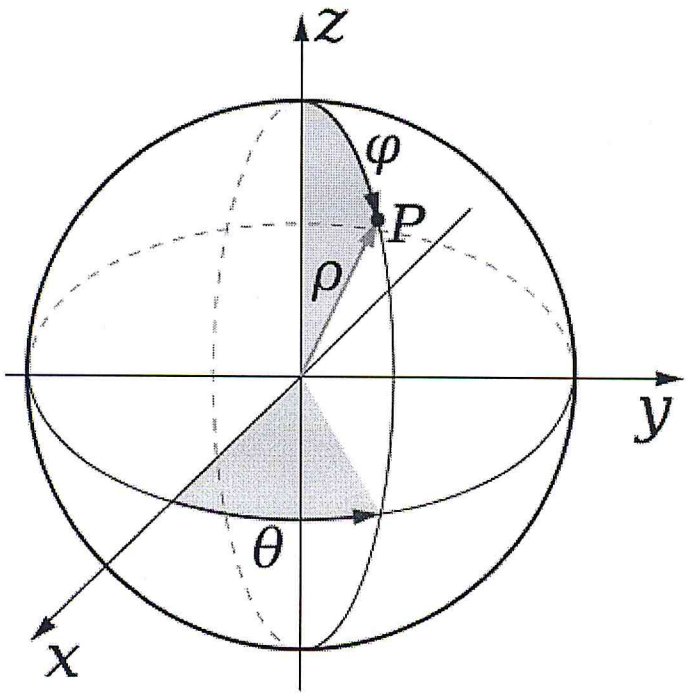
L'espace géométrique euclidien est un outil toujours efficace aux vastes domaines d'applications. Par exemple, l'espace des physiciens, l'astronomie, mécanique du solide etc. [2]

[Figure 1] Un objet solide possède 6 degrés de liberté

1.1 Les coordonnées sphériques

Les **coordonnées sphériques** est un système de coordonnées dans l'espace qui généralisent les coordonnées polaires du plan (2D). Un point de l'espace P est repéré par la distance à un pôle et deux angles. Ce système est d'emploi courant pour le repérage géographique : l'altitude, la latitude, et la longitude sont une variante de ces coordonnées.

Plusieurs systèmes de coordonnées sphériques sont également employés en astrométrie.



[Figure 1.1-1] En coordonnées sphériques, la position du point P est définie par la distance ρ et par les angles θ (longitude) et φ (colatitude).

Étant donné un repère cartésien (O, x, y, z), les coordonnées sphériques (ρ, θ, φ) d'un point P sont définies par (Figure 1.2-1):

- ρ est la distance (rayon) du point P au centre O et donc $\rho > 0$
- φ est l'angle non orienté formé par les vecteurs z et $\boldsymbol{OP}$, appelé angle zénithal ou colatitude
- θ est l'angle orienté formé par les demi-plans ayant pour frontière l'axe vertical et contenant respectivement la demi-droite $[O, x)$ et le point P. Si H est le projeté orthogonal de P dans le plan horizontal (O, x, y), alors θ peut être défini comme l'angle formé par les vecteurs x et $\boldsymbol{OH}$

Par convention, et pour assurer l'unicité de ρ, l'angle φ est compris entre 0 et π radians (0 et 180°) et θ entre 0 et 2π radians (0 et 360°). En conséquence la relation de passage aux coordonnées cartésiennes s'écrit :

$$\begin{cases} x = \rho \cos\theta \sin\varphi \\ y = \rho \sin\theta \sin\varphi \\ z = \rho \cos\varphi \end{cases}$$

[Figure 1.1-2] Les équations de coordonnées sphériques

1.2 Matrice de Rotation

Une **matrice de rotation** Q est une matrice orthogonale de déterminant 1, qui vérifier les équations suivantes :

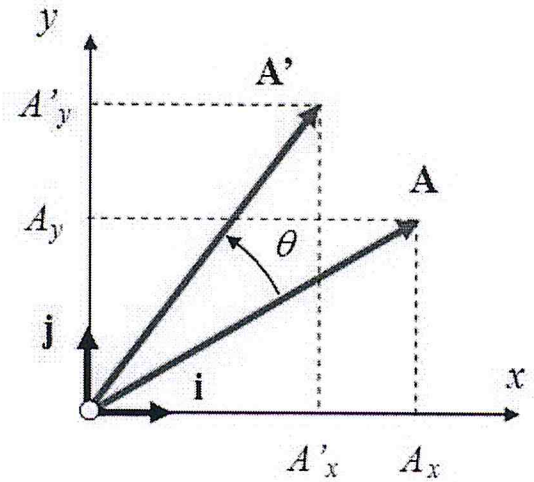
$Q^tQ = I = QQ^t$ et $\det(Q) = 1$
tel que Q^t est la transposée de la matrice Q, et I est la matrice unité.

Ces matrices rend compte du changement de base des coordonnées d'un point ou un repère lié à un solide de R vers R' [3, 4, 5].

En 3D, ces matrices sont utilisées intensivement pour les calculs de géométrie, de physique et en infographie.

$$\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

[Figure 1.2-1] Expression de la matrice de rotation d'angle θ dans un plan



[Figure 1.2-2] Schéma représente la rotation du point A vers A' d'angle θ

$$\begin{pmatrix} A'_x \\ A'_y \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} A_x \\ A_y \end{pmatrix}$$

[Figure 1.2-3] Calcul de la rotation d'angle θ du vecteur $\overrightarrow{OA}$ à l'aide de la matrice de rotation

Rotation en 2D

En 2D, les matrices de rotation ont la forme suivante :

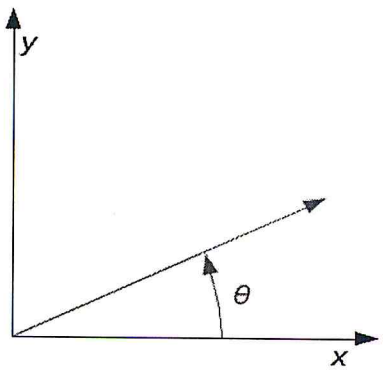
$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

[Figure 1.2-4] Matrice de rotation d'angle θ en 2D

Cette matrice fait tourner le plan d'un angle θ. Elle fait tourner l'axe x en direction de l'axe y.

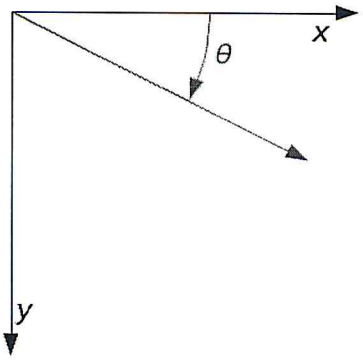
Orientation d'un plan

Il existe deux conventions majeures sur l'orientation du plan, la plus habituelle est (x vers la droite et y vers le haut), cette rotation se fait dans le sens anti horaire (Figure 1.3-5)



[Figure 1.2-5] Orientation du plan, x vers la droite et y vers le haut

La deuxième convention est l'inverse du première (x vers la droite et y vers le bas), cette rotation se fait dans le sens horaire (Figure 1.3-6)



[Figure 1.2-6] Orientation du plan, x vers la droite et y vers le bas

Pour se convaincre qu'il s'agit bien de la même rotation, on n'a qu'à imaginer le plan comme une feuille de papier que l'on regarderait alternativement par au-dessus et par en dessous, par transparence.

Dans le domaine de mathématiques et de physique, la première convention est la plus utilisable. Par contre, en imagerie numérique, la convention opposée est la plus fréquente, qui présente l'avantage d'être compatible avec le sens d'écriture des scripts occidentaux de gauche à droite et de haut en bas. Dans la plupart des logiciels, les rotations se font dans le sens horaire.

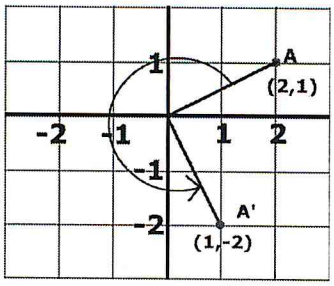
Supposons que l'on adopte l'orientation usuelle du plan. Pour obtenir une rotation dans le sens horaire, on remplace simplement θ par $-\theta$:

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

[Figure 1.2-7] Rotation antihoraire d'angle θ

$$R(-\theta) = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}$$

[Figure 1.2-8] Rotation horaire d'angle θ



[Figure 1.2-9] Rotation de 270° de A vers A'

Rotation en 3D

Dans un espace euclidien à 3-dimensions, les matrices de rotations suivantes correspondent à des rotations autour des axes x, y et z (respectivement) :

$$R_{x(\theta)} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{pmatrix}$$

$$R_{y(\theta)} = \begin{pmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{pmatrix}$$

$$R_{z(\theta)} = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

[Figure 1.2-10] Les rotations opèrent ainsi : R_x tourne l'axe y vers l'axe z, R_y tourne l'axe z vers l'axe x et R_z tourne l'axe x vers l'axe y

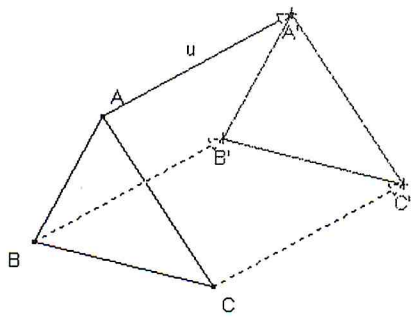
1.3 Translation

Une **translation** ou un déplacement est une transformation géométrique qui correspond au glissement d'un objet, sans rotation ni déformation de cet objet. Une translation de vecteur $\vec{u}$ est une transformation qui, à tout point *M*, associe le point *M'* tel que :

$$\overrightarrow{MM'} = \vec{u}$$

M' est une image de Ma par cette translation.

Une translation de vecteur $\overrightarrow{AB}$ rend le point M en un point M' tel que ABM'M forme un parallélogramme (Figure 1.4-1).



[Figure 1.3-1] Déplacement d'un triangle par le vecteur de translation $\vec{u}$

Une translation déplace tous les points d'un objet géométrique d'une même distance selon une direction dans le même sens (suivant un vecteur) (Figure 1.4-1).

En géométrie plane et en géométrie d'espace, une translation se traduit par un déplacement de toute la figure sans changement de la direction, ni du sens, ni des longueurs.

Construire l'image d'une figure par une translation revient à la faire glisser dans une direction, un sens et avec une longueur donnée.

Un tel glissement conserve l'objet de la déformation, le changement de disposition ; donc, les longueurs, la perpendicularité et les angles sont conservés [6, 7].

La construction d'une image d'une figure géométrique nécessite la construction de ses points caractéristiques :

- Pour un segment, ses extrémités

- Pour un triangle, ses trois sommets
- Pour un cercle, son centre et son rayon etc.

Coordonnées cartésiennes

Dans le plan (2D), la translation de vecteur $\vec{u}(a, b)$, transforme le point $M(x , y)$ en $M'(x' , y')$ tel que

$$x' = x + a$$

$$y' = y + b$$

[Figure 1.3-2] La translation dans un plan 2D

Dans l'espace (3D), la translation de vecteur $\vec{u}(a, b, c)$ transforme le point $M(x , y, z)$ en $M'(x' , y', z')$ tel que

$$x' = x + a$$

$$y' = y + b$$

$$z' = z + c$$

[Figure 1.3-3] La translation dans un espace 3D

1.4 Échelle

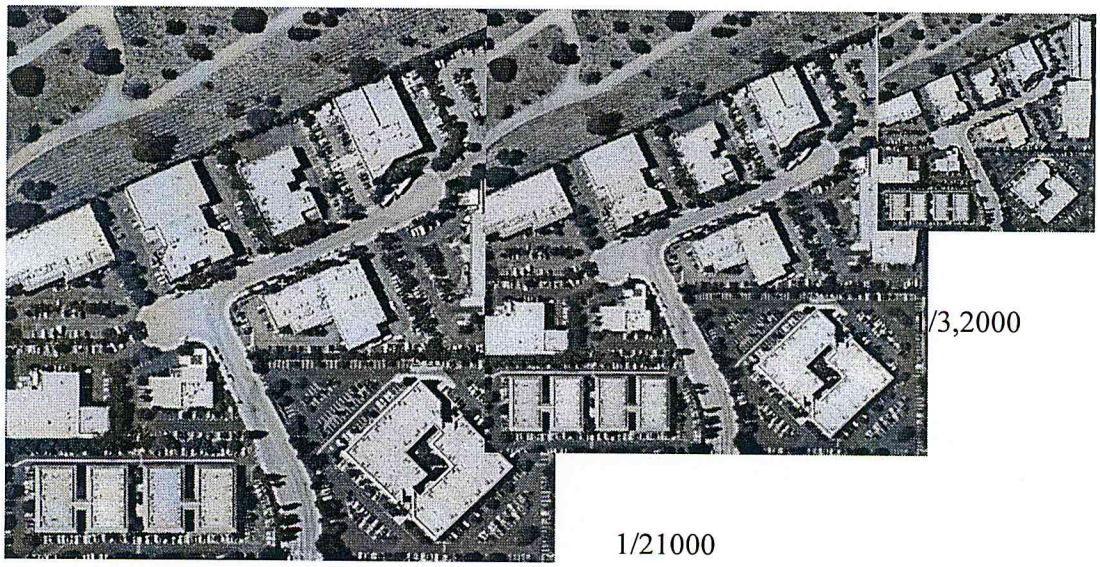
Une **échelle** est le rapport entre la mesure réelle d'un objet et la mesure de sa représentation (carte géographique, maquette, etc.).

Une échelle 1/1000 (équivalent à « 1 :1000 » ou « au 1000^{e} ») implique la formule suivante :

Dimension apparente = dimension réelle x (1/1000)

Dans ce cas, la représentation est 1000 fois plus petite que l'objet réel (1 centimètre pour 100 centimètres dans la réalité, soit 1 mètre).

L'expression à grande échelle désigne donc une échelle qui se rapproche de 1/1, soit une représentation relativement précise d'une réalité peu étendue. Au contraire, à petite échelle désigne une échelle de valeur élevée, soit une représentation de taille réduite.



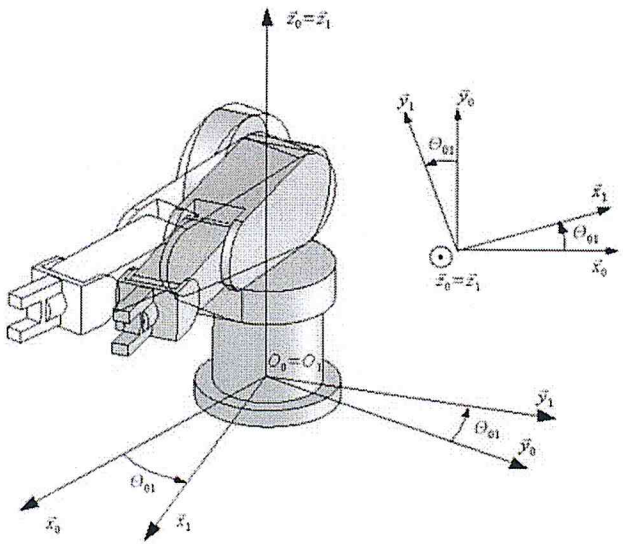
[Figure 1.4-1] Schéma représente les différentes échelles d'une partie d'une ville

1.5 Changement de repère

Pour passer d'un repère à un autre ou bien un **changement de repère**, on passe par un ensemble des opérations permettant ce passage. Un changement de repère n'est pas un changement de référentiel, le premiers changement n'a pas d'influence sur les propriétés des objets.

