Apport des SIG et de la réalité virtuelle à la modélisation et la simulation du trafic urbain

Julien Richard

Université Paris-Est
École Doctorale Mathématiques et Sciences et Technologies de l'Information et de la Communication
Laboratoire en Sciences et technologies de l'information géographique (LaSTIG)

Apport des SIG et de la réalité virtuelle à la modélisation et la simulation du trafic urbain

Par Julien RICHARD

Thèse de doctorat de Sciences et Technologies de l'Information Géographique

Dirigée par Benoît DEFFONTAINES et François BOUILLE

Devant un jury composé de :

M. MEAR Yann	Rapporteur
Mme. PIROT Françoise	Examinateur
M. POLIDORI Laurent	Rapporteur
M. PRIOU Denis	Examinateur
M. RIBSTEIN Pierre	Examinateur
M. BOUILLE François	Directeur de thèse
M. DEFFONTAINES Benoît	Directeur de thèse

Apport des SIG et de la réalité virtuelle à la modélisation et la simulation du trafic urbain.
Tome 1

Julien Richard

Remerciements

Cet itinéraire scientifique a été possible grâce à quelques personnes sur lesquelles j'ai pu compter et même me reposer quelques fois. Tout d'abord, mes directeurs de thèse :

M. Benoît Deffontaines sans qui tout cela n'aurait pas abouti. Il m'a soutenu, supporté et aidé pendant ces années pour passer les virages les plus serrés et je lui en suis très reconnaissant.

M. François Bouillé, je suis heureux de l'avoir croisé sur la route de mes études. Il a été bien plus qu'un pilier pour moi. Il a su me guider tout au long de ce travail, me donnant des pistes et des conseils et je lui dis merci pour cela. Grâce à lui et à sa conviction scientifique j'ai pu terminer ce travail.

Je remercie par cette courte page le jury de cette thèse me permettant de présenter mes résultats.

Toute ma gratitude va également à l'école doctorale MSTIC pour m'avoir accueilli dans leurs dossiers dans des circonstances plus que difficiles. Vous m'avez littéralement sauvé !

Il y a eu ensuite des petites mains qui m'ont aidé, je pense bien sûr à Marie, fidèle amie depuis quelques années maintenant. Elle m'a grandement soutenu, tant scientifiquement que moralement, merci infiniment.

Il y a aussi ma mère, relectrice assidue de mes écrits. La cartographie n'a désormais plus de secrets pour elle !

Pour finir, ma femme, Agathe, qui, en plus de m'avoir épaulé sur mon travail de thèse, a su me donner du courage durant ces années. Elle a toujours été présente pour moi et je l'aime pour ça.

A celui ou celle que je ne connais pas encore...

Notice

Afin de mieux appréhender ce document, voici quelques indications :

— *Un glossaire et une liste d'acronymes se trouvent en fin de document. Un lien hypertexte existe avec le glossaire et la première utilisation du mot.*

— *Si la lecture est faite sur pdf, de nombreux liens hypertextes permettent de circuler facilement entre les titres et les chapitres, entre les références et les figures ou à la bibliographie par exemple.*

— *Un CD contenant une démonstration vidéo est disponible avec ce manuscrit.*

Table des matières

Résumé

La cartographie ainsi que les sciences qui s'y rattachent, sont en constante évolution et s'adaptent aux nouvelles technologies et les domaines d'application sont de plus en plus variés. Nous souhaitons illustrer, dans ce manuscrit, l'application des systèmes d'informations géographiques à la simulation du trafic urbain en 4D grâce aux nouvelles technologies telles que les casques de réalité virtuelle. Le flux routier se traduit en équations via des simulations discrètes (voiture par voiture) et continues (assimilées à l'écoulement d'un fluide).

Dans une première partie, nous allons étudier l'historique de la cartographie, plus particulièrement la représentation de la ville au cours du temps. La gestion de trafic urbain est un élément crucial pour les urbanistes et sa représentation a évolué tant par l'utilisation d'outils de plus en plus précis, que par les enjeux actuels. L'urbanisation croissante nous conduit à être de plus en plus prévoyants sur les flux urbains. Il n'est pas seulement question de réseaux routiers, ce problème d'urbanisation influence d'autres réseaux tels que les systèmes d'assainissements qui sont sous dimensionnés face à l'augmentation de foyers et de surfaces imperméables. Il est en est de même pour les réseaux d'eau potable qui doivent être sans cesse renouvelés pour subvenir aux besoins des habitants, et plus globalement avec tous les réseaux enterrés. [1].

Nous étudierons aussi dans cette partie la modélisation du réseau routier via les graphes et les hypergraphes, cela nous permettra d'optimiser le code. En effet, la modélisation choisie, développée par M. Bouillé, la représentation HBDS, se rapproche de l'écriture de code en orienté objet et permet de bien structurer un réseau.

Dans la partie suivante, nous décrirons les critères de développement en passant par le choix des données sources et des langages informatiques. Celui de la donnée source est très important pour une simulation qui s'approche au plus près de la réalité. Faire des simulations sur diverses villes du monde, sans se limiter à la France, est un des buts de ce travail. Nous ferons donc une analyse des données disponibles afin de trouver les meilleures informations pour alimenter les simulations.

Ensuite, nous exposerons les méthodes et les réalisations que nous avons mises en place pour cette étude. Dans cette partie, on trouvera l'organisation du code ainsi que les différents outils de géomatique permettant la simulation de trafic en ville. Nous avons créé de nombreux algorithmes avant de développer, pour optimiser le temps de conception et renforcer le modèle créé. De plus, au cours de cette partie, nous nous intéresserons à l'apport d'un outil d'aide à la décision dans ce contexte via la mise en place d'outils :

— De simulation informatique,
— De Système Expert avec la création d'un module d'Intelligence Artificielle.

Pour finir, les résultats visuels et les perspectives de ce travail seront abordés. Nous allons décrire l'interface homme-machine qui se devait d'être la plus intuitive possible. En effet, les interfaces des SIG propriétaires sont souvent très complexes, ils doivent être accessibles pour

proposer des outils variés selon les domaines d'intérêts des utilisateurs. Dans le cadre de notre thématique, nous pouvons limiter les interactions avec l'utilisateur et nous concentrer sur des usages plus ciblés sur la simulation. Nous verrons aussi l'utilisation des principes d'immersion visuelle, comme la stéréoscopie, principe encore sous exploité dans les SIG actuels [2].

Abstract

Mapping and spatial data visualization are increasingly used to communicate to a wide audience, while providing specific expertise. We want to illustrate the application of geographical information systems to 4D urban traffic simulation thanks to new technologies such as virtual reality headsets. Road flow can be described in equations by discrete simulation (car by car) and continuous simulation (as a fluid flow).

Firstly, we study the cartography history, more particularly the city representation over time. Urban traffic management is a critical piece for urban planners. Its representation has changed both with precise tools uses, and with current issues. An increase in urbanization leads us to be more and more farsighted of urban flows. It's not only a road networks question. These urbanization problems impact other networks as sanitation which are undersized dealing with population and surfaces damp-proofing increases. It's the same problem with the water supply which has to be replaced to cover the population needs, and more generally with all underground networks.

We also study in this part the road network model via graphs and hypergraphs to optimize the code. Indeed, the chosen model, developed by Mr. Bouillé, the HBDS representation, is close to the object oriented code writing and helps to well structure a network.

Afterwards we describe the development criterion through the raw data choice and the computer languages. Raw data choice is important to get the most realist simulation. The fact to make simulation all over the world, and not only in France, is one of the aims of this work. That's why we do a data analysis to find the best data to supply simulations.

Then we expose methods and achievements that we implement for this study. In this part, we present the code organization and the geomatic tools helping to the city traffic simulation. We build many algorithms before coding, to optimize the conception time and to strengthen the created model. Moreover we talk about the benefits of a decision support tool in this context via the implementation tools :

— Computer simulation,

— Expert System with Artificial Intelligence creation.

At least, visual results and perspectives are discussed. We describe the graphical user interface which had to be user-friendly. Indeed, owner user interfaces are often complicated. It has to be approachable to offer wide tools depending on users fields. As part of our thematic, we can limit interactions with the user and focus on targeted uses on simulation. We can also see the immersion view uses as the stereoscopy, technique underused in actual GIS.

Introduction

La modélisation du trafic urbain, en tant qu'outil d'aide à la décision de grande efficacité, ne peut se satisfaire de circuits simplistes. Elle doit être la plus exacte possible pour représenter le plus justement la réalité quelqu'en soit la complexité. La Géomatique et les Systèmes d'Information Géographique (SIG) constituent à la fois un ensemble de méthodes et une source abondante de données concrètes, à utiliser dans un contexte de réalité virtuelle / réalité améliorée. Il s'agit donc d'interfacer un certain nombre de domaines scientifiques, d'outils informatiques utilisés en géomatique, et de les appliquer à des données géoréférencées issues de sources SIG. Toutefois celles-ci ne fournissent généralement pas directement le lot de données nécessaires, et il convient de leur faire subir diverses transformations et raffinages. Un premier axe de travail consiste à inventorier ces diverses sources de données, sachant qu'elles ne sont généralement pas à la même échelle, n'ont pas la même orientation, et nécessitent des méthodes d'accès différentes.

La modélisation de la nature, des activités humaines et des nombreux réseaux s'entrecroisant, ne peut plus être assurée par un certain nombre d'outils obsolètes, mais encore utilisés dans certains projets, tels que l'arborescence ou la grille. Il en est de même du contexte 2D, dépassé depuis bientôt deux décades. La vraie 3D, avec perspective et vision stéréoscopique, est indispensable, tout comme la faculté pour l'usager de s'immerger au sein de l'univers ainsi créé, et de s'y déplacer aisément via des outils ergonomiques. Il faut donc faire appel aux méthodes classiques de la réalité virtuelle. Pour modéliser un réseau, quelqu'il soit, il faut partir des outils de la théorie des graphes, incluant les concepts de multigraphe et hypergraphe. Un réseau routier peut alors être modélisé par un multigraphe muni de valuations sur ses arcs et sur ses sommets. Il se plaque au sein d'un tissu urbain constitué entre autre, de bâtiments assimilables à des polyèdres plus ou moins complexes.

Le contexte actuel des SIG mis à disposition du grand public ou des spécialistes a vu le développement de logiciels aussi divers que Google Earth (pour le grand public) ou encore de logiciels nettement plus complexes chez les éditeurs de SIG (ESRI, Map-Info, INTERGRAPH, AUTODESK,...). Or, généralement, la donnée est essentiellement vectorielle et 2D, et il faut lui appliquer diverses méthodes de conversion. La base de données du bâti est en 2D avec des coordonnées x,y, mais elle est superposable à une autre, celle de la topographie, qui permet de disposer d'une coordonnée z, correspondant à la base des bâtiments. Encore faut-il disposer de la hauteur du bâti. On peut alors constituer un polyèdre (pas nécessairement convexe) par extrusion. Ces bâtis ont tous leurs toits plats, comme c'est le cas dans certaines régions de la planète où il ne pleut que très rarement. On peut disposer d'une série de modèles de toits adaptables et directement applicables sur les immeubles. Sur ces bâtis, il est également possible d'appliquer automatiquement des textures issues d'une base de données, puis divers éclairages variant avec le moment (saison, heure) et les conditions climatiques. Il s'agit donc là d'un deuxième axe de travail.

Disposant d'une ville 3D, pour l'instant immobile, il faut y faire passer le réseau routier (les autres réseaux peuvent d'ailleurs s'y insérer par la suite selon un processus assez similaire). Les routes sont définissables selon un multigraphe issu de données en mode vectoriel, et munies de données dites attributaires, ce qui correspond, sur le plan théorique, à multivaluer ce multigraphe. Parmi ces nombreuses valuations, on s'intéresse à la largeur de la route, ainsi qu'à son nombre de voies. Un problème complexe consiste à considérer la route non plus comme une suite de segments mais comme une bande continue possédant une surface, constituée de polygones 3D, parfois curvilignes, se raccordant parfaitement. On peut alors appliquer une ou plusieurs textures sur cette chaussée. La réalisation d'un tel réseau constitue un autre axe de travail.

L'étape suivante est la réalisation du trafic, à partir de mobiles qui sont des objets se déplaçant avec leurs caractéristiques propres, sur le réseau. Chaque trajectoire est constituée

par la combinaison de translations et de rotations, en contexte 3D, et il est indispensable d'appliquer l'emploi des quaternions. La simulation du trajet d'un mobile étant assurée, et représentant un travail de simulation continue, on doit alors vérifier la cohabitation de plusieurs véhicules (puis de nombreux) constituant autant de processus, ce qui nécessite la mise en oeuvre d'un échéancier et relève de la simulation discrète. L'interaction de la simulation discrète et continue est donc le quatrième axe de ce travail.

Il faut alors insérer divers mobiliers (feux de signalisation, par exemple), ainsi que des incidents et/ou accidents générés aléatoirement, venant perturber les flux du trafic. Tout ceci doit être géré selon des règles qu'il convient de formuler. Un moteur d'inférence doit donc être développé et doit interagir avec le moteur de simulation discrète et celui de simulation continue. Ce thème relevant du système expert est donc le cinquième axe de la thèse.

Enfin, et quoi qu'apparaissant en dernier, le mode de visualisation doit être pris en compte dès la conception initiale du travail, l'usager en immersion devant pouvoir se déplacer selon deux modes différents mais compatibles :

— soit à l'aide d'un véhicule circulant sur les routes,
— soit au-dessus du modèle, comme un hélicoptère, et en suivant éventuellement un flux de voitures ou une voiture particulière.

Il faut donc mettre en œuvre un quatrième moteur pilotant les trois précédents. Ce dernier axe de recherche doit offrir la possibilité de la stéréoscopie (non limitée à une simple vision anaglyphique).

Les données utilisées pour la création automatique de villes en 3D, ainsi que pour la modélisation de la circulation, sont issues de la base de données Open Street Map (OSM) pour les informations vectorielles et du projet Shuttle Radar Topography Mission (SRTM) pour les données raster d'élévation. La disponibilité mondiale et gratuite de ces données permet d'avoir autant de cas d'études que nous le souhaitons.

Première partie

Représenter la ville

Chapitre 1

Un peu d'histoire

A l'origine des Sciences de l'Information Géographique, les cartes apportent un regard neutre sur des problématiques techniques et variées. Et si, aujourd'hui, on les utilise sous diverses formes (papier ou numérique) ou pour diverses raisons (s'orienter, représenter un phénomène physique, visible ou non), la cartographie est une science dont l'histoire n'est pas linéaire.

1.1 La représentation cartographique à travers le temps et les civilisations

Cette histoire s'étale sur plusieurs siècles, et même millénaires puisqu'une des premières représentations topographiques a été retrouvée dans une grotte en Italie du Nord et daterait de 2000 ans avant JC. Une des plus anciennes cartes date du Vème siècle avant J.C., elle décrit la Mésopotamie, et met Babylone au centre de l'Univers. La civilisation grecque va ensuite révolutionner la cartographie :

— Pythagore et la sphéricité de la Terre,
— Mise en place d'un quadrillage pour déterminer les latitudes et longitudes,
— Erathostène et ses calculs de précision pour mesurer la circonférence de la Terre.

Par la suite, l'ouvrage de Claude Ptolémée (IIème siècle après J.C.) fera référence pendant plusieurs siècles en répertoriant tout ce que les Anciens savaient du monde à cette époque. Entre le IVème siècle et le XIIème siècle, le déclin du commerce maritime en Europe favorisera alors le développement de cartes symboliques et religieuses avec les diagrammes en T dans l'O centrés sur Jérusalem. A partir du XIIème siècle, de plus en plus de scientifiques s'intéressent aux cartes, ainsi Al Idrisi (XIIème siècle) compile les connaissances du monde de l'Islam pour le roi de Sicile.

Au XIIIème siècle, les portulans indiquent toutes les informations nécessaires à la navigation. A cette époque, le Traité de géographie de Ptolémée est traduit en latin (via l'arabe) et vers la fin du XIVème siècle les Européens redécouvrent cette réflexion et enrichissent alors les cartes du siècle suivant avec un monde nouveau : America. C'est à partir du XVIème siècle que les types de planisphères se multiplient et que le monde se dessine sous la forme actuelle : le Nord en haut et une Europe au centre. Vers la fin du XVIIème siècle et au cours du XVIIIème, les cartes se feront plus précises et les cartes thématiques (forestières, minéralogiques,...) se développeront. Au XIXème siècle, la carte se veut être le miroir de la réalité territoriale et c'est à partir de cette période que la carte devient un outil administratif tel que nous l'utilisons encore aujourd'hui.

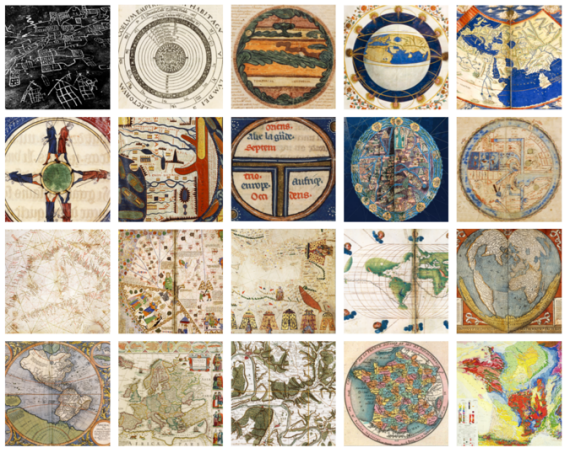


FIGURE 1 – Évolution des cartographies à travers le temps, Source : BnF

Toute cette évolution des cartes dépend des mathématiques. En effet, représenter le monde tridimensionnel sur un planisphère ne peut se faire sans déformation, sans projection. Certaines propriétés géométriques peuvent être conservées séparément et dans certaines combinaisons sur des projections cartographiques. Cela dit, les principales, telles que la superficie, la forme, la distance et l'orientation ne peuvent pas être conservées simultanément. Le choix du type de projection doit prendre en compte la perte d'une ou plusieurs de ces propriétés. Sachant cela, il est donc normal que plusieurs systèmes de projections aient été proposés depuis la début de l'histoire de la cartographie [3]. Voici quelques unes des projections créées depuis plus de deux millénaires (voir figure 2) :

Gnomonique : Du fait de sa découverte via la portée de l'ombre d'un gnomon (instrument d'astronomie permettant de connaître la hauteur du soleil) cette projection est attribuée à Thalès en 580 av J-C environ. C'est une projection azimutale utilisant le centre de la Terre comme point de perspective.

Conique équidistante : Cette projection, créée en l'année 100 et basée sur la 1ère projection de Ptolémée, s'appuie sur un ou deux parallèles de référence. Tous les parallèles concentriques sont équidistants le long des méridiens.

Mercator : La projection de Mercator est une projection conforme, en d'autres termes, elle conserve les angles. Elle a été crée par Geradus Mercator en 1569. Elle est souvent utilisée comme standard pour les cartes mondiales de par sa grande utilisation dans les voyages maritimes. Cependant, elle n'est pas valable pour la conservation des aires, en effet, le Groenland parait aussi grand que l'Afrique sur ce type de projection alors que ce pays est 14 fois plus petit.

Cassini : C'est une projection cylindrique transverse qui conserve l'échelle le long du méridien central et de toutes les lignes qui lui sont parallèles. Elle est adaptée à la cartographie des zones orientées du nord au sud sur de grandes échelles. Elle a été décrite par César-François Cassini de Thury en 1745.

Transverse Universelle de Mercator (UTM) : Cette projection, créée en 1822 par Carl Friedrich Gauss et Johan Heinrich Louis Krüger, corrige partiellement les problèmes liés aux aires de la projection de Mercator. La projection se fait à partir du centre de la Terre sur un cylindre tangent à l'équateur. Dans cette projection les parallèles et les méridiens se coupent à angle droit. La Terre est divisée en 60 bandes de largeur constante du nord au sud numérotées de 1 à 60, elles mêmes divisées en 20 bandes depuis le parallèle 80 ° sud jusqu'au parallèle 84 ° nord, désignées par une lettre. Il existe donc 600 zones UTM.

Goode : La projection décrite par John Paul Goode en 1923 est utilisée de nos jours pour créer les planisphères car elle permet la conservation des aires. Elle est une combinaison de deux projections, la projection sinusoïdale et la projection de Mollweide [4].

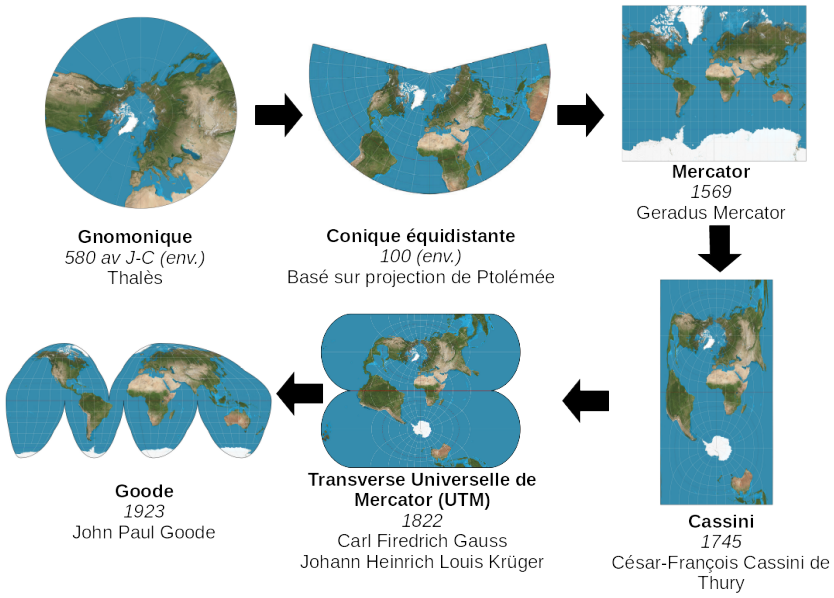


FIGURE 2 – Évolution des systèmes de projection

Comme nous l'avons remarqué, de nombreuses projections existent et aucune n'est vraie, du fait que le passage à la 3D à une carte 2D ne peut se faire sans déformations. Le tableau 1 recense ces différences sur les projections citées précédemment.

	Forme	**Surface**	**Direction**	**Distance**
Gnomonique	Distorsion accentuée depuis le centre	Distorsion accentuée plus l'on s'éloigne du centre	Exacte à partir du centre	Aucune échelle
Conique équidistante	Formes fidèles au niveau des parallèles de référence	Distorsion constante au niveau de chaque parallèle	Varie localement le long des parallèles	Vraie pour les méridiens et les parallèles de référence
Mercator	Conforme	Surface de plus en plus déformée vers les régions polaires	Toute ligne droite dessinée sur cette projection représente un relèvement au compas réel	Vraie le long de l'équateur ou le long des latitudes sécantes
Cassini	Distorsion nulle au niveau du méridien central mais augmente plus l'on s'éloigne de celui-ci	Distorsion nulle au niveau du méridien central mais augmente plus l'on s'éloigne de celui-ci	Déformée	Distorsion plus l'on s'éloigne du méridien mais échelle respectée le long du méridien central
Transverse Universelle de Mercator (UTM)	Conforme	Distorsion augmente plus l'on s'éloigne du centre	Angles fidèles en tout point	Échelle exacte au niveau du méridien central si le facteur d'échelle est de 1
Goode	Aucune distorsion	Surfaces exactes	Angles correctes le long des méridiens centraux et le long de l'équateur mais faux ailleurs	Échelle précise le long des parallèles dans la partie sinusoïdale et des méridiens centraux des lobes de la projection

Tableau 1 – Différences géométriques entre les projections. Source : ArcMap

1.2 La géomatique, outil de création cartographique / usage des données cartographiques via la géomatique

Avec les avancées technologiques, l'utilisation de la carte et des données géographiques se développent. De plus l'évolution de l'informatique, d'internet et des technologies d'obtention de données précises révolutionnent la cartographie. Ces informations obtenues sont qualifiées de géographiques car elles disposent de coordonnées dans l'espace et sont généralement représentées sur la carte par des formes géométriques simplifiées (points, lignes ou polygones). On peut les regrouper en trois grandes catégories :

— les Modèles Numériques de Terrain,
— les données "vectorielles",
— les nuages de points.

Le traitement de ces données est ainsi un enjeu essentiel du XXIème siècle. C'est ce que propose la Science de l'Information Géographique : la géomatique, via le déploiement des Systèmes d'Informations Géographiques (SIG).

Ces SIG doivent permettre la génération de nouvelles informations géographiques à partir de données connues et localisées. Ils se situent alors à l'interface entre un outil de communication et un outil d'aide à la décision et permettent de représenter de nombreux phénomènes comme la surveillance des traits de côtes ou encore la modélisation des flux (routiers par exemple).

La modélisation des flux urbains est souvent traitée dans les outils de géomatique. Par exemple le projet Terra Dynamica[1] du LIP6, Unité Mixte de Recherche de l'Université Pierre et Marie Curie et du Centre National de la Recherche Scientifique et de Thalès, a pour but de modéliser les flux des piétons et des voitures dans une ville virtuelle (voir figure 3). Il prend en compte les interactions entre les piétons et les voitures mais aussi les mouvements de foules, les passages de bus,... le tout géré par une intelligence artificielle.

FIGURE 3 – Captures du logiciel Terra Dynamica. Source : LIP6

D'autres modèles de trafic urbain existent tels que Symuvia [5], développé par le LICIT depuis 2005, ou encore TransNetSim [6] avec une approche multi-échelle du trafic urbain. Tous ces modèles utilisent des données SIG de différents types tels que décrits ci-dessous.

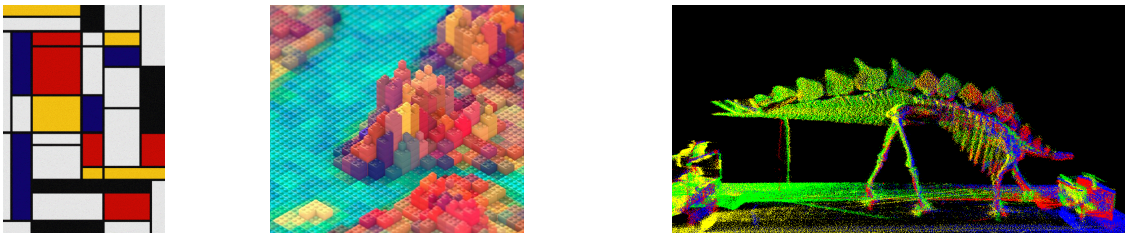
1.3 Apport de la 3D dans la représentation cartographique

La visualisation 2D atteint ses limites pour un certain nombre de représentations (maritime, souterraine ou aménagement de la ville). Pour cela, il est nécessaire de travailler à partir de trois grandes catégories de données :

1. http://www-poleia.lip6.fr/ corruble/TerraDynamica/accueil.html

— 3D : Coordonnées X, Y et Z,
— 2,5D : Coordonnées X et Y ainsi qu'une information de hauteur (en mètres ou en nombre d'étages par exemple),
— 2D : Coordonnées X et Y qui, à partir d'autres informations 3D, peuvent être valorisées pour l'affichage tridimensionnel de la scène (par exemple un linéaire de route en 2D à compléter par un Modèle Numérique de Terrain, par définition 3D).

Ces données, avec des précisions qui leurs sont propres, permettent de visualiser et de simuler diverses domaines, un exemple pour chaque type de donnée est illustré dans la figure 4.



(a) (b) (c)

FIGURE 4 – Différentes catégories de données : (a) Le tableau de Mondrian est un exemple parmi d'autres de données 2D, chaque polygone peut être décrit par des coordonnées X et Y. (b) On reconnaît New York en brique Légo®. Les bâtiments sont extrudés selon une hauteur, ceci est une donnée 2,5D. (c) Un squelette de stégosaure scanné par Lidar. Chaque point correspond à des coordonnées X, Y et Z.

La combinaison de ces types de données permet de compléter une visualisation et/ou d'affiner un modèle 3D. La 3D est de plus en plus utilisée, avec en tête dans le domaine de la géomatique, l'application Google Earth qui permet à un utilisateur de visualiser en 3D la Terre ainsi que certaines planètes du système solaire depuis peu. D'autres SIG 3D existent sans avoir l'utilité que l'on a décrit jusque là : les jeux vidéos. De nombreux jeux vidéos utilisent des données réelles pour contextualiser l'action. Les jeux de la franchise Assassin's Creed sont représentatifs de cela, chaque volet correspondant à une région du globe et à une époque. Par exemple Paris à l'époque de la révolution a été recréé, avec ses rues et ses bâtiments les plus significatifs. Il en a été de même pour Londres à l'époque industrielle (voir figure 5).

(a) (b)

FIGURE 5 – Capture de jeux vidéo : (a) La ville de Paris a été modélisée en 3D pour Assassin's Creed Unity avec une gestion du flux des piétons. Il en a été de même avec Londres (b) avec l'ajout de la gestion des véhicules dans Assassin's Creed Syndicate

Chapitre 2

La ville en 3D - Etat de l'art

2.1 Les attraits de la modélisation 3D

La modélisation a souvent été bidimensionnelle pour beaucoup de domaines et l'est encore de nos jours. Prenons la plus connue d'entre toutes, la météo. Elle est présentée et interprétée en 2D pour la vulgarisation des informations alors que les données existantes sont des informations 3D. En effet, les circulations d'air ou encore la forme des nuages sont obtenues en 3D et leurs modélisations sont également réalisées suivant les trois axes. Cependant, la représentation telle qu'on la connaît aujourd'hui, est une carte en 2D "vue du dessus". Il en est de même pour le trafic urbain, où les cartes illustrant les zones encombrées sont en 2D avec un code couleur bien connu : rouge pour les bouchons, orange pour les ralentissements et vert pour les zones fluides (voir figures 6).

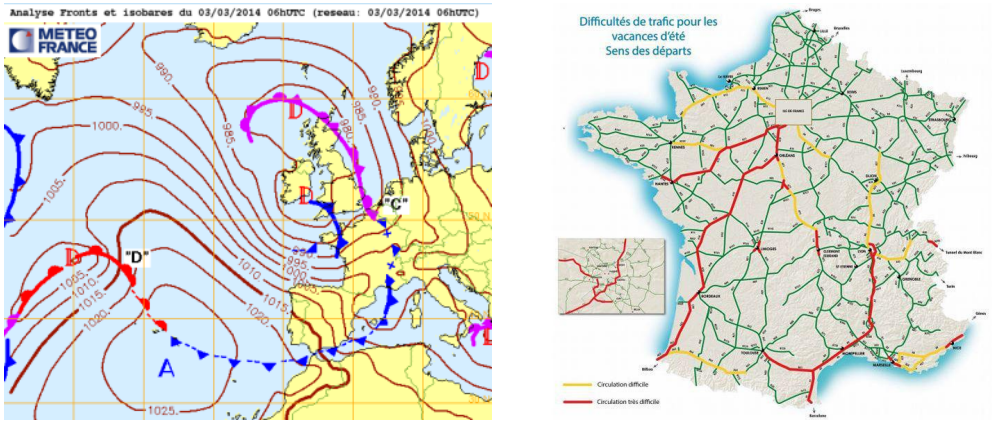


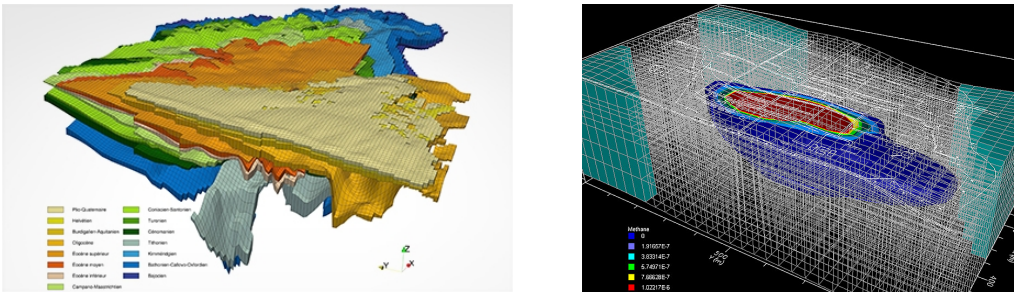
FIGURE 6 – Vues 2D d'information sur la météo et le trafic routier, Sources : Météo France et Bison Futé

Ce type de carte 2D est idéale pour le passage d'informations rapides, les données sont peu importantes et très claires. Lorsque l'on souhaite approfondir la réflexion et la compréhension d'un phénomène, il peut alors être utile de rajouter une troisième dimension à la modélisation.

La géologie est un domaine où l'intérêt de la 3D est intuitif dans les modélisations. Prenons en exemple les écoulements hydrologiques. Ces écoulements ne peuvent être appréhendés dans leur globalité sans la 3D. Le logiciel MARTHE du BRGM[1] permet de visualiser ces phéno-

1. http://www.brgm.fr/

mènes en 3D (voir figure 7). Les données sont plus facilement compréhensibles avec ce type de visualisation, il en est de même avec le modèle MODFLOW[2] de l'USGS, United States Geological Survey, permettant de simuler et de visualiser de nombreux fluides tels que les poches de méthane visibles sur la figure 7.



(a) (b)

FIGURE 7 – (a) Vue 3D du logiciel Marthe, Source : BRGM. (b) Vue 3D du logiciel MODFLOW, Source : USGS

Ces vues 3D nous permettent de nous rendre compte de l'utilité de la 3D pour explorer les résultats scientifiques. Nous allons recenser les principaux logiciels dédiés à la reconstruction de ville en 3D dans la partie suivante.

2.2 Les logiciels existants

Autonomes ou modules complémentaires, les logiciels du marché s'appliquent à divers domaines. Nous allons en décrire certains ayant comme attrait principal, la modélisation de villes virtuelles ou non en 3D.

On retrouve ainsi :

LandSim 3D : La société Bionatics avec son logiciel LandSim 3D. Cette société propose à tout type de client de créer des paysages 3D à partir de données géospatiales (images, terrains, SIG,...) couplé avec des objets 3D ou des dessins industriels (CAO). Les données ne sont pas pré-calculées pour une transformation en polygones mais sont calculées en temps réel. Le logiciel prend aussi en compte la dimension temporelle, permettant de suivre l'évolution de l'environnement. De nombreuse villes l'ont utilisé pour modéliser leur agglomération telles que Bordeaux, Chartres ou Marseille. L'intégration de la végétation en fait un outil majeur pour les régions naturelles comme pour Mayotte qui a modélisé plus de 400 km^2 en temps réel.

FIGURE 8 – Exemple de rendu du logiciel LandSim3D, Source : Bionatics

2. https://water.usgs.gov

VirtualGeo : La solution de DIGINEXT est VirtualGeo. Elle permet de survoler via la souris ou des joysticks des paysages 3D. Ils sont constitués de données topographiques, d'orthophotographies, de données vectorielles ou même d'objets 3D. Sa force est dans la fusion de toutes ces données et dans leur optimisation pour une meilleure compréhension. Son utilisation peut être diverse, allant de la planification militaire à la gestion de phénomènes naturels en passant par des projets d'urbanismes. Les données peuvent être exportées et partagées en ligne en format image dans une version allégée tandis que des vidéos peuvent être réalisées dans une version professionnelle.

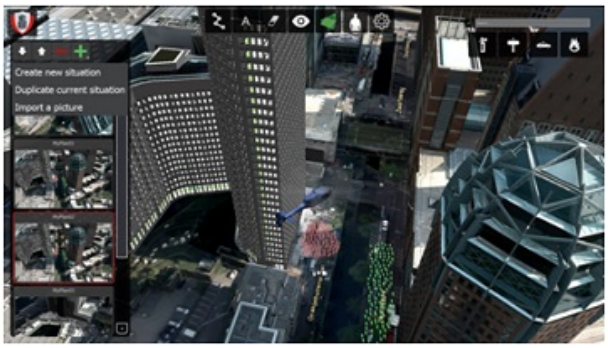


FIGURE 9 – Exemple de rendu du logiciel VirtualGeo, Source : Virtual-Geo

SpacEyes3D : SpacEyes3D est une suite logicielle de la société GEOIMAGE. Elle est composée de nombreux outils permettant de visualiser, construire et de stocker des informations 3D. De plus un SDK est disponible pour une intégration des fonctionnalités de SpacEyes3D dans une application tiers. SpacEyes Viewer permet, en plus de visualiser des modèles pré-construits, d'analyser des données provenant de MNT ou de faire des recherches multicritères. SpacEyes Builder permet de construire des modèle 3D d'environnement à l'aide de multiples sources de données (SIG, BDD, WMS, ...). Toutes les fonctionnalités de bases d'un SIG sont présentes (requêtes SQL, édition vecteurs, filtres, recherches, ...). SpacEyes Server permet de mettre à disposition des travaux réalisés par SpacEyes Builder avec un système de streaming permettant d'utiliser des machines multicoeurs pour l'affichage de modèles complexes.