L'ensemble des opérations a affecté à un changement de repère sont :

- la rotation, opérée par une matrice de rotation (voir 1.3)
- la translation (voir 1.4)
- le changement d'échelle (voir 1.5)



[Figure 1.5-1] Le changement de repère en pratique dans la mécanique du solide

1.6 La Distance entre deux Points

La notion de distance se situe au cœur de toute conception de l'espace. L'une des faiblesses structurelles de la géographie a longtemps été de ne pas avoir de réflexion théorique sur la distance, ce qui l'empêchait de tirer parti des philosophies de l'espace et des concepts de distance d'autres disciplines. Une distance est une application qui formalise l'idée intuitive de distance, c'est-à-dire la longueur qui sépare deux points [9].

1.6.1 La Distance Euclidienne

Il existe plusieurs manières pour calculer la distance entre 2 points :

- Distance de Manhattan
- Distance euclidienne
- Distance de Minkowski
- Distance de Tchebychev

Pour notre cas on choisi la distance euclidienne (ou 2-distance) car elle permet de généraliser le théorème de Pythagore dans un espace de dimension n et plus intuitive.

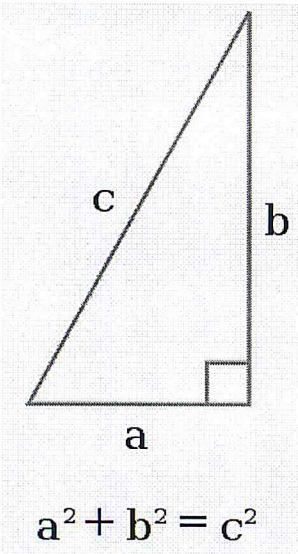
On suppose deux points de E^n, $A(x_1, x_2, \dots, x_n)$, $B(y_1, y_2, \dots, y_n)$

La distance entre le point A et le point B est noté par $d(A, B)$ par la formule suivante :

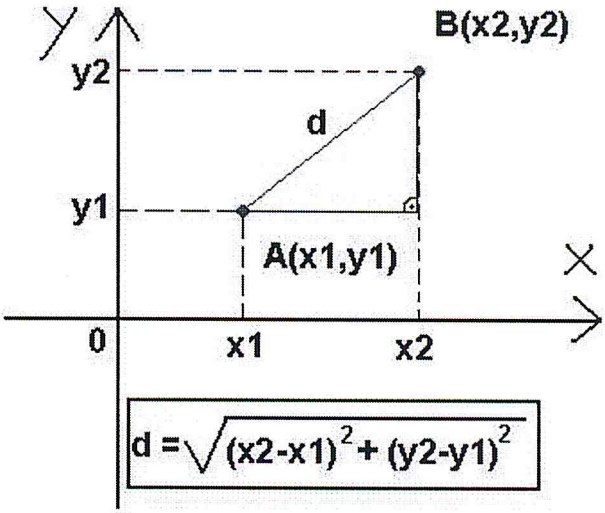
$$d(A,B) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

[Figure 1.6.1-1] Formule de la distance euclidienne

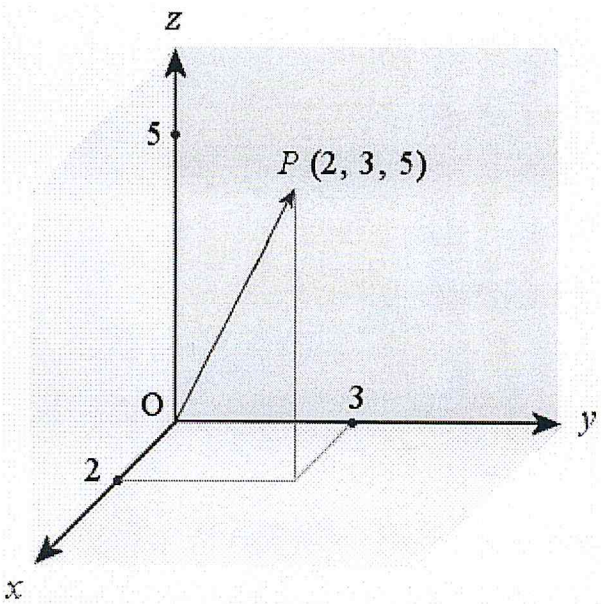
1.6.2 Applications



[Figure 1.6.2-1] Théorème de Pythagore



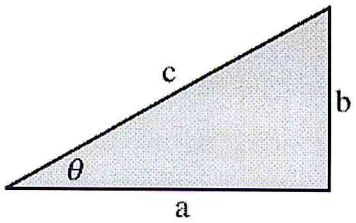
[Figure 1.6.2-2] Distance euclidienne entre A(x1, y1) et B(x2, y2) dans E^2



[Figure 1.6.2-3] Distance euclidienne entre l'origine et le point P dans E^3

1.7 Angle d'inclinaison

Soit la figure suivante :



[Figure 1.7-1] Triangle

Pout trouver l'angle θ on utilise les fonctions trigonométriques de base, et leurs fonctions inverse

$$\sin\theta = \frac{b}{c} = \frac{côté\ opposé\ de\ \theta}{hypoténuse}$$

$$\cos\theta = \frac{a}{c} = \frac{côté\ adjacente\ de\ \theta}{hypoténuse}$$

$$\tan\theta = \frac{\sin\theta}{\cos\theta} = \frac{b}{a} = \frac{côté\ opposé\ de\ \theta}{côté\ adjacente\ de\ \theta}$$

$$\theta = \tan^{-1}\frac{b}{a}$$

Pour le cas où $a = 0 \Longrightarrow \theta = 90° = \frac{\pi}{2}$

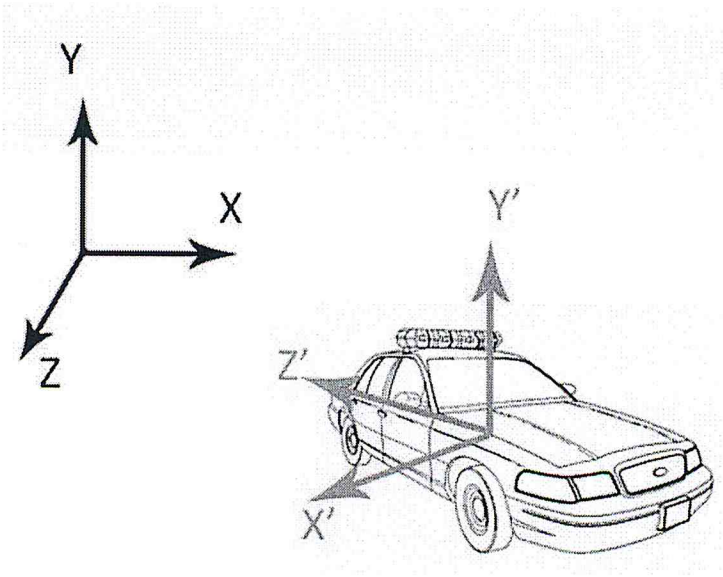
Si x appartient à]-π/2 ; π/2[et y appartient à $\mathbb{R}$:

$$y = \tan x \Leftrightarrow x = \tan^{-1} y$$

Conclusion

Dans ce chapitre, nous avons traité les différentes bases de la géométrie sous forme des formules mathématiques afin de pouvoir implémenter dans notre projet :

- Dans une scène 3D, les coordonnées sphériques (voir 1.2) sont utilisées dans le mouvement d'un objet par rapport à son centre
- Si le centre se déplace, on applique la notion de translation (voir 1.4)
- Au cours du mouvement, l'objet peut se déplacer (voir 1.4) et/ou change son angle d'inclinaison (voir 1.3) et trouver cette angle (voir 1.8)
- En vue 2D, il est préférable de réduire la dimension (voir 1.5) pour avoir une vue globale de la scène
- Les opérations de rotation (voir 1.3), translations (voir 1.4), changement d'échelle (voir 1.5) provoquent un changement de repère
- Le calcul de la distance entre deux points, ou entre deux objets (voir 1.7)



[Figure 2] Dessin représente un véhicule dans un autre repère

Chapitre 2
OpenGL

Introduction

Quand on parle de bibliothèque de développement 3D, les mots qui reviennent le plus souvent sont Direct3D et OpenGL. Direct3D est une API développée par Microsoft pour son system d'exploitation Windows. Le concurrent de Direct3D est la bibliothèque OpenGL.

Dans ce chapitre, nous allons donner un aperçu de ce à quoi sert L'OpenGL, de son historique et les différentes fonctionnalités offerts.

2.3 Présentation générale sur l'OpenGL

OpenGL (Open Graphique Library, ce que se traduit en bon français par Bibliothèque Graphique Ouverte est une API développée depuis 1992 par Silicon Graphics, un des grands ténors de l'informatique graphique professionnelle.

Originellement développé pour les stations graphiques haut de gamme Silicon Graphics, l'aspect ouvert d'OpenGL lui a valu d'être porté sur la plupart des plateformes modernes, et sur la plupart des systèmes d'exploitation. Aujourd'hui, OpenGL est reconnu par les professionnels comme un standard, et est utilisé par tous les logiciels professionnels de synthèse d'images (3DSMax, Maya, Softimage...), de CAO (ProEngineer, Catia...), mais aussi dans le domaine des jeux vidéos 3D (Quake (1, 2,3), Unreal...) [10].

OpenGL permet de visualiser une scène 2D ou 3D. Les objets de cette scène peuvent être composés de points, de lignes et de polygones. Ils possèdent des attributs graphiques : couleur, texture. L'éclairage de la scène réalisé par des lumières de différents types.

OpenGL est résolument orienté vers l'interaction temps réel, c'est à dire qu'un programme conçu avec OpenGL permet en général `a l'utilisateur d'agir par l'intermédiaire de périphériques d'entrée (clavier, souris, trackball...) sur la scène (déplacement de caméra, changement de focale, déplacement des objets....), et d'en visualiser immédiatement l'effet [11].

2.2 Les Bibliothèques coexistant avec OpenGL

Il existe plusieurs bibliothèques développées à partir d'OpenGL afin d'apporter des fonctionnalités qui ne sont pas disponible dans la bibliothèque OpenGL elle-même, ces bibliothèques doivent être ajoutés à l'application, parmi ces bibliothèques:

2.2.1 OpenGL Library (GL)

GL est une libraire standard proposant les fonctions de base pour l'affichage en OpenGL, elle ne contient pas de fonction pour la construction d'une interface utilisateur (fenêtres, souris, clavier, ...).

2.2.2 OpenGL Utility Library (GLU)

Glu, Il s'agit d'une bibliothèque officielle de fonctions pour OpenGL, fournissant des méthodes de plus haut niveau pour par exemple initialiser les matrices de projection/transformation ou dessiner des surfaces complexes [12].

2.2.3 OpenGL Utility Toolkit (GLUT)

Cette libraire écrite pour rendre simple la programmation des petites applications dans le cadre d'interfaces graphiques interactives simples, c'est elle qui gère l'interaction entre l'OpenGL et le système.

Proposant des fonctions pour :

- La gestion d'une fenêtre d'affichage.
- La gestion de la souris.
- La gestion du clavier.
- La gestion des menus.
- La gestion des environnements multifenêtres.
- La gestion des périphériques d'entrée supplémentaires.

FreeGlut est une réécriture de la bibliothèque GLUT, les nouveautés apporté par cette bibliothèque c'est essentiellement de la correction des bugs.

2.2.4 OpenGL Extension Wrangler Library (GLEW)

Une bibliothèque multiplateforme qui permet la gestion simple des extensions OpenGL, qui apporte une solution à la complexité de la gestion des extensions OpenGL.

A l'exécution, elle détecte les extensions supportées par la machine, et initialise les fonctions associé à ces extensions.

2.3 Les bases d'OpenGL

Dans cette partie, nous allons citer les fonctionnalités vues et acquises en OpenGL.

2.3.1 Les fonctions d'initialisation d'OpenGL

Il existe deux fonctions qui assurent l'initialisation d'OpenGL, la première permet de spécifier la couleur du fond avec le mode de spécification de couleur RGB et un composant supplémentaire alpha utilisé pour la gestion de la transparence des objets. La deuxième permet de spécifier la taille des sommets à l'écran.

2.3.2 Les fonctions d'initialisation du GLUT et de la gestion d'une fenêtre

Pour l'initialisation du GLUT, il faut ajouter des fonctions permettent de:

- Initialise la bibliothèque *GLUT* et négocie une session avec le système de fenêtrage.
- Régler les paramètres liés à l'affichage.
- Spécifier la position du coin supérieur gauche de la fenêtre OpenGL par rapport au coin supérieur gauche de l'écran.
- Définir la largeur et la hauteur de la fenêtre.
- Créer une fenêtre OpenGL.

2.3.3 Les fonctions de rappel associé aux évènements

GLUT offert des fonctions qui sont associées aux différents types d'événements envoyés par le système, ces fonctions spécifient :

- La fonction à appeler lorsque la fenêtre est retaillée.
- La fonction à appeler lorsque la fenêtre est appuyée.
- La fonction à appeler lorsqu'un bouton de la souris est appuyé.
- La fonction à appeler lorsque la souris est déplacée tout en gardant un bouton appuyée.

2.3.4 Gestion d'un processus en tache de fond

Ce processus s'exécute s'il n'y a pas d'événement en attente Cette fonction est fréquemment utilisée pour programmer des animations.

2.3.5 Dessin d'objets 3D

GLUT possède des routines pour afficher des objets 3D en fil de fer ou en volumique, nous allons lister l'ensemble de ces objets :

Sphère, cube, tore, cône, tétraèdre, octaèdre, icosaèdre, dodécaèdre.

2.3.6 Le double tampon

Le double buffer est une technique ayant pour but l'amélioration de la qualité et de la vitesse d'affichage lors de la réalisation des animations : montrer à l'écran une image correspondant à une première zone mémoire (buffer) et dessiner les objets dans une deuxième zone mémoire qui n'est pas encore à l'écran, et qui sera affiché lorsque la scène entière y sera calculée.

2.3.7 Le tampon de profondeur

Pour résoudre le problème des faces cachées, OpenGL propose une technique extrêmement répandue en synthèse d'images : le tampon de profondeur, plus connu sous son nom anglais : Zbuffer. Le tampon de profondeur est une technique simple et très puissante. L'idée principale est de créer en plus de notre image un tampon de même taille. Ce tampon va servir à stocker pour chaque pixel une profondeur, c'est à dire la distance entre le point de vue et l'objet auquel appartient le pixel considéré.
A l'origine, le tampon est rempli avec une valeur dite de profondeur maximale: l'image est vide [10].