FIGURE 10 – Exemple de rendu du logiciel SpacEyes, Source : SpacEyes

ELYX3D : 1Spatial, spécialisé dans la gestion des données géospatiales, propose la solution ELYX 3D. Une version dédiée à la visualisation est disponible et peut être alimentée avec des données 3D (Collada, 3DS, CityGML). Sa version complète permet de générer des terrains ou de créer des extrusions de données vectorielles (cas des bâtiments). Il permet également de construire des maquettes 3D afin de mieux appréhender les problèmes en milieu urbain, domaine cible de la société 1Spatial.

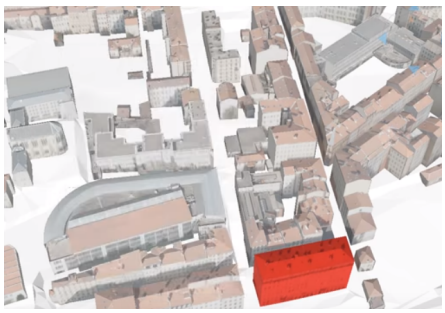


FIGURE 11 – Exemple de rendu du logiciel ELYX3D, Source : 1Spatial

LOD1 3D : Siradel, filiale d'Engie, propose une suite logicielle basée sur la modélisation 3D d'environnement. L'outil LOD1 3D permet de générer des villes avec un degré de réalisme à la demande du client, allant d'un modèle simpliste à une reconstruction virtuelle d'une ville. Les modèles sont alimentés par les données standards (MNT, données vectorielles, CityGML, ...). De nombreuses villes sont déjà générées telles que Rennes, Nîmes ou San Francisco. Des réalisations sur demande sont également possibles.

FIGURE 12 – Exemple de rendu du logiciel LOD1 3D, Source : Siradel

VirtuelCity : VirtuelCity est le spécialiste français des modélisations 3D géoréférencées. Elles mettent à disposition de données payantes comprenant un MNT, des orthophotographies, des bâtiments texturés avec extrusion à la bonne hauteur. Les livraisons sont faites par dalle dans des formats génériques (3DS, CityGML, ...) et peuvent être réalisées à la demande.

FIGURE 13 – Exemple de rendu du logiciel VirtuelCity, Source : SIMVIR

3D for Geoconcept : La société GeoConcept, entreprise dédiée à la conception et à l'édition de technologies d'optimisation géographique, a un module pour générer des modèles 3D à partir de MNT, des données vectorielles ou 3D. Ces modèles peuvent être exportés vers Google Sketchup. Ils sont complets et permettent de créer plusieurs types de simu-

lation de navigation voire même de créer des profils topographiques le long d'un trait de coupe.

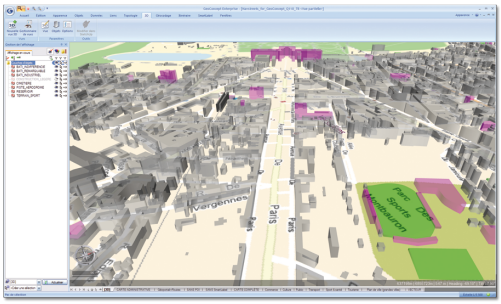


FIGURE 14 – Exemple de rendu du logiciel de la société GeoConcept, Source : GeoConcept

RhinoCity : L'entreprise RhinoTerrain, fondée en 2008, est spécialisée dans la création d'outils dédiés à la modélisation 3D. Un de ceux-ci est RhinoCity, qui est l'addition de RhinoTerrain, dédié à la topographie, et de bâtiments 3D importés en format CityGML.

FIGURE 15 – Exemple de rendu du logiciel RhinoCity, Source : RhinoTerrain

Street Factory : La société Airbus propose des représentations 3D acquises par imageries obliques (terrestre ou aérienne). Les calculs pour traiter ces images sont directement réalisés sur les GPU afin d'optimiser le temps de traitement. Les résultats peuvent être exportés directement vers OpenSceneGraph, logiciel open source dédié aux données 3D.

FIGURE 16 – Exemple de rendu du logiciel StreetFactory, Source : Airbus

ESRI : ESRI propose également deux projets :

CityEngine : CityEngine est une solution logicielle du groupe ESRI, leader mondial du SIG. Elle permet de travailler à partir de données 2D et/ou 3D (de formats divers : KML, SHP,...) et permet des exports vers OBJ, 3DS, DXF, KML,... selon la version basique ou avancée du logiciel. Des outils dédiés à la génération pseudo-aléatoire de villes sont proposés aux utilisateurs tels que l'emplacement de corniches

sur certains types de bâtiments, ou alors sur la création de balcons sur certains types de façades. Nous pouvons aussi noter que ce logiciel permet de créer des modèles 3D non géoréférencés.

ArcGIS 10 : Le logiciel le plus connu d'ESRI est ArcGIS. Ses dernières versions intègrent une visualisation 3D. Il permet de visualiser de la donnée mais aussi d'en créer en modifiant à la volée l'extrusion de bâtiments ou la topographie d'un simple clic. Des analyses peuvent être faites sur des données d'import et bien sûr un export est possible afin de partager les modèles 3D créés au préalable.

(a) (b)

FIGURE 17 – (a) Exemple de rendu du logiciel CityEngine. (b) Exemple de rendu du logiciel ArcGIS 10, Source : ESRI

ExperienCity : ExperienCity est une solution logicielle de Dassault Systèmes. Elle est axée sur la simulation de scénarios dans des environnements 3D au préalable créés via des données d'acquisitions propriétaires ou alors par des données sources.

FIGURE 18 – Exemple de rendu du logiciel ExperienCity, Source : Dassault Systèmes

Chapitre 3

Structuration des axes routiers/Topologie

Modéliser l'organisation des réseaux routiers nécessite de s'intéresser à leur structuration. Les données sources que nous décrirons en détail dans le chapitre 9 sont bien spécifiques et leur utilisation brute n'est pas envisageable pour une simulation de trafic routier. Nous allons nous baser sur les principes de la topologie et des graphes qui en sont tirés pour bâtir notre base de données dédiée à la simulation.

3.1 Définition

Les premiers travaux sur la topologie débutent avec Leonhard Euler [7] [8] et le problème des sept ponts de Königsberg. Cette étude permet de trouver une méthode définissant l'existence, ou non, d'un chemin à partir d'un point choisi et passant par chacun des ponts de la ville pour revenir à son point de départ sans repasser sur un de ceux déjà empruntés. La configuration de la ville bien particulière ne permet pas de définir un chemin mais Leonhard Euler a mis au point la théorie des graphes. Il a formulé qu'un graphe ne peut être parcouru en entier avec un seul passage par arc lorsqu'il est constitué d'un nombre pair d'arcs.

3.2 La structure du réseau

Comme nous l'avons décrit précédemment, un territoire peut être représenté par des formes géométriques simples. Il en est de même pour le réseau routier qui est constitué de :

— panneaux routiers, feux de signalisation, réverbères,... identifiables comme ponctuels,
— voies de circulation représentées par des linéaires,
— bornes de péages, stations essence par des polygones.

(a) (b) (c)

FIGURE 19 – Éléments constitutifs d'un réseau routier : (a) : Cédez le passage, assimilé à un point, (b) Route, assimilé à une ligne et (c) un péage d'autoroute assimilé à un polygone.

Ainsi, afin de décrire un/des réseau(x) routier(s), et en partant de cette description, il est possible de le(s) schématiser via des graphes. En effet, la carte des routes peut être vue comme étant une succession d'arcs (par exemple l'autoroute A34, la départementale D8051A,...), chaque arc étant délimité par deux sommets (intersections, carrefours, ronds points,...) et pouvant être découpé par divers Points Annexes (feux de circulation par exemple) (qui découpe cet arc en segments). Sur ces segments, il est également possible de retrouver un certain nombre de Points Complémentaires (panneaux d'affichages, bornes d'appels d'urgence,...) qui les subdivisent alors en tronçons.

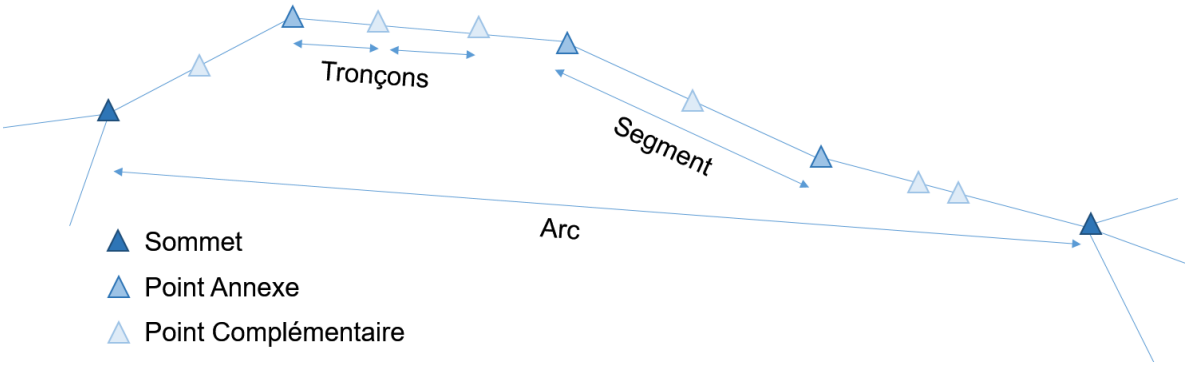


FIGURE 20 – Structure générale d'un graphe

Il est évident que pour chaque type de véhicules, ces éléments peuvent être différents. En effet, certains tronçons seront découpés différemment selon le véhicule. Le tableau 2 résume les différences de découpages par type de véhicules.

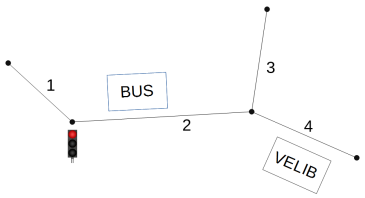
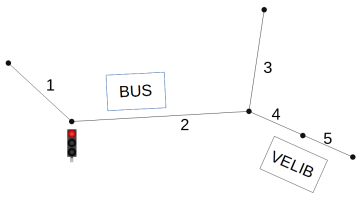
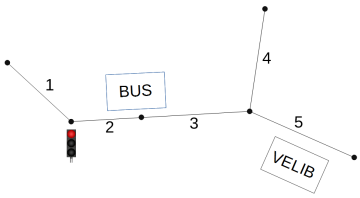
Véhicule	Découpage selon	Illustration
Voiture	Feux de signalisation, intersections, cédez le passage, stop, passage piétons, changements de directions	
Vélo type vélib	Même mobilier + borne vélib + parc à vélos	
Bus	Même mobilier + arrêt de bus	

Tableau 2 – Différence de découpages selon le type de véhicules

3.3 La modélisation HBDS

Une fois cette structure de graphe identifiée, il est alors intéressant de travailler sur la modélisation informatique de ces éléments. Pour illustrer nos choix de structuration de données nous utiliserons le modèle Hypergraph Based Data Structure (HBDS) introduit par les travaux de M. Bouillé [9].

La méthode HBDS est très utile pour représenter les hypergraphes et est particulièrement adaptée à une conversion dans un langage informatique orienté objet. Tout d'abord, le vocabulaire est très proche et ensuite, la structuration des éléments les uns avec les autres est similaire.

On retrouve principalement les classes, les attributs et les liens (voir figure 21). Les classes intègrent, ou non, des attributs et interagissent entre elles via des liens. Parallèlement, des hyperclasses peuvent être créées pour regrouper des classes ayant des attributs en commun.

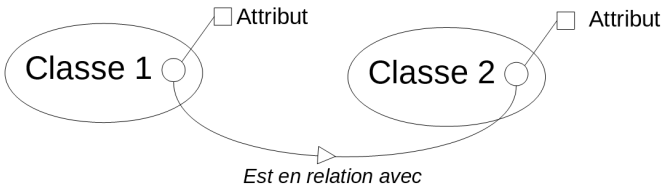


FIGURE 21 – Principaux éléments du la modélisation HBDS

Cette technique de structuration a pour avantage de mettre en évidence toutes les structures associées à ces réseaux routiers tout en évitant les redondances et en y intégrant les relations existantes entre divers objets/classes/hyperclasses.

Ainsi un graphe, quelle que soit sa finalité, peut être représenté selon la structure illustrée sur la figure suivante.

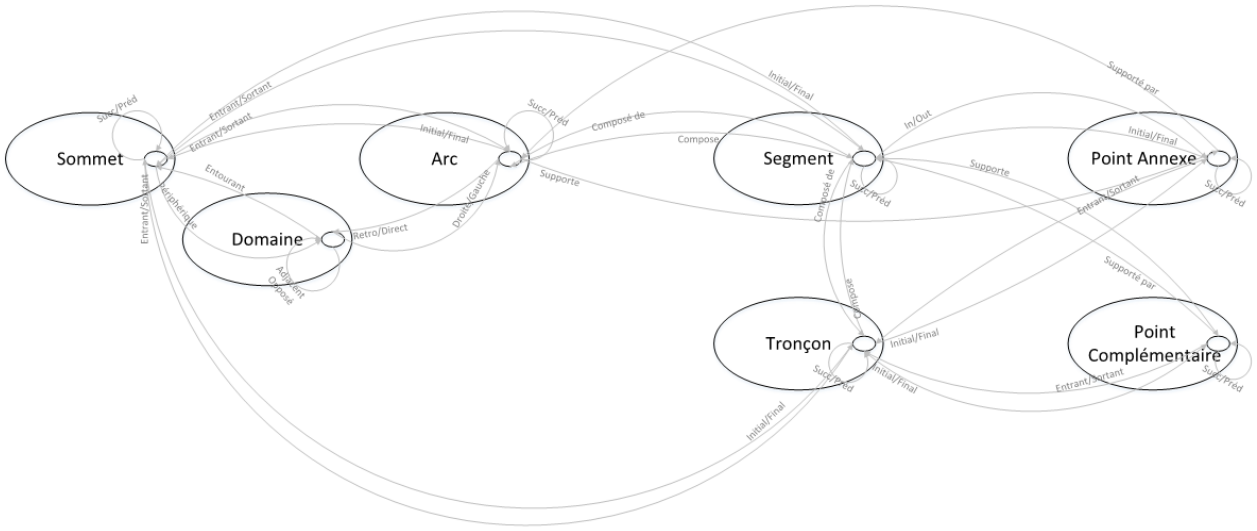


FIGURE 22 – Structure HBDS d'un graphe

Une autre façon de structurer ce phénomène via la méthode HBDS se trouve sur la figure 23. Les éléments comprenant tout ce qui a trait à la ville sont regroupés dans une hyperclasse "éléments du réseau routier", tandis que les éléments géométriques sont regroupés dans une

autre hyperclasse dédiée. Entre ces deux classes nous trouvons les tronçons qui composent les routes et qui sont eux même composés d'arcs. Nous pouvons aussi noter la classe "Topographie" qui est composée de triangles et qui permet le calcul de l'élévation de chacun des sommets.

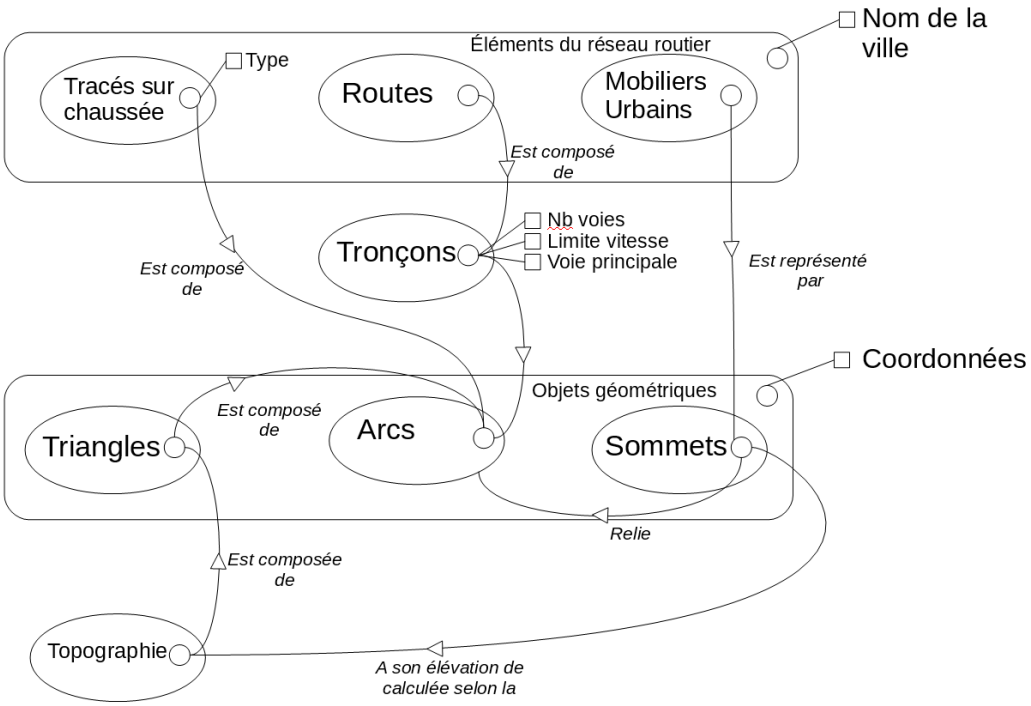


FIGURE 23 – Représentation HBDS du réseau routier

Deuxième partie

Critères de développement

Chapitre 4

Le choix de la licence de distribution du logiciel créé

Au début de cette thèse s'est posée la question du choix de la licence de distribution de ce programme. En effet, il était possible de créer le logiciel à partir d'outils Open Source, et donc de proposer un logiciel lui même modifiable et à disposition de toute personne, ou alors partir d'outils propriétaires. Nous avons choisi par défaut la solution Open Source du fait du contexte actuel permettant l'accès à de nombreuses données ou d'outils de manière libre et gratuite. Dans cette partie sont alors décrites les principales licences libres connues et reconnues dans les domaines informatiques.

4.1 Les différents types de licences

Si analyser les données/logiciels fourni(e)s par différents organismes passe par un certain nombre de critères (précision, surface couverte, communauté associée,...) cela passe également par l'étude de leur distribution et donc par leurs types de licences. En effet les licences sont nombreuses et ont leurs spécificités. Certaines sont des licences payantes (dont la plus connue est le CopyRight) et d'autres sont dites libres (selon différents critères). Ce sont ces licences que nous allons décrire ci-après.

4.1.1 Les licences Creative Commons

Les licences Creative Commons sont liées au droit d'auteur. Elles permettent de partager des créations, ainsi leurs reproductions, leurs diffusions et leurs modifications. Bien sûr, elles font économiser les coûts de transactions et légalisent la diffusion via le peer to peer. Elle sont découpées en plusieurs catégories :

BY : Le titulaire des droits autorise toute exploitation de l'œuvre,

SA : Share Alike, si l'oeuvre est modifiée, elle devra être diffusée sous la même licence,

NC : Non Commercial, il ne peut y avoir de diffusion commerciale de l'oeuvre,

ND : No Derivative, l'oeuvre ne peut être modifiée.

Des associations peuvent être faites avec ces catégories pour avoir 6 types de licences :

BY : Le titulaire des droits autorise toute exploitation de l'œuvre,

BY + ND : Le titulaire des droits autorise toute utilisation de l'oeuvre originale (y compris à des fins commerciales), mais n'autorise pas la création d'œuvres dérivées,

BY + NC + ND : Le titulaire des droits autorise l'utilisation de l'œuvre originale à des fins non commerciales, mais n'autorise pas la création d'œuvres dérivées,

BY + NC : le titulaire des droits autorise l'exploitation de l'œuvre, ainsi que la création d'œuvres dérivées, à condition qu'il ne s'agisse pas d'une utilisation commerciale (les utilisations commerciales restant soumises à son autorisation).

BY + NC + SA : Le titulaire des droits autorise l'exploitation de l'œuvre originale à des fins non commerciales, ainsi que la création d'œuvres dérivées, à condition qu'elles soient distribuées sous une licence identique à celle qui régit l'œuvre originale.

BY + SA : Le titulaire des droits autorise toute utilisation de l'œuvre originale (y compris à des fins commerciales) ainsi que la création d'œuvres dérivées, à condition qu'elles soient distribuées sous une licence identique à celle qui régit l'œuvre originale. Cette licence est souvent comparée aux licences « copyleft » des logiciels libres. C'est la licence utilisée par Wikipedia.

4.1.2 Les licences GNU

Les licences GNU ont été fondées par Richard Stallman. Il en existe divers types :

GPL : General Public License, la plus utilisée de nos jours, détermine des conditions de distribution qui garantissent les libertés de l'utilisateur. Elle met en place la notion de copyleft, jeu de mots anglo-saxons faisant référence au copyrigth. En plus des droits d'auteur, des points sur la liberté d'utilisation, de l'étude, de modification et de diffusion ont été ajoutés,

LGPL : Lesser General Public License (Licence publique générale limitée), elle est moins restrictive que la première, permettant aux logiciels libres sous cette licence de pénétrer des domaines où les licences libres ne sont pas les bienvenues. En d'autres termes, des parties du logiciels peuvent être sous des licences non GPL. Elle est souvent utilisée au niveau des bibliothèques,

GFDL : Free Documentation License, rend tout manuel, livre ou document écrit « libre d'utilisation ». Elle ne s'applique que sur les documents imprimés ce qui explique les nombreuses critiques envers elle,

AGPL : Affero General Public License, ajoute le côté "réseau" à la licence GPL, obligeant les services accessibles par ce dernier à publier leur code source.

4.1.3 Autres licences libres

Les licences libres ne se limitent pas aux licences Creatives Commons et aux licences GNU, de nombreuses autre licences existent, soit pour pallier à des manques des premières citées, soit pour les appliquer à des domaines bien spécifiques. On y retrouve :

MPL : Mozilla Public License, qui, commme son nom l'indique, concerne uniquement la distribution de codes source, binaire ou paquetage de l'application Mozilla,

CeCILL : **CE**A**C**NRS **I**NRIA **L**ogiciel **L**ibre, créé par des organismes de recherche français afin de garantir aux créateurs et aux utilisateurs de logiciels libres, le respect du droit français,

BSD : qui diffère de la licence GNU GPSL parce que non contaminante. Elle peut être modifiée sans restriction pour correspondre au mieux aux besoins du logiciel créé,

Apache V2 : incompatible avec la licence GNU GPL (cas de résiliation de brevet),

EGPL : Exception General Public Licence, basée sur la licence GPL, en ajoutant une restriction d'utilisation aux entités militaires et leurs fournisseurs,

ODBL : Open DataBase License, qui permet à chacun d'exploiter à des fins commerciales ou non, des bases de données, à condition de maintenir la licence sur la base de données. Sa version française provient d'une collaboration entre le projet ParisData[1] de la mairie de Paris et l'association VeniVidiLibri[2], diffuseur de licences libres.

4.2 Quel choix de distribution pour ce projet ?

Comme cité au début de ce chapitre, nous étions dans l'optique de choisir une licence libre. Pour cela, il a fallu regarder dans un premier temps les licences derrière les librairies et les langages que nous allons utiliser.

Le langage C++ n'est pas propriétaire et est complètement Open Source, sans licence, il en est de même avec la librairie OpenGL. Ces deux éléments ne sont donc pas restrictifs. L'API Qgis est, elle, sous licence GNU et la librairie dédiée aux traitements des images, SDL, est sous licence LGPL. La dernière bibliothèque utilisée est ASSIMP, bibliothèque permettant l'import d'objets 3D est sous licence BSD.

Afin de les harmoniser, nous avons opté pour une licence GPL pour ce logiciel.

1. https://opendata.paris.fr/page/home/
2. http://vvlibri.org/fr

Chapitre 5

Le choix des langages et logiciels utilisés

5.1 Présentation du matériel de développement

Pour développer le logiciel SIG dédié à cette thèse nous avons à disposition un ordinateur portable assez ancien n'ayant que 3 Go de Ram, un processeur 2 cœurs de 2 GHz et une carte graphique intégrée avec 1Go de Ram dédiée. Cela implique que le programme de simulation ne soit pas trop gourmand et soit optimisé, et qu'il puisse fonctionner sur la plupart des ordinateurs. C'est pour cela qu'une étape algorithmique préliminaire a été faite avant de réaliser un développement informatique sur IDE. Nous avons choisi l'ADL (Algorithm Description Language) du fait de notre formation, le master IASIG. Le système d'exploitation est Debian Jessie (Debian 9). Ensuite, nous avons un smartphone classique associé à un système de stéréoscopie que nous utilisons pour l'affichage en mode "casque de réalité virtuelle".

5.2 Choix du langage informatique

L'utilisation d'un tel appareil met déjà quelques langages informatiques de côté tel que le Visual Basic ou le .NET. Il en reste de nombreux disponibles, le choix se fera sur l'ergonomie, la rapidité et le côté multiplateforme de ceux-ci.

	Mac			Windows		
Langage	Version/Compiler	Temps	Temps Réél	Version/Compiler	Temps	Temps Réel
C++	GCC-4.9.0	0.73	1.00	Visual C++ 2010	0.76	1.00
	Intel C++ 14.0.3	1.00	1.38	Intel C++ 14.0.2	0.90	1.19
	Clang 5.1	1.00	1.38	GCC-4.8.2	1.73	2.29
Fortran	GCC-4.9.0	0.76	1.05	GCC-4.8.1	1.73	2.29
	Intel Fortran 14.0.3	0.95	1.30	Intel Fortran 14.0.2	0.81	1.07
Java	JDK8u5	1.95	2.69	JDK8u5	1.59	2.10
Julia	0.2.1	1.92	2.64	0.2.1	2.04	2.70
Matlab	2014a	7.91	10.88	2014a	6.74	8.92
Python	Pypy 2.2.1	31.90	43.86	Pypy 2.2.1	34.14	45.16
	CPython 2.7.6	195.87	269.31	CPython 2.7.4	117.40	155.31
R	3.1.1, compilé	204.34	280.90	3.1.1, compilé	184.16	243.63
	3.1.1, script	345.55	475.10	3.1.1, script	371.40	491.33
Mathematica	9.0, base	588.57	809.22	9.0, base	473.34	626.19

Tableau 3 – Comparaison de vitesse de calculs entre différents langages informatiques [10]

De nombreuses études ont été réalisées pour connaître les rapidités de calculs des différents langages. Sachant que les conversions de données et les simulations à réaliser seront gourmandes en calculs, nous préférons utiliser le langage le plus rapide. Une étude de 2014 [10] compare les principaux langages tels que Java, Fortran 2008, C++ ou Matlab et il en ressort que le C++ et le Fortran 2008 sont en tête de liste (voir tableau 3).

Ces deux langages sont orientés objets et sont utilisés par la communauté scientifique. Ce travail de thèse se rapproche de la création des jeux vidéos à monde ouvert, nous nous orientons vers le C++, langage informatique de prédilection pour la création de jeux. De plus, OpenGl, librairie dédiée à la création de vues 3D, est souvent associée au C++ ce qui en fait la librairie la plus utilisée pour les simulations 3D [11].

5.3 Choix des librairies

Pour gérer la partie 3D, la partie 2D, et l'interface, des librairies ont été sollicitées. Nous en avons déjà citées précédemment, en voici la liste :

OpenGL est une librairie dédiée à la 3D. Elle n'est pas soumise à une licence. Elle a été lancée par Silicon Graphics, société spécialisée dans l'infographie, le traitement vidéo et le calcul haute performance. Elle permet de définir des objets par des vecteurs, des lignes ou des points et d'ajouter des textures. La librairie gère automatiquement les calculs de projections pour l'affichage à l'écran prenant en compte la distance, l'orientation, les ombres et de la transparence.

L'API QGIS : est une interface permettant la gestion de données référencées vectorielles et rasters. Elle est sous licence GNU et est codée en C++. Elle est compilée sur le même système d'exploitation de notre projet afin d'accéder directement depuis le code aux fonctionnalités internes de Qgis.

Qt est une bibliothèque permettant la création de d'interface logicielle, appelée GUI (Graphical User Interface). En plus de cela, elle supporte OpenGL et donc, gère les fenêtres incluant des vues traitées par cette librairie. Il y a bien d'autres modules que nous n'utilisons pas tels que le module SQL, dédiés aux requêtes qur les bases de données ou le module XML, pour l'échange de données.

SDL (Simple Directmedia Layer) est une bibliothèque offrant une abstraction sur des éléments de programmation généralement considérés comme non-portables : vidéo, audio, entrées, threads, accès aux fichiers et gestion du temps. Nous l'utilisons pour la gestion des entrées clavier et souris pour l'interaction avec la simulation.

ASSIMP est une bibliothèque sous licence BSD permettant d'importer des objets 3D et de les convertir dans un format compréhensible par OpenGL.

Une fois cette configuration choisie, il nous fallait nous tourner vers un environnement de développement (IDE). Ayant choisi le C++ et la librairie OpenGl, nous privilégions le logiciel QtCreator qui propose dans sa version de base un compilateur C++ (GCC) et l'association avec OpenGl. Comme son nom l'indique, QT est aussi incluse nativement dans cet IDE. Cette configuration est couramment utilisée dans de nombreux domaines et est présente dans nos ordinateurs et nos smartphones. Cela permet de créer facilement une solution multiplateforme.

Comme énoncé précédemment, la partie SIG est supportée par l'API QGIS, du fait de son côté multiplateforme et open source. Malgré la multitude des outils présents dans QGis, il a fallu créer des fonctions dédiées à nos problématiques afin d'avoir des résultats plus rapides et plus spécifiques.

Chapitre 6

Les données nécessaires

Pour aborder cette thèse le premier problème était le choix de la donnée de base pour le réseau routier, pour les délimitations des différents bâtis, ou encore pour la topographie. L'époque actuelle veut que la plupart des données soient accessibles à tous et en particulier pour les travaux de recherche, ce qui nous donne un large éventail d'informations à portée de main. Le problème du choix du langage informatique s'est aussi posé, tant pour la partie calcul que la partie interface ou pour la 3D. L'explication de ces choix est détaillée dans les parties ci-après.

6.1 État de l'art sur les données de réseau routier

Sachant que nous allons utiliser les données du réseau routier pour faire des calculs d'itinéraires nous avons concentré nos recherches uniquement sur les données vectorielles, données de type géométrique contenant des informations alphanumériques. De plus, nous ne voulons aucune contrainte de secteur d'étude, privilégiant les caractères de libre accès et mondiaux de la donnée.

6.1.1 Les données de l'Institut de l'Information Géographique et forestière (IGN)

L'IGN propose plusieurs données à différentes échelles concernant les réseaux routiers : la BD TOPO®, la BD CARTO®et la BD routière. Cette dernière regroupe les bases de données de la BD TOPO et de la BD CARTO, c'est pourquoi le choix de cette base de données apparaît le plus judicieux. La BD routière est disponible à plusieurs échelles, les deux principales sont la BD ROUTE500®et la BD ROUTE120®. La BD ROUTE500 étant plus précise, nous la privilégions pour cette étude comparative. ROUTE500[1] est accessible pour tous les usages selon les termes de la licence ouverte version 1.0.[2]. Cette donnée, comme toutes les données provenant de l'IGN est nationale. Cette dernière est sous forme vectorielle et très détaillée au niveau des attributs dont les principaux se réfèrent au nombre de voies, le sens et le numéro de la voie. La donnée est vraiment très détaillée pour les principaux axes mais ne contient pas ou peu les routes les moins importantes. Elle est adaptée pour les échelles de travail comprises entre 1/200.000 au 1/500.000.

1. http://professionnels.ign.fr/route500#tab-3
2. https://www.etalab.gouv.fr/wp-content/uploads/2014/05/Licence_Ouverte.pdf

6.1.2 Here Navstreets™

Here est le nouveau nom de la société Navteq, connue pour ses données routières dédiées aux systèmes de navigation embarquée. Cette donnée mondiale est l'une des plus précises du marché avec celle de Google Maps [12]. Elle propose trois types de précisions[3] incluant ou non l'élévation, les adresses ou les positions des parkings. Cependant elle est payante. Dans les trois cas, cette donnée est très précise et contient tous les attributs possibles pour la gestion de trafic : la position des feux de signalisation, des passages pour piétons, la vitesse maximale sur chaque tronçon,.. Elle inclut de même l'empreinte au sol des bâtiments voisins du chevelu routier. Du fait de sa constante mise à jour (25% du réseau est remis à jour tous les ans) sa précision est très fine. Cette donnée est fournie à de nombreux organismes tels que Microsoft pour alimenter Bing Maps.

6.1.3 TomTom et Téléatlas

L'entreprise TomTom qui a racheté la société Télé Atlas, fournit ses données à de nombreuses entreprises telles que Apple pour Apple Map. Sa précision est décamétrique sur quasiment tout le globe ce qui en fait une donnée utilisée par de nombreux systèmes embarqués. Tout comme la donnée Here, elle possède tous les attributs nécessaires à la simulation de trafic urbain. Elle se caractérise par l'abondance des points d'intérêts (POI) tels que l'emplacement des stations d'essence, des hôtels, des parkings.

6.1.4 OpenStreetMap (OSM)

OpenStreetMap (OSM)[4] est un ensemble de données ouvertes, disponibles sous licence libre ODbL[5]. Ces données mondiales sont alimentées par la communauté d'utilisateurs ainsi que par de grands organismes. Par exemple, l'entreprise néerlandaise Automotive Navigation Data (AND) a importé sa donnée dans OSM, tout comme les Etats Unis avec la base de données TIGER. La donnée routière peut être de très bonne qualité dans certains pays et quasi inexistante dans d'autres. Les grandes villes sont bien renseignées et on y retrouve les attributs nécessaires à la simulation de trafic urbain. Certaines villes proposent même l'emplacement des feux tricolores, des passages pour piétons voire même des arrêts de bus. Quand la donnée est bien complète, les largeurs des tronçons, la vitesse maximale y étant autorisée, leur nombre de voies sont indiqués. La précision est décamétrique ce qui en fait une donnée idéale pour le sujet de cette thèse. De plus OpenStreetMap est en constante évolution et le nombre d'entrées ne fait qu'augmenter depuis sa création [13] [14].

6.1.5 Google Maps

La donnée de Google est difficilement accessible pour des travaux de recherches. Cependant, elle reste une donnée très fournie et détaillée sur tout le globe. On y retrouve les attributs de vitesse maximale, de largeur de route, ou encore d'élévation.

6.1.6 Récapitulatif et conclusion

Parmi toutes les données disponibles ressortent les données de l'IGN et OSM (voir tableau 4). En effet, pour les deux nous avons une précision suffisante avec les informations caractéristiques pour créer une simulation de trafic urbain. La seule différence est le fait que la donnée de L'IGN

3. http://navmart.com/here-navstreets-comparison/
4. http://www.openstreetmap.org/copyright
5. https://opendatacommons.org/licenses/odbl/1.0/

est nationale alors qu'OSM propose une couverture mondiale. Pour ne pas avoir de contraintes sur le choix d'un cas d'étude nous optons pour la donnée d'OSM. Cependant, nous pourrons utiliser la donnée de l'IGN pour un cas d'étude sur une ville française. Afin de voir ce que ces deux données peuvent apporter en terme de précision, passons un peu de temps sur leur comparaison.

Donnée	Résolution de travail	Secteur	Information suffisantes	Accès de la donnée
ROUTE500®	décamétrique	National	Oui	Licence Ouverte
Here Navstreets™		Mondial		Payant
TomTom		Mondial		Payant
OSM		Mondial		Licence libre
Google Maps		Mondial		Non Accessible

Tableau 4 – Tableau récapitulatif des principales données routières

Comparaison d'OSM et de la BD Route500

Pour faire cette comparaison nous choisissons trois tailles de villes :
— une grande ville, Lyon,
— une moyenne, Reims
— et une plus petite, Charleville-Mézières.

Nous prenons en compte la précision de la donnée, le nombre d'attributs renseignés pour les tronçons d'une zone choisie ainsi que les lacunes que chaque donnée peut avoir, en ajoutant pour chacune des cartes la position des feux de signalisation, des passages pour piétons et des stops stockés dans la base de données d'OSM (points en bleu). On peut d'ores et déjà constater qu'ils sont nombreux dans chacune des villes choisies. Les différences d'attributs entre chaque donnée sont visibles dans un tableau comparatif. Pour chaque attribut, le pourcentage d'informations est calculé.

Cas de Lyon

Nous avons choisi volontairement une importante partie de Lyon en prenant dans la zone d'étude quatre arrondissements (Le 1er, le 3ème, le 5ème et le 6ème) afin de mettre en avant les différences entre les deux données. Nous observons sur la figure 24 qu'il y a une grande différence de recouvrement entre les deux fichiers. En effet, la BD Route500 compte seulement 66 tronçons contre 2146 pour la base de donnée d'OSM (voir tableau 5). En considérant les attributs, des différences nettes sont aussi visibles. La donnée d'OSM n'offre pas la précision sur le nombre de voies alors qu'elle est renseignée à hauteur de 90% dans la donnée de l'IGN. Pour ce qui est du sens de circulation, les deux données sont égales contrairement à l'indication de la vitesse maximale autorisée qui n'est pas présente dans la donnée IGN.

Attributs	Nombre de voies	Sens de circulation	Vitesse autorisée
OSM (2146 tronçons)	0%	100%	28%
BD ROUTE500 (66 tronçons)	90%	100%	0%

Tableau 5 – Comparaison des attributs pour la ville de Lyon

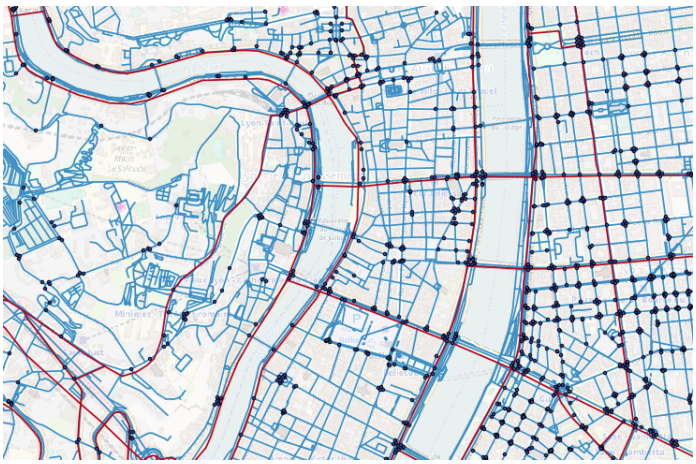


FIGURE 24 – Comparaison de l'OSM et de la BD Route500 sur la ville de Reims : En bleu la donnée OSM et en rouge celle de la BD Route500. Les points proviennent d'OSM et correspondent au mobilier urbain

Cas de Reims

Sur la ville de Reims, la différence est toujours flagrante. Chaque tronçon de route semble être représenté dans la base de données d'OSM (figure 25). Pour une petite zone de Reims, 365 tronçons sont répertoriés dans OSM contre 21 dans la BD Route500. Tout comme pour Lyon, la donnée d'OSM ne présente pas d'information sur le nombre de voies, même pour celles coïncidant avec les données de l'IGN qui elles, sont renseignées à 100%. Les deux données se rejoignent au niveau de l'attribut "Sens de circulation" où l'information est toujours présente. La dernière différence vient de la vitesse maximale autorisée. Renseignée à 11% dans la donnée OSM, elle n'est jamais présente dans la BD Route500 (voir tableau 6).

FIGURE 25 – Comparaison de l'OSM et de la BD Route500 sur la ville de Reims : en bleu la donnée OSM et en rouge celle de la BD Route500. Les points proviennent d'OSM et correspondent au mobilier urbain

Cas de Charleville-Mézières

La différence est encore plus remarquable dans le cas de Charleville-Mézières (figure 26). En effet, les quelques tracés correspondant à la donnée de la BD Route500 sont en rouge alors

Attributs	Nombre de voies	Sens de circulation	Vitesse autorisée
OSM (365 tronçons)	0%	100%	11%
BD ROUTE500 (21 tronçons)	100%	100%	0%

Tableau 6 – Comparaison des attributs pour la ville de Reims

que la donnée d'OSM paraît bien plus complète, ici en bleu. De plus, le mobilier urbain est toujours renseigné.

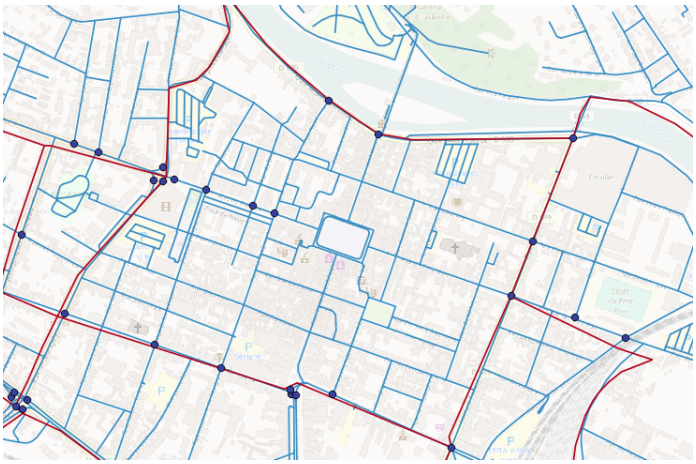


FIGURE 26 – Comparaison de l'OSM et de la BD Route500 sur la ville de Charleville-Mézières : en bleu la donnée OSM et en rouge celle de la BD Route500. Les points proviennent d'OSM et correspondent au mobilier urbain

Attributs	Nombre de voies	Sens de circulation	Vitesse autorisée
OSM (267 tronçons)	0%	100%	5%
BD ROUTE500 (18 tronçons)	100%	100%	0%

Tableau 7 – Comparaison des attributs pour la ville de Charleville-Mézières

Conclusion

La différence entre les deux données est grande au niveau de la précision et des attributs présents. Bien sûr, elle serait bien moindre avec la donnée de la BD TOPO, mais celle-ci n'est pas distribuée avec une licence ouverte. Le géoportail[6] permet de visualiser cette donnée (figure 27). La BD TOPO est affichée sur le fond de plan d'OSM et toutes les rues semblent coïncider entre les deux données. N'ayant pas pour but de travailler avec des données sous licence payante, nous choisissons la donnée d'OSM. De plus cette dernière propose le nombre de voies de façon indirecte car son attribut "Sens de circulation" indique le sens et aussi le fait qu'il y ait un double sens, donc des voies multiples. De plus un attribut concernant le type de route (*tag : highway*) est quelques fois présent, nous renseignant sur le nombre de voies et sur la vitesse autorisée par défaut. C'est moins précis que sur la BD Route500 mais cela est satisfaisant à cette étape de l'étude. Autre point, et non des moindres, la donnée d'OSM est mondiale, ce qui permet de choisir n'importe quelle ville pour tester les diverses conversions et simulations.

6. https://www.geoportail.gouv.fr/carte

FIGURE 27 – Représentation du réseau routier de Charleville-Mézières de la BD TOPO. Le fond de plan est issu de la BD d'OSM afin de faciliter la comparaison des deux données.

6.2 Données topographiques

La donnée topographique est également disparate et peut provenir de diverses sources, payantes ou non. Présentons une liste non exhaustive de ces dernières afin de choisir la plus appropriée à notre problématique.

6.2.1 Les données de l'Institut de l'Information Géographique et forestière (IGN)

L'IGN propose de nombreuses données de topographie sous forme de grilles avec différentes résolutions. La plus petite résolution est de 1m en latitude/longitude et la plus grande peut aller jusqu'à 250m de résolution. Ces données ne sont pas gratuites. Les plus fines résolutions ne sont pas sous licence ouverte et sont accessibles soit par achat soit par licence pour l'enseignement ou la recherche. La donnée topographique accessible est celle de résolution de 25m mais ces fichiers ne sont disponibles que pour la France. Sachant que nous souhaitons trouver des données présentes mondialement, nous ne les écartons pas pour autant car elles permettent d'avoir un référentiel assez précis pour le territoire français.

6.2.2 WorldDEM d'Airbus

La donnée WordDEM d'AIRBUS est disponible sur le globe avec une précision de 12 x 12m depuis 2014. Cette donnée est payante et n'est pas sous licence ouverte. Acquise par balayage satellitaire, elle est utilisée dans de nombreux domaines, militaires, agricoles, ou encore dans la prospection d'énergies fossiles.

6.2.3 SRTM Data

La donnée SRTM - Shuttle Radar Topography Mission - est disponible sous licence ouverte sur le globe avec une résolution de 90 x 90m. Une version à 250m a été aussi mise à disposition. Cette donnée est une interpolation de données acquises par satellite [15]. Elle est aussi complétée avec la donnée ASTER DEMs, notamment pour certaines zones du Sahara. La donnée se télécharge facilement par zone géographique en format **asc** ou en **geotiff**.

6.2.4 Conclusion sur la topographie

Les données topographiques se ressemblent car l'information est simple, la seule différence vient de la résolution et de l'accessibilité. Les contraintes que nous nous sommes imposées filtrent grandement le choix de la donnée. Afin de visualiser les différences entre les sources nous avons choisi de présenter sur le tableau 8 l'exemple de la côte bretonne.

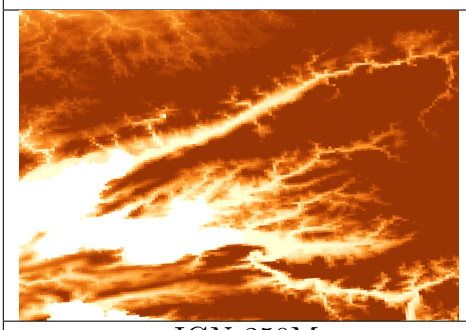
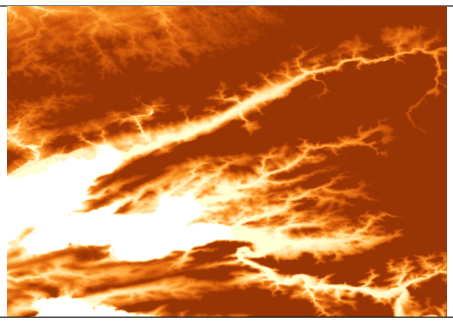
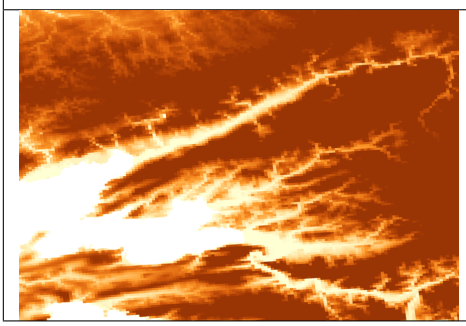
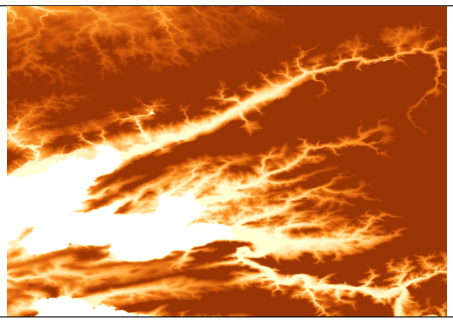
SRTM 250M	SRTM 90M
	
IGN 250M	IGN 75M
	

Tableau 8 – Comparaison des fichiers d'élévation pour une zone donnée

Comme nous pouvons le constater, plus la résolution est petite, plus les détails sont élevés. Cependant la résolution de 75m ou de 90m ne sera pas suffisante pour une visualisation correcte. Un interpolation de cette donnée doit être faite et est traitée dans le chapitre 7.

Chapitre 7

Conversion de la donnée en 3D

Nous avons vu dans la partie précédente quelles données SIG nous allons utiliser durant ce travail. Ces données vectorielles sont considérées dans la plupart des cas comme de la donnée 2.5D :

— La géométrie de chaque objet correspond à la partie 2D
— L'information concernant sa hauteur correspond à la demi-dimension

La hauteur du bâtiment donnée en addition à la forme plane de l'objet nous permet d'implémenter une extrusion selon l'axe des Z.

7.1 Différences entre les données 2.5D et 3D

La donnée vectorielle est décrite avec des formes planes (points, lignes et polygones) et avec des attributs. Tous les objets d'une couche vectorielle ont au moins un attribut, la clef primaire, qui permet de les rendre uniques. De plus, d'autres données peuvent être attribuées à chacun de ces éléments afin de mieux les définir. Par exemple, pour un fichier contenant des données vectorielles délimitant des habitations, nous pouvons avoir :

— le nombre d'étages,
— la date de construction,
— le matériau utilisé (béton, briques, etc.),...

Généralement une donnée attributaire d'élévation est liée à ce type de données qui correspond :

— soit à la base du bâtiment,
— soit à l'élévation totale du bâtiment
— soit à sa hauteur.

Cette information, considérée comme une demi-dimension, nous permet d'appliquer une extrusion verticale de chaque bâtiment. Par contre, cela ne nous permet pas d'avoir des "trous" (tel qu'un pont) dans la structure car un seul (z) (ou H) est attribué par objet. Les données 3D sont décrites différemment. Pour tous les points (x,y) un (z) est attribué. Dans notre exemple des habitations, cela reviendrait à avoir la description des contours des fenêtres, ou le renfoncement de la porte d'entrée. Pour avoir une donnée aussi précise, la technologie Lidar est généralement utilisée. Celle ci est basée sur le principe des ondes réfléchies. Lorsqu'une acquisition Lidar est réalisée, tous les objets autour de l'appareil d'acquisition sont géo-référencés avec un (z) pour chaque sommet. Une fois les sommets stockés, la donnée peut être utilisée de façon brute, ou alors un traitement peut être fait pour découper chaque objet 3D. Pour avoir une donnée de

haute qualité, le dispositif Lidar est déplacé en différents points pour avoir chaque côté de l'objet étudié. Le résultat d'un objet 3D traité au Lidar est illustré dans la figure 28. De plus une quatrième information est souvent stockée lors de l'acquisition via cette méthode : la couleur. Ceci donne alors un nuage de points colorés qui reconstitue un objet de façon très réaliste.

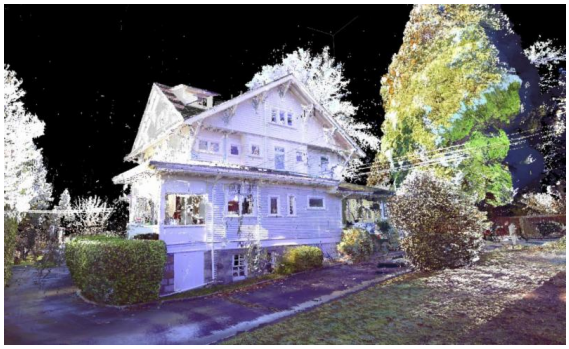


FIGURE 28 – Représentation d'une capture 3D par Lidar d'une façade de maison

On observe la façade avec ses renfoncements, son toit de forme complexe, ou bien son chien assis. Ces détails sont précis et permettent une visualisation proche de la réalité. N'ayant pas accès à cette donnée 3D au cours de cette étude, l'extrusion sera employée sur la modélisation des bâtiments. Afin de garder un certain réalisme, un algorithme d'application de textures aléatoires sera appelé pour chaque bâtiment, celui-ci sera expliqué dans la partie 7.3.1.

7.2 Interpolation de la topographie

La donnée choisie pour représenter la topographie est celle de la base de données de la Nasa SRTM. Cette donnée mondiale a une résolution de 90m en horizontale et 1m en verticale. Nous pourrions l'utiliser telle quelle, mais pour avoir une meilleure visualisation, et moins pixelisée, nous devons l'interpoler. De nombreux algorithmes d'interpolation existent comme par exemple l'interpolation pondérée par l'inverse de la distance [16] (IDW) ou l'interpolation triangulaire [17]. Ces interpolations permettent d'affiner une topographie grossière de manière efficace. L'interpolation de Laporte (INTERXY) [18] permet également d'affiner un maillage tout en présentant la particularité de garder la continuité topographique au niveau des lignes de crêtes ou dans les talwegs. Afin de comprendre en détail les différences entre les algorithmes, considérons une zone dans le massif Himalayen, afin d'avoir des changements de pentes flagrants, ainsi que les interpolations IDW et INTERXY avec des résolutions de 50m, 10m, 2m et 1m. En commençant par la topographie voici la zone d'étude avec sa résolution native illustrée dans la figure 29.

Le résultat de ces interpolations est proposé dans le tableau 9. Tout d'abord remarquons des similitudes :

— En dessous de 10 m de résolution il y a peu de différences pour chacune des interpolations.
— Les courbes globales sont gardées quelles que soient les résolutions.

Par contre il existe des différences permettant de trancher sur l'utilisation d'une interpolation ou d'une autre :

— Les courbes semblent moins géométriques avec l'interpolation INTERXY.
— Le temps de calcul est bien plus court avec l'interpolation IDW.
— A petite échelle la différence est plus notable. Des artefacts sont présents avec l'interpolation IDW (par exemple en bas à gauche de l'interpolation à 1m, les contours sont très droits).

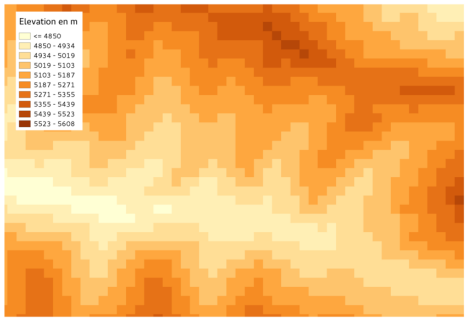


FIGURE 29 – Zone d'étude provenant de la base de données SRTM avec une résolution de 90m

Afin d'avoir une bonne représentation de la topographie, choisissons l'interpolation de Laporte [18]. Le temps de calcul est plus long mais sachant que l'interpolation de la donnée est faite en pré-processing, cela n'a que peu d'importance. Comme dit précédemment, la différence est minime pour une résolution inférieure à 10m, gardons alors cette valeur par défaut pour toutes les interpolations.

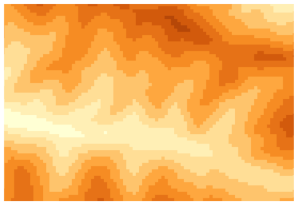
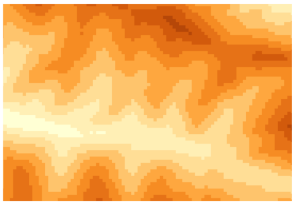
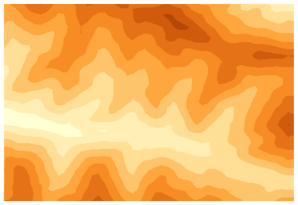
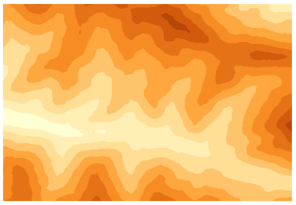
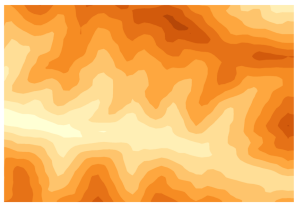
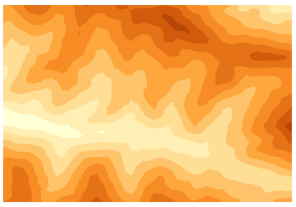
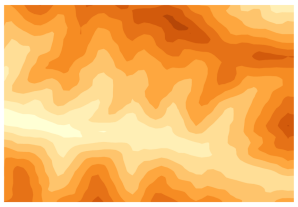
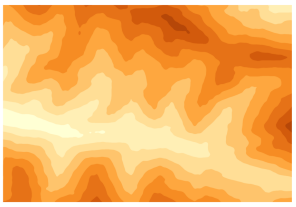
Résolution	INTERXY	IDW
50m		
10m		
2m		
1m		

Tableau 9 – Comparaison des interpolations IDW et INTERXY à différentes résolutions

7.3 Conversion de la donnée SIG en 3D

7.3.1 Cas des bâtiments

Le but de ce travail est de démontrer que la donnée SIG permet de modéliser des villes en 3D et d'y intégrer des moteurs de simulation. De nombreux algorithmes ont été créés afin d'automatiser les processus de conversion de la donnée 2.5D en 3D. L'API QGIS permet, dans le programme créé, d'extraire, pour le fichier contenant les polygones des bâtiments, chaque sommet (x,y). Nous stockons ces sommets dans un vecteur, ici considéré comme un tableau (voir figure 30). Une fois ces coordonnées extraites, nous rattachons l'élévation contenue dans la donnée SRTM à chacun des points via les algorithmes G3D et INTERXY (voir annexes C et D). Ces algorithmes ont pour but de prendre en entrée la position du sommet extrait et d'interpoler son élévation selon les 16 sommets les plus proches de ce point. L'importance de l'algorithme INTERXY est qu'il prend en compte la présence de crêtes ou de talwegs et ne lisse pas la topographie. Cet algorithme à été créé par Laporte en 1980 [19]. Une fois le traitement réalisé pour tous les points nous avons une table Tbati (de NBati) liée à la table TPoints (figure 30).

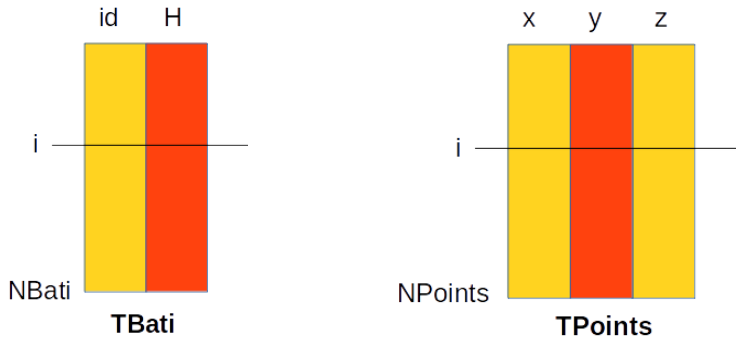
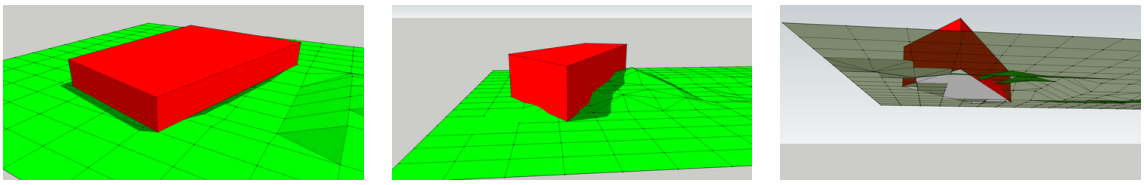


FIGURE 30 – Tables TBati et TPoints créées par les algorithmes d'extraction des coordonnées

L'application de cet algorithme, à tous les bâtiments, entraîne deux limitations de visualisation liés à la résolution de la grille raster de topographie (voir la figure 31).



(a) (b) (c)

FIGURE 31 – La figure (a) illustre un bâtiment (ou une route), ayant une taille supérieure à la résolution de la grille, et un creux dans celle-ci fait apparaître un espace dessous. La figure (b) illustre le cas ou il y a une élévation de la topographie. Le bâtiment la cache et le résultat n'est pas limitant visuellement si l'on ne tient pas compte de la vue du dessous (c).

La solution simultanée de ces deux problèmes est de découper les bâtiments (ou les tronçons de routes) selon la résolution de la topographie. Pour cela, nous avons créé un algorithme d'intersection entre les contours de polygone 2D avec la grille de topographie (H). L'objet 3D est donc plus complexe et suit maintenant les différences de topographie (voir figure 32).

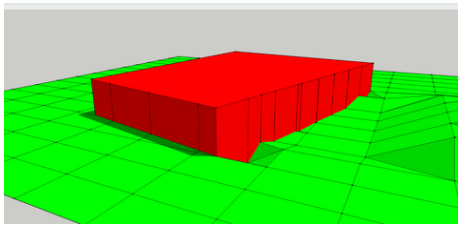


FIGURE 32 – Objet 3D épousant la topographie sous-jacente. A chaque intersection entre un segment et les triangles formant la topographie, un nouveau segment est créé afin de pouvoir lui attribuer une élévation, ce qui permet à l'objet de suivre parfaitement la topographie.

Maintenant, à l'aide de ces tables de points remplies, nous pouvons calculer la hauteur d'extrusion de chaque bâtiment [20]. Afin d'avoir le contour des toits droit par rapport à la topographie qui, elle, peut être accidentée, prenons le point le plus haut de la base du bâtiment pour avoir une valeur de référence. Ensuite nous calculons le Z correspondant pour tous les points du toit. Notre table TPoints a maintenant une nouvelle colonne correspondant à l'élévation du toit (figure 33).

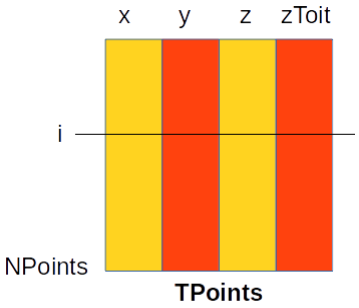


FIGURE 33 – Table TPoints avec l'élévation du toit pour chaque point du bâtiment

Cette extrusion est parfaitement illustrée par le groupe F4[1]. Cette société, basée à Paris, est spécialisée dans les applications web utilisant la 3D. La principale application, F4map, utilise les données Open Street Map et les affiche en 3D.

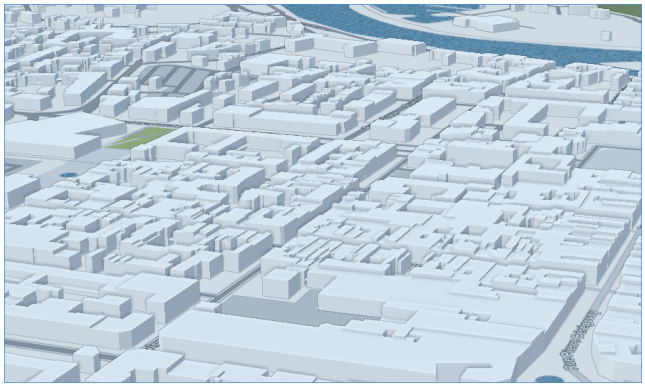


FIGURE 34 – Extrusion des bâtiments de Charleville-Mézières stockés dans OSM par la web-application F4map

1. http://map.f4-group.com

La figure 34 illustre l'extrusion d'une partie de Charleville Mézières. La donnée OSM, pour cette ville, ne contient pas la hauteur des bâtiments.

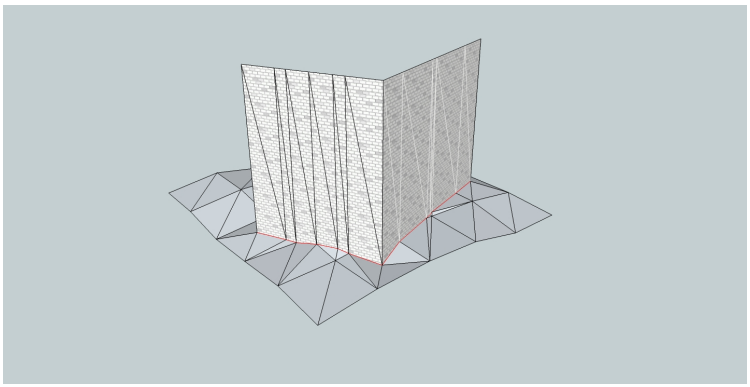


FIGURE 35 – Triangulation des murs d'un bâtiment

Texturage des bâtiments

Les données SIG d'OSM ne contiennent pas de textures, mais une donnée attributaire concernant sa couleur globale existe quelquefois. Lorsque celle ci est renseignée, l'algorithme de préparation de la donnée la prend en compte et affiche donc en 3D le bâtiment entier avec cette couleur. Si cette donnée est inexistante, une texture aléatoire est appliquée.

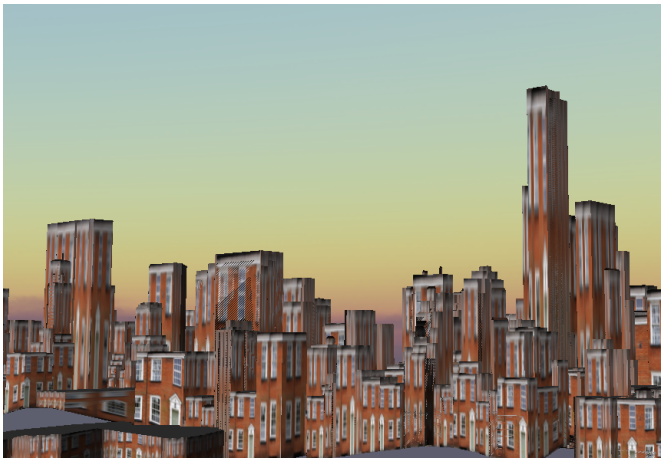


FIGURE 36 – Une seule texture image pour les bâtiments

On remarque plusieurs problèmes sur la figure 36 lorsqu'une même texture est appliquée sur l'ensemble des bâtiments. Premièrement, les bâtiments les plus grands ont une texture complètement difforme. Cela provient de la méthode d'application de la texture choisie. OpenGL permet d'appliquer une image aux dimensions fixes sur une surface. Si la surface n'a pas la même dimension que l'image originale, elle sera étirée. Le second problème est que tous les bâtiments se ressemblent. Pour remédier à cela nous avons créé des jeux de textures selon les hauteurs de bâtiments. De plus, pour avoir une certaine diversité, ces jeux de textures sont eux même de plusieurs types. Nous avons divisé les hauteurs selon trois catégories, les bâtiments moins de 20m, entre 20m et 40m, et plus de 40m. Les jeux de textures sont représentés dans la figure 37. On peut remarquer que le jeu de textures des bâtiments de plus de 40m parait compressé. Cela permet d'atténuer l'effet d'étirement lors de l'application de la texture.

FIGURE 37 – Fichiers contenant les textures de chaque catégorie de bâtiments selon la hauteur. *A gauche < 20m, au centre entre 20 et 40m et à droite > 40m*

Une fois ces textures appliquées sur les bâtiments, un rendu pseudo réaliste est visible. La figure 38 affichant les immeubles newyorkais nous l'illustre.

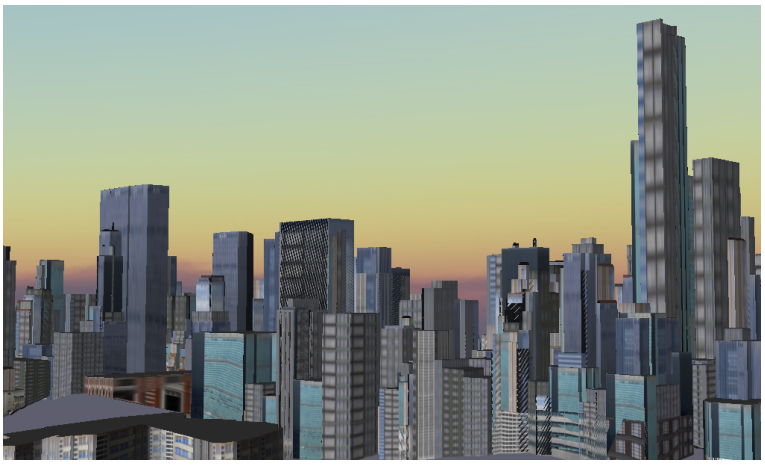


FIGURE 38 – Plusieurs jeux de textures selon la hauteur des bâtiments

7.3.2 Cas des routes

Les routes sont décrites selon un chevelu de lignes. Chaque tronçon est géoréférencé et peut contenir des informations intéressantes pour leur conversion en 3D. En premier, la largeur de chaque tronçon doit être connue. Si elle n'est pas référencée dans les attributs du fichier contenant le chevelu routier, une autre information va être recherchée à la place : celle du nombre de voies. En prenant en compte une largeur par défaut de 3,5 mètres, nous arrivons à établir une largeur moyenne de la chaussée. Par contre, si aucune information n'est présente, nous considérons les tronçons comme des voies à double sens, donc avec une largeur de 7m. Ce processus est couplé à un rattachement à la topographie sous-jacente. Afin qu'elle suive l'élévation du sol, une interpolation est calculée pour chaque sommet de chaque arc. Ce calcul n'est pas le seul lien entre la donnée topographique et la donnée routière. En effet, les triangles 3D générés pour afficher la topographie ne prennent pas en compte la présence de route ou de tout autre objet. Un problème survient lors de la superposition de ces deux données : les tronçons de routes ne sont pas découpés selon le découpage de la topographie. Nous nous retrouvons face au même problème que pour les bâtiments. Pour pallier ce défaut, les tronçons de routes sont découpés avant la conversion 3D selon les triangles de la topographie. L'étape d'après est la création de la topologie. Celle ci nous sert pour la partie simulation mais aussi pour la conversion en 3D du chevelu.

De la 2D à la 3D

Pour convertir ces tronçons linéaires en polygones, dans notre cas en triangles, considérons ces segments comme le centre de la chaussée. Pour chacun d'eux une mesure de l'angle est faite entre le Nième et son suivant . Cet angle servira de référence pour dessiner une bissectrice. Il y a alors 2 cas possibles :

— Le tronçon a un prédécesseur et un successeur : la mesure de l'angle est faite avec son prédécesseur et son successeur et sont tous deux stockés dans un tableau

— Le tronçon n'a qu'un prédécesseur ou un successeur : la mesure du prédécesseur ou du successeur est réalisée et pour le sommet opposé, un angle de 90 ° est appliqué afin de clôturer la route.

Les deux cas de figures sont représentés sur la figure 39.

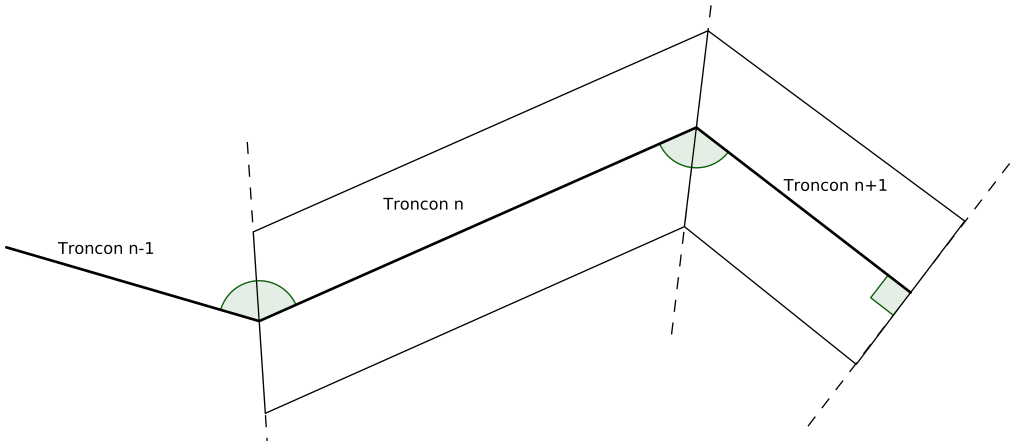


FIGURE 39 – Les bissectrices entre chaque tronçon servent à créer les polygones 2D pour la création de la route.

Cette procédure pour affecter les angles à chaque tronçon est réalisée lors de la création de la topologie sur le chevelu routier. Pour mieux appréhender la mise en place de cette procédure, l'algorithme CREER_TRIANGLE_ROUTE est présent dans l'annexe (voir E).

Une fois la conversion faite, la table contenant les routes est remplie de la manière décrite dans la figure 40. Cette table est chargée lors de la création d'itinéraires traitée dans le chapitre 9.