2.4 Vision

2.4.1 Le Principe de la vision

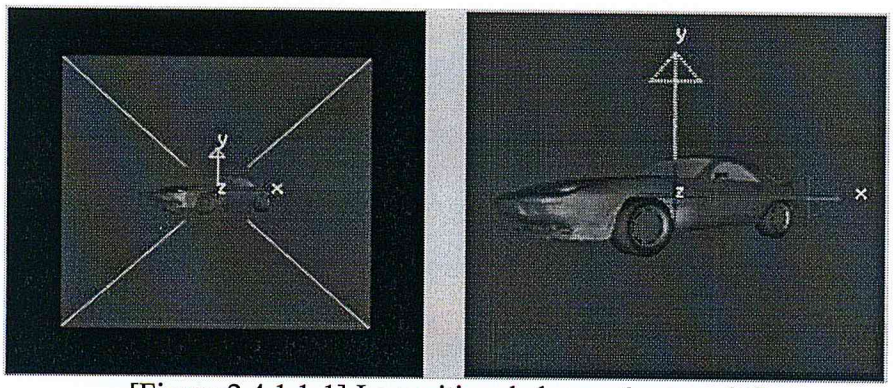
Le processus de transformation qui produit une image à partir d'un modèle de scène 3D est analogue Il comprend quatre étapes :

	Sur un ordinateur	type de transformation
1	Placer la caméra virtuelle	transformation de vision
2	Composer une scène virtuelle à représenter	transformation de modélisation
3	choisir une projection	transformation de projection
4	choisir les caractéristiques de l'image	transformation de cadrage

[Table 2.4.1-1] Table des types de transformation

2.4.1.1 La transformation de vision

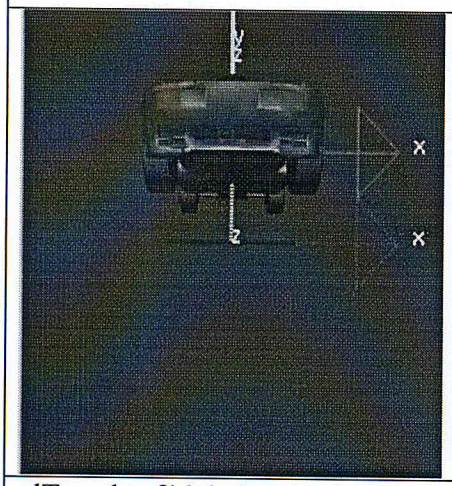
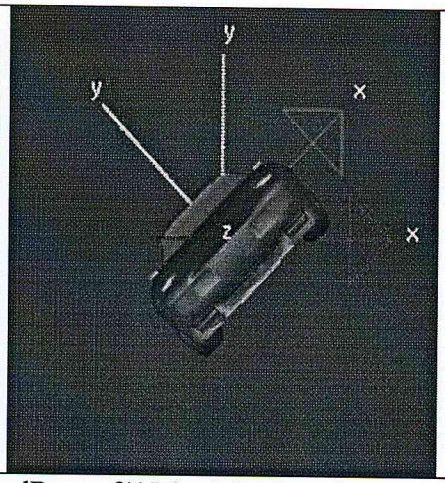
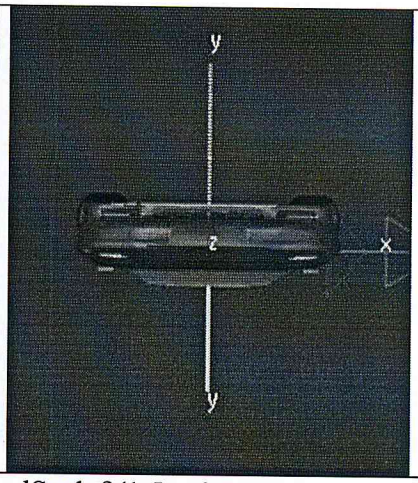
Elle modifie la position et l'orientation de la caméra virtuelle. La position par défaut de la caméra est à l'origine du repère de la scène, orientée vers les z négatifs, et la verticale de la caméra est alignée avec l'axe des y positifs :



[Figure 2.4.1.1-1] La position de la caméra par défaut

On utilise ainsi les procédures OpenGL de translation et de rotations appliquées aux objets, On peut également utiliser une procédure pour changer le point de vue de la caméra.

2.4.1.2 La transformation de modélisation

translate le repère local de l'objet du vecteur(x, y, z).	opère une rotation de l'objet autour du vecteur (x, y, z).	opère un changement d'échelle sur 3 axes. Les coordonnées en x sont multipliées par a, en y par b et en z par c.
		
glTranslatef(0.0, 0.5, 0.0)	glRotatef(45.0 , 0.0 , 0.0 , 1.0)	glScalef (1.5 , -0.5 , 1.0)

[Table 2.4.1.2-1] La Manipulation d'un objet voiture

La manipulation d'objets 3D peut mener à des réflexions complexes pour bien positionner et orienter les objets. Et l'application successive d'une translation puis d'une rotation, ou l'inverse mènent à des résultats différents.

2.4.1.3 La transformation de projection

OpenGL permet de choisir le mode d'affichage des scènes dessinées. Deux choix sont offerts :

- La projection orthographique.
- La projection en perspective.

2.4.1.3.1 Projection perspective

Peuvent être utilisées. Ces fonctions définissent le volume de vision sous la forme d'une pyramide tronquée. Elle fait apparaître les objets lointains plus petits que ceux qui sont proches.

2.4.1.3.2 Projection orthographique

La projection orthographique n'affecte pas la taille relative des objets.

2.4.1.4 La transformation de cadrage

La transformation de cadrage pour définir une zone rectangulaire de pixels dans la fenêtre dans laquelle l'image finale sera affichée.

2.6 Eclairage

2.5.1 Modèle d'éclairage OpenGL

La perception de la couleur de la surface d'un objet du monde réel dépend de la distribution de l'énergie des photons qui partent de cette surface et qui arrivent aux cellules de la rétine de l'œil. Chaque objet réagit à la lumière en fonction des propriétés matérielles de sa surface. Le modèle d'éclairage d'OpenGL considère qu'un objet peut émettre une lumière propre, renvoyer dans toutes les directions la lumière qu'il reçoit, ou réfléchir une partie de la lumière dans une direction particulière, comme un miroir ou une surface brillante. Les lampes, elles, vont envoyer une lumière dont les caractéristiques seront décrites par leurs trois composantes : ambiante, diffuse ou spéculaire [12].

OpenGL distingue quatre types de lumières :

2.5.1.1 Lumière émise Ne concernent que les objets

Les objets peuvent émettre une lumière propre, qui augmentera leur intensité, mais n'affectera pas les autres objets de la scène [12].

2.5.1.2 Lumière ambiante Concernent les objets et les lampes

C'est la lumière qui a tellement été dispersée et renvoyée par l'environnement qu'il est impossible de déterminer la direction d'où elle émane. Elle semble venir de toutes les directions. Quand une lumière ambiante rencontre une surface, elle est renvoyée dans toutes les directions [12].

2.5.1.3 Lumière diffuse Concernent les objets et les lampes

C'est la lumière qui vient d'une direction particulière, et qui va être plus brillante si elle arrive perpendiculairement à la surface que si elle est rasante. Par

contre, après avoir rencontré la surface, elle est renvoyée uniformément dans toutes les directions [12].

2.5.1.4 Lumière spéculaire Concernent les objets et les lampes

La lumière spéculaire vient d'une direction particulière et est renvoyée par la surface dans une direction particulière. Par exemple un rayon laser réfléchi par un miroir [12].

2.5.3 Les lampes

2.5.2.1 Nombre de lampes

OpenGL offre d'une part une lampe qui génère uniquement une lumière ambiante (lampe d'ambiance), et d'autre part au moins 8 lampes de GL_LIGHT0 à GL_LIGHT7 que l'on peut placer dans la scène et dont on peut spécifier toutes les composantes [12].

2.5.2.2 Couleur des lampes

Dans le modèle d'OpenGL, la couleur d'une composante de lumière d'une lampe est définie par les pourcentages de couleur rouge, verte, bleue qu'elle émet. Il faut donc définir pour chaque lampe la couleur et l'intensité des trois composantes ambiante, diffuse et spéculaire [12].

2.5.2.3 Lampes directionnelles

Il s'agit d'une lumière qui vient de l'infini avec une direction particulière. La direction est spécifiée par un vecteur (x, y, z) [12].

2.5.2.4 Lampes omnidirectionnelles et spots

Par défaut, une lampe illumine l'espace dans toutes les directions. L'autre type de lampe proposé par OpenGL est le spot. Un spot est caractérisé, en plus de sa position, par sa direction, le demi-angle du cône de lumière et l'atténuation angulaire de la lumière [12].

2.5.3 Couleur d'un matériau

Dans le modèle d'OpenGL, la couleur que l'on perçoit d'un objet dépend de la couleur propre de l'objet et de la couleur de la lumière qu'il reçoit. Par exemple, un objet rouge renvoie toute la lumière rouge qu'il reçoit et absorbe toute la lumière verte et bleue qu'il reçoit.

- Si cet objet est éclairé par une lumière blanche (composé en quantités égales de rouge, vert et bleu), il ne renverra que la lumière rouge et apparaîtra donc rouge.
- Si cet objet est éclairé par une lumière verte, il apparaîtra noir, puisqu'il absorbe le vert et n'a pas de lumière rouge à réfléchir [12].

2.5.3.1 Propriétés matérielles d'un objet

Les propriétés matérielles d'un objet sont celles qui ont été évoquées dans la partie Modèle d'éclairage OpenGL : la lumière émise, la réflexion ambiante, diffuse et spéculaire du matériau dont est fait l'objet [12].

Conclusion

OpenGL est essentiel pour l'avenir des applications graphiques parce qu'il est considéré comme un standard ouvert et non commerciale. Il est créé pour permettre aux utilisateurs sur n'importe quelle plateforme d'expérimenter des graphismes de la plus haute qualité dont leur matériel peut donner sa capacité. Nous avons étudié dans ce chapitre la bibliothèque OpenGL avec les différentes bibliothèques coexistantes avec cette dernière, dans le but de réaliser une scène en 3-dimensions qui permettront la simulation d'un trafic de véhicules.

Chapitre 3
Intelligence Artificielle
et
Détection de Collision

Introduction à l'intelligence artificielle

L'intelligence artificielle (IA) est un domaine de l'informatique qui apparait en 1956. Il s'agit des systèmes qui simulent des actions et des réactions associées au comportement humain. La différence se trouve dans la nature des problèmes à étudier. Les techniques algorithmiques classiques ne peuvent pas traiter les différents problèmes de l'intelligence.

Les premiers pas sur le sujet commencent en 1950. Les techniques de programmation résolvent des problèmes comme :

- Les jeux de stratégies (dames, échecs…)
- La démonstration des théorèmes mathématiques d'une façon automatique
- La résolution de problèmes compliqués

Des résultats satisfaisants ont été obtenus par les techniques de l'IA en ce qui concerne les domaines suivants:

- L'enseignement
- L'analyse d'images (reconnaissance de formes) et des sons
- La linguistique et les problèmes de traduction

Ces tâches sont effectuent sans efforts par les êtres humains.

3.1 Déplacement d'un objet dans l'espace

Le déplacement d'un objet dans un espace à 3-dimensions se fait par la modification de ses coordonnées qui provoque le mouvement. L'objet peut par exemple être un robot dans un labyrinthe, un avion, un humain, un véhicule, une galaxie etc.

Le déplacement de cet objet se passe selon l'ensemble de ses caractéristiques et de son comportement qui impliquent un mouvement vers une nouvelle position. Ces objets peuvent être en collision avec d'autres objets au cours de déplacement.

3.2 Etude du comportement du conducteur de véhicules

Il s'agit de mettre au point un modèle de comportement du conducteur selon chaque pays, ou bien une théorie en fonction des facteurs déterminants. Ce modèle s'efforcera de permettre une prévision du comportement en termes statistiques ou de probabilités.

De plus, à partir de la on pourra déterminer les mesures qui devront être proposées en ce qui concerne le contrôle de la circulation dans tous ses aspects, rectifications de projets, réglementation, etc. Le but final de ces études est de réduire les risques d'accidents de circulation [11].

[Figure 3.2-1] Etudes expérimentales et collection de données sur le comportement du conducteur

3.3 Comportement du Conducteur et Intelligence Artificielle

Le comportement du conducteur indique ces types de réponse de chaque situation au cours de son déplacement d'une position à une autre et la réalisation d'un système pour simuler le comportement d'un conducteur est très compliquée et reste en phase de développement; pour cela, notre système réduit cette étude en utilisant une algorithme de cheminement (plus court chemin).

3.4 Problèmes de cheminement

L'objectif est de trouver un chemin entre des sommets d'un graphe qui minimise ou maximise une certaine fonction économique.

L'interprétation des problèmes de cheminement dans les graphes finis en termes de structures algébriques a fait l'objet ; jusqu'ici, de nombreux travaux [12].

Le problème le plus classique consiste à chercher le chemin qui minimise la somme des coûts des arêtes traversées. Il existe des algorithmes polynomiaux pour résoudre ce problème, comme l'algorithme de Dijkstra [13]. En revanche, lorsqu'on ajoute des contraintes supplémentaires comme des fenêtres de temps, le problème devient NP-difficile [14].

3.5 Algorithmes de résolution du problème classique de plus court chemin

Il existe plusieurs algorithmes qui traitent le problème de cheminement, lorsqu'un graphe ne comporte pas de cycle, on utilisera l'algorithme de Bellman [15]. Lorsque les coûts sont positifs, l'algorithme le plus utilisé est l'algorithme de Dijkstra [13].

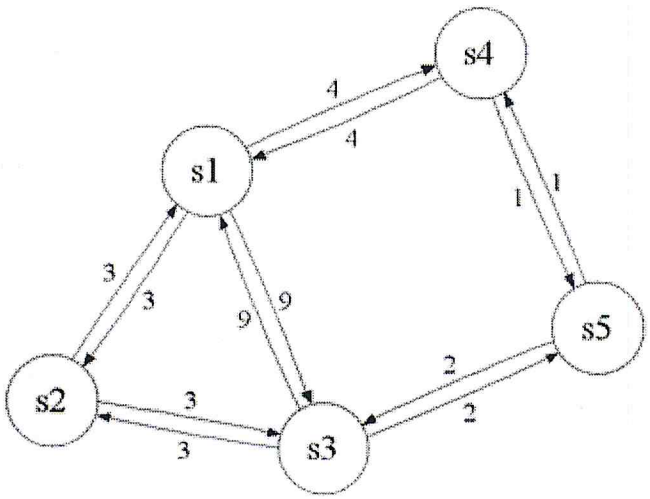
Le coût qui représente un graphe d'une ville est la distance, cette distance est toujours positive et jamais être négative.

3.6 Algorithme de Dijkstra

L'algorithme de Dijkstra ou l'algorithme qui trouve le chemin le plus court et le moins cher. Il est connu comme le problème du plus court chemin de source simple. Il calcule la longueur du plus court chemin de la source à chacun des sommets d'un graphe et minimise le cout.

Soit G = {V, E} un graphe orienté, V est l'ensemble de sommets, E est l'ensemble de arcs.

Pour chaque arc e dans E, Coût(e) est le coût de l'arc e, tous les coûts dans le graphe doit être positif.



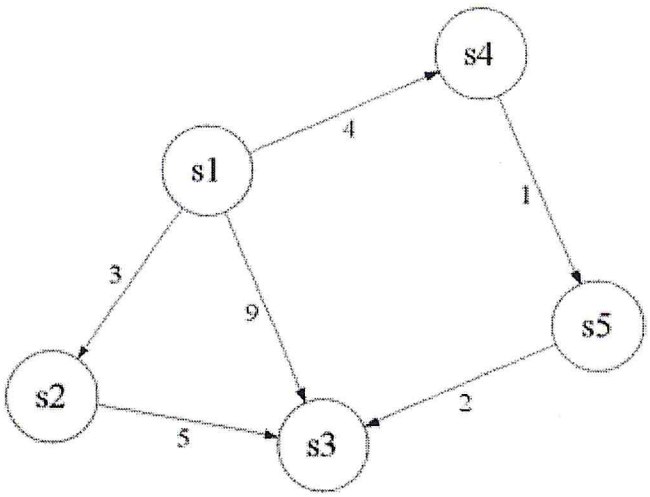
[Figure 3.6-1] Exemple d'un graphe orienté

3.6.1 Procédure

```
Function Dijkstra(V, child, start, end, distance)
    Foreach s in V
        s.visited = infinite
        s.previous= 0
    End foreach
    start.visited = 0
    sNotVisited         = V
    While sNotVisited != empty
        s1 = min(sNotVisited)
        sNotVisited.remove(s1)
        Foreach s2 in child(s1)
            If s2.visited  > s1.visited + distance(s1, s2)
               s2.visited  = s1.visited + distance(s1, s2)
               s2.previous = s1
               sNotVisited.add(s2)
            End if
        End foreach
    End while
    path = empty
    sEnd = end
    While n != début
        path.addBefore(sEnd)
        sEnd = sEnd.previous
    End while
    path.addBefore(start)
    Return path
End function Dijkstra
```

3.6.2 Illustration

Soit le graphe suivant (Figure 3.6.2-1)



[Figure 3.6.2-1] Exemple d'un graphe orienté à traiter

Recherchons des chemins de longueur minimale entre le sommet s1 et tous les autres.

Considérons l'ensemble S des sommets pour lesquels le chemin de longueur minimale recherché est connu. Au début de l'exécution de l'algorithme, cet ensemble est vide. A chaque étape, nous allons ajouter un nouveau sommet à cet ensemble.

Nous allons associer à chaque sommet s un triplet donnant les informations connues sur ce sommet, et mettre à jour ce triplet à chaque étape de l'algorithme. Ce triplet sera composé de :

- Un booléen indiquant si le sommet s appartient à S (noté T s'il vaut vrai et F s'il vaut faux)
- La longueur du plus court chemin trouvé jusqu'à présent reliant le sommet de départ (s1) au sommet s (ou inf si pas de chemin connu)
- Le prédécesseur du sommet s dans ce chemin (ou n si pas de prédécesseur) [23]

Au début de l'exécution de l'algorithme, tous les sommets seront donc associés à (F,inf,n), sauf le sommet de départ (s1) qui sera associé à (F,0,n), car il existe un chemin évidant de longueur 0 reliant s1 à s1. [23]

	étape 0
s1	F,0,n
s2	F,inf,n
s3	F,inf,n
s4	F,inf,n
s5	F,inf,n

[Table 3.6.2-1] Initialisation de Dijkstra pour la figure (3.6.2-1)

A chaque étape, on choisit un sommet s qui n'est pas encore dans S, et dont la longueur du chemin est minimale dans le tableau de l'étape précédente. On ajoute ce sommet s dans l'ensemble S (le booléen passe à T). On considère alors tous les voisins v du sommet s choisi (les sommets directement reliés à s par un arc), et on regarde si le sommet qui mène de s1 à s, puis de s à v est plus court que le plus court chemin menant se s1 à v trouvé jusqu'à présent. Si c'est le cas, on met à jour le triplet associé à v en mettant la nouvelle longueur, et en indiquant que le prédécesseur de v dans le plus court chemin menant de s1 à v est maintenant s. [23]

Lors de la première étape, on choisit donc le sommet s1 (sa longueur 0 est la plus petite), et on l'ajoute à S. Ses voisins sont les sommets s2, s3 et s4.

Le chemin qui mène de s1 à s2 est de longueur 3, qui est inférieure à inf. Le triplet associé à s2 passe donc à (F,3,1).

Le chemin qui mène de s1 à s3 est de longueur 9, qui est inférieure à inf. Le triplet associé à s3 passe donc à (F,9,1).

Le chemin qui mène de s1 à s4 est de longueur 4, qui est inférieure à inf. Le triplet associé à s2 passe donc à (F,4,1). [23]

A la fin de l'étape, on obtient donc :

	étape 0	étape 1
s1	F,0,n	T,0,n
s2	F,inf,n	F,3,1
s3	F,inf,n	F,9,1
s4	F,inf,n	F,4,1
s5	F,inf,n	F,inf,n

[Table 3.6.2-2] Première étape de Dijkstra pour la figure (3.6.2-1)

Lors de l'étape suivante, on choisit le sommet s2 (celui qui n'est pas dans S et qui est de longueur minimale), et on l'ajoute à S. Son seul voisin est le sommet s3. Le chemin qui mène de s1 à s3 en passant pas s2 est de longueur 8 (la longueur 3 du chemin qui va de s1 à s2, plus la longueur 5 de l'arc qui va de s2 à s3). Ce chemin est plus court que celui précédemment trouvé qui était de longueur 9. Le triplet associé à s3 passe donc à (F,8,2). [23]

A la fin de l'étape, on obtient donc :

	étape 0	étape 1	étape 2
s1	F,0,n	T,0,n	T,0,n
s2	F,inf,n	F,3,1	T,3;1
s3	F,inf,n	F,9,1	F,8,2
s4	F,inf,n	F,4,1	F,4,1
s5	F,inf,n	F,inf,n	F,inf,n

[Table 3.6.2-3] Deuxième étape de Dijkstra pour la figure (3.6.2-1)

Lors de l'étape suivante, on choisit le sommet s4 (celui qui n'est pas dans S et qui est de longueur minimale), et on l'ajoute à S. Son seul voisin est le sommet s5. Le chemin qui mène de s1 à s5 en passant pas s4 est de longueur 5 (la longueur 4 du chemin qui va de s1 à s2, plus la longueur 1 de l'arc qui va de s4 à s5). Ce chemin est plus court que celui précédemment trouvé qui était de longueur infinie. Le triplet associé à s5 passe donc à (F,5,4). [23]

A la fin de l'étape, on obtient donc :

	étape 0	étape 1	étape 2	étape 3
s1	F,0,n	T,0,n	T,0,n	T,0,n
s2	F,inf,n	F,3,1	T,3;1	T,3,1
s3	F,inf,n	F,9,1	F,8,2	F,8,2
s4	F,inf,n	F,4,1	F,4,1	T,4,1
s5	F,inf,n	F,inf,n	F,inf,n	F,5,4

[Table 3.6.2-4] Troisième étape de Dijkstra pour la figure (3.6.2-1)

Lors de l'étape suivante, on choisit le sommet s5 (celui qui n'est pas dans S et qui est de longueur minimale), et on l'ajoute à S. Son seul voisin est le sommet s3. Le chemin qui mène de s1 à s3 en passant pas s5 est de longueur 7 (la longueur 5 du chemin qui va de s1 à s5, plus la longueur 2 de l'arc qui va de s5 à s3). Ce chemin est plus court que celui précédemment trouvé qui était de longueur 8. Le triplet associé à s3 passe donc à (F,7,5). [23]

A la fin de l'étape, on obtient donc :

	étape 0	étape 1	étape 2	étape 3	étape 4
s1	F,0,n	T,0,n	T,0,n	T,0,n	T,0,n
s2	F,inf,n	F,3,1	T,3;1	T,3,1	T,3,1

s3	F,inf,n	F,9,1	F,8,2	F,8,2	F,7,5
s4	F,inf,n	F,4,1	F,4,1	T,4,1	T,4,1
s5	F,inf,n	F,inf,n	F,inf,n	F,5,4	T,5,4

[Table 3.6.2-5] Quatrième étape de Dijkstra pour la figure (3.6.2-1)

Lors de l'étape suivante, on choisit le sommet s3 (celui qui n'est pas dans S et qui est de longueur minimale), et on l'ajoute à S. Il n'a pas de voisin. [23]

A la fin de l'étape, on obtient donc :

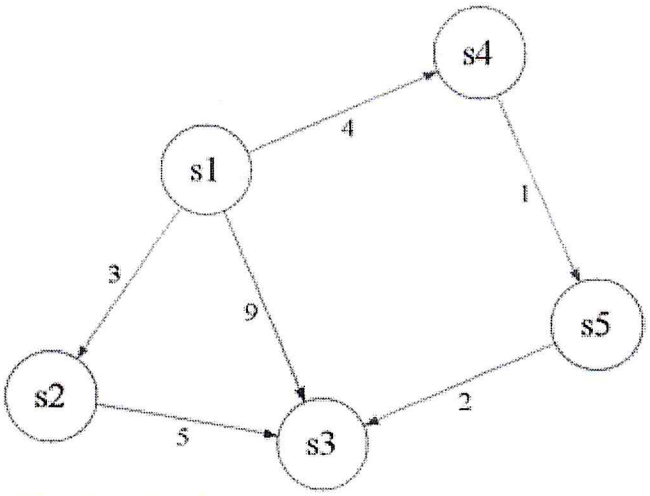
	étape 0	étape 1	étape 2	étape 3	étape 4	étape 5
s1	F,0,n	T,0,n	T,0,n	T,0,n	T,0,n	T,0,n
s2	F,inf,n	F,3,1	T,3;1	T,3,1	T,3,1	T,3,1
s3	F,inf,n	F,9,1	F,8,2	F,8,2	F,7,5	T,7,5
s4	F,inf,n	F,4,1	F,4,1	T,4,1	T,4,1	T,4,1
s5	F,inf,n	F,inf,n	F,inf,n	F,5,4	T,5,4	T,5,4

[Table 3.6.2-6] Cinquième étape de Dijkstra pour la figure (3.6.2-1)

Tous les sommets sont dans S. L'algorithme s'arrête. Il ne reste plus qu'à reconstituer les chemins. [23]

	Chemin	Longueur
s1	1	0
s2	1,2	3
s3	1,4,5,3	7
s4	1,4	4
s5	1,4,5	5

On peut résumer ce résultat par :



[Figure 3.6.2-2] Tous les chemins possibles à parcourir à partir de s1 avec un cout minimal

Les arcs rouges sont les arcs parcourus dans tous les chemins de longueur minimale trouvés.

3.7 Détection de collision

La détection de collisions est un problème classique qui a été étudié massivement ces dernières années par son implication dans de nombreux domaines : robotique et automatique, conception assistée par ordinateur, chaînes de fabrication, simulation informatique, jeux vidéos, etc. [16]

Les défis que doivent relever les algorithmes de détection de collisions sont triples :

- Rapidité, dans la mesure où ces routines sont appelées très fréquemment et souvent en considérant des objets complexes [16]
- Précision, car il est inacceptable de manquer des collisions effectives ou d'ajouter des fictives dans bon nombre d'applications [16]
- Exactitude, (ou au moins fournir une mesure quantifiable des incertitudes) pour assurer le bon fonctionnement du logiciel considéré [16]

Actuellement, il existe deux grandes familles :

- Les algorithmes s'appliquant aux objets convexes dont GJK et LIN-CANNY qui sont considérés comme les plus performants [16]
- Les algorithmes hiérarchiques basés sur des boîtes englobantes comme "Axis Aligned Bounding Boxes" ou "Oriented Bounding Boxes" [16]

3.7.1 Détection exacte

Pour déterminer si deux polyèdres convexes A et B sont en collision, il existe plusieurs familles d'approches [17] :

- Déterminer s'il existe un plan partitionnant l'espace en deux demi-espaces, l'un contenant A, l'autre B. C'est la solution proposée par Chung [18] et van den Bergen [19] (méthodes itératives).
- Calculer la distance entre ces deux objets. Si elle est inférieure ou égale à zéro, il y a collision [20, 21, 22]

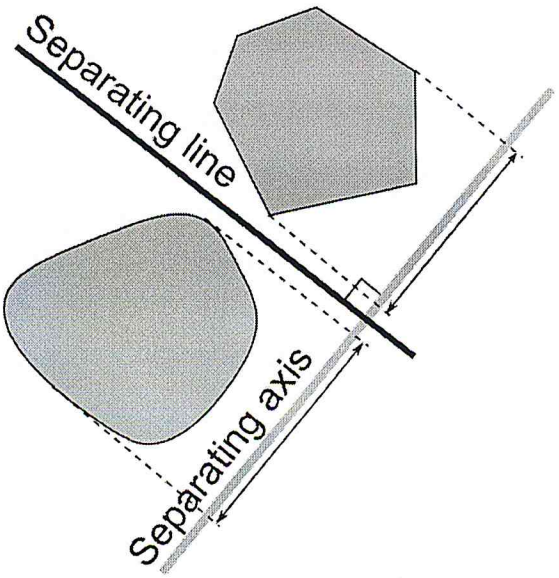
3.7.2 Algorithme de Séparation de l'Axe

3.7.2.1 Introduction

La séparation de l'axe est un algorithme utilisé pour la détection de collision dans des quelques moteurs de physique. L'algorithme est très simple, et facile à comprendre géométriquement. Il également engage un coût informatique élevé $O(mn)$, où m et n sont le nombre de visages pour chaque polygone, donc c'est la meilleure pour réaliser cet essai si nécessaire. La plupart des moteurs de physique ont quelques phases de détection de collision. Ces phases pourraient inclure un certain genre de division spatiale comme des quad-arbres dans 2-dimensions ou des oct-arbres dans 3-dimensions, ou des contrôles de bondissement de boîte ou de sphère.

3.7.2.2 La base

Si on trace une ligne entre deux polygones convexes dans 2-dimensions sans cette ligne intersecte un des polygones, alors ils n'intersectent pas. Si telle ligne n'existe pas, alors ils intersectent. (Figure 3.7.2.2-1) montre un exemple d'une paire de non-intersection de polygone. La ligne pointillée est la ligne de séparation et la ligne verte est l'axe de séparation.

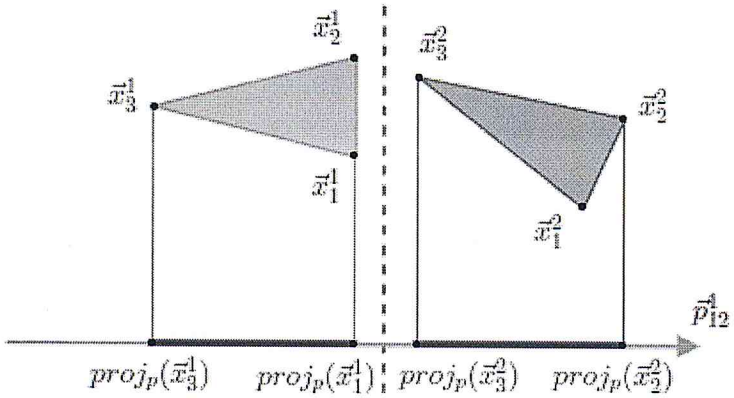


[Figure 3.7.2.2-1] Schéma représente la technique de l'axe de séparation

3.7.2.3 L'algorithme

Pour chaque deux polygone convexe, on choisit une face.

Dans la (Figure 3.7.2.2-1) chaque $\vec{x}$ est un sommet, l'indice supérieure identifié la forme, et l'indice inférieure représente le sommet, $\vec{p}$ est l'axe séparateur. Une fois qu'on a l'axe séparateur nous projetons chaque sommet de deux polygones sur l'axe de séparation.