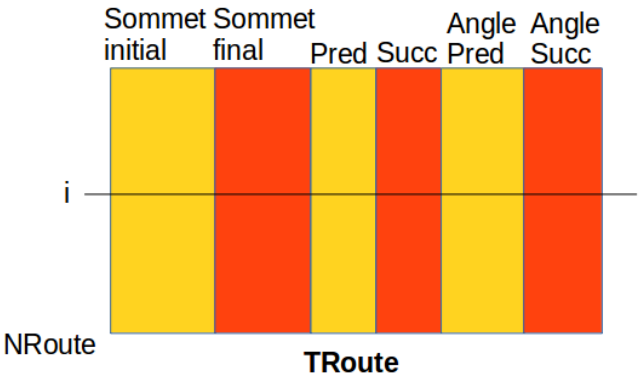


FIGURE 40 – La table route contient pour chaque tronçon, ses sommets, ses tronçons prédécesseurs et successeurs ainsi que les angles entre chacun de ces tronçons.

Les sommets de ces triangles n'ont pas d'élévation attribuée. Pour les renseigner, calculons

leur altitude avec l'interpolation IDW vue dans la partie 7.2. Nous n'utilisons pas l'interpolation de Laporte car nous avons besoin d'une valeur moyenne entre les points les plus proches afin d'avoir un calcul linéaire qui permettra de placer les points exactement sur la surface de la topographie. Puis, calculons les élévations seulement pour chaque extrémité du tronçon, c'est-à-dire que les points projetés de chacune des extrémités auront la même valeur d'élévation. Cela nous permet de ne pas avoir de pente ou de dévers sur les routes 3D, alors considérées comme plates.

Une fois les triangles de routes créés, des artefacts visuels apparaissent dans la scène 3D. En effet, chaque tronçon de route converti a une élévation égale à la topographie sous-jacente. Pour simplifier, nous avons des triangles de couleur grise plaqués sur la topographie. Afin de pallier ce problème il faudrait "creuser" la topographie afin d'inclure les routes à l'intérieur. Dans un même temps, cela permettrait la création des trottoirs de façon automatique. Le principe pour intégrer ces tronçons de route consiste à retirer dans un premier temps les triangles de topographie étant en intersection avec les triangles de routes. Ensuite, une triangulation est faite entre les triangles restants de la topographie et les triangles de routes. Pour finir, nous appliquons une profondeur à nos triangles de route pour créer les trottoirs. Nous choisissons une valeur par défaut, 20cm, afin que cela soit assez visible sur la vue 3D. Le processus, sans trottoir est illustré dans la figure 41.

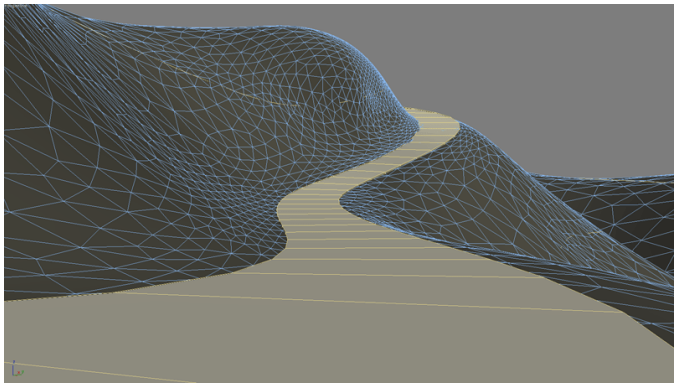


FIGURE 41 – Exemple d'incrustation de la route sur la topographie, Source : CGtricks.

Troisième partie

Méthodes et réalisations

Chapitre 8

Organisation du code

Nous avons choisi des librairies, un langage de programmation et un IDE, penchons nous maintenant sur l'organisation du code. Afin d'être le plus structuré et le plus clair face à une possible reprise du projet par un tiers, nous avons découpé le programme en blocs, chacun dédié à une partie majeure du logiciel. Ces sous-ensembles sont au nombre de quatre :

— La 3D,
— L'import de la donnée,
— L'export pour la sauvegarde de simulation
— Les outils dédiés à la géomatique et à la simulation.

8.1 La structure générale du projet

Le projet est découpé selon les fonctionnalités du logiciel lui-même. Chaque partie fonctionne de manière séquentielle avec les autres, les calculs ne sont pas faits en parallèle. Cela nous permet de suivre le cheminement de la donnée, de l'import à sa visualisation en passant par sa conversion.

8.2 Gestion des imports et de la conversion des fichiers sources

Il existe deux imports dans le logiciel, un avec des fichiers SIG, sources brutes, et l'autre permettant d'importer des conversions déjà calculées par le programme.

8.2.1 Import pour une nouvelle simulation

Les imports se font manuellement grâce à l'interface graphique, sa classe est nommée "gis4dwindow" dans le code. Les fichiers sources supportés sont en format vectoriels, pour les routes, la végétation, les surfaces d'eau et les zones herbacées, et en raster pour la topographie. Pour chaque fichier importé, des tests sur le format et son contenu sont réalisés. L'étape suivante est la copie de ces fichiers dans un dossier d'export. Cette copie nous permet de ne pas travailler sur les fichiers sources. Ainsi, l'utilisateur choisit un dossier et, à l'intérieur de celui-ci, deux nouveaux dossiers sont créés : un premier pour la donnée SIG et un second contenant les conversions réalisées au cours des prochains traitements. Nous développerons cela dans la partie 8.3.

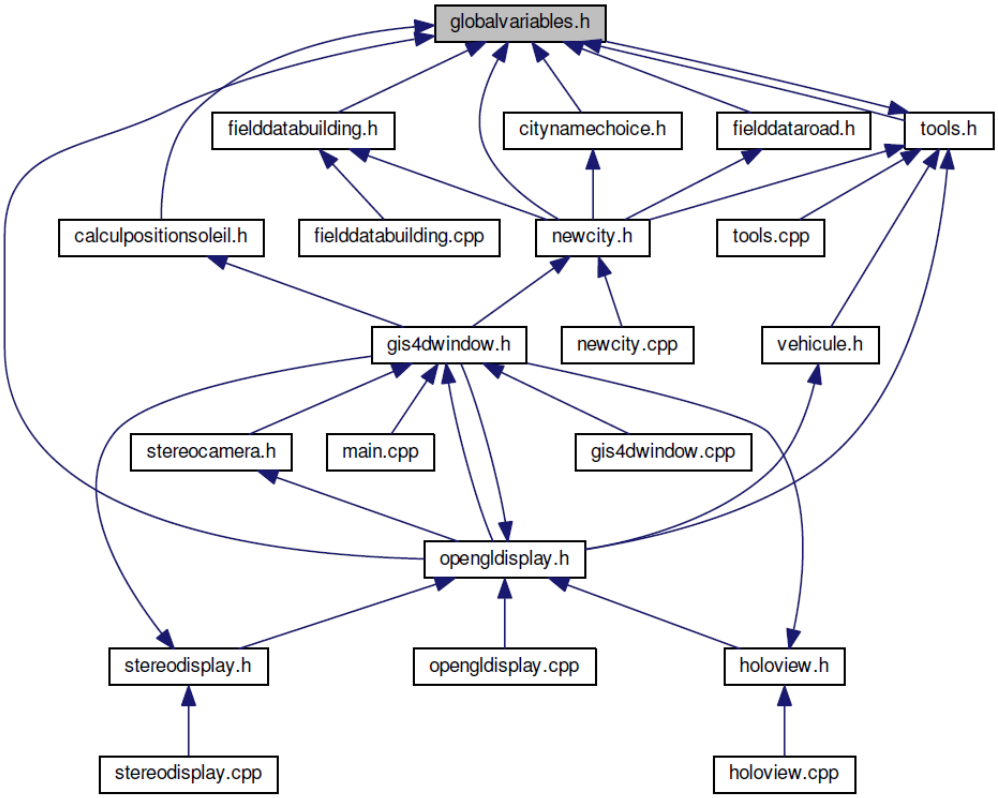


FIGURE 42 – Graphe des dépendances pour la partie import

Une fois les fichiers copiés, la phase de conversion débute. Elle est adaptée à chacune des entrées. Certains traitements sont communs pour les données de visualisation (surface d'eau, végétation ou bâti). Comme nous pouvons le voir sur le graphe des dépendances illustré dans la figure 42, nous remarquons que la classe *gis4dwindow* est en position centrale avec des accès pour la partie affichage en bas, correspondant aux appels vers les fonctions dédiés à la 3D, attachées aux boutons de l'interface. Nous y retrouvons la vue stéréoscopique (*stereocamera.h*), la vue 3D (*opengldisplay.h*) ou encore la vue holographique (*holoview.h*). La partie supérieure de ce graphe de dépendance reprend les parties dédiées à l'import. On y retrouve l'import d'une nouvelle ville (*newcity.h*) qui elle même appelle les classes gérant le nom de la ville (*citynamechoice.h*), les champs nécessaires à la visualisation des bâtiments (*fielddatabuilding.h*) ou encore les champs utiles à la simulation (*fielddataroad.h*). On peut remarquer que beaucoup d'appels sont faits sur la classe *tools.h*, ce qui est normal vu que tous les outils dédiés à la conversion sont inclus à l'intérieur, nous sommes dans la notion pure de "boite à outils".

Les données de visualisation

Les zones herbacées, les arbres et les surfaces d'eau sont des données purement visuelles. Elles sont demandées à l'utilisateur en format polygone, avec donc, pour chaque zone, des formes plus ou moins complexes pour définir l'environnement. Le traitement de ces données est celui-ci : nous stockons en mémoire chaque sommet de chaque polygone dans un tableau. Il y a bien sûr un tableau par fichier d'import permettant de faire la différence lors de l'affichage dans la scène 3D. Dans le cas des zone arborées, nous plaçons un arbre de type "objet 3D" à l'emplacement de chaque sommet de polygones (voir figure 43).

Pour les bâtiments, nous gardons aussi en mémoire la position de chaque sommet dans des

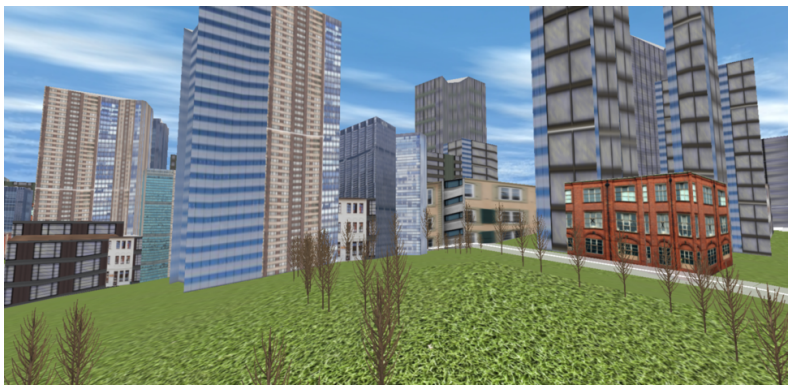


FIGURE 43 – Implantation d'arbres dans la vue 3D

tableaux et nous faisons les conversions décrites dans la partie 7. Les outils pour la conversion sont placés dans la classe *tool.h*.

Les données de simulation

Les données de simulation sont aussi utilisées pour la visualisation, et donc passent par les mêmes traitements que les données précédemment citées. Cependant, elles sont également converties pour gérer la modélisation. Toutes ces conversions sont placées dans la classe *tool.h*, comme nous pouvons le voir dans la figure 44. La fonction *getData* appelle les outils les uns après les autres pour les diverses conversions. On y retrouve les extractions des tronçons de routes, le calcul des itinéraires, de la topologie, etc...

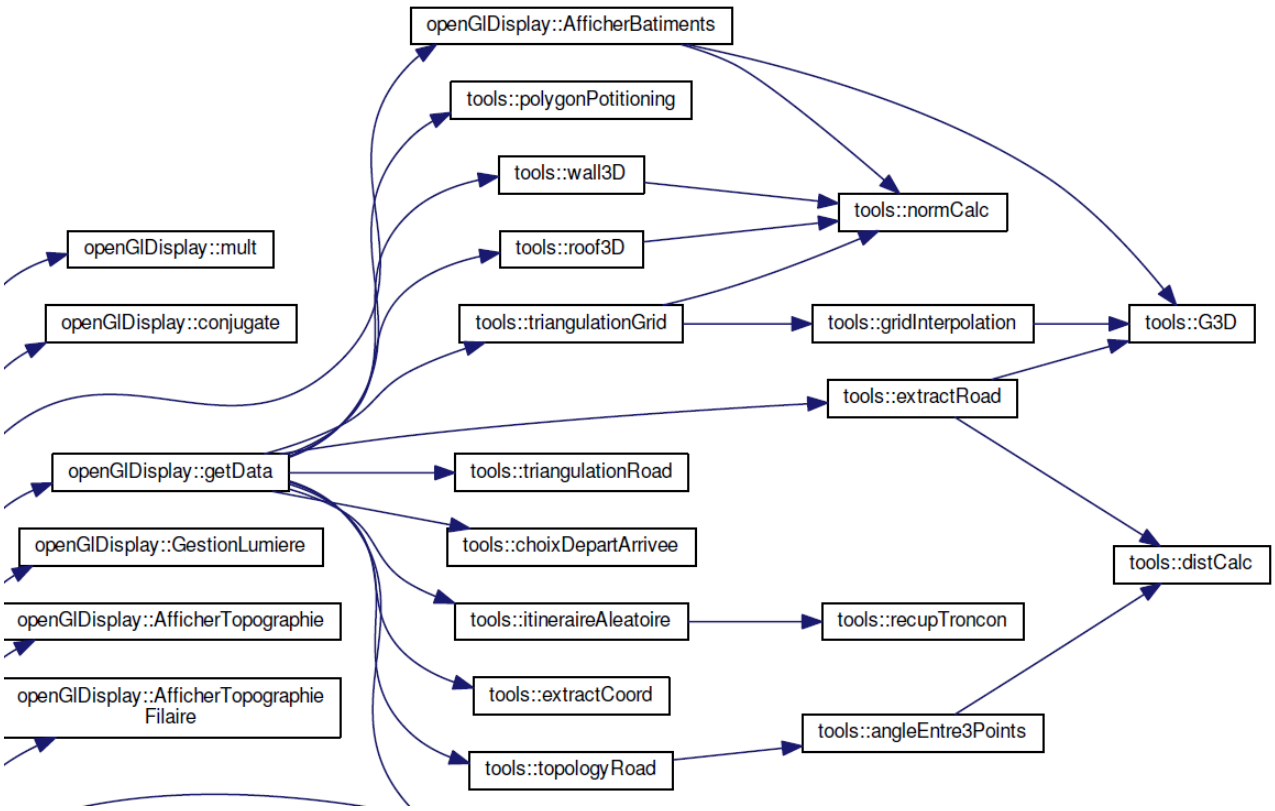


FIGURE 44 – Appels aux fonctions "outils"

8.2.2 Import d'un modèle 3D déjà existant

Lors de l'import d'une simulation déjà existante, les conversions ont déjà été réalisées et stockées dans des fichiers propres au logiciel. Le traitement est bien plus simple et rapide, l'utilisateur choisit un fichier de simulation et le programme stocke en mémoire les données dans des tableaux correspondants à chacune d'entre elles. Nous avons donc des tableaux de triangles avec des textures pour les bâtiments, des triangles pour les routes, des tronçons pour les calculs de trajets, ou même l'emplacement des arbres à afficher. Les calculs de trajets sont réinitialisés à chaque import afin de ne pas générer les mêmes scénarios d'une fois sur l'autre.

8.3 Gestion des exports

Comme abordé précédemment, des exports sont faits afin d'optimiser le traitement de données pour une simulation lors d'un second lancement. Ces données sont découpées par fichiers, il y aura donc un fichier contenant des triangles de routes, un autre avec les triangles de bâtiments, etc... Un fichier global est créé aussi, il contient les informations principales avec les chemins vers les autres fichiers ou le nom de la ville, par exemple. Ces fichiers ont une extension notée *.Traffic3D* permettant de faire un filtre lors de l'import.

8.4 Outils pour le logiciel

Les outils utilisés sont nombreux et ont des utilités diverses. Ils sont écrits dans la classe *tools.h* qui est publique afin de pouvoir y accéder depuis chaque partie du code. Les fonctions vont du calcul de normes à la création de topologie à partir d'un chevelu de tronçons. Nous avons ici déclaré des structures permettant un partage de données entre classes facilité comme par exemple la structure nommée triangleToDisplay. Ces structures contiennent :

- Les sommets avec leurs coordonnées 3D,
- La norme pour l'affichage des parties cachées et la gestion de la lumière (voir chapitre 13),
- Les coordonnées du pixel correspondant à la topographie sous-jacente pour optimiser le chargement des parties environnantes aux routes,
- La texture à appliquer dans le cas des bâtiments.

Ces informations globales sont renseignées selon le type de triangles et permettent de stocker plusieurs données à un seul et même endroit.

8.5 Gestion de la scène 3D

La scène 3D est gérée dans la classe *openglDisplay*. Elle contient des outils spécifiques à l'affichage 3D tels que ceux dédiés à la gestion des quaternions. On y retrouve aussi les appels aux classes calculant la vue anaglyphique. Toute la gestion des entrées souris se retrouve également dans cette classe, voir figure 45.

On y retrouve ici les appels vers la fonction de gestion de lumière, celle gérant le changement d'angle de vision (roulis et tangage) ou toutes les fonctions permettant d'afficher ou non les données.

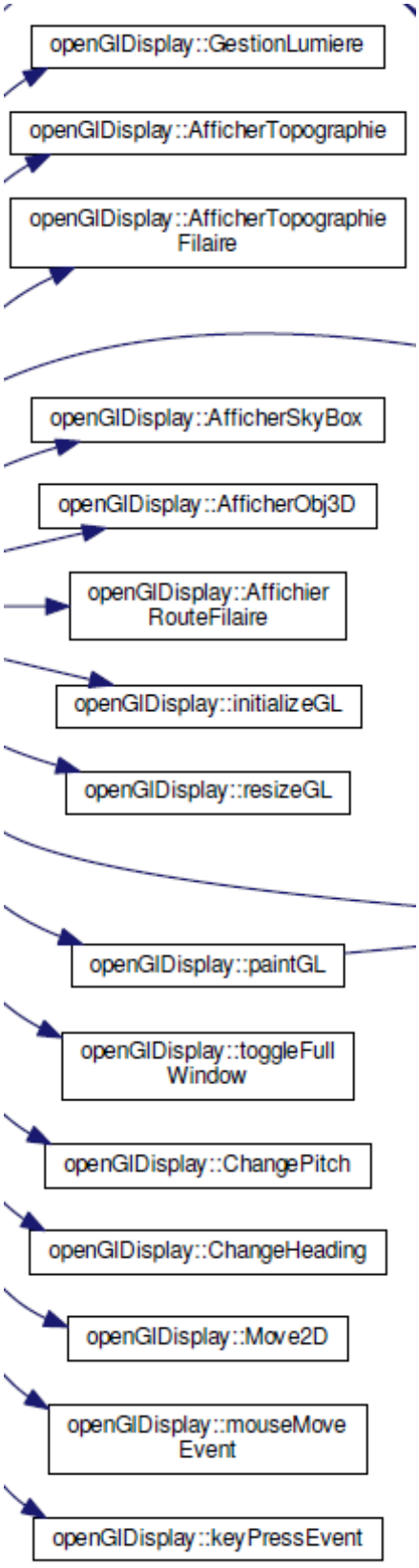


FIGURE 45 – Appels aux fonctions dédiées à la 3D

8.6 Gestion de la visualisation 2D

La visualisation 2D intègre l'API Qgis et permet l'affichage des données brutes. Nous y trouvons les liens avec les fonctions pour afficher ou non les données. Il y a aussi toute la gestion de l'interface. Elle fonctionne via des signaux et des slots. Ce sont des concepts proposés par la librairie Qt, qui permettent de gérer les événements liés à l'interface graphique. Les signaux sont les messages envoyés par un widget (boutons, liste déroulante, barre de menu, ...) lorsqu'un événement s'est produit, par exemple lorsque l'utilisateur appuie sur un bouton. Les slots sont les fonctions appelées lors de l'événement, en d'autres termes, le signal appelle le slot. Concrètement, un slot est une méthode d'une classe.

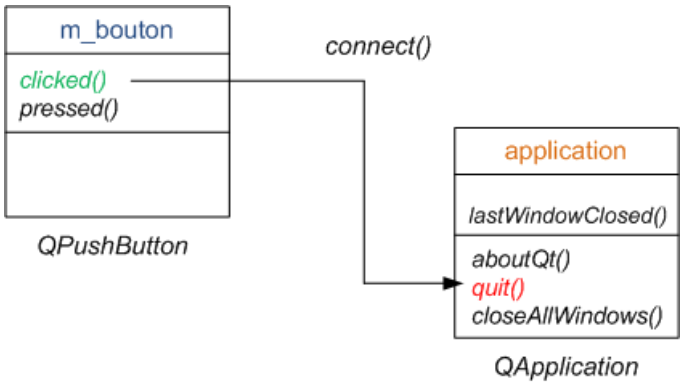


FIGURE 46 – Représentation des slots et des signaux, Source : OpenClassRoom

La figure 46 représente le fonctionnement entre les signaux et les slots. Le widget de l'exemple est un bouton sur une interface. Ce type de widget peut avoir deux signaux différents : le premier est lorsqu'il est "appuyé", c'est à dire que le clic de la souris est en cours, un des comportements usuels est alors son changement de couleur ; le second est celui de l'action décrite par le bouton, c'est-à-dire, lorsqu'il est "cliqué". Dans le cas de notre exemple, le bouton cliqué à une fonction, un slot, qui lui est reliée, appelée *quit()*. On peut imaginer que ce bouton est décrit par une petite croix ou par le mot "Fermer" dessus. L'action sera donc de clore la fenêtre courante.

Chapitre 9

Création du réseau routier et des objets 3D

9.1 Description de la donnée source

Comme vu dans le chapitre 2, la donnée source qui est utilisée pour gérer les trajets est un fichier SIG provenant d'OSM. Pour pouvoir l'utiliser, nous devons y appliquer des traitements pour la visualisation et pour la simulation. Lors de l'import des données, décrite dans le chapitre 7, de nombreux champs sont demandés à l'utilisateur pour pouvoir les utiliser lors du traitement. La donnée OSM des routes comporte de nombreuses informations appelées *tag*. Ces derniers permettent de créer les conversions pour l'affichage 3D et pour la simulation.

9.1.1 Données pour la visualisation

Délimitation des voies

Pour filtrer les éléments linéaires des données d'OSM, le *tag* principal est **highway**. Sous ce nom se trouvent tous les tronçons de routes, des autoroutes aux chemins de forêt en passant par les voies sans issue ou les chemins privés. Afin de ne garder que les éléments nécessaires, un autre filtre est réalisé sur le type. Le tableau 10 recense les principales valeurs prises en compte.

Valeur	Description	Vitesse par défaut
motorway	Autoroute : 2x2 voies	130 km/h
trunk	Voie Rapide ou voie express : 2x2 voies	110 km/h
primary	Route nationale : 2x1 voie	90 km/h ou 50km/h en ville
secondary	Route départementale : 2x1 voie	90 km/h ou 50km/h en ville
tertiary	Route ou rue reliant les villages : 2x1 voie	90 km/h ou 50km/h en ville
tertiary	Route ou rue reliant les villages : 2x1 voie	90 km/h ou 50km/h en ville
motorway_link	Bretelle d'accès ou de sortie d'autoroute : 1 voie	non renseignée
trunk_link	Voie d'accès à une voie rapide ou express : 1 voie	non renseignée
primary_link	Voie d'accès à une route nationale : 1 voie	non renseignée
secondary_link	Voie d'accès à une route départementale : 1 voie	non renseignée
tertiary_link	Voie d'accès à une route de village : 1 voie	non renseignée

Tableau 10 – Principales valeurs prises en compte pour la caractérisation des routes. Pour chacune des routes, des vitesses par défaut sont renseignées. Ce descriptif provient du wiki d'OSM[1]

Nous remarquons dans ce tableau que le nombre de voies est souvent renseigné via le type de route. Les autoroutes sont considérées comme des 2x2 voies, tout comme les voies rapides, tandis que les autres sont des voies 2x1 voie. Ces données nous permettent de dessiner de façon automatique les bordures des routes ainsi que les délimitations entre les différentes voies.

Pour commencer, les bordures sont ajoutées en utilisant le tronçon linéaire source comme référence et en appliquant une translation perpendiculaire a celui-ci pour les placer à 10cm du bord de la route.

Ensuite, le tronçon central est gardé pour faire la délimitation entre les voies lorsqu'elles ne sont pas de sens unique. Dans le cas ou plusieurs voies sont référencées, une division de la largeur est réalisée par le nombre de voies afin de placer les lignes pointillées permettant le dépassement de véhicules (voir figure 47).

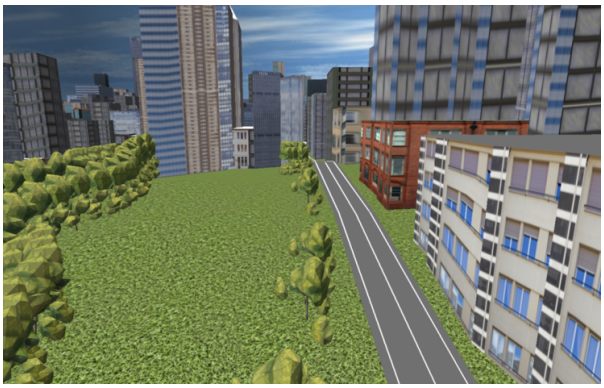


FIGURE 47 – Capture d'écran des voiries avec marquages obtenues avec le logiciel

Dans le cas ou la route change de catégorie, par exemple d'une chaussée à 2 x 1 voie à une autoroute, les délimitations vont aussi changer. De nouvelles lignes pointillées délimitant les voies apparaissent, et une ligne blanche dessine la limite entre les deux sens de circulation (voir tableau 12).

Mobiliers urbains

Pour compléter la vue 3D, nous affichons les données concernant le mobilier urbain. Ces derniers sont référencées aussi dans le fichier contenant les routes d'OSM. Le tableau 11 résume les données que nous utilisons pour placer les différents mobiliers urbains.

Valeur	**Description**
crossing	Passage pour piétons
stop	Panneau de stop
traffic_signals	Feu de signalisation
traffic_signals :direction	Direction du feu de signalisation à appliquer selon le sens du flux.
give_way	Cédez le passage

Tableau 11 – Référence des mobiliers urbains dans le fichier contenant les routes

D'autres mobiliers urbains sont référencés dans ce fichier comme les arrêts de bus, les bornes d'appel d'urgence ou les radars fixes. Dans le cadre de cette étude nous avons choisi ceux présentés dans le tableau, et en particulier les feux de signalisation et les passages pour piétons.

Pour commencer, les feux tricolores sont placés selon les positions (x,y) des points contenus dans le fichier source. Ces objets 3D proviennent d'une bibliothèque libre d'accès. Ils sont

chargés une seule fois en mémoire et sont ensuite dupliqués à chaque position référencée dans le fichier source. Leur direction nous est donnée par la valeur **traffic_signals :direction**.

Les passages pour piétons, quant à eux sont dessinés à chaque indexation dans le fichier route. Nous prenons la largeur de la route et traçons des rectangles de 2,5m perpendiculaires aux tronçons de route avec une inter-distance de 50cm. Ces données sont les minimales définies par l'instruction interministérielle sur la signalisation routière[2]. Bien sûr, ces valeurs sont valables en France, et elles devront être adaptées selon le pays dans lequel la simulation est lancée.

Les panneaux stop sont aussi placés selon leur position (x,y). La direction est calculée en considérant la perpendiculaire au tronçon concerné. Afin de compléter le visuel nous ajoutons le tracé de la bande blanche signalant le stop sur la chaussée. Tout comme les passages pour piétons, cette peinture au sol suit des règles, donc son tracé sera de la longueur de la voie avec une largeur de 50cm.

Pour finir, les cédez le passage seront placés à l'extrémité du tronçon concerné. La peinture au sol est faite avec des carrés de 50cm par 50cm espacés de 50cm tout le long de la voie.

9.1.2 Données pour la simulation

Les données utilisées pour la simulation sont basées sur les informations énoncées dans la partie précédent. Nous allons décrire ici leurs utilisations dans ce contexte.

Le chevelu routier

Comme nous l'avons déjà décrit, le réseau routier est fourni avec une vision assez simpliste. Un tronçon de route correspond à une chaussée pouvant comprendre une voie à sens unique, ou alors deux voies à sens opposés, ou même à plusieurs voies dans le cas des autoroutes ou voies rapides [21]. Visuellement, ces tronçons ne sont donc pas représentatifs de la réalité. Il faut donc en recréer selon les informations décrites sous le tag **highway**. Les références utilisées sont celles du tableau 10. Les conversions mises en place sont illustrées dans le tableau 12. Afin de garder un lien topologique (voir partie suivante) entre les tronçons suivants et prédécesseurs, nous stockons l'identifiant du tronçon source à chaque duplication (voir figure 48). En suivant la même logique nous attribuons la même limitation de vitesse à chacun des sous-tronçons.

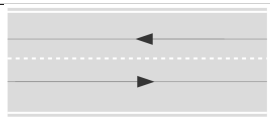
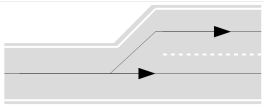
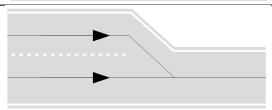
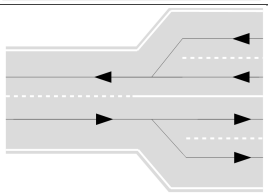
Nombre de voies	Représentation
2 x 1	
Passage de 1 voie à 2 voies	
Passage de 2 voies à une voie	
Passage de 2 x 1 voie à 2 x 2 voies	

Tableau 12 – Création de sous tronçons à partir d'un seul tronçon source

2. Instruction interministérielle sur la signalisation routière Livre 1, " 7e partie, Marques sur chaussées"

Comme on peut le remarquer dans ce tableau, le passage d'une voie à deux voies, ou le contraire, engendre la création d'un tronçon intermédiaire. En effet, afin de faire le lien entre un tronçon unique et deux autres, nous utilisons un tronçon avec un angle à 45 ° pour créer un cheminement entre le premier tronçon et la deuxième voie. Le cas le plus courant est celui d'une chaussée à 2x1 voie passant à une voie rapide ou autoroute à 2x2 voies (4ème cas dans le tableau). Dans ce cas là le décalage est fait sur la droite du tronçon, en considérant le sens de circulation. En prenant en compte une conduite à droite, la route principale sera celle longeant le bord de la chaussée et la voie de gauche sera celle de dépassement. Elle sera utilisable si la voie principale est déjà occupée. Pour différencier les deux, un nouveau champ est ajouté dans la table TRoute (voir figure 48). La valeur par défaut choisie à zéro correspond au chemin principal et les voies secondaires, dites de dépassements, sont notées à 1 ou plus dans le cas où nous avons plus de voies dans la même direction. Le fait d'avoir plus de 2 voies n'est pas mis en place actuellement mais nous gardons cette possibilité dans le code.

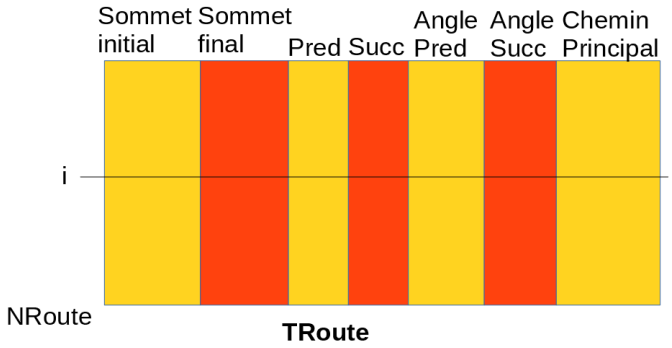


FIGURE 48 – Structure de la table Route : afin de garder la continuité entre les tronçons nous ajoutons l'identifiant du tronçon principal dans la table, noté *ID tronc Principal*. De plus le champ *Chemin Principal* permet de connaître la voie principale et les voies secondaires.

9.2 Création de la topologie

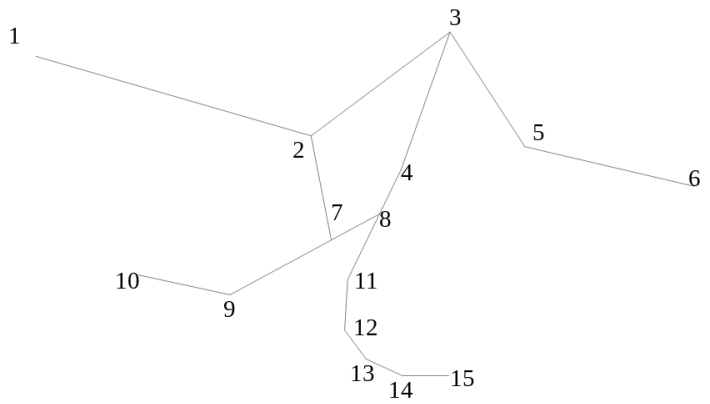
La création de la topologie est basée sur les principes apportés par la modélisation HBDS introduits dans le chapitre 3.

9.2.1 Application sur le réseau routier

Pour simplifier sa conceptualisation, chaque tronçon de route est assimilé à un arc et ses extrémités à des sommets (voir chapitre 3) [22]. Ainsi, chaque arc a deux sommets, et chaque sommet peut avoir un arc ou plus (cas des intersections). En appliquant des algorithmes recherchant les prédécesseurs et les successeurs de chaque arc une classification topologique des tronçons de route est obtenue. Cet algorithme est appliqué une seule fois, lors de l'import de la donnée source. Il a comme principe de rechercher pour chaque tronçon son prédécesseur et son successeur (voir annexe F). Pour cela, les coordonnées d'un sommet initial est testé avec les coordonnées des autres tronçons. Si aucun tronçon n'est trouvé, nous sommes face à une voie sans issue, il n'y aura donc pas de tronçon successeur. Nous faisons de même sur les sommets finaux. Bien sûr, des vérifications sont faites dans cet algorithme pour ne pas avoir de doublons.

Afin de concrétiser cet algorithme, prenons une ville fictive avec son réseau routier. Une table contient chaque arc avec, pour chacun,son sommet initial et son sommet final. Une table

de classement des arcs est créée pour avoir la succession géographique des tronçons les uns après les autres. Le schéma explicatif 49 illustre ce classement.



Sommet initial	Sommet final
1	2
2	3
3	5
5	6
2	7
3	4
4	8
8	7
7	9
9	10
8	11
11	12
12	13
13	14
14	15

FIGURE 49 – Schéma d'un réseau routier. Les arcs de la première ligne du tableau et de la suivante se suivent car le sommet final du premier est le même que le sommet du deuxième. Il en est de même pour l'arc ayant comme sommet initial le 4 et comme sommet final le 8, les arcs connectés à ce dernier sont les arcs 8 – 11 et 8 - 7

Avec ce type de classement, un itinéraire peut être généré selon un point de départ et un point d'arrivée.

9.2.2 Génération des itinéraires aléatoires

Afin d'avoir de multiples trajets pendant la simulation, nous créons des itinéraires cibles. Ces itinéraires ne prennent pas en compte la simulation lors de leur création, mais peuvent être modifiés lorsque la simulation est en cours. Leurs générations suivent quelques règles :

— Ils commencent tous par un sommet initial différent
— Ils ont un nombre aléatoire de tronçon maximum à atteindre
— Si le parcours arrive sur un tronçon n'ayant pas un tronçon successeur, on recherche un autre chemin à partir du sommet précédent.

Chaque trajet est stocké dans un tableau. Le nombre de trajets par défaut est de vingt mais peut être choisi par l'utilisateur lors du lancement de la simulation. Ce nombre est modifiable via l'interface au démarrage de la simulation. Comme expliqué précédemment le trajet aléatoire est régi par quelques règles dont une à propos du nombre de tronçons à parcourir. Suite à de nombreux tests itératifs pour obtenir une simulation de circulation urbaine fluide, ce nombre a été fixé à 50 tronçons (plus ou moins un facteur aléatoire généré par le programme). En effet, plus le trajet est long, plus la simulation est lisible.