[Figure 3.7.2.3-1] Schéma montre l'intervalle de la projection en rouge

La bonde de projection de chaque forme est appelée l'**intervalle de projection** représenté en rouge (Figure 3.7.2.3-1). Si ces deux intervalles ne s'interfèrent pas on dit que les deux polygones n'intersectent pas.

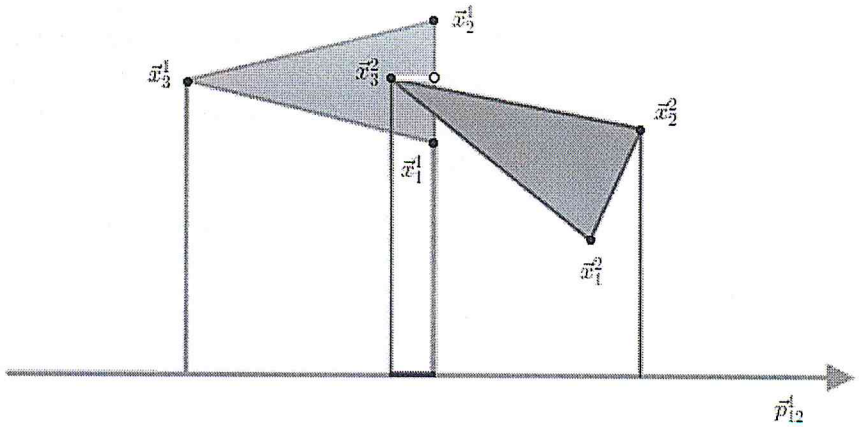
Pour chaque sommet, on calcule sa longueur projeté sur l'axe séparateur

$$d = |proj_p(\vec{x})|$$

Pour chaque forme, d_{min} et d_{max} représente l'intervalle de projection

Si $d_{min}^2 < d_{max}^1$ et $d_{min}^1 < d_{max}^2$ les deux intervalles s'interfèrent (Figure 3.7.2.3-2)

Sinon ils n'interfèrent pas



[Figure 3.7.2.3-2] Schéma montre la collision entre les deux formes

3.7.2.4 Avantages et Inconvénients

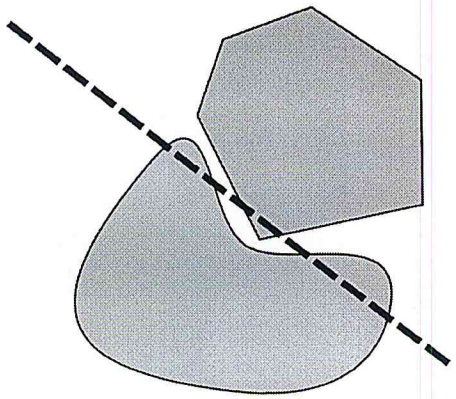
Comme toutes les techniques de détection de collision, cet algorithme a des avantages et des inconvénients. Voici quelques [24]:

Avantage [24]**:**

- Il est rapide – il emploie des maths assez fondamentales, le test termine dès qu'il trouve une collision éliminant des calculs inutiles
- Il est précis

Inconvénients [24]**:**

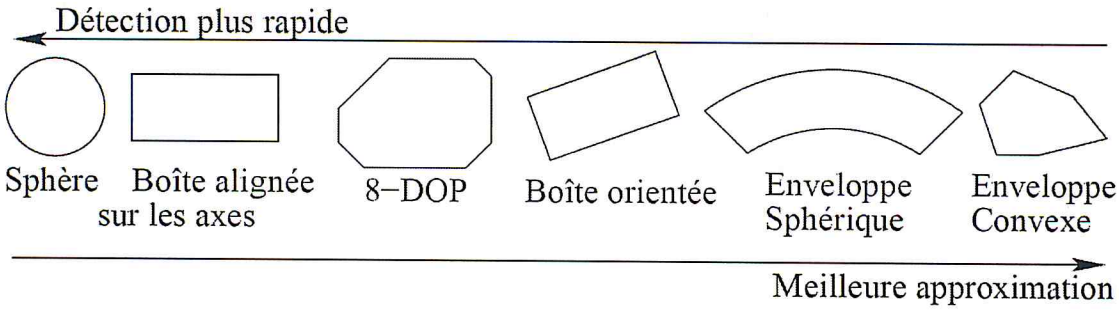
- Il fonctionne seulement avec les polygones convexes (Figure 3.7.2.4-1) – pour que sa fonctionne avec les formes complexes, il faut diviser la forme principale en sous formes convexes après on refait les tests individuels
- Il n'indique pas quels côtés touchent – seulement à quelle distance ils recouvrent et les distances les plus courts pour les séparer



[Figure 3.7.2.4-1] Le théorème ne s'applique pas si un des corps n'est pas convexe

3.7.2.5. Complexité de la détection

Lorsque l'application contient un nombre d'objets relativement élevé, il devient impossible d'utiliser cet algorithme sur la totalité des couples. Il existe d'autres techniques pour accélère la détection comme la détection approximatives ou la recherche approximative.



[Figure 3.7.2.5-1] Exemples de volumes, plus le volume est complexe, mieux il approxime l'objet mais plus longue est la détection de collisions entre deux volumes de ce type (et réciproquement)

Conclusion

L'algorithme de Dijkstra est très important dans la théorie des graphes, on peut le trouver par tout, dans les protocoles de routages, le système GPS, le développement des routières… etc.

Dans la simulation du trafic de véhicules on a besoin de:

- L'intelligence artificielle qui représente le conducteur et son comportement
- La détection de collision qui sert à contrôler le trafic

Chapitre 4
Modélisation du Framework

Introduction

Dans ce chapitre Nous allons proposer une architecture pour notre système de simulation du trafic, nous allons modéliser notre système avec le langage de modélisation UML sachant que la modélisation d'un Framework n'a pas le même processus de la modélisation d'une application, nous allons voir le diagramme de cas d'utilisation, le diagramme de classe et le diagramme de séquence. Avant de passer à la modélisation, nous allons introduire une définition d'un Framework.

4.1 Définition d'un Framework

Le terme Framework est fréquemment utilisé dans des contextes différents mais il peut être traduit par cadre de développement.

Les Framework se présentent sous diverses formes, qui peuvent inclure tout ou partie des éléments suivants :

- un ensemble de classes généralement regroupées sous la forme de bibliothèques pour proposer des services plus ou moins sophistiqués
- un cadre de conception reposant sur les design patterns pour proposer tout ou partie d'un squelette d'application
- des recommandations sur la mise en œuvre et des exemples d'utilisation
- des normes de développement
- des outils facilitant la mise en œuvre

L'objectif d'un Framework est de faciliter la mise en ouvre des fonctionnalités de son domaine d'activité. Il doit permette au développeur de se concentrer sur les tâches spécifiques à l'application à développer plutôt qu'à des tâches techniques récurrentes telles que :

- l'architecture de base de l'application
- l'accès aux données
- l'internationalisation
- la journalisation des événements (logging)
- la sécurité (authentification et gestion des rôles)
- le paramétrage de l'application
- ...

La mise en œuvre d'un Framework permet notamment :

- de capitaliser le savoir-faire sans "réinventer la roue"
- d'accroître la productivité des développeurs une fois le Framework pris en main.
- d'homogénéiser les développements des applications en assurant la réutilisation de composants fiables
- donc de faciliter la maintenance notamment évolutive des applications

Cependant, cette mise en œuvre peut se heurter à certaines difficultés :

- le temps de prise en main du Framework par les développeurs peut être plus ou long en fonction de différents facteurs (complexité du Framework, richesse de sa documentation, expérience des développeurs, ...)
- les évolutions du Framework qu'il faut répercuter dans les applications existantes [25].

Un Framework est habituellement implémenté à l'aide d'un langage à objets, bien que cela ne soit pas strictement nécessaire : un Framework objet fournit ainsi un guide architectural en partitionnant le domaine visé en classes et en définissant les responsabilités de chacune ainsi que les collaborations entre classes [26].

4.2 Architecture du système

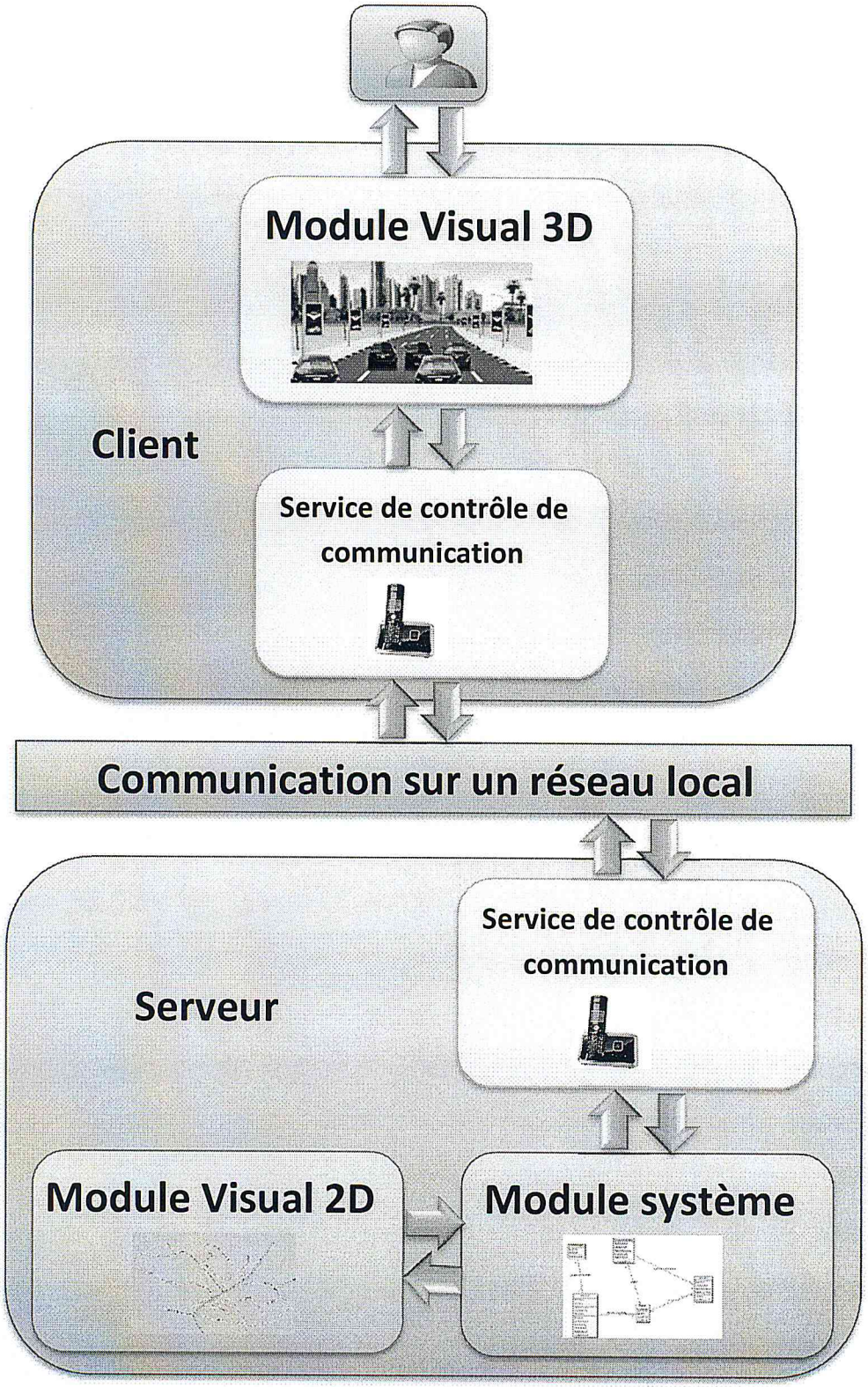


Figure 4.2-1] Architecture Global du Framework

4.3 Description des couches du système

Dans la figure précédente, on a représenté l'architecture globale de notre Framework où on a montré les principaux composants, client et serveur, qui se communiquent sur un réseau local, la connexion se fait avec les sockets, le client est programmé avec C/C++, le serveur est programmé avec JAVA.

4.3.1 Client

Le client est en mode de communication directe avec l'utilisateur et en mode de communication à travers un réseau local avec le serveur, le client est composé de deux sous composants, un module visuel 3D et un service de contrôle de communication avec le serveur.

4.3.1.1 Module Visuel 3D

Ce module contient une interface graphique pour l'utilisateur, un corps 3D qui manipule et qui programme l'environnement en OpenGL, ce module communique avec le service de contrôle de communication.

4.3.1.2 Service de contrôle de communication (client)

Ce service joue le rôle d'intermédiaire entre le serveur et le module Visual 3D, il sert à manipuler l'ensemble des requetés envoyés du module Visual 3D au serveur et l'ensemble des réponses reçus par ce dernier à partir de threads.

4.3.2 Réseau local

Ce réseau local assure la connexion entre le client et serveur.

4.3.3 Serveur

Le serveur est en mode de communication à travers un réseau local avec le client, le serveur est composé de trois sous composants, module système, un service de contrôle de communication et un module visuel 2D.

4.3.3.1 Module système

Le module système qui répond aux requetés reçus du client sur le service de contrôle de communication, il représente le noyau du notre Framework, il est aussi composé d'un module classes et un module IA (contrôleur de collision, et un détecteur du plus court chemin), ce module est en interaction directe avec le module visuel 2D.

4.3.3.2 Module Visuel 2D

Ce module contient une interface graphique pour l'utilisateur du serveur (développeur de l'intelligence artificielle), un corps 2D qui schématise la ville choisi en un graphe et manipule les objets véhicules à partir de threads.

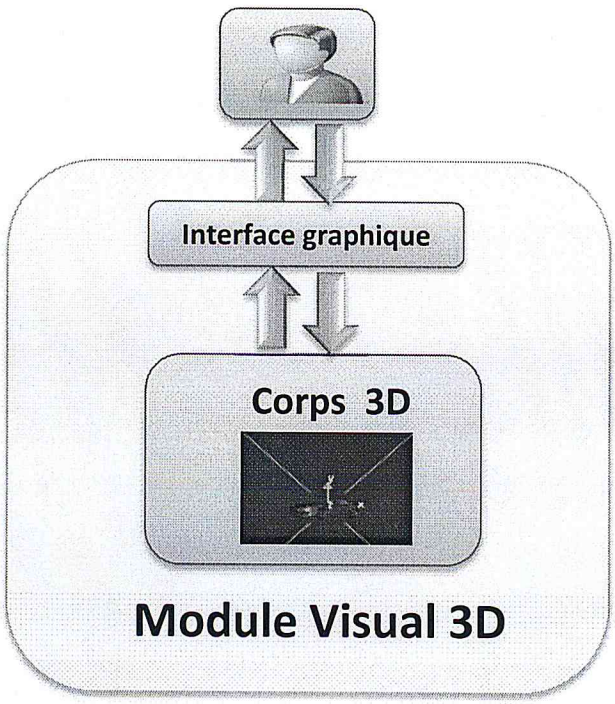
4.3.3.3 Service de contrôle de communication (serveur)

Ce service joue le rôle d'intermédiaire entre le serveur et le module Visual 3D, il sert à manipuler l'ensemble des réponses envoyés du serveur au module Visual 3D et l'ensemble des requetés reçus par le module visuel 3D à partir de threads.

4.4 Description des modules

4.4.1 Module Visual 3D

4.4.1.1 Architecture du Module Visual 3D



[Figure 4.4.1.1-1] Architecture du Module Visual 3D

4.4.1.2 Description du Module Visual 3D

4.4.1.2.1 Interface graphique

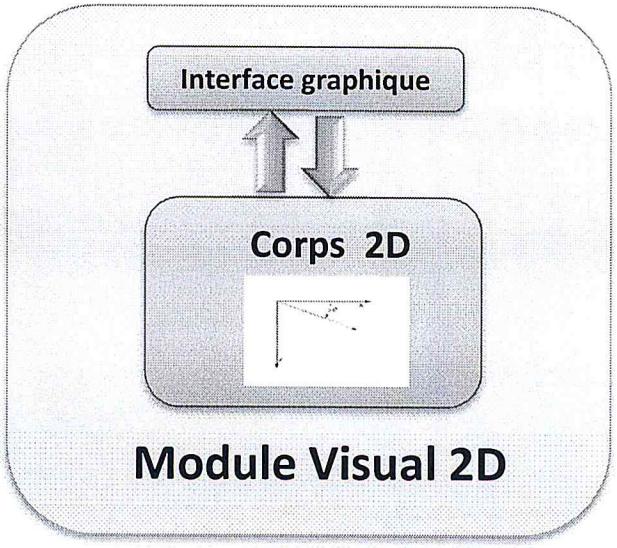
L'interface graphique pour l'utilisateur du système afin qu'il puisse voir l'évolution du monde 3D, il a la possibilité de choisir entre plus qu'une ville.

4.4.1.2.2 Corps 3D

Le corps 3D programme l'environnement en OpenGL et envoie les requetés au serveur et manipule les objets véhicules.

4.4.2 Module visuel 2D

4.4.2.1 Architecture du module visuel 2D



[Figure 4.4.2.1-1] Architecture du Module Visuel 2D

4.4.2.2 Description du module visuel 2D

4.4.2.2.1 Interface graphique

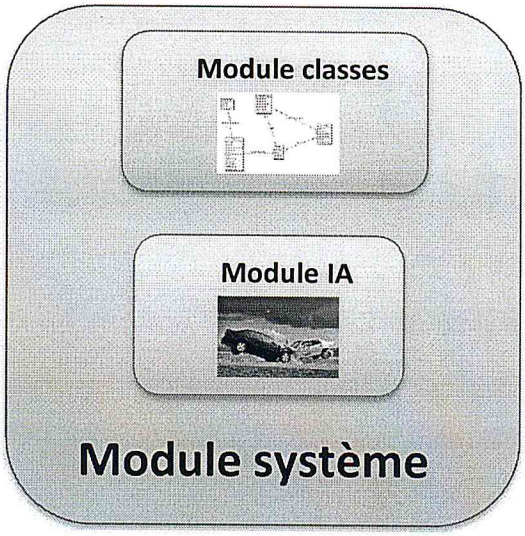
L'interface graphique est pour l'utilisateur du serveur (développeur de l'intelligence artificielle) afin qu'il puisse voir l'évolution du monde 2D

4.4.2.2.2 Corps 2D

Le corps 2D schématise la ville choisi pour la simulation en un graphe et manipule les objets véhicules

4.4.3 Module système

4.4.3.1 Architecture du module système



[Figure4.4.3.1-1] Architecture du Module système

4.4.3.2 Description du module système

4.4.3.2.1 Modules classes

Ce module *contient* les différentes classes qui représentent les objets de notre système.

4.4.3.2.2 Module IA

Le module IA génère un contrôleur de collision qui permet d'effectuer un traitement lors la détection de collision, et un détecteur du plus court chemin qui spécifie le chemin de véhicules dans la scène.

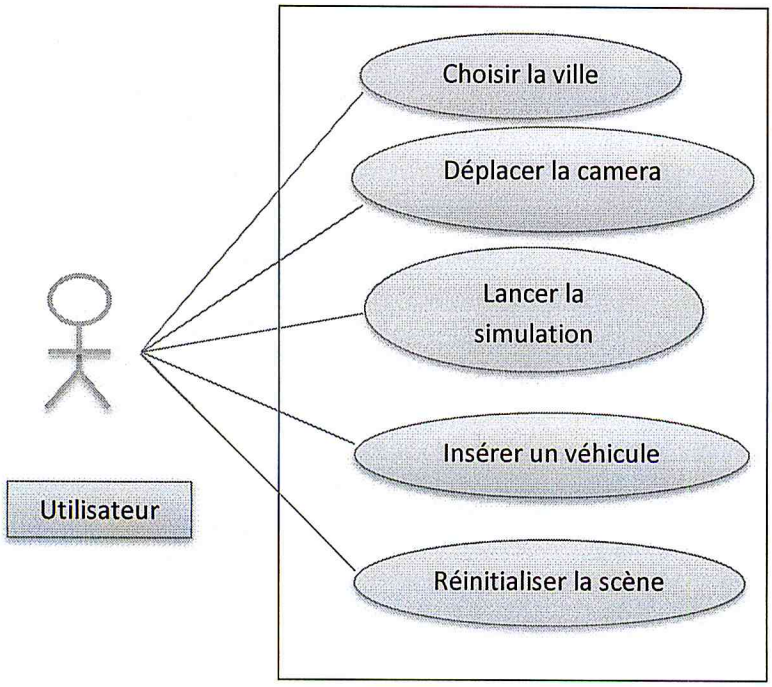
4.5 Modélisation

Dans la partie de modélisation, nous allons utiliser trois types de diagrammes pour concevoir notre système.

4.5.1 Le diagramme de cas d'utilisation

Le diagramme de cas d'utilisation permet d'exprimer les besoins des utilisateurs d'un système, ces cas d'utilisations ont donc une vision d'orientation pour les utilisateurs non informaticiens.

Dans notre diagramme de cas d'utilisation, on a un seul Acteur qui interagit avec le système (l'utilisateur).



[Figure 4.5.1-1] Diagramme de cas d'utilisation pour la simulation d'un trafic de véhicules

Dans ce diagramme de cas d'utilisation, nous avons capturé les différentes fonctionnalités du système, tel qu'un utilisateur externe le voit :

- Le premier cas d'utilisation offre à l'utilisateur la possibilité de choisir la ville, où se fait la simulation.

- Le deuxième cas d'utilisation offre à l'utilisateur la possibilité de déplacer la caméra dans la scène 3D.

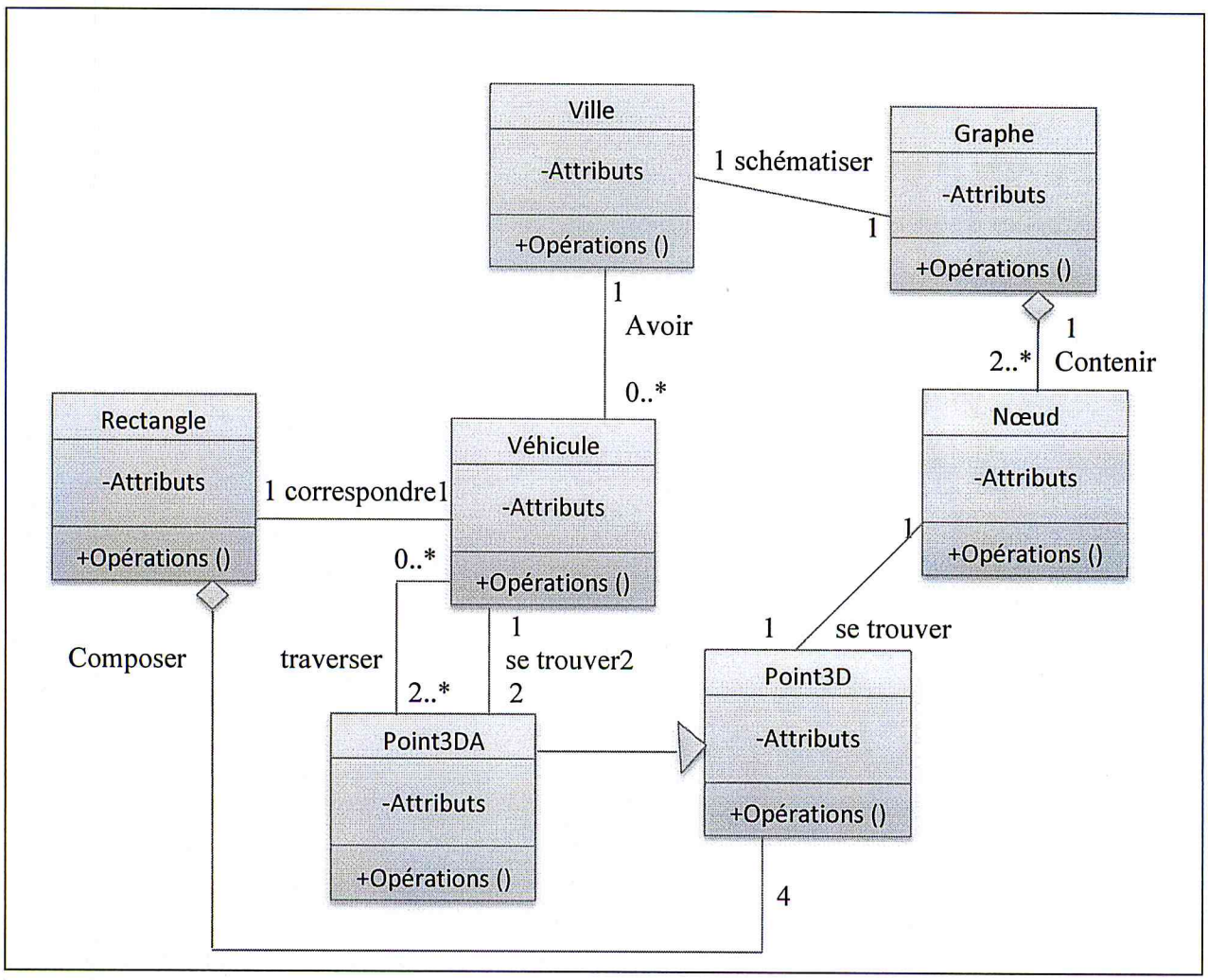
- Le troisième cas d'utilisation permet l'utilisateur de lancer la simulation

- Le quatrième cas d'utilisation permet l'utilisateur à insérer un véhicule dans la scène 3D.

- Le cinquième cas d'utilisation permet l'utilisateur de réinitialiser la simulation sur la scène.

4.5.2 Diagramme de classe

Le diagramme de classe représente les Classes d'objets de notre système dans le serveur.



[Figure 4.5.2] Diagramme de classe du système

4.5.2.1 Liste des attributs des classes

Nom classe	Attributs	Description
Véhicule	-Id -Position x, y, z -Couleur -Shortestway -started -speed -maxSpeed	-Identifiant de véhicule -position de véhicule -la couleur de véhicule -le plus court chemin -le nœud de départ -la vitesse de véhicule -la vitesse maximale

Ville	-Emplacement	-Emplacement de la ville
Graphe	-Position	-Position du graphe
Nœud	-Identifiant - Coordonnées x, y, z	-Identifiant du nœud
Point3D	- Coordonnées x, y, z	/
Point3DA	- Coordonnées x, y, z -angle	/ /
Rectangle	-Valeur	-Intervalle de projection

[Table 4.5.2.1-1] liste des attributs des classes

4.5.2.2 Liste des Méthodes des classes

Classe	**Operations**	**Description**
Véhicule	-getWay() -setspeed(speed) -buildWay() -setPosition(position) -setMax(max) -isStarted() -hasFinish()	-avoir le plus court chemin -affecter une vitesse au véhicule -construire le chemin - affecter une position - affecter une vitesse -vérifier si le véhicule à démarrer -vérifier si le véhicule a atteint leur destination
Ville	-addCar(indentifiant) -getCityLocation() -isLoaded() -reset() -buildGraphe()	-ajouter un véhicule -avoir l'emplacement de la ville -vérifier si la ville est chargée -réinitialiser la ville -construire le graphe
Graphe	-Getters and setters	/
Nœud	-Reset ()	-réinitialiser le nœud
Point3D	-Distance (point1, point2)	-calculer la distance entre 2 points
Point3DA	/	/
Rectangle	-Distance(point1, point2)	-calculer la distance entre 2 points

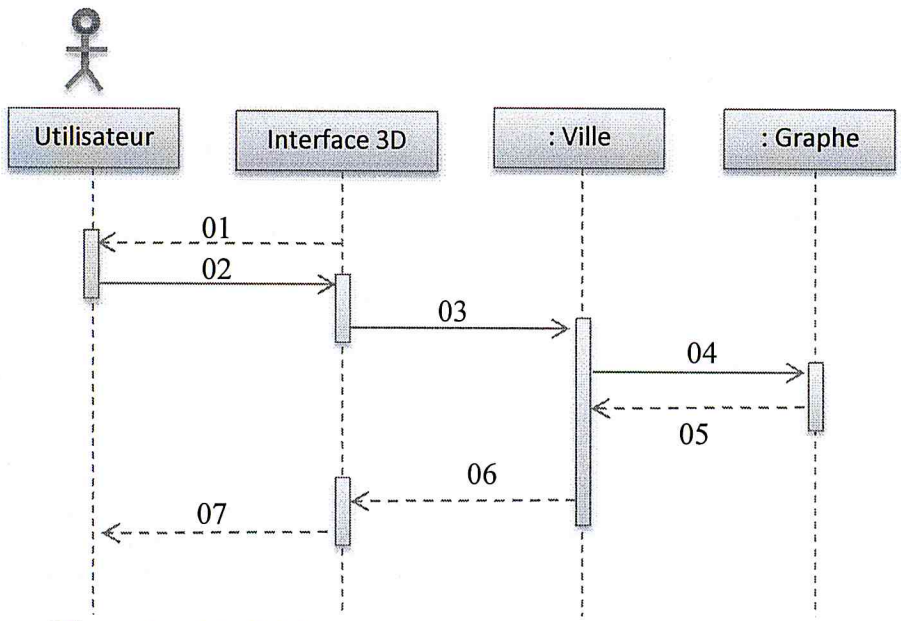
[Table 4.5.2.2-1] liste des méthodes des classes

4.5.3 Le diagramme de séquence

Un diagramme de séquence est une représentation séquentielle du déroulement des traitements et des interactions entre les éléments du système et /ou de ses acteurs. Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel on y met l'accent sur la chronologie des envois de messages.

Dans ce qui suit nous allons présenter les diagrammes de séquence afin de formaliser les scénarios des cas d'utilisation vus précédemment. Nous allons voir le système comme un ensemble d'objet en interaction.