Chapitre 10

Simulation discrète et continue

Réaliser une simulation informatique consiste à développer un modèle en vue de représenter un phénomène physique que nous pouvons rencontrer dans la réalité et qui est généralement complexe. Ainsi, parmi les différents phénomènes représentables par cet outil, la modélisation du trafic routier est un outil d'aide à la décision qui semble intéressant de développer pour les différents métiers concernés (collectivités locales, mairies, ...), d'autant plus en utilisant une modélisation 3D de la ville d'intérêt.

Pour ce faire, la simulation du trafic routier comprend plusieurs volets au-delà de la représentation du chevelu routier. En effet, le déplacement des véhicules le long d'un itinéraire comporte la nécessité de définir différents paramètres. La détermination de l'itinéraire est en elle-même déjà une part importante de ce travail puisqu'il faut prendre en compte les aspects de distances, de pentes, d'accessibilité et de saturation des parties du réseau à parcourir. Le choix final de l'itinéraire le plus performant devra donc tenir compte de l'importance donnée à chaque critère et de leur hiérarchisation influençant de trajet suivi.

Une fois le trajet déterminé, le mouvement des véhicules doit répondre à certaines règles, physiques d'une part, et usuelles d'autre part. Ces deux aspects du déplacement mettent en jeu des caractéristiques et des modélisations très différentes. La première implique des processus physiques descriptibles de manière continue par des équations paramétrées selon les connaissances effectives du milieu. La deuxième, quant à elle, implique de répondre à une succession de tests permettant de faire coïncider le déplacement d'un véhicule donné en fonction d'autres. C'est dans cette partie qu'interviendront certaines règles de conduite dictées par le Code de la route. Cette partie du travail est également celle qui permet de poser des contraintes particulières telles que la présence de travaux ou l'intensification du trafic selon le moment de la journée (heure de pointe vs. nuit), de la semaine (semaine vs. week-end) ou de l'année (départs et retours de vacances).

La diversité des problématiques de modélisation liées à la simulation du trafic implique de choisir la modélisation la plus adaptée à chaque partie du programme et de permettre la communication constante entre chacune de ces parties. Avant d'entrer dans les détails de la simulation au sein de ce travail, il convient donc de préciser rapidement les différents types de simulation existants pour mieux cerner les différences de fonctionnement qui leur sont propres.

10.1 Définition des différents types de simulation

Modéliser un système, un phénomène, et surtout son évolution probable dans le temps, nécessite de simuler des phénomènes plus ou moins complexes via un programme informa-

tique. Du point de vue de la simulation du trafic routier, trois grandes catégories peuvent être différenciées :

— La simulation continue,
— La simulation discrète,
— La simulation mixte.

10.1.1 La simulation continue

La simulation continue peut être définie comme une simulation à pas de temps régulier. Ainsi à chaque pas de temps peut avoir lieu un événement. Ce type de simulation peut parfaitement être utilisée pour des phénomènes à pas de temps fixe comme un feu de signalisation qui a un rythme donné, ou encore l'écoulement de fluide. Elle est utilisée pour simuler un phénomène continu. Dans notre cas la simulation continue est basée sur le temps, à chaque intervalle de temps nous exécutons un déplacement des véhicules sur les tracés prédéfinis. Ce pas de temps peut varier selon le choix de l'utilisateur. En effet, nous lui laissons la possibilité d'accélérer ou de ralentir la simulation. Il y a donc une variable permettant de changer la vitesse de cette simulation, cela permet dans un sens de mieux comprendre les phénomènes, via un ralentissement, et dans l'autre sens, de faire des simulations sur des périodes plus longues en accélérant les déplacements.

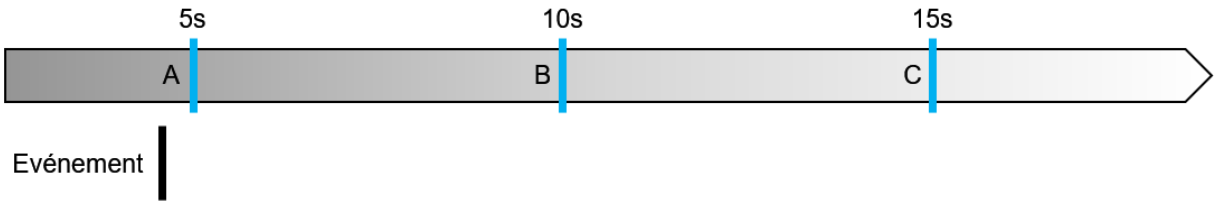


FIGURE 50 – Principe de l'évolution par pas de temps régulier dans la simulation continue

10.1.2 La simulation discrète

La simulation discrète, quant à elle, est basée sur une succession d'événements, n'ayant pas forcément une régularité temporelle entre eux. Cet outil est idéal pour représenter certains phénomènes comme par exemple : les collisions en voiture, les interactions avec les mobiliers (feu de signalisation, stop...)

FIGURE 51 – Principe de l'évolution par événement dans la simulation discrète

Cette simulation est basée sur trois éléments [23] :

— Des processus,
— Un échéancier ("sequencing set" ou SQS [24] [25] [26] [27]), qui est une liste où sont stockés les différents processus selon leurs horloges respectives,

— Des coroutines, composées de trois éléments : un "Local Sequence Counter" (LSC) avec l'adresse du prochain point de reprise, une éventuelle séquence d'initialisation
— et enfin un "body" qui est la succession d'instructions (avec des points de reprises ou non) et dont les processus sont capables de gérer divers événements (mise en attente, activation,...).

10.1.2.1 Les coroutines

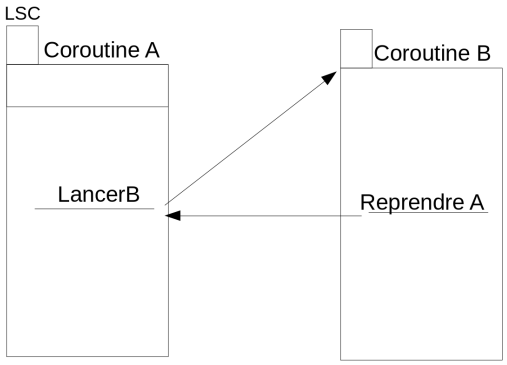


FIGURE 52 – Schéma d'une coroutine : La coroutine A appelle B au bout de N lignes de code puis la coroutine relance la coroutine A à l'endroit où le processus s'est détaché.

Les coroutines peuvent avoir des configurations bien différentes, amorçant le concept de travail en parallèle (multithreading).

10.1.2.2 Les processus

Un processus dans une simulation discrète est un objet (d'une classe de processus) qui évolue au cours de la modélisation à la suite d'événements, sans avoir de pas de temps régulier. Cette succession d'événements est générée par les processus (sur eux-mêmes ou sur ceux avec qui ils interagissent). Un processus agissant comme une coroutine, il est également composé :
— d'un BODY,
— d'un LSC qui mémorise là où s'est arrêté le dernier déroulement de son BODY,
— d'une horloge qui évolue de façon discrète (et non pas continue).
Au cours de ces appels, le processus peut passer par quatre états :
— actif (plusieurs processus ne peuvent être simultanément actifs),
— suspendu (le processus est rangé dans l'échéancier et attend d'être activé),
— passif (la date de réactivation du processus n'est pas connue/prévue, il ne fait donc pas partie de l'échéancier),
— terminé (état irréversible pour le processus).

Les processus peuvent agir via huit primitives, trois concernent le processus lui-même, mais ont des conséquences sur d'autres, et cinq agissent sur d'autres processus (soit dans l'état suspendu, soit passif) [28] [29] :
— SUSPENSION : le processus actif est envoyé dans le SQS (selon son horloge interne à laquelle on ajoute un dt) et le premier processus dans le SQS est alors automatiquement activé,
— PASSIVATION : le processus actif est mis dans l'état passif, son horloge n'a donc plus d'horloge valide tant qu'aucun autre processus ne le réactive ; le premier processus dans le SQS est alors automatiquement activé,

— TERMINAISON : le processus est envoyé dans l'état terminé, ce qui est irréversible ; le premier processus dans le SQS est alors automatiquement activé,
— RETARD : le processus actif contient, dans son BODY, une instruction pour retarder, d'un délai dt, un autre processus,
— ANTICIPATION : le processus actif contient, dans son BODY, une instruction pour avancer, d'un délai dt, un autre processus,
— ANNULATION : le processus actif contient, dans son BODY, une instruction pour mettre un autre processus en état passif,
— SUPPRESSION : le processus actif contient, dans son BODY, une instruction pour mettre un autre processus (déjà à l'état suspendu ou passif) en état terminé ce qui est irréversible,
— REACTIVATION : le processus actif contient, dans son BODY, une instruction pour mettre un autre processus (déjà à l'état passif) en état actif à la date courante ou à une date donnée.

Ces successions d'états sont rassemblées et schématisées dans la figure ci-après.

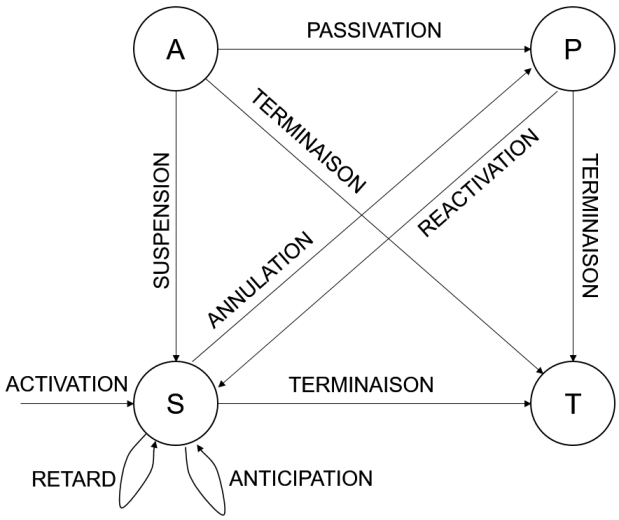


FIGURE 53 – Représentation schématique des relations entre les primitives lors d'une simulation

10.1.3 La simulation mixte

Enfin, la simulation mixte, comme son nom l'indique, se situe à l'interface entre la simulation continue et la simulation discrète. En effet elle permet de représenter, à la fois des phénomènes périodiques mais également des événements ponctuels via l'utilisation du "sequencing set" défini précédemment.

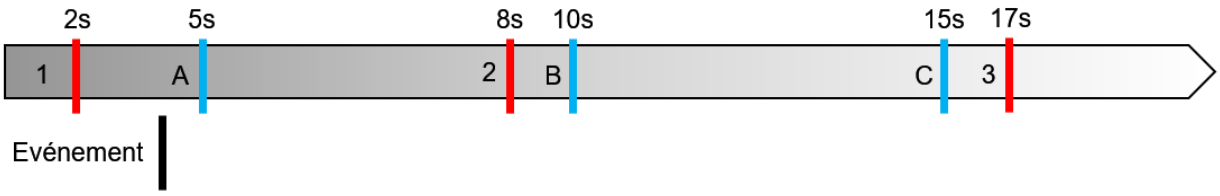


FIGURE 54 – Principe de l'évolution temporelle dans la simulation mixte

10.2 La simulation liée à la gestion du trafic urbain

Simulation discrète, continue ou mixte, à chacune ses qualités et ses applications. Mais quelle est celle qui répondra aux attentes de la modélisation de la circulation urbaine ?

10.2.1 Pourquoi le choix de la simulation ?

De nombreux éléments entrent en jeu pour simuler le trafic urbain et chacun d'entre eux a sa propre gestion temporelle. De plus certains interagissent entre eux, ce qui est donc également à prendre en compte. Ce sont ces éléments et ces relations que nous allons lister et définir dans cette partie.

Nous pouvons différencier trois grands sous-ensembles :

— Le premier élément qui vient à l'esprit pour une gestion temporelle lors d'une simulation de trafic est le déplacement des éléments mobiles, en particulier les véhicules et les piétons. En effet, il est nécessaire de travailler sur le déplacement de l'objet en fonction de sa vitesse mais également du taux de rafraîchissement de la fenêtre de visualisation.

— Le second élément à considérer est la gestion des mobiliers environnants qui peuvent avoir leur propre "chronomètre".On peut noter en particulier les feux de signalisations ou les réverbères.

— Enfin nous pouvons identifier les interactions entre les mobiliers et les véhicules. Ces mobiliers sont variés et on peut retrouver, en plus des feux de signalisation et réverbères, les panneaux de signalisation, les passages pour piétons,...

La simulation choisie doit alors prendre en compte la circulation du trafic ainsi que les mobiliers ayant leur propre "horloge" mais également ceux ayant des interactions avec les véhicules ; c'est pour cela que nous avons opté pour la simulation mixte, choix expliqué dans la partie suivante.

10.2.2 Quelle(s) simulation(s) pour ce projet ?

A partir des informations temporelles que nous avons décrites dans la partie précédente, nous avons pu identifier les intérêts des simulations continues et discrètes. Dans les parties ci-après nous allons présenter ce que chacune de ces deux simulations peut apporter pour la modélisation du trafic urbain ainsi que mettre en évidence l'intérêt d'avoir travaillé sur une simulation mixte pour y mêler tous ces avantages.

La simulation continue pour le trafic urbain

La simulation continue semblait la plus adéquate pour représenter les événements chronologiques liés aux feux de signalisation ou aux réverbères. Il en est de même pour déplacer des véhicules sur des tronçons de route avec une vitesse constante. Le temps de la simulation continue est lié au rafraîchissement de l'affichage, c'est à dire que chaque pas de temps correspond de base à $1/24^{\text{ème}}$ de seconde, en référence à l'affichage de 24 images par seconde. Ajouté à ce pas de temps, une variable, mise en facteur à ce pas de temps, permet de contrôler la vitesse. Ces deux paramètres nous permettent de gérer les différents mobiles ou objets durant la simulation continue :

Les véhicules se déplacent selon leurs itinéraires à chaque pas de temps,

Les feux de signalisation passent du rouge au vert toutes les 30 secondes,

Les réverbères s'allument dès que la nuit tombe et s'éteignent quand le jour se lève,

Les piétons utilisent les passages pour piétons toutes les minutes avec un facteur aléatoire pour que les passages pour piétons ne soient pas utilisés tous en même temps.

La simulation discrète pour la gestion des évènements liées à la circulation

Sachant que nous avons différentes simulations continues pouvant se croiser du fait de notre contexte, il y a donc des interactions entre elles, non liées directement au pas de temps. C'est pour cela que nous utilisons la simulation discrète pour nous permettre de modéliser les événements liés aux interactions entre mobiles (véhicules et piétons) ou entre les mobiles et les mobiliers. En effet, représenter le fonctionnement et l'interaction entre une voiture, un vélo, un piéton ainsi qu'un feu de signalisation nécessite de mettre en place quatre coroutines, une pour chacun de ces éléments. Mais traiter de la simulation du trafic urbain nécessite de prendre en compte de nombreux mobiles, et il est impensable de construire une coroutine pour chacun d'entre eux, d'autant plus que leurs structures sont similaires pour chacune de ces quatre catégories, seules certaines propriétés et relations différent au sein d'un même sous-ensemble. Les deux processus suivants illustrent le fonctionnement des coroutines pour la gestion des feux de signalements et des passages pour piétons :

```
ProcessusPiétons(objVoiture, listePassagePietons, listeDtPassagePietons) :[[ ©Pour chaque
piétons© {i : 0 nbPassagePietons ? ©le passage pour piétons est inoccupé©
reactiverVoiture(numVoiture) | ©le passage pour piétons est occupé©
TronçonEnArret(numTronçon)¿; }i ]]

ProcessusSignalisation (objVoiture, listeSignalisations, listeDtSignalisation) :[[ ©Pour
chaque feu de signalisation© {i : 0 nbSignalisations dt = 30 ? ©le feu passe au vert©
reactiverVoiture(numVoiture) | ©le feu passe est rouge© TronçonEnArret(numTronçon)¿;
}i ]]
```

La simulation mixte pour répondre aux attentes

Au cours des deux parties précédentes, nous avons donc identifié que certains éléments nécessitaient de passer par une modélisation via la méthode de la simulation continue (en particulier les déplacements des véhicules et la gestion des feux de signalisation) et d'autres via celle de la simulation discrète (avec la gestion des interactions entre mobiles). Ainsi, comme nous avons défini au début de ce chapitre, la simulation mixte est à l'interface entre ces deux simulations. Ce choix semble le plus approprié pour notre étude. Il va falloir cumuler ces deux simulations en les faisant interagir l'une avec l'autre. De plus il ne faut pas oublier le moteur 3D qui nous permet d'afficher les résultats à l'écran. Lui est basé sur le pas de temps de base, c'est-à-dire 24 images par seconde. Il doit donc être en lien avec ces deux types de simulation. La mise en place de cette simulation mixte sera expliquée dans la section suivante.

10.3 La simulation au cours de cette étude

Pour cette étude, différentes simulations ont été étudiées puis mises en œuvre pour la représentation réaliste du trafic urbain. Dans cette partie nous expliquerons tout d'abord la structure générale de la simulation temporelle puis celle permettant la simulation de l'éclairage de la scène avant de présenter comment s'interfacent les moteurs de simulation discrète et continue au cours de cette étude. Mais avant cela, illustrons succinctement les différents processus qui seront pris en compte en adoptant la forme des coroutines expliquées précédemment et leurs relations.

10.3.1 Les éléments à considérer

Afin de gérer l'animation et la simulation à l'intérieur du modèle 3D des classes de processus ont été créées. Elle régissent le déplacement, ainsi que les divers objets pouvant changer d'état au cours du temps. Au cours de ce travail, quatre classes de processus sont traitées :

— Le déplacement des véhicules,
— L'occupation ou non des passages pour piétons,
— Le changement d'état des feux de signalisation,
— Le changement d'état des tronçons suivants dans l'itinéraire.

Les relations qui existent entre ces différents éléments sont reportées sur la figure suivante.

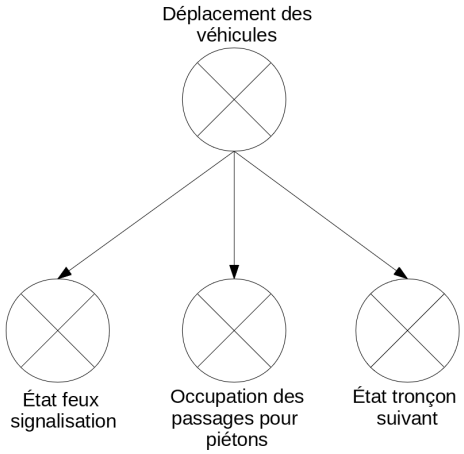


FIGURE 55 – Représentation des relations entre les divers processus présentés au cours de ce travail de recherche

Comme le souligne l'illustration 55, la structuration des relations entre les différents processus mis en évidence peut être réalisée pour un très grand nombre d'objets (ici appartenant à quatre classes de processus). Il semble alors évident qu'un grand nombre de classes de processus peuvent alors être également intégrées à cette modélisation, comme par exemple :

— la création d'une hyperclasse Véhicules qui pourra intégrer, à la fois la classe Voiture actuelle, mais également d'autres éléments comme les véhicules de service (pompiers, police, ambulance, éboueurs,...), les trains, les véhicules militaires (tanks, camions blindés,...), mais également des véhicules aquatiques (paquebots, remorqueurs,...) ou aériens (avions, hélicoptères,...),
— la prise en compte de phénomènes naturels comme les éboulements, les incendies ou les inondations,
— la mise en place de nouveaux mobiliers (comme par exemple des passages à niveau),...

Ces perspectives d'amélioration de la modélisation via la création de nouveaux processus sera présentée au cours du chapitre 15.

10.3.2 La gestion temporelle

Le gestion du temps est primordiale lorsqu'on réalise des simulations. Si l'échelle de temps est inadéquate, le phénomène ne peut pas être compris, voire visualisé. Prenons par exemple le cas de la simulation dans le domaine de la géologie. Si l'on souhaite simuler la dérive des continents, il faudra choisir un pas de temps adapté au phénomène et donc accélérer les processus. Il en est de même pour les phénomènes quasi instantanés comme la collision des photons par exemple. Leurs simulations doivent être ralenties pour mieux appréhender les interactions entre les particules.

Nous avons la même problématique pour la simulation de trafic urbain. Par défaut, le temps de la simulation est celui de nos montres, mais le pas de temps de simulation peut être modifié à la volée par l'utilisateur. Comme expliqué précédemment, cela affecte uniquement l'affichage, les simulations sont donc accélérées ou ralenties. Il faut aussi ajouter que le changement de pas de temps n'influence pas non plus le nombre d'images par seconde de l'affichage 3D, fixé à 24 images par seconde, pour avoir un rendu toujours fluide à l'écran.

Gérer le temps est donc une des préoccupations principales dès lors que l'on choisit de réaliser une simulation, et en particulier dans le cas des trafics urbains. Nombreux sont les éléments qui vont se déplacer et les relations entre les divers objets (fixes ou mobiles) sont multiples. Ainsi, nous allons définir dans ce paragraphe les éléments indispensables à prendre en compte pour la gestion temporelle (et donc de vitesse) des véhicules et des piétons au cours de cette simulation. Tout d'abord, traitons du cas des piétons. Si on estime la vitesse moyenne d'un individu à 4km/h, cette vitesse est dépendante d'un grand nombre de facteurs :

L'individu : en effet, selon l'âge ou la taille il existe une variabilité de la vitesse de marche, ainsi des personnes âgées ou de jeunes enfants n'auront pas (forcément) la même allure qu'une personne sportive d'âge moyen,

Les groupes : ainsi si des personnes se déplacent en groupe, il n'est pas évident que ces individus se déplacent à leur allure habituelle. Personnes avec de jeunes enfants, groupe d'amis, foules importantes dans la rue (comme les jours de soldes ou dans les périodes de fêtes comme Noël par exemple)... autant de variations qui peuvent entraîner des ralentissements dans la marche,

Le but de la sortie : la plus forte variabilité est probablement liée à cet élément. En effet des personnes peuvent choisir de faire un footing, d'autres courent après le bus, certains marchent à leur cadence, d'autres marchent les yeux rivés sur leur smartphone, mais on rencontre également des gens qui flânent, font du "lèche-vitrines", ou encore font des arrêts touristiques,

La météo : il n'est pas rare de constater que selon les conditions climatiques les gens ne circulent pas à la même vitesse, ceci peut donc également être pris en compte pour cette simulation.

Tous ces éléments sont donc à considérer pour une modélisation réaliste du trafic piétonnier. Le second élément à étudier est la modélisation des vitesses de déplacement des véhicules. Sont à prendre en compte :

Le type de véhicule : effectivement selon si le véhicule est motorisé (voiture, camion, tracteur, moto,...) ou non (vélos, voitures à chevaux, trottinettes,...) les vitesses ne seront pas comparables, de même les conditions de circulation et itinéraires peuvent varier,

Le mode de conduite : là encore une grande variabilité peut être observée, comme les dépassements de vitesse (ou à l'inverse les conducteurs à "conduite lente" comme les jeunes conducteurs par exemple), les augmentations de vitesse lors de dépassements, les arrêts fréquents (bus, tramways ou taxis par exemple),...

Les changements de vitesse liés aux conditions météorologiques : en effet en cas de verglas ou de pluies torrentielles les vitesses moyennes ne seront pas les mêmes qu'en cas de météo plus clémente.

Ces listes, non exhaustives, d'éléments peuvent donc être rapportées à la modélisation du trafic urbain afin de simuler au plus juste la circulation dans la ville.

10.3.3 Gestion de la lumière

Lors d'une simulation ayant une longue durée, il peut exister une influence des facteurs environnementaux, tel est le cas du cycle nycthéméral. Son influence est d'autant plus grande dans notre cas car, la nuit, les véhicules sont moins nombreux, roulent avec les feux allumés, les piétons sont quasi-inexistants tout comme les bus ou les vélos. La simulation doit prendre cela en compte afin d'être au plus proche de la réalité.

Afin de visualiser ce phénomène, un module gérant l'éclairage a été mis en place avec un calcul de position réelle du soleil selon une date choisie et une heure. Vu que nous sommes dans un système géoréférencé, l'azimut, la hauteur et l'angle du soleil peuvent être calculés précisément. Cette problématique astronomique n'est pas nouvelle. Depuis des siècles les astronomes cherchent à déterminer la position des astres et donc du soleil par les calculs. Tout d'abord intéressons nous au calendriers car les calculs sont basés sur ces dernier pour déterminer l'azimut et la déclinaison du soleil. Voici quelques exemples de ces calendriers :

Calendrier Chaldéen ou Babylonien : -4000 avant J.C., les Chaldéens ou Babyloniens connaissaient le mouvement des planètes, les éclipses, la précision des équinoxes. Ils utilisaient le gnomon et le cadran solaire. La position du soleil était devinée selon les douze parties d'un zodiaque ; Bélier, Taureau, Cancer, ... Aucune prédiction n'était possible, seulement de l'observation.

Calendrier Maya : -3113 avant J.C., les Mayas construisaient leur calendrier à partir de la date légendaire de 3113 av. J.-C., et ils utilisaient des unités de temps plus importantes telles que le baktun, période cyclique comprenant 20 *katuns*, (c'est-à-dire 400 années mayas, correspondant à 394 de nos années).

Calendrier Chinois : -2000 avant J.C., les astronomes chinois ont divisé l'année en 12 mois alternativement de 29 et 30 jours et pour rester en accord avec les saisons, ils ajoutaient un mois de temps en temps.

Calendrier Grec : -433 avant J.C, le calendrier grec fut d'abord purement lunaire, avec des mois de 30 jours, puis avec des mois alternés de 29 et 30 jours. Ceci donnait une année de 6 x 30 + 6 x 29 = 354 jours. La solution fut une période de 19 ans nommée cycle de Méton qui indique que 19 ans = 235 lunaisons. Après 19 ans, les phases reviennent à l'initial. En 130 avant notre ère, Hipparque fut le premier à découvrir que l'année est plus courte que 365,25 jours. Sur 4 cycles de Callipe, il retrancha encore un jour. Après quelques calculs la durée d'une lunaison moyenne est de 29 jours 12 h 44 min 2 s ce qui correspond à moins d'une seconde d'erreur par rapport à la valeur calculée de nos jours.

Calendrier Persan : 1079, les Perses ont adopté un calendrier solaire, ce qui est beaucoup plus évident à mesurer qu'un calendrier lunaire du fait de la grande régularité du mouvement apparent du Soleil, si l'on parvient à mesurer la durée de l'année au niveau des Tropiques. Le nombre de jours ajoutés en 33 ans est de 8, donc la durée moyenne de l'année s'écrit aussi 365 + 8 / 33 = 365,24242424 jours.

Calendrier Julien : -46 avant J.C., pour calculer le calendrier, Jules César fit venir l'astronome Egyptien Sosigène. Ils choisirent une durée de l'année moyenne de 365,25 jours, bien qu'un siècle plus tôt Hipparque savait déjà que cette valeur était trop forte. On a donc un cycle comprenant 3 années de 365 jours, suivies d'une année de 366.

Pour calculer la position du soleil, nous nous basons sur le calendrier Julien. Pour cela nous avons besoin de l'heure, du jour, du mois et de l'année et nous avons déjà, grâce à l'aspect SIG du modèle développé, les coordonnées de la ville. A partir de toutes ces informations nous calculons l'heure sidérale, nous permettant d'avoir l'angle horaire du soleil. Cet angle nous permet ensuite de connaître l'altitude de l'astre, et enfin son azimut (voir figure 56).

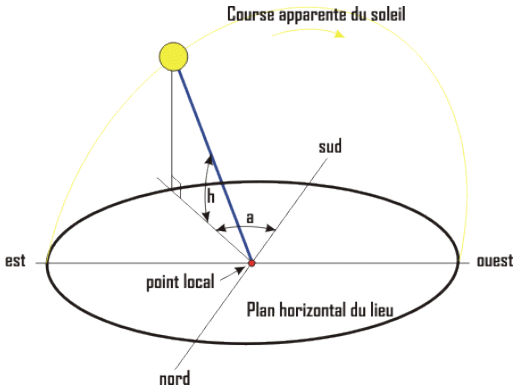


FIGURE 56 – L'azimut solaire (a) est l'angle horizontal formé par le plan méridien (axe nord-sud) et le plan vertical du vecteur "point local->soleil". Le signe de l'azimut est le même que celui de l'angle horaire. La déclinaison (Dec) est l'angle que forme le vecteur "centre de la terre->soleil" et le plan équatorial de la terre. L'angle horaire (Ah) mesure le mouvement du soleil par rapport à midi qui est l'instant où le soleil passe au plan méridien du lieu (zénith). Cet angle horaire est négatif si le temps solaire est inférieur à 12 h. a = ArcSin((Cos(Dec) x Sin(Ah)) / Cos(h)), Source : Delta-Calor

Ces paramètres sont appliqués à chaque pas de temps sur le moteur 3D. L'utilisateur peut à tout moment changer le mois ou l'heure.

La figure 57 illustre la ville de New York en plein jour et lors du coucher.

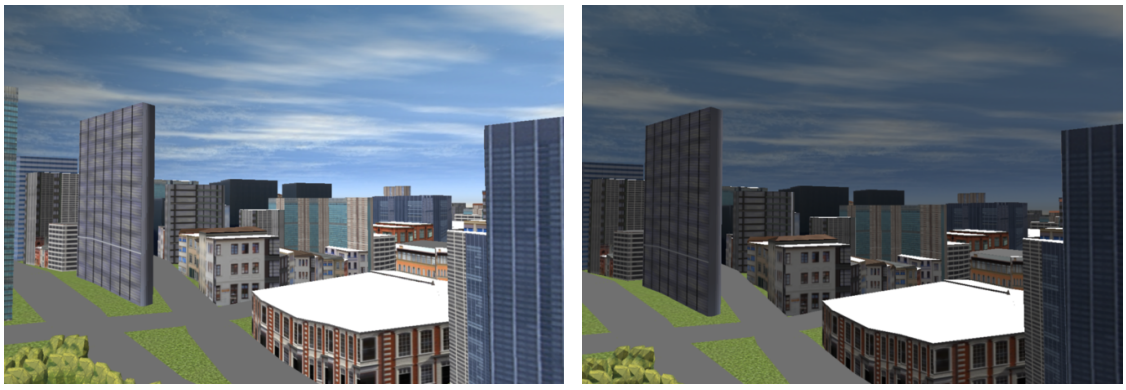


FIGURE 57 – Gestion de la lumière du jour

L'influence du cycle jour/nuit sur le trafic n'est pas encore implémenté dans le logiciel et sera étudié dans le chapitre consacré aux perspectives de ce travail (chapitre 15).

10.3.4 Interfacer les moteurs de simulations discrète et continue

Au cours de cette section, nous allons, tout d'abord, présenter un certain nombre d'exemples permettant d'illustrer l'intérêt d'interfacer des éléments de simulation continue et des éléments de simulation discrète avant d'expliquer comment vont interagir ces deux types de simulations,

pour obtenir ce qu'on appelle un moteur de simulation mixte, dans le cadre de ce projet de recherche.

10.3.4.1 Structurer la simulation mixte

Afin de structurer un moteur de simulation mixte, il est primordial de commencer par identifier différents éléments de simulation que nous aurons à modéliser. Au cours de ce paragraphe seront exposés un certains nombre d'exemples qui mettront en avant l'usage de la simulation mixte. Ces exemples seront illustrés grâce à la représentation dite des "graphes de Pétri"[30]. Il faut ajouter à tous ces exemples le temps de circulation sur chaque tronçon, temps non définissable car les tronçons sont tous de tailles différentes.

Cas d'un feu de signalisation sur un trajet : Le trajet comprend un feu de signalisation. Deux cas s'offrent au véhicule, soit le feu est vert, et il passe avec aucune attente soit le feu est rouge et il attend 30 secondes (voir figure 58). Il peut avoir des valeurs intermédiaires si la voiture arrive après le passage au rouge.

FIGURE 58 – Cas d'un trajet avec un feu de signalisation

Cas d'un feu de signalisation et d'un stop sur un trajet : Prenons le cas où le trajet se complexifie avec la présence d'un feu de signalisation et d'un stop. Le trajet peut prendre 3 secondes en plus du déplacement normal lorsque le feu est vert et que le tronçon qui suit le stop est libre. Dans le cas où le feu et rouge et que le tronçon suivant le stop est libre, le temps supplémentaire peut être de 33 secondes (3 + 30). Si le tronçon après le stop est occupé, le temps peut aller jusqu'à 210 secondes (voir figure 59).

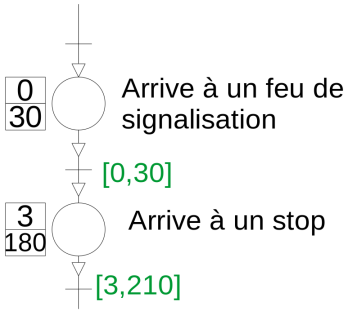


FIGURE 59 – Cas d'un trajet avec un feu de signalisation suivi d'un stop

Cas d'un stop et d'un passage pour piétons sur un trajet : Dans ce cas, le trajet comporte un stop suivi d'un passage pour piétons. Dans tous les cas, le trajet sera "retardé" de 3 secondes du fait de la présence d'un panneau stop. Si le tronçon est occupé dans son temps maximum et qu'un piéton traverse ensuite, le temps à rajouter au déplacement sera de 195 secondes (voir figure 60).

Cas avec toutes les interactions possibles : Considérons la circulation d'une voiture sur un trajet ayant, à la suite un feu de signalisation, un passage pour piétons, un stop

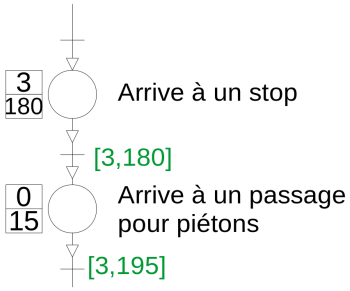


FIGURE 60 – Cas d'un trajet avec un stop suivi d'un passage pour piétons

et un cédez le passage. Le graphe de Pétri de la figure 61 illustre les temps minimums et maximums pour le passage de cette voiture. Dans le cas où tout se passe bien c'est-à-dire que le feu de signalisation est au vert, qu'il n'y a pas de piéton, que les tronçons situés après le stop ne sont pas occupés ainsi que pour le cédez le passage, la voiture aura attendu 3 secondes. Dans le cas contraire, le mobile peut attendre jusqu'à 305 secondes (5min05). Le temps d'attente pourrait même être infini en considérant que les tronçons situés après le stop et/ou le cédez de passage soient toujours occupés.

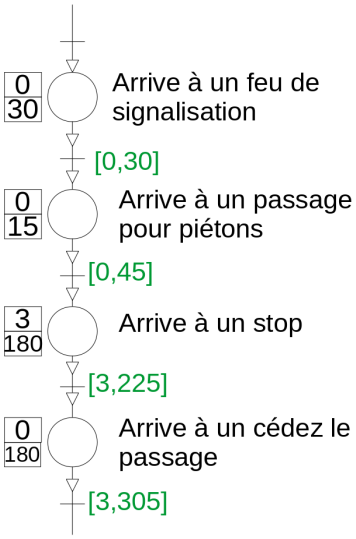


FIGURE 61 – Succession de processus pour la circulation d'une voiture

10.3.4.2 La réalisation du moteur de simulation mixte

L'affichage 3D est constant dans le temps, nous avons fixé son pas de temps et il n'est pas modifiable par l'utilisateur. Il est donc notre base temporelle pour les différentes interactions. La simulation continue est aussi régulière mais elle peut être accélérée ou ralentie par l'utilisateur. Son point d'appel est lancé à chaque image affichable à l'écran, c'est-à-dire toutes les $24^{\text{ème}}$ de seconde. Ayant mis en place cela nous créons une interaction avec la simulation discrète. Ses processus sont appelés à chaque pas de temps pour contrôler l'état des mobiles et des mobiliers. Le schéma 62 illustre ces interactions.