4.5.3.1 Diagramme de séquence « chargement de la ville »



[Figure 4.5.3.1-1] Diagramme de séquence « chargement de la ville »

4.5.3.2 Scenario du Diagramme de séquence « chargement de la ville »

01 : Afficher l'interface client

02 : choisir la ville

03 : charger la ville

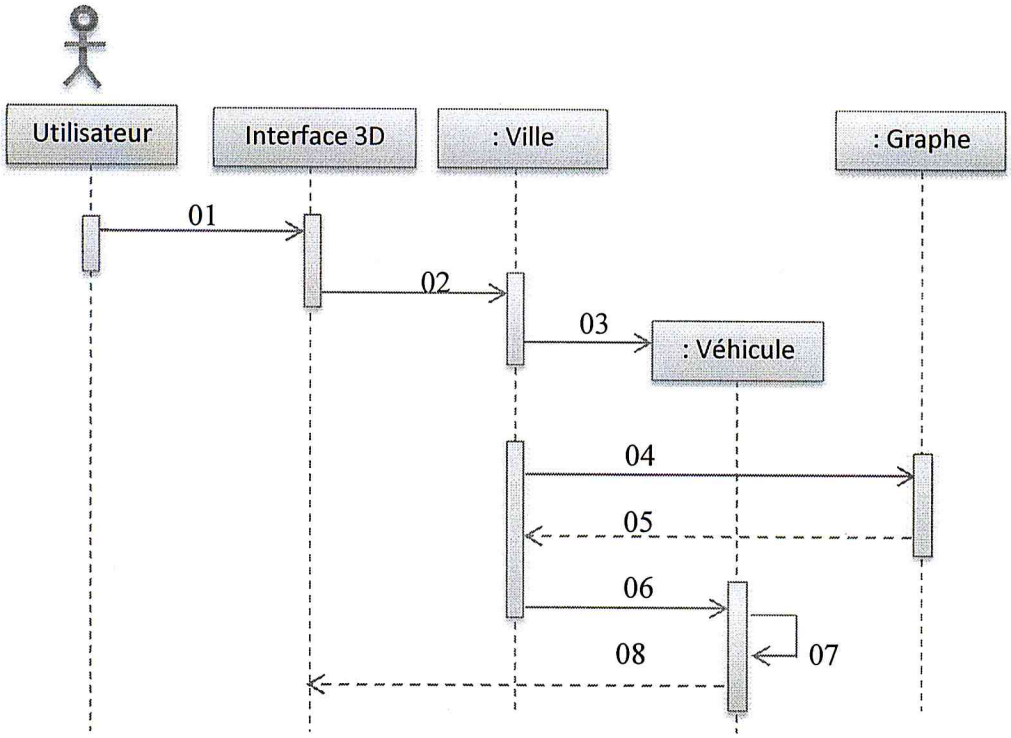
04 : charger le graphe de la ville

05 : retourner un graphe

06 : retourner une structure du graphe

07 : Affichage du résultat

4.5.3.3 Diagramme de séquence « Insertion d'un véhicule»



[Figure 4.5.3.3-1] Diagramme de séquence « chargement de la ville »

4.5.3.4 Scenario du Diagramme de séquence « Insertion d'un véhicule»

01 : Insérer un véhicule

02 : demande à la ville d'insérer un véhicule

03 : créer un véhicule dans la ville

04 : Demander le plus court chemin pour le véhicule

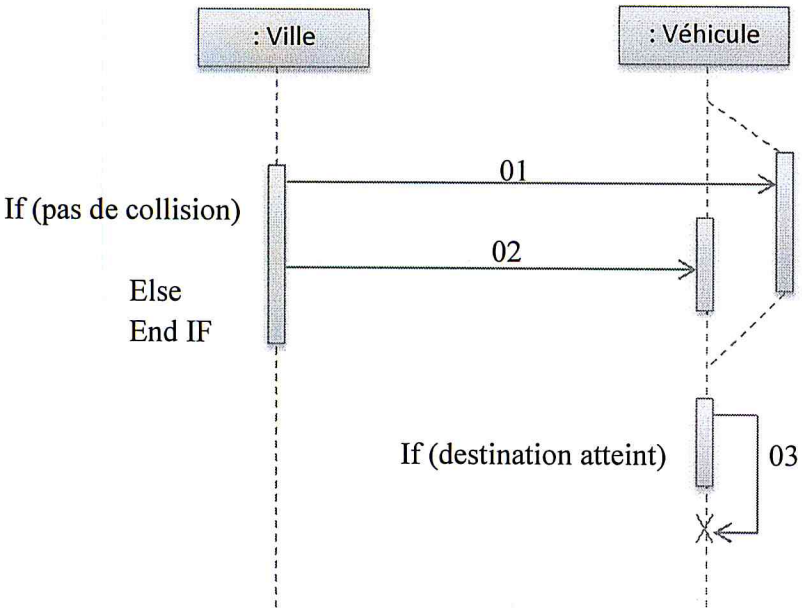
05 : retourner le plus court chemin

06 : Affecter le plus court chemin au véhicule

07 : lancer le véhicule

08 : Affichage de résultat

4.5.3.5 Diagramme de séquence « simulation d'un trafic de véhicules »



[Figure 4.5.3.5-1] Diagramme de séquence «simulation d'un trafic de véhicules »

4.5.3.6 Scenario du Diagramme de séquence «simulation d'un trafic de véhicules »

01 : si pas de collision le véhicule continue son chemin
02 : sinon elle ralentit ou elle s'arrête
03 : si le véhicule atteint son destination, l'objet véhicule se détruit

Conclusion

Dans ce chapitre nous avons présenté une modélisation détaillé notre Framework, cette étude conceptuelle nous a permis de mettre en évidence les étapes nécessaires pour la création du Framework.

La prochaine étape consiste à la réalisation de notre système tout en présentant les outils que nous utiliserons pour l'implémenter.

Chapitre 5
Implémentation

Introduction

Dans cette partie de mémoire, et après la conception du notre Framework, il est le temps de passer à la partie technique de l'informatique et présenter ainsi les principales composantes qui constituent notre Framework et les technologies qui nous ont permis de les développer.

5.1 Environnement et outils de développement

Pour l'implémentation de notre Framework, on a utilisé le langage C/C++ sous un environnement Visual C++ Express et le langage JAVA sous l'environnement Eclipse.

5.1.1 Langage utilisé

5.1.1.1 JAVA

Java est le nom d'une technologie mise au point par Sun Microsystems qui permet de produire des logiciels indépendants de toute architecture matérielle. Java est à la fois un langage de programmation et une plateforme d'exécution. Le langage Java a la particularité principale d'être portable sur plusieurs systèmes d'exploitation tels que Windows, Mac OS ou Linux. C'est la plateforme qui garantit la portabilité des applications développées en Java.

Il permet de créer des applications autonomes et de doter les documents html de nouvelles fonctionnalités : animations interactives, applications intégrées, modèles 3D, etc. Ce langage est orienté objet et comprend des éléments spécialement conçus pour la création d'applications multimédia [27].

5.1.1.2 C/C++

Apparu au début des années 90, le langage C++ est actuellement l'un des plus utilisés dans le monde, aussi bien pour les applications scientifiques que pour le développement des logiciels. En tant qu'héritier du langage C, le C++ est d'une grande efficacité. Mais il a en plus des fonctionnalités puissantes, comme par exemple la notion de classe, qui permet d'appliquer les techniques de la programmation-objet.

On trouve sur le marché un grand nombre de compilateurs C++ destinés à différents Microprocesseurs ou microcontrôleurs. Le langage C++ possède assez peu d'instructions, il fait par contre appel à des bibliothèques, fournies en plus ou moins grand nombre avec le compilateur [28].

5.1.2 Les outils utilisés

5.1.2.1 Eclipse

C'est une plateforme de développement écrite en Java, fruit du travail d'un consortium de grandes entreprises (IBM, Borland, Rational Rose, HP...). Il en résulte un IDE performant et open Source, qui a su trouver sa place comme l'un des environnements de développement Java les plus populaires. Elle intègre pour cela la prise en charge des outils comme Ant, SVN, JUnit...

Pour ce qui est de l'ergonomie, Eclipse n'a rien à envier à ses concurrents. Cette plateforme contient en effet toutes les fonctionnalités indispensables (création de projet, de template, refactoring, debuggage). Elle est également très aisée à prendre en main. Mais la grande force de cet IDE réside dans l'ouverture de son noyau qui permet l'ajout de très nombreux plugins. Il est par exemple possible d'intégrer des éditeurs XML, HTML, JSP, etc. ou encore de déployer ses applications vers le quasi totalité des serveurs du marché.

Eclipse est distribué sous la forme de bundles, qui contiennent un certain nombre de plugins préconfigurés pour une tache donnée comme par exemple eclipse-jee pour le développement Java EE ou encore eclipse-sdk pour le développement de plugins.

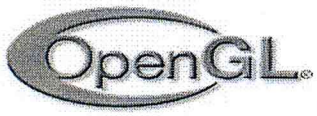
Enfin, Eclipse n'est pas limité au développement Java mais supporte aussi d'autres langages comme PHP (via PDT) et C/C++ (via CDT), etc. [3]

5.1.2.2 Visual Studio 2010 Express

Visual C++ 2010 Express fait partie de la gamme Visual Studio 2010 Express, une série d'outils offerte gratuitement à tout développeur Windows qui cherche à créer des applications personnalisées moyennant des réglages de base ou avancés. Visual C++ est un puissant langage de programmation conçu pour vous donner une perspective approfondie et détaillée lorsque vous concevez des applications natives Windows (COM+) ou des applications Windows gérées .NET Framework [4].

5.1.3 Les APIs

5.1.3.1 OpenGL



Open Graphic Library (ou OpenGL) est une API graphique multiplateforme open-source, de bas niveau, dédiée pour les applications générant des images 2D ou 3D. OpenGL est sortie en 1992 par la Sillicon Graphics Inc. (ou SGI) en 1992, et est actuellement géré par la société - à but non lucratif - Khronos Group. La version actuelle de l'API est le 4.2 (sortie le 8 août 2011).

OpenGL est conçu pour développer des applications graphiques portables, et elle favorise l'innovation et la simplicité d'utilisation en offrant aux développeurs de nombreuses fonctionnalités de visualisations. Etant libre, ouvert et portable, la bibliothèque graphique a pu se vendre dans le milieu scientifique et industrielle, et être utilisé dans des nombreuses applications artistiques ou de traitement d'images.

Du fait de ses performances, l'API OpenGL est aussi présent dans le marché des jeux-vidéo ludiques, en concurrence direct avec la bibliothèque graphique propriétaire DirectX. En ajout, OpenGL est de plus en plus utilisé dans les applications web (en particulier grâce à la bibliothèque WebGL), et est présent dans les systèmes embarqués (OpenGL ES, ou OpenGL for Embedded Systems), dont les téléphones portables (les systèmes Android et iPhone) et les PDA [5].

5.1.3.2 assimp

ASSIMP est une bibliothèque pour importer divers formats bien connus de modèles 3D («actifs») d'une manière uniforme. Assimp vise à fournir un pipeline complet de conversion actif pour une utilisation dans les moteurs de jeux / systèmes de rendu en temps réel de toute nature, mais n'est pas limité à cette audience [6].

5.1.3.3 JGraphT

JGraphT est une bibliothèque libre de graphique de Java qui fournit les objets et les algorithmes mathématiques de graphique-théorie. JGraphT soutient de divers types de graphiques comprenant:

- graphiques dirigés et non dirigés.
- graphiques avec bords pesés/non pondérés/marqués ou tous définis pour l'utilisateur.
- diverses options de multiplicité de bord, incluant : simple-graphiques, multi graphes, pseudo graphs.

- les graphiques non modifiable - permettre aux modules de fournir l'accès « inaltérable » aux graphiques internes.

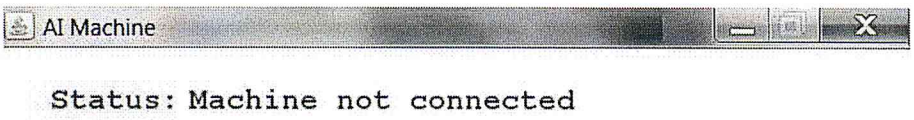
- les graphiques écoutable - permettre aux auditeurs externes de dépister des événements de modification.

- les graphiques de sous-graphes qui automatique-mettent à jour sous-graphe regarde sur d'autres graphiques [7].

5.1.4 Présentation de l'application

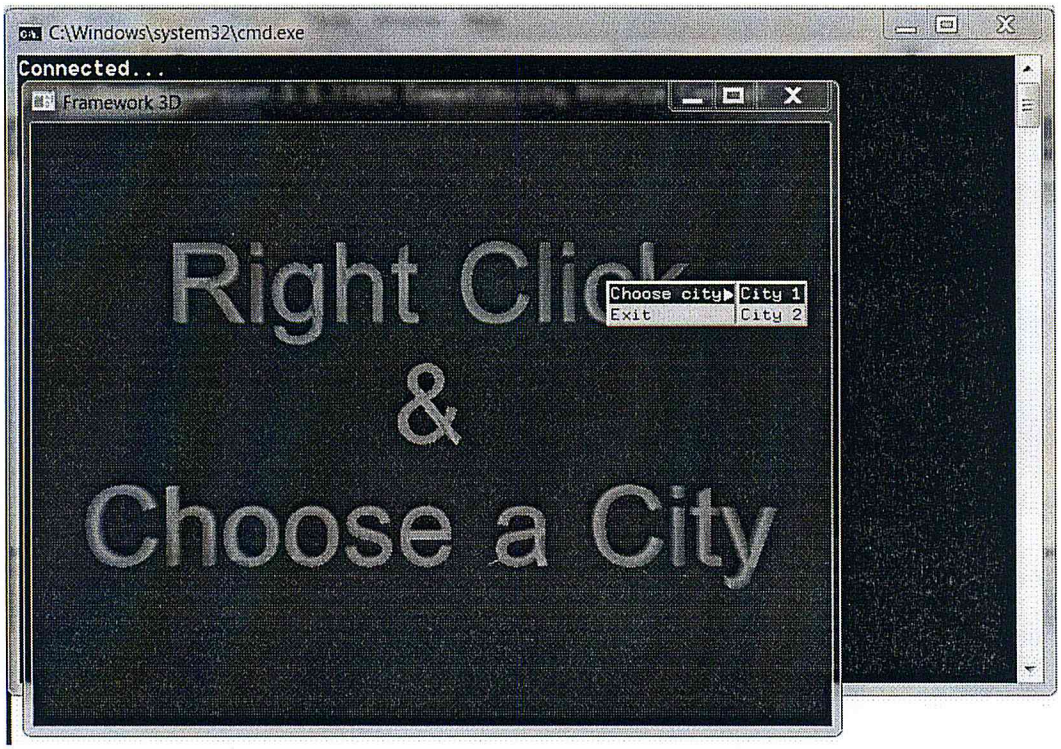
Dans cette partie, nous allons présenter les interfaces graphiques de notre framework.

Comme notre framework fonctionne en mode client/serveur, nous devons lancés le serveur qui permet la mise en place de la connexion avec le client et vérifier si le client est connecté ou non.