Pour illustrer son fonctionnement, imaginons que la scène est placée dans une bande dessinée comme illustré dans la figure 63. Chaque vignette correspond à une image affichée à l'écran, à laquelle nous ajoutons la référence du temps. Pour la première le temps est de 0 seconde, suivi de la deuxième qui se trouve à $1/24^{\text{ème}}$ de seconde. Entre les deux vignettes, les différentes simulations s'interfacent. Pour commencer, le moteur 3D, gérant le temps, appelle le moteur de

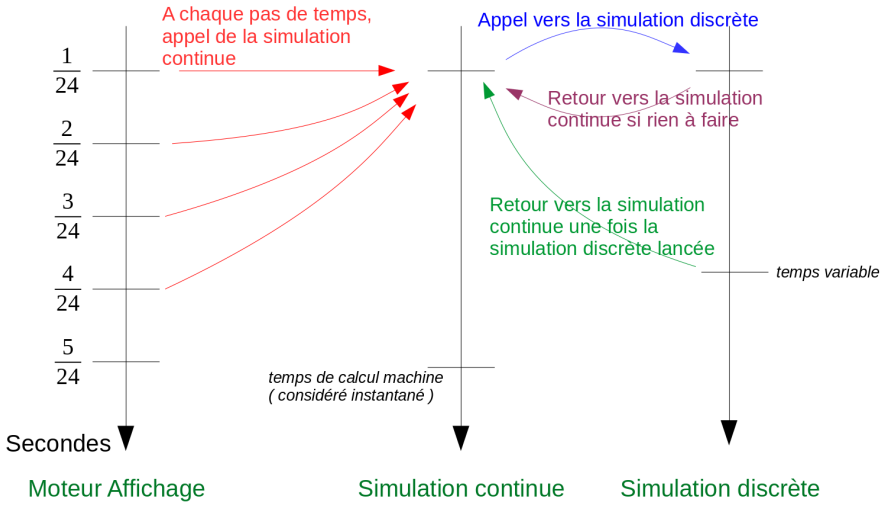


FIGURE 62 – Schéma d'interaction entre les différentes simulations

simulation continue. Celui-ci a pour but de faire avancer le temps dans la simulation mixte, c'est à dire qu'il va incrémenter le temps de changement d'état des feux de signalement et de même pour les passages pour piétons. Son rôle principal est aussi de déplacer les véhicules le long des tronçons. A ce moment là la simulation continue passe le relais à la simulation discrète pour connaître les états des feux de signalisations, des passages piétons et des tronçons constituants les trajets. Selon les retours, le déplacement sera possible ou non et l'affichage de la seconde image sera réalisée. Le moteur 3D reprend la suite pour de nouveau appeler la simulation continue, etc... Nous pouvons noter que les passages pour piétons sont occupés aléatoirement et qu'un temps de 30 secondes est attribué pour le passage d'un piéton. Cela se traduit par une sous méthode gérant l'état aléatoire des passages pour piétons et une autre gérant son occupation. Le schéma représenté ci-après illustre la méthode permettant l'incrémentation du temps de passage du piéton. Ainsi, la simulation mixte permet de gérer d'un côté la circulation en ville et de l'autre les événements pouvant la modifier.

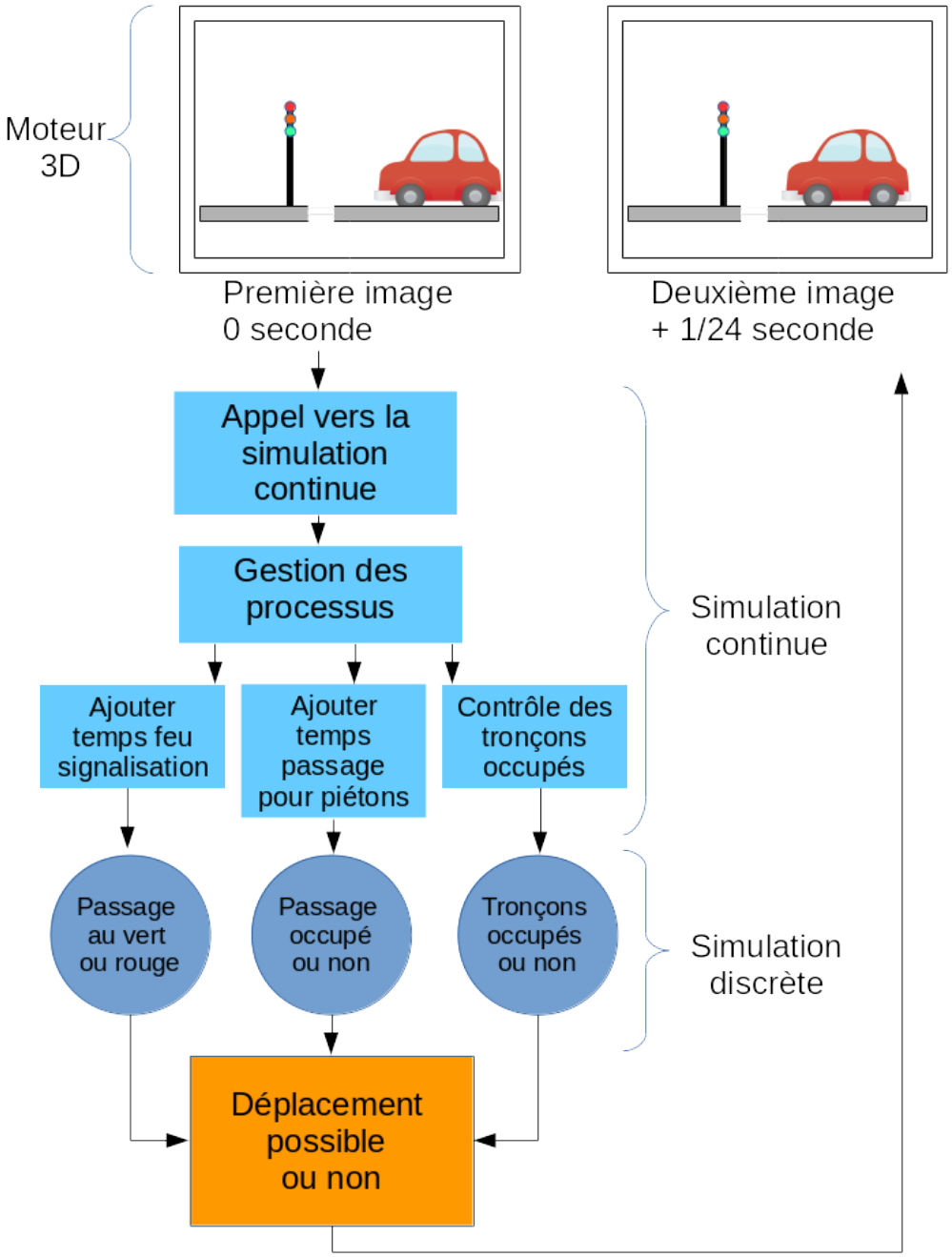


FIGURE 63 – Illustration des interactions en moteurs de simulation

Chapitre 11

Système expert

Dans le contexte de la gestion du trafic, un outil d'aide à la décision se doit de pouvoir représenter les différents aspects de la circulation routière et de permettre d'évaluer les performances de cette circulation par des méthodes statistiques lors de l'altération de l'environnement (travaux de la voie publique, accidents, pannes,...).

Actuellement, les modèles macroscopiques existants se basent surtout sur le calcul d'itinéraires ou l'analyse statistique du trafic ([31], [32]) pour en prévoir la fluidité. Dans les deux cas, le temps de trajet individuel (temps de parcours d'une distance donnée pour un véhicule donné) sera le caractère déterminant pour le grand public alors que la fluidité, définie par le débit, la concentration et la vitesse du flot de véhicules, le sera pour les gestionnaires du trafic routier ([33]). L'analyse statistique de la fluidité du trafic routier nécessite l'obtention de données qui sont facilement obtenues sur le terrain au travers du comptage de véhicules qui peut se faire manuellement (comptage au passage d'un péage par exemple) ou automatiquement par des boucles électromagnétiques. La problématique de ce travail est légèrement différente car elle vise à fournir une visualisation immersive et une réponse en temps réel de la modification des conditions de circulation, sans pour autant être alimentée par des données recueillies sur le terrain. La mise en place d'un trafic routier pour cette étude sous-entend donc la mise en place de véhicules mobiles indépendants qui respectent les règles usuelles de conduite. Ces règles, définies en France par le Code de la route, permettent, si elles sont respectées par tous les utilisateurs, d'éviter les accidents. Leur implantation se traduit par la mise en place d'une suite de tests répétés pour chaque véhicule en circulation et pour chaque itération de déplacement, ce qui permet de définir si le mouvement du véhicule est possible ou non. De plus le mouvement d'un véhicule, le long d'un trajet, est défini à chaque instant par la disponibilité du tronçon de route qu'il doit emprunter. A terme, le déplacement d'un véhicule, indépendant du déplacement des autres véhicules et des règles de conduite, amène à une certaine redondance de la libération des tronçons. En effet, la condition de disponibilité d'un tronçon de route est à la fois une condition pour le déplacement mais aussi une contrainte implicite de l'application du Code de la route. Ces différentes règles seront régies par une Intelligence Artificielle : le Système Expert.

11.1 Structure d'un Système Expert

C'est au cours des années 70 que les Systèmes Expert ont fait leur apparition. Ils ont pour objectif de représenter/reproduire le raisonnement schématique d'un expert au sujet d'une question donnée.

11.1.1 Définition

Cet outil est donc capable de résoudre des questions complexes à partir d'informations qu'il connaît puisque renseignées par un développeur à partir des connaissances d'un expert dans le domaine d'intérêt. Ces Systèmes Experts suivent trois grands modes de raisonnements :

Le chaînage avant : méthode de déduction appliquant des règles en partant d'une proposition et aboutissant à de nouvelles conclusions. Exemple : Si X roule, alors X est un véhicule.

Le chaînage arrière : cette méthode part de la conclusion pour connaître le cheminement vers celle-ci. Elle est appliquée dans le domaine des échecs servant à déterminer les coups joués pour atteindre la position donnée.

Le chaînage mixte : comme son nom l'indique, est à l'interface des deux méthodes précédentes en permettant de saturer la base de faits par chaînage avant et de chercher les faits encore éventuellement déductibles par chaînage arrière.

Ainsi, un Système Expert est constitué de trois grands ensembles (voir figure 64) :
— Une base de faits, ce sont les éléments renseignés par l'utilisateur pour répondre à la question qu'il pose,
— Une base de connaissances, où sont rassemblées les règles liées aux connaissances de l'expert/des experts du domaine,
— Un moteur d'inférence, capable de faire le lien entre ces divers éléments pour répondre à la requête posée.

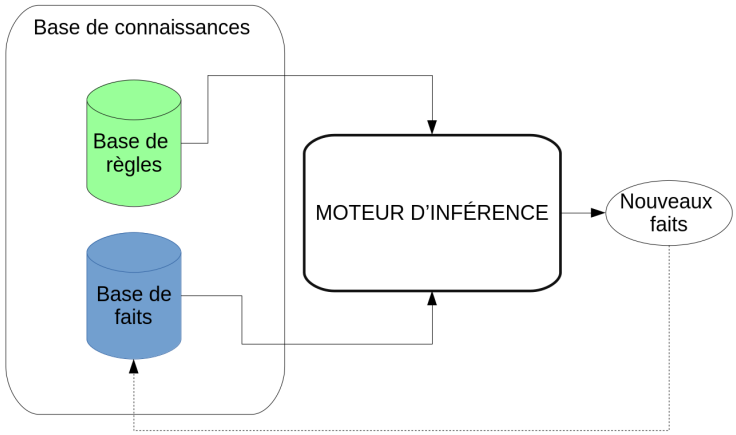


FIGURE 64 – Structure d'un Système Expert

11.1.2 Quelques exemples

Les Systèmes Experts existent depuis plus de 50 ans et dans de nombreux domaines, en voici quelques uns :

Dendral : 1965, détermination de composés chimiques à partir de données spectrométriques,

Mycin : 1970, expertise en bactériologie (diagnostic et traitement),

Prospector : 1978, évaluation géologique de sites (probabilité de présence de gisements et de minerais),

Cyc : 1984, inventaire des termes appartenant à l'ontologie dans le but de permettre à des applications d'Intelligence Artificielle de raisonner d'une manière similaire à l'être humain,

CIME : 1988 [34], destiné à piloter et à optimiser l'activation séquentielle de traitements numériques en vue de cartographies thématiques,

Sachem : 1990, conçu pour piloter des hauts-fourneaux en analysant les données fournies en temps réel par un millier de capteurs,

Akinator : 2007, est un jeu basé sur un système expert pour deviner un objet ou un personnage à partir de questions générées par une Intelligence Artificielle.

11.2 Trafic urbain et Système Expert

Après avoir traité des généralités liées aux Systèmes Experts, nous allons démontrer, dans cette section, l'intérêt de mettre en place cet outil pour la représentation du trafic urbain mais aussi quels sont les critères qu'il devra revêtir.

11.2.1 Pourquoi un Système Expert dans le cadre de la modélisation du trafic urbain

La modélisation de trafic en ville est propice à la mise en place d'un Système Expert. En effet, la circulation de voitures en milieu urbain est réglementée selon le Code de la route, donc des règles. Ces règles peuvent être inscrites dans une base de connaissances pour les appliquer sur des mobiles dans la modélisation de la ville 3D. Cela nous permet d'automatiser la circulation. Voici une liste des règles que le Système Expert doit prendre en compte afin de permettre une modélisation la plus réaliste possible :

— Les véhicules doivent s'arrêter aux feux de signalisation tout comme aux stops,
— Les véhicules doivent ralentir à un cédez le passage pour vérifier si la route est occupée ou non,
— Les véhicules s'arrêtent à un cédez le passage si la route est occupée,
— Les véhicules s'arrêtent si un passage pour piétons est occupé,
— Les véhicules allument leurs feux de croisement lorsqu'il fait nuit,
— Les véhicules ne doublent pas sur une ligne blanche,
— ...

Ces règles sont celles d'une conduite dite de sécurité. La mise en place d'un Système Expert peut aussi être intéressant dans le cas où la conduite est aléatoire. Prenons en exemple une personne dangereuse au volant, pour n'importe quel motif. Si cette personne décide de rouler à contre-sens sur l'autoroute, il peut être intéressant d'avoir des règles à suivre dans ce cas là afin d'être sûr de réagir dans la bonne mesure, ce qui nous donnerait des règles comme celles-ci : **Si** un véhicule se trouve en contre sens, **alors**

— les véhicules dans la bonne direction sortent à la première sortie,
— les véhicules décelèrent et s'arrêtent dès que possible,
— les véhicules font des appels de phares pour prévenir du danger,
— ...

Bien sûr cette liste ne suit pas une logique écrite dans un Code de la route car ce sont des cas extrêmes mais le fait de pouvoir alimenter une base de connaissance avec le plus de cas particuliers peut amener à une modélisation au plus près d'un comportement humain. Cela peut être très à propos dans le cas des voitures autonomes.

11.2.2 La structure générale choisie

Ce Système Expert doit permettre de faire le lien entre les différentes règles du Code de la route et les informations liées à la route où circule le véhicule (ou le trottoir pour le piéton) pour réaliser le calcul du déplacement. A partir de ces éléments, on peut identifier les grands ensembles qui structurent le Système Expert et en particulier :

— La base de règles avec le Code de la route,
— La base de faits avec les éléments liées au tronçon de route.

Comme écrit dans la partie ci-dessus, le Code de la route alimente la base de règles / connaissances. Elle permet au moteur d'inférence de décider ce qu'il faut faire dans un cas ou dans un autre. La base de faits permet, quant à elle, de déduire un comportement à partir de faits. C'est à dire de décider en connaissance de cause. Elle va nous être utile dans les dépassements par exemple. En effet, si un véhicule doit en dépasser un autre, il doit savoir si le tronçon de dépassement est libre ou non. Regarder les faits et les connaissances pour choisir une solution nous font tendre à choisir un Système Expert à chaînage avant.

11.3 La réalisation du Système Expert

11.3.1 Le Moteur d'Inférence

Comme exprimé dans les paragraphes précédents, le Système Expert est composé de trois sous-ensembles :

— Une base de faits (avec les informations liées au tronçon où circule le véhicule ou le piéton),
— Une base de connaissances (la liste des règles liées au Code de la route),
— Un moteur d'inférence.

Le moteur d'inférence peut être considéré comme un système à deux phases : la phase d'évaluation et la phase d'exécution (voir figure 65).

La phase d'évaluation consiste en un choix de règles qui vont être utiles à un moment donné. En effet, lors de l'écriture du code, nous ne pouvons pas définir à l'avance quelles règles doivent être utilisées, ainsi que l'ordre dans lequel elles doivent être appliquées. Le déclenchement de telle ou telle règle se fera en fonction de la situation donnée. C'est ce qui différencie l'Intelligence Artificielle d'un algorithme.

La phase d'exécution consiste à exécuter la partie "action" des règles déclenchées et à modifier en conséquence la base de connaissances et/ou la base de faits qui pourront être réutilisées dans la phase d'évaluation.

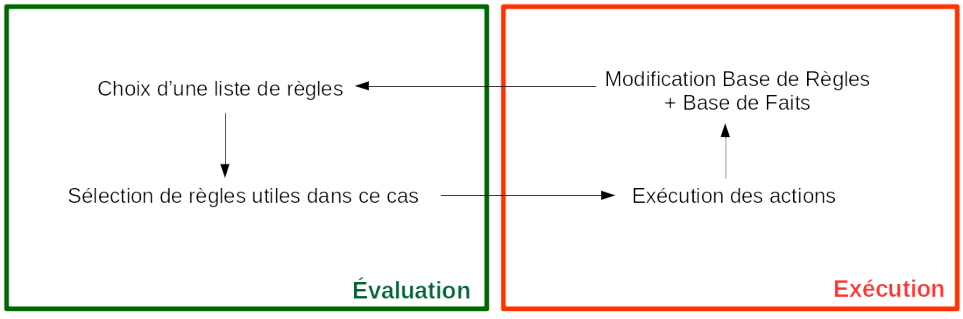


FIGURE 65 – Schéma du moteur d'inférence

Ce moteur d'inférence permet donc de définir le déplacement du véhicule ou du piéton en fonction du Code de la route et des tronçons traversés.

11.3.2 La base des règles

Les règles applicables lors d'une simulation comprennent le code de la route ainsi que les comportements humains. Les règles de comportement permettent d'avoir des évènements liés à une conduite dangereuse ou non, ou alors une conduite éco-responsable. Nous verrons dans les prochains paragraphes les différentes règles mises en place ainsi que leurs implémentations.

Principe général des informations gérées par la base de règles

La base de règles permet de stocker tous les cas liés à la conduite et à sa gestion. Elle alimente le moteur d'inférence afin de répondre à une problématique à un moment donné.

Passons à la description des processus de décision de chaque mouvement vers l'avant d'un véhicule le long de l'itinéraire prédéfini.

Le premier test consiste à déterminer les changements de tronçons. En effet, au sein d'un même tronçon, il n'y a aucune raison que les conditions de circulation changent.

Si le véhicule entre dans un nouveau tronçon de route, il peut faire face à diverses situations. Le cas le plus simple reste que le tronçon suivant corresponde au prolongement du précédent (c'est-à-dire que l'on reste sur une même route sans modification mais qu'un tronçon a dû être créé pour gérer un changement de propriété comme un changement de direction par exemple).

La présence d'un stop ou d'un feu rouge implique pour le véhicule la même réaction, à savoir un arrêt pendant un temps donné. Ce temps d'arrêt est déterminé, au cours de ce projet, comme une durée fixe de trois secondes. Bien qu'assez éloignée de la réalité, cette durée permet d'introduire un délai dans le flux de véhicules tout en conservant une certaine simplicité dans les conditions demandées. En effet, la durée réelle des feux de signalisation est une information très peu renseignée et implique dans tous les cas la mise en place d'une valeur par défaut. La sortie de cette condition renvoie vers le test initial de libération du tronçon suivant pour permettre l'avancée du véhicule.

Pour connaître la présence d'un piéton sur un passage pour piétons, nous avons placé un booléen de présence sur ces derniers. Il permet de définir si le véhicule peut avancer ou doit s'arrêter. Actuellement, la présence de piétons est générée aléatoirement à chaque temps de calcul. L'arrêt, quant à lui, est géré de la même manière que précédemment.

Le dernier cas est le plus complexe à mettre en œuvre. Il consiste à représenter le comportement que peut adopter le conducteur dans certaines situations. Cette partie est, de loin, celle qui demande le plus grand temps de calculs. En effet, la présence d'un cédez le passage exige le ralentissement du véhicule et l'estimation de la disponibilité du tronçon à l'intersection. Ces deux actions doivent être réalisées dans un même temps.

Présentation de la Base de Règles associée au Code de la Route

La base de règles traitant des informations du Code de la route étant composé d'une vingtaine de règles, il n'est pas envisageable d'en présenter l'ensemble dans ce chapitre. Cependant, à titre d'exemples, voici quelques unes des règles choisies pour appliquer le Code de la route via notre Système Expert :

- **Si** un véhicule arrive à un stop **alors** il s'arrête
- **Si** un véhicule arrive à un feu de signalisation **alors**
  - **Si** le feu est rouge **alors** il s'arrête
  - **Si** le feu est vert **alors** il continue
- **Si** un véhicule arrive à un cédez le passage **alors**

    - **Si** le tronçon en intersection est vide **alors** il passe
    - **Si** le tronçon en intersection est occupée **alors** il s'arrête
- **Si** un véhicule arrive à un passage pour piétons **alors**
    - **S'il** n'y a pas de piéton **alors** il passe
    - **S'il** y a au moins un piéton **alors** il s'arrête

Bien sûr des passerelles entre les règles existent. Par exemple, lorsque le véhicule est arrêté à un feu de signalisation :

- **Si** le feu passe au vert et que le passage pour piétons est libre, **alors** le véhicule passe.
- **Si** le tronçon suivant est libre, alors le véhicule reprend sa circulation **Si** le tronçon est occupé, **alors** le véhicule reste sur place et applique l'état du feu de signalisation

Comme il l'a été mentionné précédemment, le déplacement d'un véhicule donné est avant tout conditionné par la disponibilité du tronçon qu'il doit emprunter :

- **Si** le tronçon est libre, **alors** le véhicule peut avancer sans autre condition.

Dans le cas inverse :

- **Si** le tronçon est déjà occupé, **alors** la distance séparant les deux véhicules est calculée en fonction de leurs positions sur le tronçon.

Cette distance est ensuite comparée à la distance de sécurité. Le véhicule avance librement si la distance de sécurité est supérieure à celle qui le sépare du véhicule qui le précède. Dans le cas contraire, deux cas peuvent être envisagés :

- **Si** le mode collision est désactivé, **alors** la vitesse du véhicule est adaptée pour que la distance de sécurité soit respectée.
- **Si** le mode collision est activé, **alors** la vitesse du véhicule reste la même. (Le véhicule avance alors librement sauf si la distance avec le véhicule qui le précède devient nulle. Dans ce cas, les deux véhicules s'arrêtent, simulant ainsi un accident. Le tronçon est alors considéré comme bloqué à la circulation. De ce fait, les itinéraires des autres véhicules doivent être recalculés pour éviter ce tronçon).

La partie amorçant le Système Expert est la partie décrite précédemment avec la gestion de la disponibilité des tronçons. C'est uniquement dans le cas où le véhicule peut circuler librement que la partie concernant l'application du Code de la route peut être appliquée.

Une fois la décision prise sur l'avancement du tronçon, nous greffons la partie dédiée aux interactions avec le mobilier urbain et les piétons. C'est-à-dire que :

- **Si** le tronçon suivant n'est pas occupé, **alors** nous regardons s'il existe un mobilier urbain sur un sommet de ce tronçon.
    - **Si** un mobilier existe, **alors**
        - **Si** c'est un passage pour piétons, **alors** on regarde s'il est occupé,
        - **Sinon si** c'est un feu de signalisation, **alors** on regarde son état,
        - **Sinon si** c'est un cédez le passage, **alors** on regarde l'état du tronçon suivant,
        - **Sinon si** c'est un stop, **alors** on s'arrête et on regarde l'état du tronçon suivant

        Sinon on avance sur le tronçon.

L'association de ces deux modes de fonctionnement permet de représenter une circulation urbaine basée sur des itinéraires indépendants pour chaque véhicule. Cette approche permet une représentation du trafic visuellement réaliste indispensable à l'immersion de l'utilisateur.

De tels systèmes sont caractérisés notamment par leur complexité. Celle-ci se définit sur différents plans (comme l'algorithme employé, les actions simulées ou les choix des comportements à mettre en œuvre selon les influences de l'environnement) et peut être quantifiée par différentes méthodes d'évaluation comme par Yoann Kubera [35] qui tient compte de la complexité inhérente à la multiplication des agents logiciels et de leurs interactions mutuelles sur leur(s) environnement(s).

Présentation de la Base de Règles associée aux comportements humains

Au cours de ce chapitre, nous avons illustré que les règles qui devaient régir ce Système Expert étaient de deux ordres :

— les règles du Code de la route, présentées dans la section ci-dessous,

— les règles liées aux comportements humains.

Ce sont de ces règles dont nous allons traiter dans cette section. Si définir des règles liées au Code de la route représente une "traduction" permettant de passer d'un texte juridique à un texte compréhensible par un ordinateur ou une unité de calcul, il n'en est pas de même pour les comportements humains. En effet ces comportements ne suivent pas des règles bien identifiées puisqu'elles sont, par définition, liées au vécu de l'individu. On peut alors schématiser ces comportements comme étant :

— soit dans la logique commune de l'utilisation de la route (pour le cas qui nous intéresse),

— soit contraires aux habitudes usuelles que doivent suivre les usagers de la route (aussi bien conducteurs que piétons ou cyclistes).

Les règles que nous avons répertoriées au cours de ce travail sont au nombre d'une trentaine, dont un certain nombre seront présentées ci-après.

— **Si** un véhicule ne circule pas à la même vitesse sur le même tronçon **alors** regarder si le véhicule est devant
  — **Si** le véhicule est devant, **alors** le véhicule peut éventuellement le doubler

— **Si** une double voie est disponible, **alors** le véhicule peut entamer une manœuvre de dépassement

— **Si** la voie de dépassement est libre, **alors** le véhicule peut se déporter et se mettre à la vitesse réglementaire

— **Si** la voie de dépassement est trop courte pour doubler (calcul de la distance), **alors** le véhicule freine et se rabat

— **Si** la voie principale est occupée, **alors** le véhicule reste sur la voie de dépassement

— **Si** la distance minimale avec le véhicule suivant est dépassée, **alors** le véhicule freine

— **Si** un véhicule est arrêté, **alors** le véhicule s'arrête avec une distance de sécurité plus courte.

Ces règles s'appliquent dans le cadre d'une conduite réglementée, sans prise de risque. Pour accentuer un mode de conduite plus dangereux, nous pouvons modifier les distances de sécurité puis appliquer des règles différentes comme par exemple :

— **Si** la voie de dépassement est trop courte, **alors** le véhicule accélère sans limitation de vitesse pour pouvoir doubler

— **Si** au tronçon suivant un stop est libre, **alors** le véhicule ne s'arrête pas (application de la règle du cédez le passage).

— **Si** la limitation de vitesse est atteinte, **alors** une accélération aléatoire est appliquée.

Chapitre 12

L'Intelligence Artificielle à l'interface entre simulations et système expert

Parce que ce projet s'adresse aussi bien aux besoins des mairies, qu'aux attentes très diverses des métiers de la circulation routière, la méthodologie développée doit permettre de rassembler toutes les techniques utilisables pour faciliter la visualisation des phénomènes routiers et orienter la décision d'une ou plusieurs personnes. L'outil dont nous présentons les méthodologies au cours de ce manuscrit permet à la fois de visualiser des informations tridimensionnelles à l'échelle de la ville, mais également de proposer une simulation du trafic urbain. Proposer un tel appui d'aide à la décision, nécessite de développer un outil d'Intelligence Artificielle qui sera à la frontière entre simulation et Système Expert.

12.1 Simulation et Système Expert, deux modules complémentaires

Avant de décrire comment a été développé ce module d'Intelligence Artificielle à l'interface des simulations discrètes et continues ainsi que du Système Expert, présentons, dans les paragraphes suivants, un rapide récapitulatif de ces moteurs qui aideront à une meilleure compréhension des techniques d'interfacages entre ces modules.

12.1.1 Le moteur de Simulation Mixte

Au cours du chapitre 10, il a été présenté un certain nombre d'éléments à prendre en compte pour la simulation du trafic. Ainsi nous avons identifié :

— des éléments à modéliser via la simulation continue,
— des éléments à modéliser via la simulation discrète.

Ceci nous a conduit à réfléchir à une simulation dite mixte qui a pour avantage de gérer, au sein d'un même échéancier, à la fois des éléments à pas de temps réguliers et d'autres à pas variable, selon des événements ayant lieu au cours de la modélisation.

L'organisation du moteur de simulation mixte se déroule en plusieurs phases :

1. Le temps est fixé par le moteur 3D, chaque image affichée fixant le pas de temps,
2. A chaque pas de temps, la simulation continue est lancée,
3. Les évènements sont gérés par la simulation discrète,

4. La simulation discrète appelle le moteur d'inférence pour la gestion des règles,
5. Les évènements sont appliqués à la simulation continue,
6. La scène 3D est mise à jour,
7. Les étapes 1 à 6 sont réalisées en boucle jusqu'au temps final de la simulation.

Ce module de simulation mixte développé permet alors :
— de gérer la circulation comme un mouvement fluide,
— d'appliquer les règles de conduite au cours du temps.

Nous remarquons sur la figure 66 que la simulation continue relance le moteur 3D. C'est le cas à chaque fin de traitement des évènements. Ce principe est possible du fait que les calculs sont en temps réel, ceci afin de ne pas ralentir le temps d'affichage.

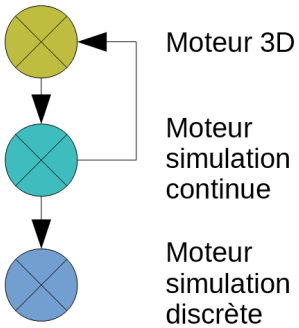


FIGURE 66 – Schéma du moteur de simulation

12.1.2 Le moteur du Système Expert

Le Système Expert étudié au cours de ce mémoire, quant à lui, permet de décrire et définir les comportements qu'adoptent les divers mobiles présents dans la scène en fonction :
— du Code de la route,
— des comportements humains.

L'organisation du Système Expert se déroule en plusieurs phases :

1. Pour chaque mobilier ou tronçon, récupération de leurs états,
2. Sélection des règles depuis la base de règles via le moteur d'inférence selon les informations présentes dans la base de faits,
3. Application des règles les unes après les autres,
4. Retour des évènements à appliquer vers le moteur de simulation

Le Système Expert développé permet alors :
— De sélectionner des règles à appliquer en fonction de cas particuliers,
— De les appliquer pour générer une action par le moteur de simulation.

La figure 67 illustre le fonctionnement du Système Expert. Pour la liste de faits trouvés à un temps donné, une liste de règles est choisie parmi la base de règles. Une fois les règles appliquées par le passage dans une boucle de calculs, les actions à réaliser sont prêtes à être appliquées.

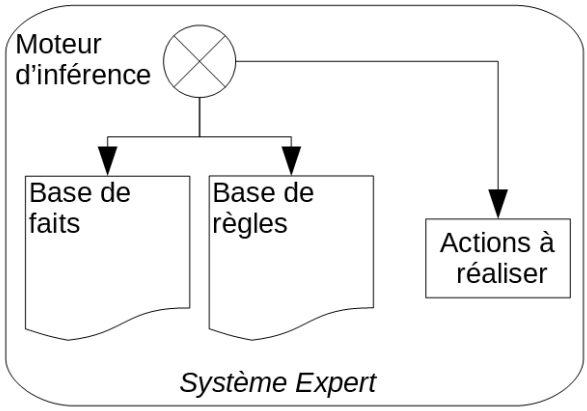


FIGURE 67 – Schéma du moteur du Système expert

12.2 L'Intelligence Artificielle à l'interface de ces modules

L'Intelligence Artificielle (IA) permet d'interfacer la partie simulation et le Système Expert. On retrouve donc :

— les classes dédiées à la simulation,

— les éléments constituant le Système Expert

Elle nous permet d'automatiser l'échange entre les deux parties majeures de notre travail. A chaque cas particulier de la circulation urbaine, elle va permettre de faire le lien entre la déclaration des cas et leur résolution.

La figure 68 illustre cette interaction.

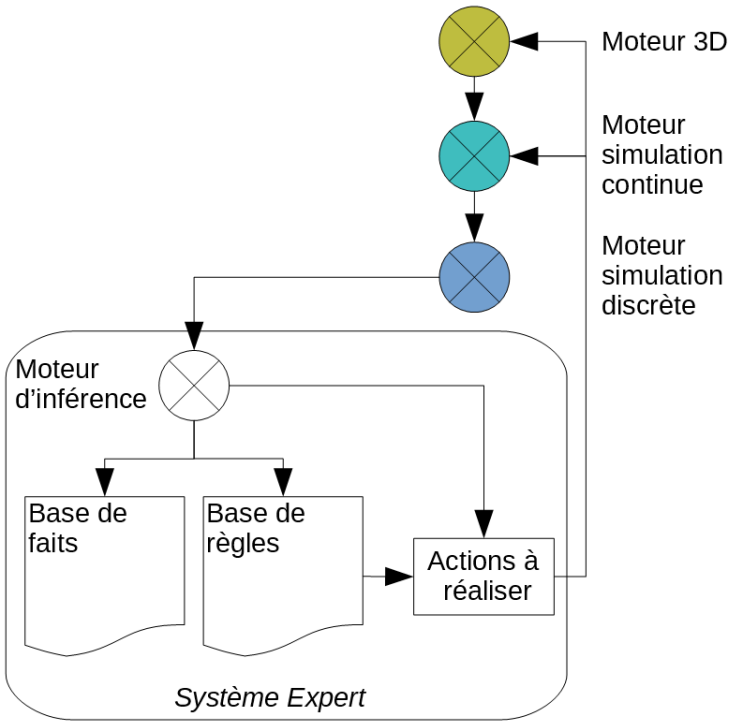


FIGURE 68 – Schéma du moteur de l'Intelligence Artificielle

Ainsi, lorsque le moteur de simulation discrète lance les routines gérant les possibles interactions entre les mobiles ou celles avec le mobilier, l'IA appelle le moteur d'inférence, point

d'entrée du Système Expert, pour résoudre ces interactions. La base de faits va alors être alimentée de ces informations. Par la suite, le moteur d'inférence va les traiter en choisissant parmi la base de règles celles qui les concernent. Une fois choisies, les actions à réaliser sont envoyées à la simulation continue pour la gestion des mobiles via l'IA. Enfin, l'affichage se met à jour.

Si simuler le trafic urbain intéresse de nombreux métiers (comme les gestionnaires de voiries, les aménageurs urbains, les mairies,...), il n'existe pas encore de méthodologies permettant d'aboutir à un logiciel capable de prendre en compte à la fois :

— la modélisation tridimensionnelle de la zone d'intérêt,
— les règles du Code de la route,
— les comportements individuels dans la ville,
— les différences de trafic en fonction de l'heure de la journée ou même selon la période de l'année,
— la création de scénarios de trafic avec de la donnée modifiée.

Le regroupement de tous ces critères dans un seul et unique outil d'Intelligence Artificielle, à l'interface entre la simulation et le Système Expert, permet de proposer aux multiples utilisateurs un véritable outil novateur facilitant la décision. De plus, le système d'import de la donnée permet une prise en main intuitive de ces outils qui sont masqués dans l'Interface homme-machine. Cette interface "en modules indépendants" permet à un utilisateur "développeur" de débrancher certains composants afin de générer d'autres types de simulations. Il pourra également ajouter de nouvelles règles dans la base afin d'affiner les réponses créées par le Système Expert ou de l'adapter à de nouvelles thématiques.

Chapitre 13

Principe et optimisation de la visualisation

L'information géographique est souvent assimilée à de la donnée cartographique, c'est à dire, à plat. Dans le cadre de simulation, la cartographie à plat peut être un bon moyen de synthétiser une information et de la rendre plus claire mais on comprend vite qu'elle peut avoir une limite. C'est pour cela, comme nous l'avons vu dans les parties précédentes (voir chapitre 7), que nous avons décidé de convertir la donnée SIG en 3D pour obtenir une simulation en 3D. En plus des problèmes liés à cette conversion, des questions sur la prise en main de la 3D et de la navigation sur ces trois axes se sont posées. Nous exposerons dans ce chapitre les différentes techniques utilisées pour la réalisation de cette vue 3D.

Lorsque l'on navigue sur une carte en 2D, le point de vue peut bouger selon 2 axes, X et Y, mouvements assez aisés et très contrôlables. De plus le curseur de la souris n'influence pas le point de vue, il est utilisé seulement pour interagir avec la carte. Lorsque l'on passe en 3D, en plus de rajouter un axe Z, nous avons aussi le curseur de notre souris qui se transforme en 'viseur' et donc qui influe directement la visualisation. La première problématique qui s'est posée est l'optimisation des mouvements. En effet, chaque mouvement de la souris, en plus de ceux de la navigation (via les flèches du clavier) génère des rotations de la caméra. Ces rotations sont souvent calculées via des matrices de rotations 4 x 4, celles-ci nous permettent de faire des mouvements ne suivant pas un axe de rotation contrairement aux matrices 3 x 3. Pour trouver la nouvelle position lors d'un mouvement, une multiplication des matrices de rotations doit être réalisée, sachant que 2 matrices ont 16 éléments, cela revient à faire 16 * 16 = 256 calculs pour un mouvement. Nous voyons vite se profiler un problème majeur : le temps de calcul. Pour pallier ce problème, une solution utilisée dans les projets d'infographie, de dynamique moléculaire ou de navigation doit être appliquée : l'utilisation des quaternions.

13.1 Utilisation des quaternions

Avant de décrire l'utilisation des quaternions, voici quelques rappels sur les termes utiles à la navigation dans une scène 3D :

Le roulis est un mouvement de rotation d'un mobile autour de son axe longitudinal (axe de roulis).

Le lacet est le mouvement de rotation horizontal d'un mobile autour d'un axe vertical.

Le tangage est un mouvement de rotation autour de l'axe transversal d'un objet en mouvement.

Ces termes sont illustrés dans la figure 69.

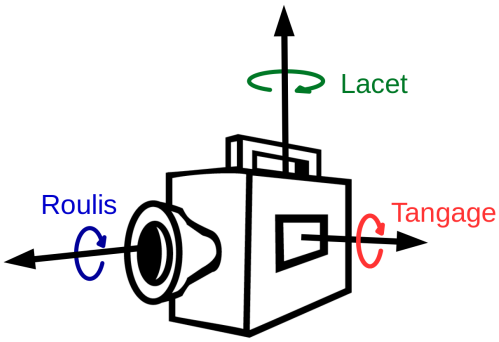


FIGURE 69 – Différents mouvements possibles pour dans un espace 3D

Les quaternions ont été inventés par Hamilton en 1843 [36]. Durant ses travaux, Hamilton voulait trouver un moyen d'étendre les nombres complexes à une dimension supérieure à 2. La troisième dimension ne l'a pas amené à une solution mais l'ajout d'une quatrième lui a permis de découvrir les quaternions. Il a inventé dans le même temps le terme 'verseur'. Les quaternions peuvent être décrits par une succession de 4 nombres réels, avec le premier correspondant à la partie scalaire et les 3 derniers à la partie vecteur.

$$q = w + ix + jy + kz \tag{13.1}$$

Avec w, x, y et z composant un système à 4 scalaires et 1, i, j et k étant 3 quaternions unitaires (ou verseurs)[36]

Les quaternions unitaires suivent cette formule :

$$i^2 = j^2 = k^2 = ijk = -1 \tag{13.2}$$

Avec :

$$\begin{aligned} ij = k, ji = -k, \\ jk = i, kj = -i, \\ ki = j, ik = -j \end{aligned}$$

On peut aussi écrire l'équation de cette façon :

$$q = (w[x, y, z]) \tag{13.3}$$

13.1.1 Utilisation dans une visualisation 3D

Les angles d'Euler

Cette méthode représente l'orientation d'un objet comme la succession de trois rotations selon les axes orthogonaux, appelés x, y et z (Figure 70).

La rotation selon les angles d'Euler peut être écrite de cette façon :

$$Euler(\psi, \theta, \phi) \tag{13.4}$$

Avec ψ, θ, et ϕ correspondant respectivement aux rotations selon les axes X, Y et Z

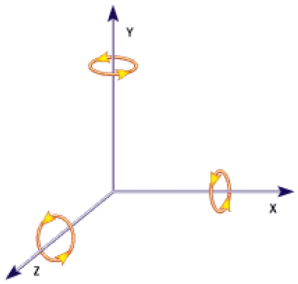


FIGURE 70 – Représentation des angles d'Euler

Interpolation avec les angles d'Euler

Les trois rotations n'étant pas indépendantes, un problème apparaît quant à l'interpolation entre deux positions. Pour expliquer ce problème, nous allons traiter deux cas. Pour le premier, admettons une rotation le long de l'axe y de 180 ° , nous écrirons alors la formule d'Euler comme ceci :

$$Euler\_1\ (0,\ 180,\ 0)$$

Si on considère un cube sur lequel on applique cette rotation, il sera retourné le long de l'axe Y. Cette position finale peut être atteinte via d'autres rotations, par exemple :

$$Euler\_2\ (180,\ 0,\ 180)$$

Si l'on affiche seulement ces deux rotations l'une après l'autre, le mouvement paraîtra saccadé. C'est pour cela que l'on calcule des positions intermédiaires. Prenons une position intermédiaire correspondant à la demie rotation de chacun des angles d'Euler de notre exemple, ils auront comme formule :

$$Euler\_1\_Inter\ (0,\ 90,\ 0) \text{ et } Euler\_2\_Inter(90,\ 0,\ 90)$$

On remarque dans la figure 71 que les positions intermédiaires sont différentes. Cela générera des artefacts visuels vu que deux "chemins" sont disponibles entre deux points. Un autre problème s'ajoute à ces rotations : le blocage de cardan.

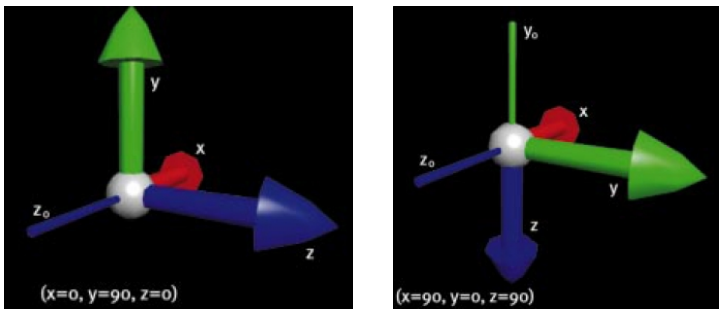


FIGURE 71 – Positions intermédiaires pour les angles Euler_1 et Euler_2. La figure à gauche représente *Euler_1(0, 90, 0)* et la partie droite *Euler_2(90, 0, 90)*.

Le blocage de cardan

Allan et Mark Watt [37] ont défini ce phénomène tel que ceci : "Le blocage de cardan est un terme dérivé de la mécanique propre d'un cardan pouvant accueillir un compas ou un gyroscope. Il est généralement constitué de 3 cercles concentriques qui peuvent tourner indépendamment les uns des autres. Lorsqu'ils sont alignés, il y a une perte d'un degré de liberté - on appelle cela un blocage de cardan. Cette perte de degré de liberté peut être démontrée par l'application d'une rotation équivalente à $\pi/2$ sur un axe quelconque. Prenons par exemple un objet sur lequel on applique une rotation de 90 ° autour de l'axe Y, alors l'axe X' projeté sera confondu avec l'axe Z, illustré dans la figure 72. Une fois ces axes confondus, aucune rotation ne peut être affectée à l'un ou l'autre, il ne restera que la rotation selon l'axe des Y dans notre cas.

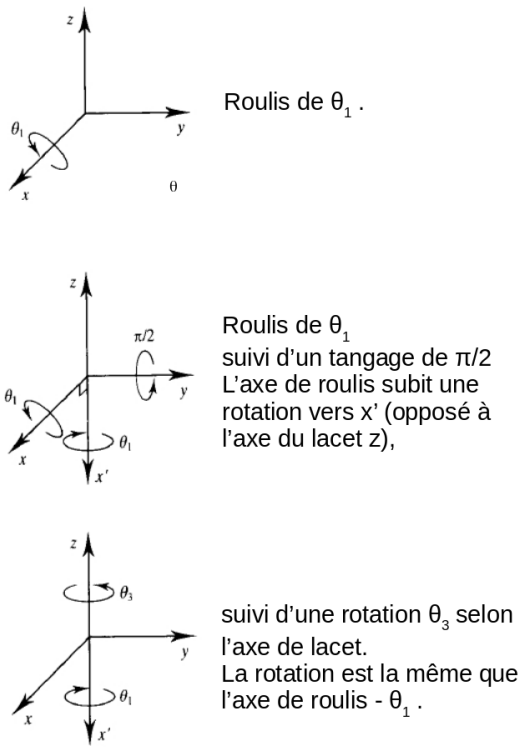


FIGURE 72 – Illustration du blocage de cardan

Les matrices de rotation

Les matrices de rotation permettent de regrouper toutes les transformations (rotation, translation et homothétie) applicables dans l'espace. L'avantage de ces matrices est l'unification des transformations dans l'espace. Par contre, c'est très coûteux en terme de mémoire sachant qu'une matrice de rotation se stocke dans un tableau à deux dimensions d'une taille 3x3 ou 4x4 dans le cas de coordonnées homogènes [38]. Une rotation d'un angle θ autour de l'axe X peut être décrite de cette façon :

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \tag{13.5}$$

Pour chacun des axes, une matrice telle que celle-ci doit être calculée et pour la succession de deux rotations une multiplication des deux doit être faite, ce qui implique un nombre conséquent de calculs. Tout comme les angles d'Euler, pour obtenir une rotation complète, l'objet passe

par les rotations de l'axe X, puis de l'Y et enfin du Z. En fait, les matrices de rotation sont des formes décomposées d'une rotation réalisée grâce aux angles d'Euler, et donc, nous retrouvons le problème de blocage de cardan. Sachant que dans les librairies 3D, telles que OpenGl ou DirectX, gérant l'affichage tridimensionnel les rotations sont calculées via des matrices, pour contourner leurs faiblesses il va falloir contourner l'usage des matrices par des quaternions.

Représentation des quaternions dans l'espace

Afin de ne pas être bloqué à utiliser un axe de rotation suivi d'un autre pour appliquer un changement de position, je vais utiliser les quaternions. Ils peuvent être représentés de cette façon :

$$
\begin{aligned}
x &= VecteurDeRotation.x * \sin(AngleDeRotation/2) \\
y &= VecteurDeRotation.y * \sin(AngleDeRotation/2) \\
z &= VecteurDeRotation.z * \sin(AngleDeRotation/2) \\
w &= \cos(AngleDeRotation/2)
\end{aligned}
$$

On peut simplifier ces formules en disant que les quaternions stockent un axe de rotation et un angle de rotation. Prenons par exemple un quaternion défini par [0.7 , 0 , 0 , 0,7]. Les trois premiers nombres correspondent à l'axe de rotation avec les composantes, x, y et z. Le dernier correspond au cosinus de l'angle de rotation divisé par 2 (voir équation). Dans les trois premiers termes, seul x possède une valeur donc, la rotation se fera exclusivement autour de cet axe et l'angle de rotation sera égal à :

$$
\begin{aligned}
&0,7 = \cos(AngleDeRotation/2)) \\
&\arccos(0,7 = AngleDeRotation/2) \\
&z = VecteurDeRotation.z * \sin(AngleDeRotation/2) \\
&doncAngleDeRotation = 2 * \arccos(0,7) = 1,59radians = 90
\end{aligned}
$$

La compréhension visuelle est un peu plus difficile que celle des angles mais les quaternions permettent de s'affranchir des rotations successives des méthodes présentées précédemment. Pour représenter ces quaternions, nous pouvons dessiner un vecteur correspondant à l'axe de rotation autour duquel on applique une rotation (voir figure 73).

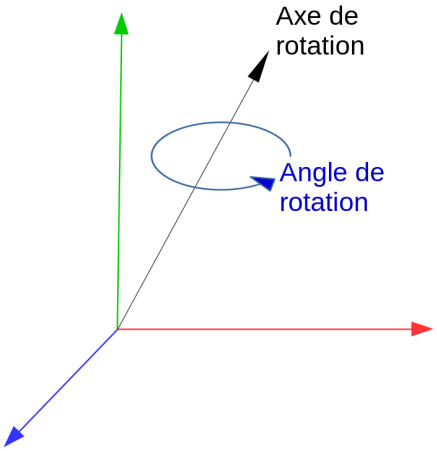


FIGURE 73 – Représentation des quaternions

13.1.2 Conversion entre matrices de rotation et quaternions

Pour pouvoir utiliser ces quaternions et appliquer mes rotations à la librairie OpenGl, je vais devoir faire des conversions entre ces deux formats et faire tous mes calculs entre quaternions. Je vais brièvement expliquer ces conversions ci-dessous.

Des quaternions aux matrices

Une fois les calculs réalisés avec des quaternions, nous devons les convertir en matrices de rotation afin qu'OpenGl les traite. La conversion d'un quaternion en matrice de rotation a été démontrée par Stanley Wayne Shepperd [39] et son explication globale est décrite dans l'article de Ken Shoemake [40] . L'auteur a écrit : en prenant en compte que $w^2 + x^2 + y^2 + z^2 = 1$ pour un quaternion quelconque $q = (w, [x, y, z])$, la composition de la matrice de rotation sera :

$$M = \begin{bmatrix} 1 - 2y^2 - 2z^2 & 2xy + 2wz & 2xz - 2wy \\ 2xy - 2wz & 1 - 2x^2 - 2z^2 & 2yz + 2wx \\ 2xz + 2wy & 2yz - 2wx & 1 - 2x^2 - 2y^2 \end{bmatrix} \tag{13.6}$$

La plupart des termes sont le produit de deux facteurs, dont un qui est doublé. Shoemake a conclu sur sa conversion qu'elle utilise seulement 9 multiplications et 15 additions.

Des matrices aux quaternions

Afin de pouvoir calculer d'autres rotations nous récupérons les matrices de rotation et les convertissons en quaternions pour refaire les opérations.

$$C = A * B$$

Ce qui donne :

$$\begin{aligned} C.x &= A.w * B.x + A.x * B.w + A.y * B.z - A.z * B.y \\ C.y &= A.w * B.y - A.x * B.z + A.y * B.w + A.z * B.x \\ C.z &= A.w * B.z + A.x * B.y - A.y * B.x + A.z * B.w \\ C.w &= A.w * B.w - A.x * B.x - A.y * B.y - A.z * B.z \end{aligned}$$

13.1.3 Interpolation entre deux positions grâce aux quaternions

Un des grands apports de la mise en place des quaternions est la possibilité de créer des mouvements fluides entre deux positions calculées. C'est en partie pour cela que ces objets mathématiques sont utilisés dans la création des jeux vidéos afin d'avoir un mouvement continu et non saccadé de la caméra ou des objets 3D se déplaçant dans la scène 3D.

Ken Shoemake a introduit aussi l'interpolation entre deux positions dans le contexte d'animation : le **Slerp** pour **S**pherical **l**inear int**erp**olation, soit interpolation sphérique linéaire. Pour comprendre cette interpolation, prenons en exemple le trajet illustré en noir dans la figure 74.

Si on applique les déplacements les uns après les autres, on aura des changements brutaux entre chaque segment. L'interpolation Slerp permet de lisser le parcours et donc de générer le tracé en rouge sur la même figure que précédemment. Cette interpolation prend en compte la longueur du vecteur de rotation qui peut se traduire par la vitesse à laquelle le mobile (ou la caméra) se déplace. Le trajet sera donc différent selon le déplacement courant ce qui donne un ressenti d'inertie du mouvement.

Une fois les quaternions mis en place il reste toujours des problèmes d'affichage dus au grand nombre d'éléments à afficher. Les triangles générés par les processus de conversion sont stockés dans la mémoire allouée à la carte graphique et lorsqu'ils sont en trop grand nombre l'affichage est saccadé. Pour ne pas charger la totalité des objets 3D, nous mettons en place un affichage avec cône de vision. Son implantation est décrite ci-après.

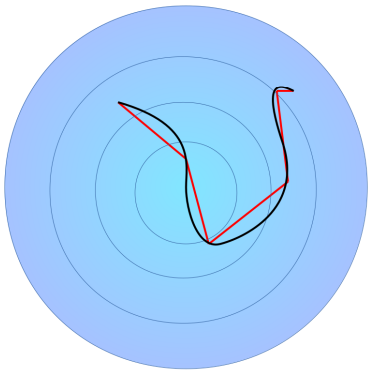


FIGURE 74 – Tracé aléatoire d'un trajet noir superposé au trajet rouge interpolé par la méthode Slerp

13.2 Cône de vision et faces cachées

La visualisation 3D amène toujours la problématique de ce qui est visible ou invisible. En effet, des parties sont cachées dans la scène 3D telles que les façades à l'arrière des maisons. Afin d'alléger la mémoire de la carte graphique, il serait bien de ne pas afficher ces parties. De plus les parties non présentes dans la fenêtre de visualisation n'ont pas besoin d'être chargées.

13.2.1 Cône de vision

Pour bien comprendre l'affichage 3D, il faut garder en tête que la scène vue à l'écran par l'utilisateur provient d'une caméra virtuelle que ce dernier peut bouger comme il le souhaite. Une caméra, comme celle illustrée dans la figure 75, possède plusieurs caractéristiques dont :

- *La profondeur de champ p* : elle correspond à la profondeur maximum du champ de vision. Les éléments présents au delà de cette limite ne sont plus affichés
- *la focale f* : elle correspond à la distance entre l'objectif et l'obturateur de la caméra. Elle a une incidence directe sur l'angle de champ de vision. Plus la focale est petite, plus l'angle du champ de vision est grand, et plus elle est grande, plus l'angle du champ est petit. Bien sûr, elle dépend de la surface de l'objectif, qui dans le cas de notre caméra virtuelle, peut être aussi modifiée.
- *l'angle de la caméra* : il est lié à la direction et au sens du vecteur définissant la caméra.

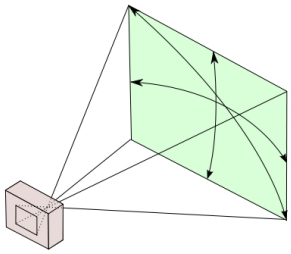


FIGURE 75 – Représentation du champ de vision de la caméra

Ces éléments permettent de définir un cône de vision. Nous calculons pour chaque élément affichable les centres et pour chacun nous calculons l'appartenance au cône de vision. Pour simplifier les calculs, considérons le cône de vision comme un triangle 2D avec une vue du dessus (voir figure 76), en effet, le cône de vision sera dans la plupart des cas orienté sur la ville et non dans le ciel, donc nous n'avons pas besoin de la 3D du champ de vision pour le calcul d'appartenance des éléments affichables.

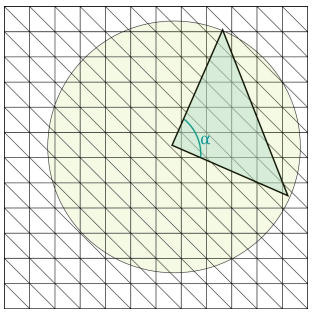


FIGURE 76 – Le cône de vision correspond au champ de vision et la caméra et les triangles sous-jacents aux triangles constituant la cartographie

Nous ne prenons donc en compte que la profondeur et la focale pour obtenir les coordonnées des sommets du triangle. Pour illustrer cette appartenance au champ de vision, considérons le triangle ABC comme champ de vision et le point P, le point de test (voir figure 77).

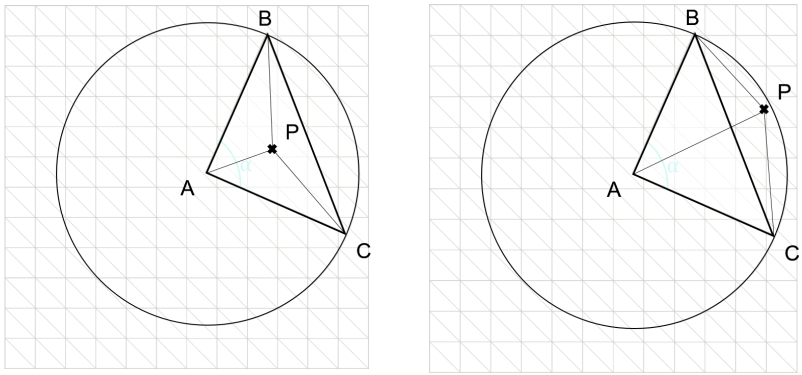


FIGURE 77 – Détermination de l'appartenance d'un point P au champ de vision

Nous appliquons ensuite un principe de l'algèbre linéaire, les coordonnées barycentriques. Une des règles utiles de ces coordonnées est la suivante : en calculant l'aire de chacun des triangles ABC,PBC,APC et ABP et en appliquant ces calculs :

$$\alpha = aire(P,B,C)/aire(A,B,C)$$
$$\beta = aire(A,P,C)/aire(A,B,C)$$
$$\gamma = aire(A,B,P)/aire(A,B,C)$$

Cela est valable si et seulement si la somme des coordonnées α, β, γ est égale à 1, le point est dans le triangle ABC (voir algorithme en annexe G). Afin de ne pas avoir des zones non chargées sur le bord de l'écran vu que nous ne prenons que les centres des éléments, nous avons volontairement élargi le champ de vision pour avoir une zone tampon. Cette zone permet aussi de limiter les zones non chargées lors de mouvements de la caméra. Le résultat des triangles pris en compte est dessiné dans la figure 78.

13.2.2 Cas des faces cachées

Afin de ne pas charger en mémoire ces faces cachées un principe mathématique simple est mis en place. En effet, la caméra utilisée pour visualiser la scène 3D se sert d'un vecteur correspondant à la direction de la vue, en d'autres termes, le point de fuite. Ce vecteur a une direction, un sens et une longueur. La direction et le sens sont utiles au déplacement mais

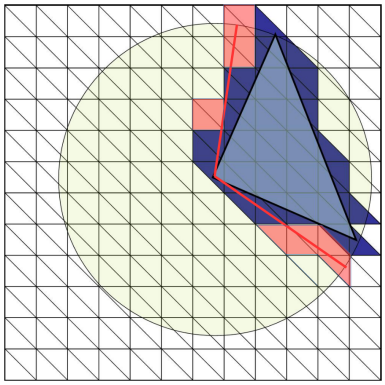


FIGURE 78 – Une fois les calculs des coordonnées barycentriques faits, l'appartenance des triangles de topographie est trouvée. Les zones en rouge correspondent à l'élargissement volontaire de la zone d'affichage

également pour savoir si les faces doivent être affichées. Sachant que chaque face comprend une normale pour la gestion de la lumière (voir partie 10.3.3) en calculant le produit scalaire entre ces deux vecteurs, deux cas sont possibles :

— En premier, le résultat est positif, cela veut dire que les deux vecteurs ont des directions et sens similaires donc que la face a la même orientation que le vecteur de vision de la caméra.
— Dans le deuxième cas, le produit scalaire est négatif, ce qui se traduit par un sens opposé, alors le plan fait face à la caméra, celui-ci est alors visible.

Pour illustrer ce comportement prenons comme exemple une maison à quatre murs avec, pour chacun, une norme calculée. Lorsque que la norme a un sens opposé à celui du vecteur de vision, les façades sont visibles, les autres sont cachées. Ce test est fait pour chaque mouvement de caméra pour recalculer les produits vectoriels entre chacune des façades. Dans le cas de notre exemple, sans prendre en compte le cas des toits, seulement deux murs sont gérés par la carte graphique (voir figure 79). Sachant que la plus grande donnée visualisée est constituée de bâtiments, nous pouvons approximer que nous diminuons de moitié le nombre de triangles à afficher.

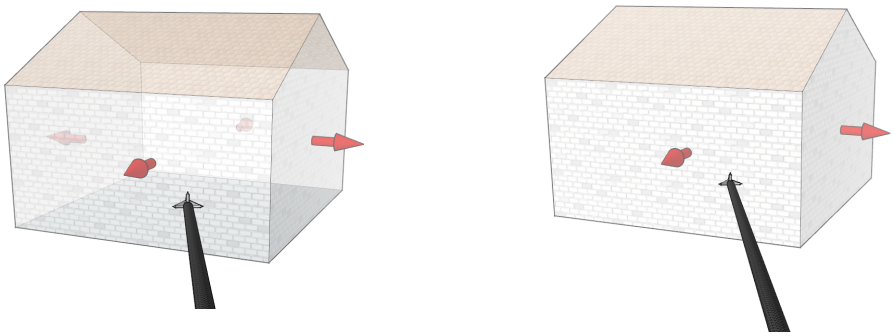


FIGURE 79 – Exemples de gestion de faces cachées grâce aux normales : les flèches rouges correspondent aux normales et la noire au point de vue de la caméra. A gauche la maison est dessinée avec ses 4 murs, une fois la prise en compte des normales, seulement 2 façades sont prises en compte pour l'affichage

13.3 Rendu visuel et objets 3D

En plus de la conversion 3D des données SIG, il a été nécessaire d'utiliser des techniques et des outils provenant de jeux vidéos afin d'avoir un rendu visuel agréable et plus immersif. Pour cela, le choix a porté sur la mise en place des objets 3D, telles que des voitures, des feux de signalisation ou des arbres, un rendu de ciel prenant en compte le cycle jour et nuit, des choix de vues différentes ainsi que des vues immersives grâce à la stéréoscopie ou aux hologrammes. Les prochaines parties vont détailler ces points de décor.

13.3.1 Skydome

Une scène 3D générée par défaut par OpenGl ne contient pas d'environnement, tout est à construire. Afin d'avoir un affichage plus réel, un ciel doit être présent, ainsi qu'un horizon. Une construction en 3D avec des objets est possible mais peut être gourmande en ressource. Une solution plus simple, et souvent utilisée dans les environnements 3D, est la mise en place d'une skybox ou d'un skydome. Ces deux types de rendu ont le même principe : placer la scène dans une immense boite (skybox, voir figure 80) ou sous un dôme (skydome,voir figure 81) et plaquer dessus une texture de ciel. Bien sûr afin de ne pas avoir des déformations de texture, il faut que l'image du ciel soit adaptée à une boite ou à un dôme. Nous avons utilisé une sphère avec un plaquage de texture tout autour. La figure 81 illustre le rendu avec la mise en place d'un skydome.

FIGURE 80 – Texture pour une skybox

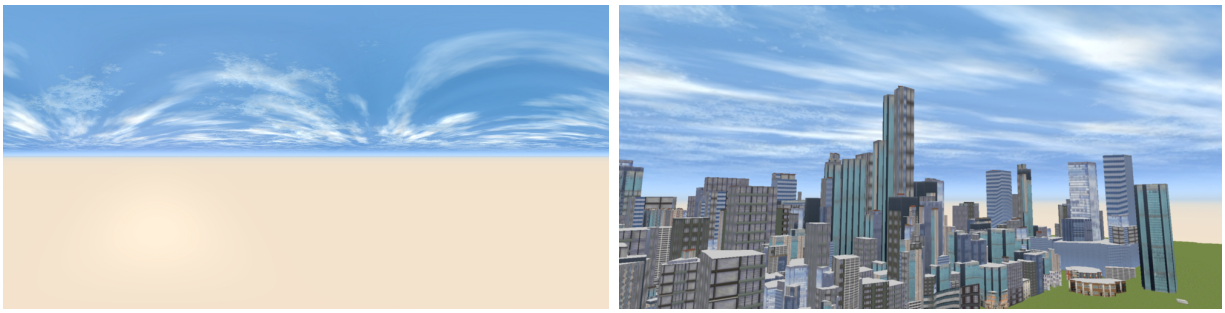


FIGURE 81 – A gauche, la texture de ciel prenant en compte la déformation pour application sur une sphère. A droite, la même texture plaquée sur une sphère

Afin de ne pas ressentir que la scène 3D se trouve dans un espace fini, nous avons centré la sphère sur la position de la caméra. Ainsi, la "bulle" entourant l'observateur se déplace à chaque mouvement. Le résultat se traduit par un effet statique du ciel sans la perception de se rapprocher du bord de l'affichage.

13.3.2 Utilisation d'objets 3D

Un objet 3D est un objet créé par des outils tels que Blender®ou Sketchup®. Ces objets sont décrits par des faces plus ou moins petites. Pour chacune, leur normales sont calculées pour la gestion de la lumière. Toutes ces faces, découpées en triangles pour une meilleure gestion par les cartes graphiques, sont enregistrées dans un fichier texte. Pour les objets les plus complets, des textures sont aussi disponibles. Tout comme le texturage des bâtiments, décrit dans la partie 7.3.1, un fichier avec toutes les parties de l'objet 3D existe (voir figure 82).

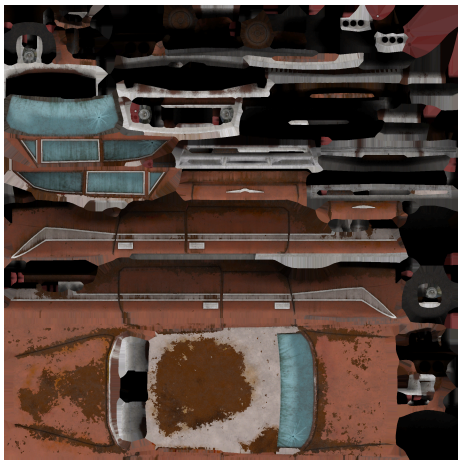


FIGURE 82 – Image contenant toutes les parties à texturer de la voiture

Le placage est fait automatiquement grâce aux coordonnées stockées dans le fichier de l'objet 3D. Ce procédé permet un affichage des objets 3D. Ces objets sont mémorisés une seule fois sans que cette opération ne gène un affichage multiple.

Nous obtenons ainsi une bibliothèque d'objets 3D, avec des feux de signalisation, des arbres et des véhicules que nous pouvons appliquer à la ville (voir figure 83).

FIGURE 83 – Différents objets 3D utilisés dans la simulation

Quatrième partie

Validation de la méthodologie

Chapitre 14

Résultats visuels

14.1 Visualisation 3D sur ordinateur

Nous avons vu jusqu'ici la mise en place des conversions de données, des différents moteurs de simulation et de l'Intelligence Artificielle. Nous allons maintenant illustrer ces parties par des résultats visuels, obtenus à partir du logiciel développé grâce aux méthodologies décrites précédemment. Nous verrons la partie 2D, sur laquelle se basent tous les calculs, et la partie 3D avec ces différentes immersions possibles.

14.1.1 Vue 2D

La vue 2D, incluse dans la fenêtre principale, est gérée par l'API Qgis. Les couches vectorielles et raster sont gérées dans une "pile" d'éléments affichables et peuvent être visibles ou non via un panneau dédié à la visualisation. Nous avons donc une sous-fenêtre pour l'affichage à plat de la donnée comme illustré par la figure 84.

La vue est orientée avec le nord en haut de l'affichage. La projection choisie est celle référencée dans les fichiers SIG importés au début du processus. Cette vue nous permet de visualiser dans sa globalité les données. En effet, elle permet une meilleure appréhension des informations dans un premier temps. Nous verrons dans le chapitre 15 consacré aux perspectives la mise en place de la simulation 2D en parallèle de celle visualisée en 3D.

FIGURE 84 – Données 2D sur la ville de New York

14.1.2 Vue aérienne (mode hélicoptère)

Tout comme la vue 2D décrite précédemment, la vue aérienne est utile pour pouvoir appréhender globalement un phénomène. Ainsi, elle permet de regarder la scène 3D sous différents angles et de mieux comprendre la simulation. Pour avoir ce sentiment de survol, l'utilisateur a la possibilité de se déplacer librement dans la vue 3D. Les déplacements sont ceux appliqués sur la caméra, c'est-à-dire le roulis, le lacet et le tangage. Ainsi, la souris permet de changer ces angles tandis que les flèches de direction du clavier permettent d'avoir des translations dans toutes les directions. De plus, les touches A et Q permettent respectivement de monter ou de descendre le long de l'axe Z. Ce mode de visualisation est activé par défaut et peut être appelé depuis une autre vue via la touche H.

FIGURE 85 – Exemple de résultat obtenu via la visualisation aérienne

Cette vue est très utile pour visualiser une ville en entier, et voir macroscopiquement le trafic urbain.

Pour l'utilisateur, choisir cette visualisation a pour avantage de permettre une observation en temps réel et globale des trajets réalisés par les véhicules. Elle permet principalement de définir les flux principaux qui en découlent sans avoir un affichage précis des mobiliers, en particulier les feux de signalisation et les panneaux de signalisation, non visibles à cette échelle.

Cette visualisation globale, qui permet de simuler des flux principaux sans avoir à rentrer dans des détails, permet de simplifier la compréhension d'un phénomène simulé. Ce mode de représentation peut alors être largement employé par divers métiers comme par exemple :

- les collectivités et mairies qui souhaitent appuyer un scénario mis en place sur la structure globale de l'agglomération,
- les agents immobiliers pour permettre aux éventuels futurs acquéreurs de mieux appréhender le quartier où s'implante le projet,
- les architectes pour présenter aux décideurs l'implantation globale du projet étudié,
- les gestionnaires de voiries qui veulent connaître la structure globale des flux qui auront lieu en cas de travaux ou de déviations,...

Cette modélisation générale à l'échelle du quartier, voire de la ville, va alors de pair avec une représentation plus détaillée, à l'échelle de la rue, de la simulation de trafic urbain permise par le mode "terrestre" illustré dans la partie ci-après.

14.1.3 Vue depuis une voiture

Ainsi, en plus de la vue aérienne, une vue au niveau terrestre est possible. Soit la vue est libre, soit elle est liée au déplacement d'un véhicule. Ces visualisations nous permettent de nous représenter l'espace en tant que piéton ou comme conducteur. Dans ce dernier cas, nous pouvons être à la place du conducteur à l'intérieur de la voiture, ou alors au niveau du toit à l'aide de la touche I. Ce mode de visualisation est illustré sur la figure 86.

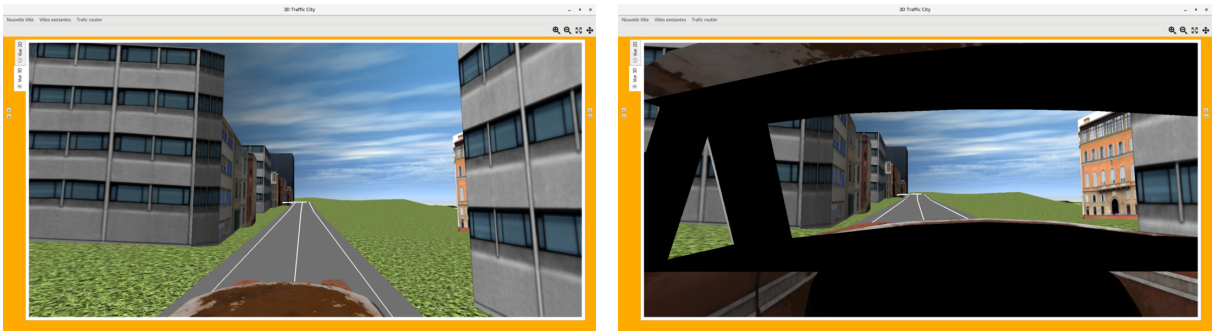


FIGURE 86 – Exemples de résultats obtenus via la visualisation terrestre

Pour activer cette modélisation, la touche V du clavier doit être actionnée. Dès lors, l'utilisateur verra un changement de mode de caméra qui sera déplacée pour se trouver au niveau du toit d'un des véhicules. Au cours de cette représentation, la caméra suit les mouvements du véhicule qui se déplace le long des routes suivant l'itinéraire qui lui a été affecté. Il est évident que tout au long de ce choix de visualisation de la simulation, l'utilisateur peut déplacer le champ de vision en même temps, et ce tout au long du parcours pour mieux appréhender l'environnement simulé.