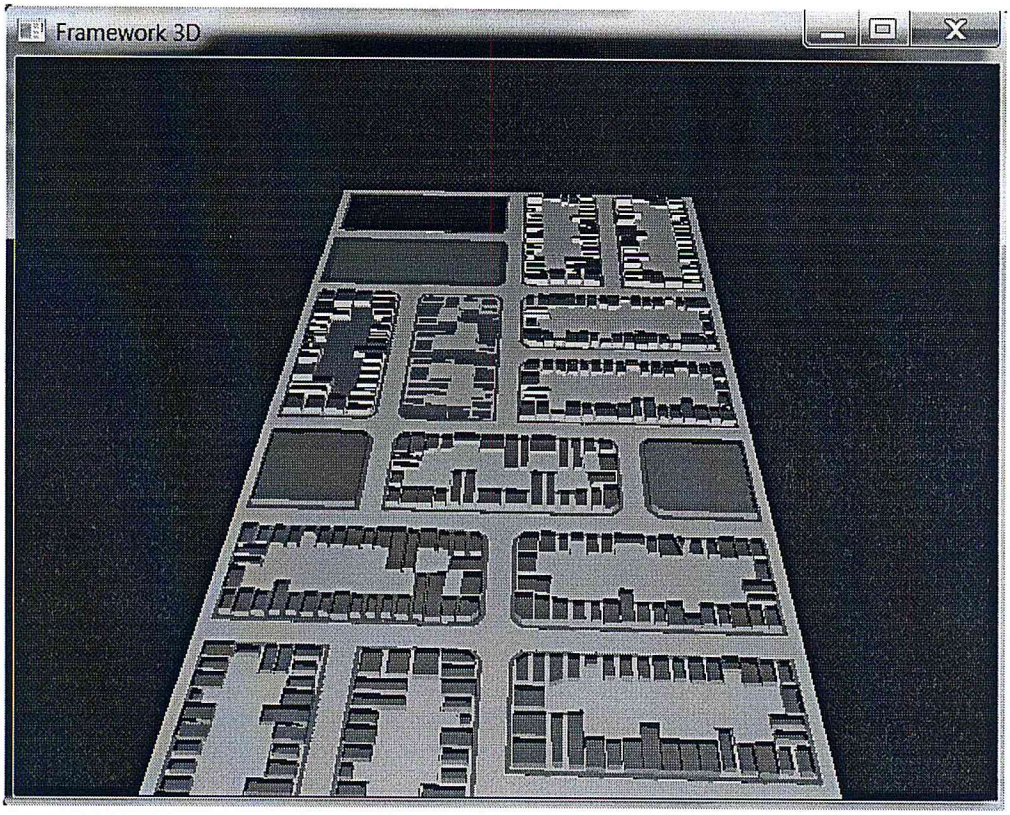
[Figure 5.1.4-1] L'interface du serveur avant la connexion du client

A ce niveau on lance le Client qui envoie une demande d'établissement de connexion au serveur et offre à l'utilisateur une Interface graphique, qu'elle lui permet de choisir la ville qu'il veut afficher sur la scène 3D.



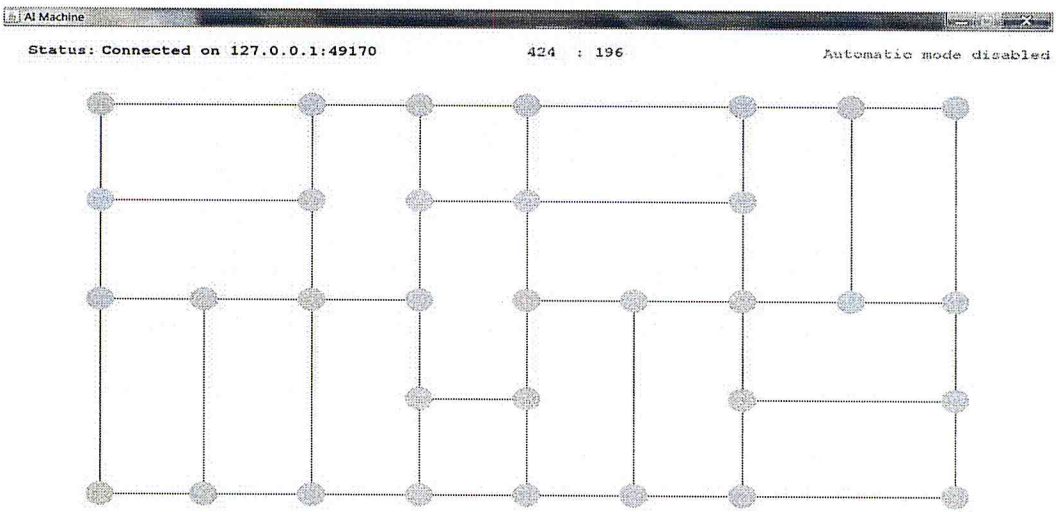
[Figure 5.1.4-2] Interface client

Le client envoie une requête au serveur et lui demandé l'adresse du fichier de la ville choisie, puis l'afficher en mode 3D.



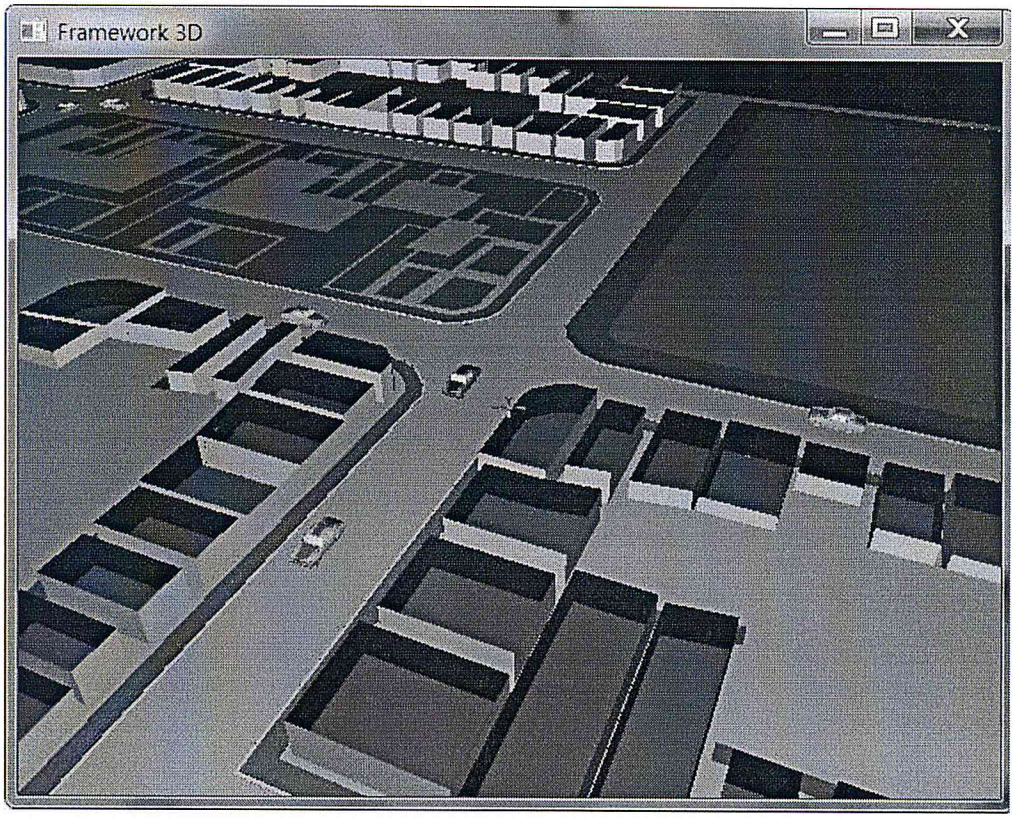
[Figure 5.1.4-3] Chargement de la ville demandé par client

Le serveur nous offre aussi un affichage de la ville en 2D.



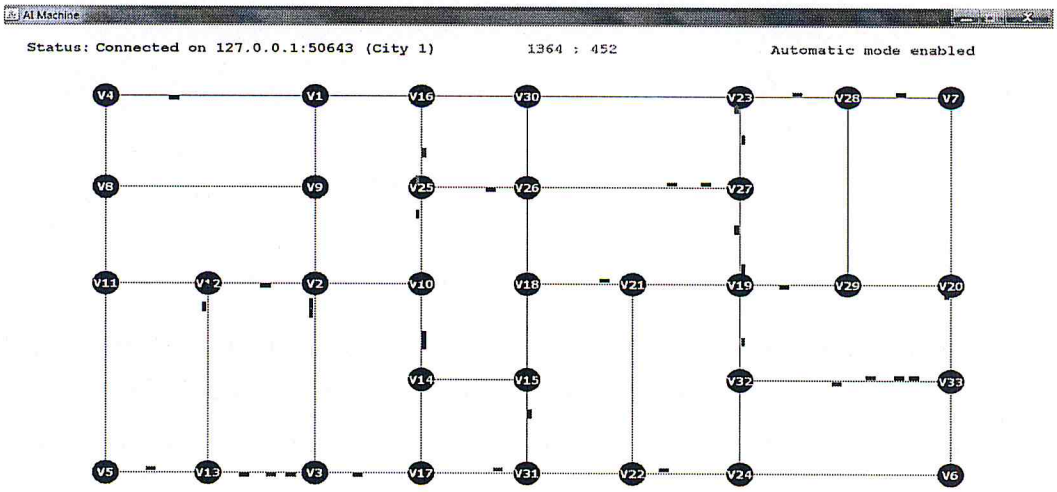
[Figure 5.1.4-4] Interface serveur après la connexion du client

L'utilisateur peut ajouter des véhicules dans la ville 3D et voir la simulation du trafic de véhicules.



[Figure 5.1.4-5] La Simulation d'un trafic de véhicules en 3D

Un autre utilisateur à la possibilité de voir la simulation du trafic de véhicules en mode 2D, c'est le développeur du comportement humain.



[Figure 5.1.4-6] La Simulation d'un trafic de véhicules en 3D

Conclusion

Dans ce chapitre, nous avons présenté l'implémentation de notre framework et les principales technologies que nous avons utilisées, et nous avons notamment présenté les composantes du framework. Enfin, en conclusion nous avons présenté les interfaces de notre framework.

Conclusion générale

Conclusion générale

La simulation du trafic de véhicules reste un problème d'actualité pour cela il était indispensable la mise en œuvre d'un Framework qui sera réutiliser, au long de ce mémoire nous avons identifié plusieurs problèmes liés à la simulation du trafic de véhicules, nous avons proposé des solutions liées à la détection de collision et la détection du plus court chemin.

Dans ce mémoire, nous avons créé un Framework qui assure la réutilisation de ces composants, il assure la portabilité, en utilisant le langage java, C/C++ et la bibliothèque OpenGL.

L'algorithme Dijkstra nous a permis de trouver une solution du plus court chemin pour la simulation du trafic de véhicules, au fil de la simulation Les composants Framework peuvent détecter une collision et traiter le cas de collision, chaque véhicule nécessite un thread.

Nous avons utilisé un service qui assure la communication entre le client et le serveur en utilisant les threads.

A l'issue de ce travail, nous croyons pouvoir affirmer la bonne simulation du trafic de véhicules.

Perspectives

Dans le cadre d'améliorer notre Framework, nous proposons les perspectives suivantes :

- Utiliser les designs patterns.
- Connecter une intelligence artificielle variée.
- Faciliter la maintenance.

Webographie :

[1] "*Wiki Mathématique*", http://fr.math.wikia.com/wiki/ Géométrie_(définition)

[2] "*Géométrie euclidienne*", http://www.techno-science.net?onglet=glossaire&definition=5502

[3] Gilbert VINCENT, "*Cours de mécanique du point*", 2007-2008, http://dlst.ujf-grenoble.fr/data/cours/documents/2008052208461915201770124013 8.pdf

[4] Claude SAINT-BLANQUET, www.sciences.univ-nantes.fr/sites/claude_saintblanquet/synophys/13mesol/13mesol.htm

[5] Bernard BAYLE, "*Introduction à la Robotique*", 2004, http://eavr.u-strasbg.fr/~bernard/education/master_gsb/slides_robotique_master_gsb.pdf

[6] Jean Louis COLLET, "*Les mécanismes de déformation d'un acier TWIP FeMnC : une étude par diffraction des rayons X*", 9 mars 2009, http://tel.archives-ouvertes.fr/docs/00/37/64/31/PDF/theseCollet.pdf

[7] Jean-Claude Charmet, "*Elasticité-Plasticité-Rupture*", http://www.pmmh.espci.fr/fr/Enseignement/Archives/MecaSol/Cours_Mecasol_0.pdf

[8] Cynthia A. Brewer & Barbara P. Buttenfield, "*Mastering map scale: balancing workloads using display and geometry change in multi-scale mapping*", 2008, http://www.spatial.cs.umn.edu/Courses/Fall11/8715/papers/msd-7.pdf

[10] Pierre-Yves Bischoff, Philippe Colantoni, "Programmation OpenGL", 2002-2003

[11] http://icwww.epfl.ch/~sam/prjinfoSV/tutorial-wxWidget/cours/openGL5/

[12] http://leri.univ-reims.fr/~bittar/cours/OpenGL/td04.html

[23] "*Recherche des plus courts chemins dans un graphe*", http://criwebpub.supelec.fr/Exercices/Dijkstra/Aide/dijkstra.html

[24] "*Separating Axis Theorem (SAT) Explanation*", http://www.sevenson.com.au/actionscript/sat/

[25] http://www.jmdoudoux.fr/java/dej/chap-frameworks.htm

[26] http://www.techno-science.net/?onglet=glossaire&definition=1471#_note-0

[27] http://www.jmdoudoux.fr/java/dej/chap-frameworks.htm

[28] http://www.techno-science.net/?onglet=glossaire&definition=1471#_note-0

Bibliographie :

[9] Sylvie Mesure, Patrick Savidan, "*Le dictionnaire des sciences humaines*", 2006

[10] Gérard Mégie, "*Réflexions sur ETHIQUE ET SCIENCES DU COMPORTEMENT HUMAIN*", 23 février 2007

[11] Lavin A, "*ESTUDIO DEL COMPORTAMIENTO DEL CONDUCTOR DE VEHICULOS MOTORIZADOS*", Sept. 1972

[12] "*Revue française d'automatique, d'informatique et de recherche opérationnelle. Recherche opérationnelle*", tome 10, N° V2 1976

[13] E. W. Dijkstra, "*A Note on Two Problems in Connexion with Graphs, Numerische Mathematik*", 1959

[14] A. Charbier, E. Danna and C. Le Pape, "*Coopération entre génération de colonnes sans cycle et recherche locale appliquée au routage de véhicules*", Charbier 2002

[15] R. Bellman, "*On a Routing Problem*", 1958

[16] Frédéric FAVIER, "*DÉTECTION DE COLLISIONS EN CONTEXTE AUTOMOBILE*", Oct. 2004

[17] Jérémie Dequidt, Laurent Grisoni, Philippe Meseure, Christophe Chaillou, "*Détection de collisions entre objets rigides convexes autonomes*", 2003

[18] CHUNG K., "*An Efficient Collision Detection Algorithm for Polytopes in Virtual Environment*", 1996

[19] VAN DEN BERGEN G., "*A Fast and Robust GJK Implementation for Collision Detection of Convex Objects*", 1998

[20] LIN M., CANNY J., "*A Fast Algorithm for Incremental Distance Calculation*", 1991

[21] GILBERT E., JOHNSON D., KEERTHI S., "*A Fast Procedure for Computing the Distance Between Complex Objects in Three-dimensional Space*", IEEE Journal of Robotics and Automation, vol. RA-4, 1988

[22] DOBKIN D., KIRKPATRICK D., "*Determining the Separation of Preprocessed Polyhedra : A Unified Approach*", ICALP, 1990, p. 400-413.