Si suivre le parcours d'un véhicule a l'avantage de se permettre une meilleure représentation de l'itinéraire suivi, l'utilisateur peut également souhaiter visualiser différents trajets. Il a alors fallu également travailler à proposer aux usagers un possible changement de véhicule au cours de la simulation. Ainsi,s'il désire changer d'itinéraire, et donc de véhicule, un nouvel appui sur la touche V permettra de passer à un autre véhicule déjà présent dans l'environnement simulé.

Ce mode de représentation présente de nombreux avantages et en particulier il permet :

— **d' illustrer** les réactions du véhicule grâce aux règles du Code le Route qui ont été intégrées au Système Expert,

— **de se représenter** les réactions d'un véhicule dans l'environnement simulé pour vérifier que les méthodologies développées semblent cohérentes avec la réalité,...

Cette approche d'une visualisation "in situ" permettra alors à de nombreux métiers une meilleure représentation de certains phénomènes comme :

— l'accessibilité de certaines routes à des véhicules particuliers (convois exceptionnels, camions de déménagements, bus,...),

— la nécessité de mettre en place de nouveaux éléments de voiries (ronds points, panneaux ou feux de signalisations,...) pour une meilleure fluidité du trafic,

— l'amélioration de la gestion de l'espace routier pour une meilleure répartition véhicules motorisés - vélos - piétons,...

Une fois la mise en place des différentes caméras réalisée, il est intéressant, pour apprécier au mieux l'immersion, d'en perfectionner le rendu. Nous nous sommes ainsi penchés sur la 3D immersive, appelée aussi réalité améliorée. En effet, la réalité virtuelle est de plus en plus commune que ce soit dans les jeux vidéos ou dans l'imagerie médicale [41] [42] [43]. Pour y arriver, plusieurs solutions sont disponibles. Nous en avons choisi deux utilisées couramment

dans les jeux vidéos : l'anaglyphe et la réalité virtuelle. De plus, nous avons également proposé d'utiliser une technique novatrice dans le domaine de la cartographie, et plus particulièrement dans celui de la géomatique avec simulation de trafic : l'hologramme. Ces méthodologies et résultats seront exposés dans les paragraphes suivants.

14.2 Autres méthodes de visualisations 3D

Alors que l'idée de se représenter des scènes en 3D date de certains dessins du XIIIème siècle, c'est à partir du XIXème siècle que se développe l'accès à la représentation 3D grâce à la photographie et aux stéréoscopes. Différentes techniques ont alors été mises en place, dont :

— la stéréoscopie,
— l'anaglyphe,
— l'holographie.

Ces techniques sont reprises dans diverses technologies qui se généralisent à l'heure actuelle, c'est pourquoi ce travail a également été traité pour répondre à ces nouvelles problématiques/ces nouveaux outils.

14.2.1 Anaglyphe

L'anaglyphe est une branche de la stéréoscopie utilisant la superposition de deux images de couleurs complémentaires afin de créer une vue en relief. Ce procédé à été découvert par Wilhelm Rollman en 1853 [44] et à été illustré à l'Académie des Sciences de Paris en 1858 par Joseph-Charles d'Almeida. Le principe est assez simple à mettre en place : deux images de la même scène sont créées avec un léger décalage. Ce décalage prend en compte l'écartement des yeux, et le point de convergence (voir figure 87). La position du point de fuite va déterminer l'effet de mise en relief ou l'effet de profondeur. Les couleurs complémentaires choisies sont le rouge (à gauche) et le cyan (à droite). D'autres couleurs peuvent être choisies mais celles-ci sont celles qui fonctionnent le mieux. L'utilisation de lunettes anaglyphiques de mêmes couleurs permet de visualiser la scène en relief.

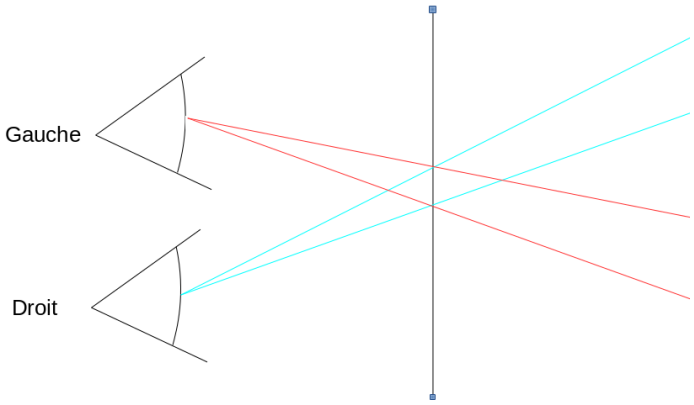


FIGURE 87 – Principe de l'anaglyphe : l'oeil droit ne voit que les parties en cyan, tandis que l'oeil gauche ne voit que les parties rouges

Le résultat de cette transformation est illustré dans la figure 88 sur la partie de gauche. Nous pouvons remarquer que tous les éléments rouges sont situés à droite des éléments cyan. Cela se traduit par le fait que le point de convergence se situe en avant de la scène et n'est pas visible. Une fois les lunettes portées, on remarque un effet de profondeur et non de relief.

FIGURE 88 – Exemples de vues anaglyphiques, Sources : i.ytimg.com, GNU

Pour avoir un effet de relief, comme si les éléments sortaient de l'écran, il faudrait placer le point de convergence plus loin dans la scène 3D. Ainsi, des éléments en cyan seront à leur tour à droite des éléments en rouge. La figure 88, à droite, illustre ce cas. Nous avons choisi de mettre un effet de profondeur qui est, semble-t-il, plus facile à supporter visuellement (cf. figure 89).

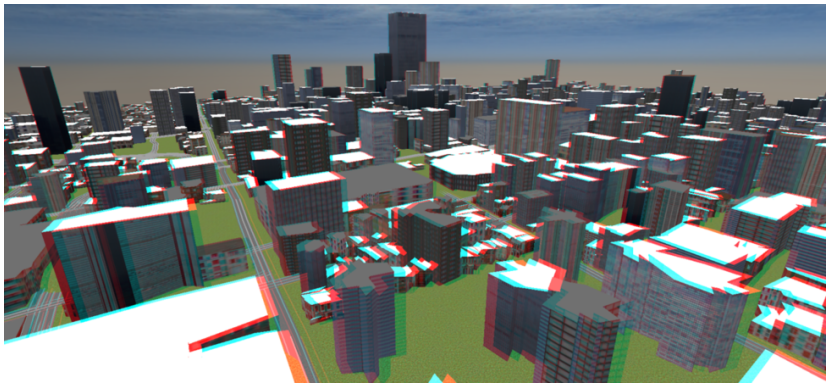


FIGURE 89 – Vue anaglyphique obtenue

Avantages

Le principal avantage de la technique de l'anaglyphe est sa mise en place. Elle est d'autant plus facile que l'on travaille dans un environnement 3D avec une gestion de caméra manuelle. Nous avons considéré l'affichage par défaut comme l'image de gauche, donc pour générer les vues stéréoscopiques rouge et cyan, on colorie la vue principale en cyan et l'on décale la caméra pour créer la vue rouge. Un autre avantage est son coût, les lunettes pour anaglyphe se trouvent facilement et sont de moindre coût.

Inconvénients

Un des inconvénients, qui à juste titre a freiné l'utilisation massive de ce procédé, est la perte de la coloration initiale. Nous avons vu précédemment que la couleur cyan est la couleur complémentaire au rouge, donc la superposition de ces deux couleurs rend tous les éléments gris. De plus, l'utilisation de textures complexes et multicolores peut mettre en défaut cette mise en place, surtout si les couleurs sont proches du rouge ou du cyan. Nous verrons dans la partie suivante la résolution de ce problème. Un autre inconvénient est de ne pas pouvoir l'utiliser sur de longues périodes. Le travail oculaire est très important avec cette méthode.

14.2.2 Stéréoscopie et vue immersive

La stéréoscopie est de plus en plus courante grâce aux casques de réalité virtuelle tel que l'oculus rift ou le Playstation VR. Cette technologie n'est pas si nouvelle car elle se base sur les travaux de Charles Wheatstone de 1853 [45], de David Brewster [46] et de Oliver Wendell Holmes de 1859 [47].

Tout comme l'anaglyphe elle repose sur le fait que l'homme a une vision binoculaire avec, a fortiori, un décalage de la vue entre les deux yeux. Cette association de deux images nous permet d'avoir une perception en profondeur ou en relief. Par contre le grand avantage par rapport à l'anaglyphe est qu'il n'y a plus de perte de couleur. En effet, le principe restant le même, deux images avec un décalage sont générées mais cette fois-ci, elles sont mises l'une à côté de l'autre. Vu que l'écartement des yeux ne change pas, un accessoire supplémentaire doit être utilisé pour superposer ces images : le stéréoscope (voir figures 90).

(a) (b)

FIGURE 90 – Exemple de vue stéréoscopique ancienne (a) et de stéréoscopes antiques (b)

Après de multiples améliorations, le stéréoscope est de nos jours un casque composé de deux lentilles une en face de chaque oeil et d'un écran (voir figure 91). Les lentilles permettent à l'oeil de ne pas forcer pour superposer les deux images.

FIGURE 91 – Photographie d'un CardBoard, Source : /www.popsci.com

La réalisation de la vue pour le casque de Réalité Virtuelle

Contrairement à la méthode de l'anaglyphe expliquée dans la partie précédente, la stéréoscopie à visualiser via un casque de Réalité Virtuelle n'est pas sur une seule image mais bien sur deux scènes côte à côte. Ceci nécessite alors de restructurer la scène d'affichage à la fois pour permettre cette double représentation, mais également pour mettre à l'échelle la scène afin d'être présentée sur des écrans beaucoup plus petits, environ 4 à 6' de diagonale. Une autre problématique est la gestion des capacités d'un téléphone ou d'un casque avec écran intégré à afficher une scène importante comme cela sera exprimé dans le sous-chapitre 15.1.2.

Générer cette double vue nécessite de prendre en compte l'écart inter-pupillaire afin de générer deux scènes légèrement décalées à l'image de ce qu'observe chacun des deux yeux d'un

individu afin que le cerveau puisse associer pour recréer une impression de relief dans la scène. Ceci est illustré sur la figure suivante.

FIGURE 92 – Double scène obtenue pour une visualisation via des casques de Réalité Virtuelle

Avantages

Cette méthode permet de garder les couleurs de l'image et permet un rendu encore plus réaliste. C'est pour cela que ce procédé est utilisé aujourd'hui dans de nombreux domaines tels que la médecine (thérapie contre les phobies [41], dépistage des signes d'Alzheimer [42] ou encore une assistance pour les opérations cardiaques [43]), l'armée (projet DAVD pour la cartographie des fonds marins, simulateur pour entraînement aux tirs [48]) ou l'immobilier et la construction (visite en réalité virtuelle, suivi de travaux [49]) pour ne citer que ceux-là.

Inconvénients

L'inconvénient majeur est son coût bien qu'il y ait des possibilités de se procurer un casque de réalité virtuelle peu cher en utilisant un smartphone comme écran. Cette solution permet de profiter d'un accès simple à la réalité virtuelle mais elle est de moins bonne qualité que les solutions complètes proposées par Facebook®(Oculus Rift), Sony®(Playstation VR) ou Valve®(HTC Vive). Elles sont comprises entre 450 euros et 800 euros sans la partie gérant le calcul (figure 93). Cela dit, les prix devraient rapidement chuter avec le nombre croissant d'utilisateurs.

(a) (b)

FIGURE 93 – Exemples de casques de Réalité Virtuelle sans écran intégré (a) ou avec écran intégré (b)

14.2.3 Vue par hologramme

La dernière visualisation que nous avons choisie est basée sur le principe des hologrammes. Par contre l'utilisation du nom *hologramme* est erronée car on ne projette pas d'image 3D. En effet, un hologramme est une représentation d'une image en trois dimensions apparaissant comme suspendue en l'air. Au cours de ces travaux, nous utilisons plus précisément le principe de réflexions d'images 2D sur une une pyramide inversée 3D constitué de 4 surfaces planes avec une certaine réflectivité.

Pour projeter les quatre images, nous utilisons un téléphone portable sur lequel on affiche quatre vues avec des angles différents (voir figure 94).

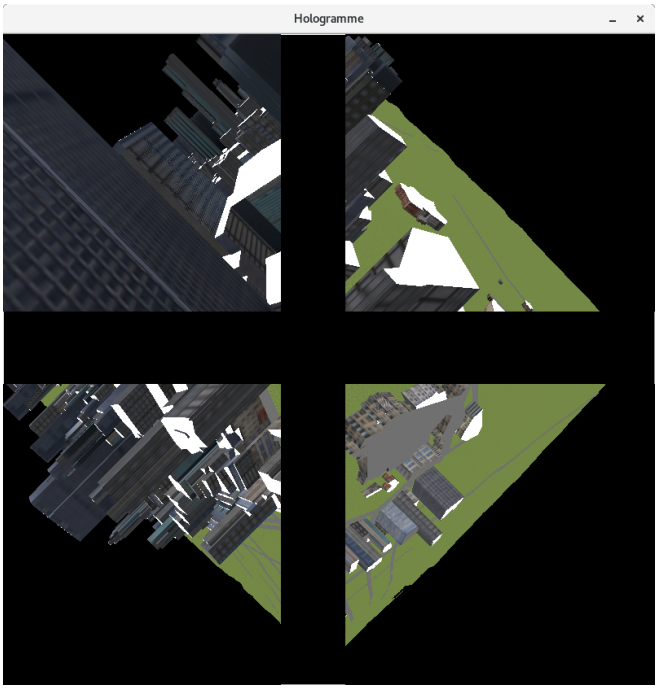


FIGURE 94 – Résultat obtenu pour proposer une visualisation holographique

Ces quatre vues sont placées dans des triangles avec une pyramide inversée, tronquée ou non, au dessus afin d'avoir une réflexion sur chacune des faces. La plupart des images projetées de cette façon sont identiques sur chacune des faces afin que les observateurs voient la même information quelle que soit leur position (Figure 95).

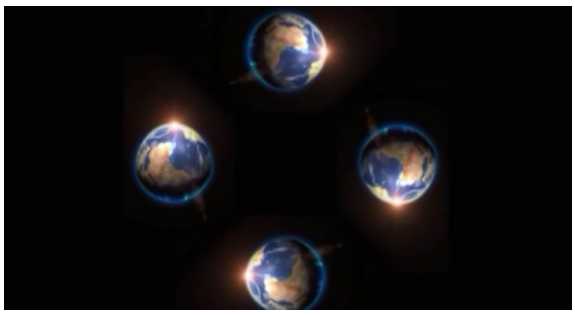


FIGURE 95 – Les quatre vues holographiques telles que proposées en ligne

Ce travail étant basé sur les sciences cartographiques, il a paru intéressant de s'éloigner de cette tendance en proposant quatre vues liées aux quatre points cardinaux (voir figure 96) afin de pouvoir regarder tout autour d'un point, en augmentant ainsi cette impression de

représentation dans l'espace. Grâce à ce choix, en tournant autour du prisme, l'utilisateur peut voir tous les éléments simulés.

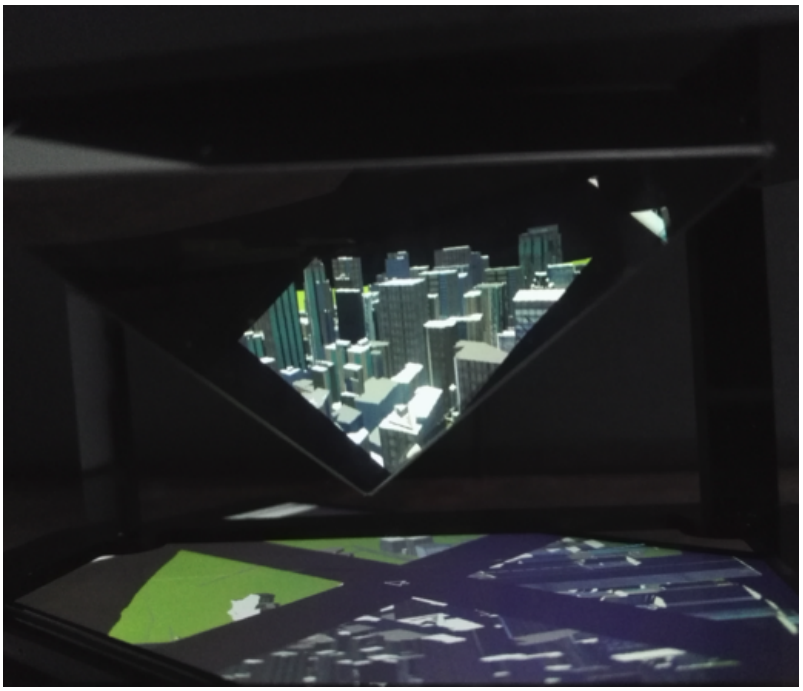


FIGURE 96 – Hologramme obtenu

Ainsi, en plus de permettre à plusieurs personnes de regarder un environnement 3D simultanément, il est possible de le faire sous différents angles pour mieux se représenter la scène étudiée dans sa globalité.

Chapitre 15

Limitations et perspectives

Au cours de notre travail de thèse nous avons dû faire des choix, aussi bien techniques que sur la réflexion, tant le sujet peut être infini. En effet, nous explorons diverses thématiques comme la conversion de données, la simulation, l'Intelligence Artificielle, la visualisation 2D ou 3D et même l'immersion via la réalité virtuelle. Nous verrons ici les limitations et les perspectives de ce travail.

15.1 Limitations

Plusieurs types de limitations nous ont contraint à prendre des directions plutôt que d'autres. Certaines sont dues au matériel tandis que d'autres sont liées à des choix d'architectures logicielles.

15.1.1 Choix matériels - Portabilité

Faire le choix de proposer cette méthodologie à des fins de créer un programme entièrement OpenSource nécessite de prendre en compte la portabilité des algorithmes, puis des codes, créés. Sachant que ce code à été pensé pour intégrer la licence GNU, nous avons gardé en tête la forte accessibilité de la structure même du code, et aussi le fait d'avoir des interactions homme-machine à bas coûts afin de toucher un large public. Ainsi une des limitations principales a été de mettre au point un code portable, facilement intégrable, aussi bien pour la partie visualisation sur ordinateur que pour celle traitant de la Réalité Virtuelle via des casques (la seule limitation étant la présence d'un gyroscope et d'un écran suffisamment grand pour l'utilisation de lunettes de type Google CardBoard).

15.1.2 Des inconvénients liés à la visualisation immersive

Dans cette partie, seront exposées un certain nombre de limitations liées à la visualisation immersive.

15.1.2.1 Capacité des appareils mobiles

En plus de la limitation de la taille de la projection (comme exprimée dans la section suivante), d'autres inconvénients se sont présentés, comme par exemple la gestion des ressources sur un téléphone mobile. En effet générer quatre scènes 3D avec des vues conjointes sur un

écran de petite taille et un processeur graphique limité représente un coût en terme de performance et d'énergie. Ainsi d'autres moyens ont dû être mis en oeuvre, comme le partage d'écran par exemple (via un outil appelé TeamViewer. Cet outil développé par *TeamViewer GmbH* permet de dupliquer l'écran d'un appareil sur un autre terminal par streaming. Il est souvent utilisé pour le dépannage informatique à distance. Cependant d'autres inconvénients liés à cette méthodologie peuvent être mis en évidence comme la perte du mouvement dans le casque de réalité virtuelle. En effet, l'affichage est dupliqué sur un autre écran, et l'échange d'informations ne peut se faire que dans un sens, donc la position via le gyroscope de l'écran cible n'influence pas l'affichage source. Le résultat est alors un écran statique où les mouvements sont possibles via le déplacement de la souris ou l'utilisation des touches du clavier de l'ordinateur source ou alors via d'autres outils que nous présenterons dans la section 15.2.6.

15.1.2.2 Taille de la projection holographique

L'inconvénient majeur est lié à la taille de la représentation. Comme expliqué dans le chapitre 14, nous utilisons un téléphone et un prisme de la même taille pour projeter les images, ainsi, l'image est très petite (de l'ordre de 4 cm). Afin de permettre une meilleure visualisation, il faudrait adapter ce principe avec un plus grand écran et une plus grande pyramide ou à d'autres techniques de projections holographiques (à l'image des réalisations de "la compagnie Adrien M. et Claire B.").

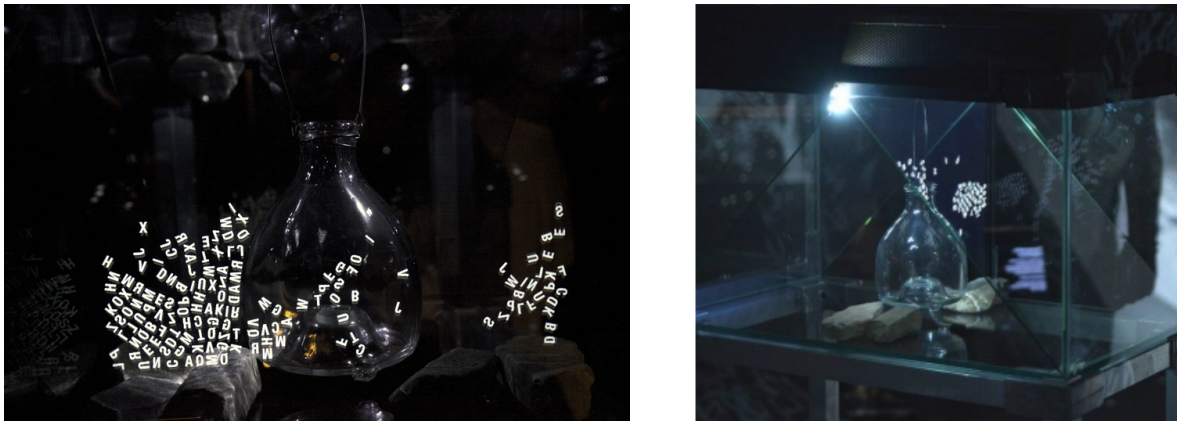


FIGURE 97 – Exemple de réalisation de la compagnie Adrien M. et Claire B mêlant Réalité Augmentée et Projection 3D

15.1.3 Des différences de résultats

Comme nous l'avions exprimé au cours du chapitre 7, certains problèmes de résolutions natives peuvent être limitants. La méthode de Laporte a été utilisée afin d'améliorer la précision selon les paraboloïdes hyperboliques. Ceci nous a permis de passer d'une résolution originelle de 90m à une résolution recalculée de 10m. Si cette méthode permet d'obtenir une meilleure visualisation plus lisse du terrain, cette précision n'est pas encore forcément adaptée à celle des éléments intégrés à la scène, et en particulier les bâtiments et les routes. De plus, il aurait pu être intéressant d'avoir la forme réelle des trottoirs ou même de la chaussée afin de vérifier la donnée d'OSM concernant les routes et leurs largeurs.

15.2 Perspectives

Au cours de cette étude beaucoup de problématiques nouvelles ont été mises en évidence et certaines restent encore à développer. Ce sont ces thématiques que nous allons exposer dans les paragraphes suivants.

15.2.1 Vérification de la modélisation

La première des perspectives qu'il semble évidente et nécessaire de développer est la mise en place d'un outil de contrôle temps réel de la simulation avec la réalité. Il s'agirait alors de proposer une double visualisation permettant de confirmer que la simulation générée semble cohérente avec un trafic urbain en situation réelle. Ceci nécessiterait alors de considérer des données 2D temps réel des tronçons routiers utilisés et/ou l'accès aux caméras de trafic temps réel. Le travail aurait pu être couplé avec l'analyse du trafic routier par caméra que la société ACIC réalise. Leurs analyses permettent de détecter les vitesses (figure 98 ainsi que le comportement des conducteurs sur certaines portions de route. Ces informations pourraient également permettre de mettre à jour régulièrement le SE avec de nouvelles règles comportementales.

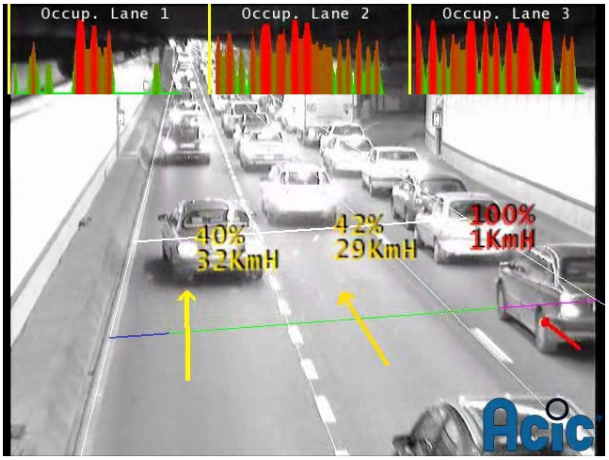


FIGURE 98 – Détection de la vitesse des véhicules par caméra

15.2.2 Amélioration de l'interface

La création de l'interface homme-machine a été une partie du travail sur laquelle nous nous sommes penchés longuement afin d'avoir une interface simple et épurée permettant de se focaliser sur la simulation. Nous pourrions encore y intégrer quelques modules utiles comme par exemple une interface dédiée à la simulation en cours avec les tronçons occupés, la vitesse de chaque véhicule ou encore la possibilité de de modifier certains paramètres à la volée comme la vitesse autorisée sur certains tronçons ou la praticabilité d'un tronçon. De plus un module d'export de la simulation en vidéo aurait pu être envisagé afin d'avoir une sortie dans un format propice à la prise de décision. Ces modules peuvent mis en place rapidement dans l'optique d'améliorer le modèle.

15.2.3 Ajout de l'orthophoto et amélioration de la donnée topographique

Une partie du travail exposé au cours de ce manuscrit traite du rendu réaliste de la scène modélisée et simulée. Mais ce rendu nécessite alors une visualisation la plus proche possible de la réalité et ceci peut passer par la prise en compte de l'orthophoto de la zone étudiée. Ces orthophotos peuvent alors être appliquées au MNT pour un sol plus réaliste (désert, goudron, prés,...). Elles sont généralement acquises par avion (vue aérienne) ou satellite (vue satellitaire). Elles se présentent sous forme de dalles géoréférencées.

Ces orthophotos peuvent être obtenues grâce à diverses entités, et en particulier l'USGS ou l'IGN. Ce travail aboutissant sur un logiciel entièrement Opensource, il est nécessaire de prendre en données orthophotos des données libres d'utilisation (ce qui peut avoir un impact sur leur précision et/ou sur les zones couvertes). certaines structures mettent à disposition ces données telles que :

IGN : Mise à disposition sous licence ouverte etalab de l'orthophoto sur le territoire français avec une résolution de 5m. Certains département sont aussi disponibles avec une résolution de 50cm.

USGS : Données sur l'ensemble des États-Unis avec des résolution allant jusqu'à 30cm.

Le résultat avec un placage est illustré dans la figure 99. Le placage a été fait selon le géoréferencement des dalles en appliquant une texture à chaque triangle formant la topographie.

FIGURE 99 – Placage 3D de dalles d'ortophoto sur un terrain, Source : projet SIG4D, promo IASIG 2012

15.2.4 Améliorer la précision du Modèle Numérique de Terrain

Des traitements plus poussés peuvent être réalisés pour améliorer la précision de la donnée topographique. En effet, les regards des réseaux enterrés ont souvent une donnée d'altitude en attribut qui est, généralement, de bonne qualité. Ces données auraient pu être intégrées à la topographie avant interpolation. Par contre, cela amènerait une nouvelle contrainte liée à la disponibilité de la donnée. Cependant, avec une présence mondiale, ces coordonnées tridimensionnelles permettraient d'améliorer grandement la précision altimétrique au niveau du réseau routier.

15.2.5 Améliorer la modélisation du trafic urbain

La modélisation elle même peut être améliorée en prenant en compte beaucoup de nouveaux paramètres et mobiles.

Rendre la simulation plus réaliste

Pour la rendre plus réaliste, on peut ne pas se limiter à une classe véhicule mais à une hyperclasse véhicule (figure 100) prenant en compte les différents véhicules comme : la police, les pompiers, les ambulances ou même les véhicules militaires.

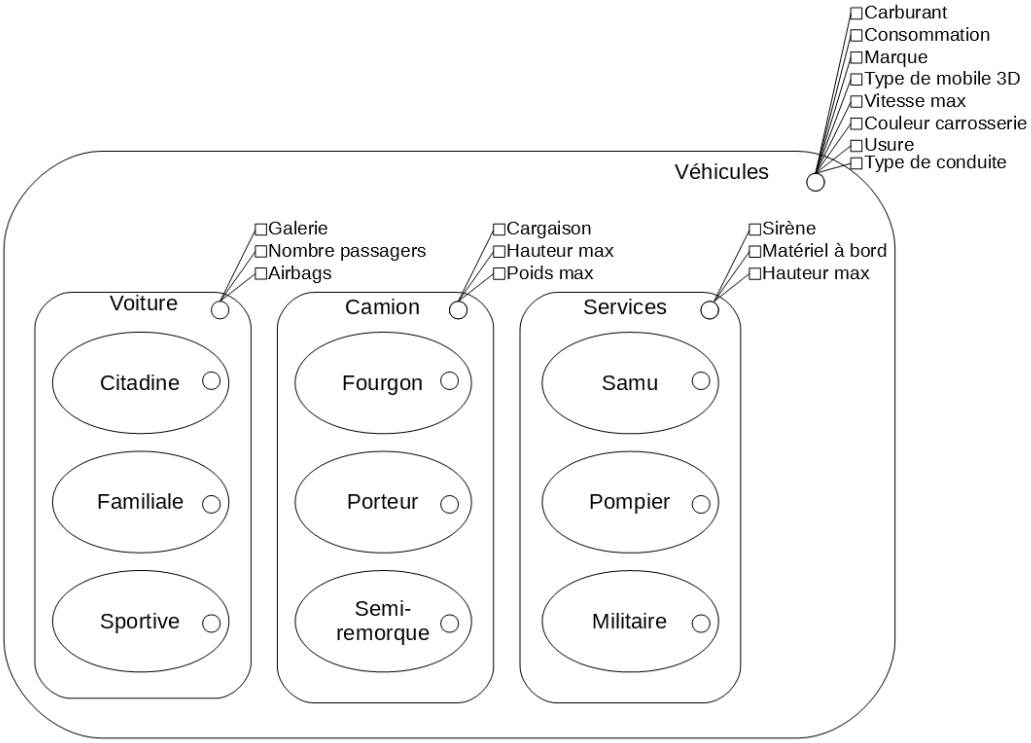


FIGURE 100 – Exemple de l'hyperclasse Véhicules

Nous pouvons noter que la structuration pourrait aller encore plus loin en ajoutant d'autres hyperclasses de véhicules (aériens ou aquatiques par exemple) ou transformant les classes présentées dans la figure précédente en hyperclasses comme l'illustre la figure 101 avec la structuration de l'hyperclasse "Militaire" qui regroupe différents véhicules de l'armée.

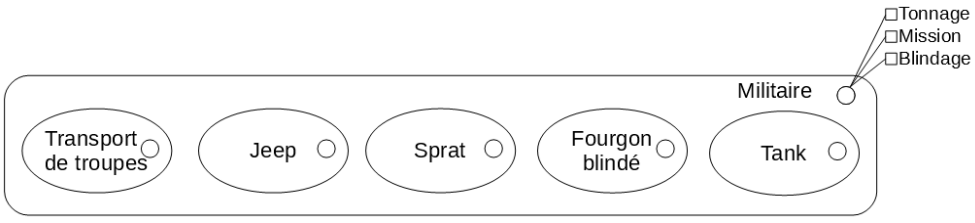


FIGURE 101 – Hyperclasse regroupant les véhicules militaires

Bien sûr ces véhicules auraient, visuellement, des objets 3D mobiles attitrés mais surtout des comportements bien différents. On peut penser qu'une ambulance pourrait avoir des trajets générés automatiquement lors d'une collision entre deux véhicules. Le trajet serait alors

construit selon la position de l'hôpital et cet accident. L'association entre les services de la police et des pompiers pourrait également être envisageable. De plus, pour les véhicules militaires, seules certaines routes seront adaptées à leurs passages (largeur des voies, renfort, hauteur des tunnels,...).

Des éléments environnementaux pourraient être aussi pris en compte. Par exemple des glissements de terrain peuvent affecter la circulation, de même pour des incendies. Comme expliqué précédemment, les secours seront "appelés" afin de venir sur les lieux rapidement. Le système expert auraient alors des règles spécifiques pour ce type de situation et auraient aussi des temps d'occupation de tels ou tels mobiles selon le type de phénomène naturel à gérer.

Enfin, la topographie pourrait être prise en compte pour le calcul de l'accélération ou du freinage des véhicules. Les mobiles les plus lourds n'auraient pas le même comportement par rapport à une petite citadine.

D'autres règles de circulation

Réaliser une simulation du trafic urbain nécessite, comme expliqué dans le chapitre 11, de prendre en considération un certain nombre de règles qui régissent le Code de la Route. Mais qu'on soit en France, en Inde, ou aux Etats-Unis, les obligations des usagers de la route (conducteurs, cyclistes, piétons,...) ne sont pas les mêmes. En effet, si certains pays (comme la France, le Canada, ou le Brésil par exemple) suivent la Convention de Vienne sur la circulation routière (avec certaines variabilités), d'autres n'ont pas de codification juridique particulière. Dans ces derniers cas, des règles assez inhabituelles existent dont voici une courte liste (Source : codeclic.com) :

Singapour : il est illégal de s'approcher d'un piéton en train de traverser une rue à plus de 50 mètres,

Kansas, à Wichita : les conducteurs doivent sortir de leur voiture, tirer trois coups de feu en l'air avant de traverser l'intersection de Douglas et Broadway,

Pennsylvanie : sur une petite route de campagne, la nuit, le code oblige à arrêter les conducteurs à chaque kilomètre et demi pour tirer une fusée, et attendre 10 minutes pour avertir les animaux du passage du véhicule.

Alabama : si une femme est au volant, un homme doit courir ou marcher en avant de la voiture en agitant un drapeau rouge pour prévenir les autres automobilistes et les piétons.

Pour ce projet, le modèle représente des règles de circulation similaires à celles de la France. Cependant, étant donné la grande portabilité du programme, nous pouvons envisager la mise place d'une simulation sur toute zone dont les données géographiques sont disponibles dans les bases de données choisies (ici OSM, donc une couverture mondiale), la possibilité de choisir les règles de conduites en fonction du lieu choisi pourra être mis en place dans une version future du modèle. Ce sera notamment le cas des conduites anglo-saxonnes, qui nécessitent la modification des sens de circulation des différentes voies.

En ce qui concerne la gestion des routes à voies multiples (voies rapides, autoroutes et grandes avenues), leur prise en compte dans la simulation du trafic impose un travail important concernant les dépassements.

Le premier travail à faire sera de définir si la voie de gauche est libre. Ce test impliquera également la possibilité, lorsqu'il y a plus de deux voies disponibles à la circulation, de représenter la courtoisie entre utilisateurs qui veut qu'un automobiliste, s'il le peut, laisse à un autre la possibilité de déboîter. Cependant, ce type de comportement étant très intimement lié au style de conduite des usagers de la route, il est très complexe à modéliser. En effet, tous les

usagers ne vont pas se déporter même s'il en ont l'occasion. Il faudra donc, lors de mise en place de ces comportements, ne les attribuer qu'à certains conducteurs.

De même, sur les routes à voies multiples, il est d'usage de se rabattre sur la voie de droite lorsque celle-ci est disponible. Cette action implique cependant de différencier les voies de roulement et les échangeurs. En effet, un véhicule ne doit pas se rabattre sur une sortie ou sur une voie d'insertion.

L'influence des tranches horaires, du cycle jour/nuit et de la météo sur le trafic

Comme nous l'avons abordé au cours du chapitre 10, différents éléments de l'environnement affectent la modélisation du trafic urbain. Ainsi au cours de ce mémoire nous avons succinctement exprimé les effets :

- Du cycle jour/nuit avec des changements à la fois sur les vitesses et les règles de circulations (plus de véhicules dans la journée, des vitesses réduites en pleine nuit,...) mais aussi sur les mobiliers (les feux de signalisation sont plus généralement au vert pour les véhicules que pour les piétons qui sont moins présents la nuit, les réverbères s'éteignent le jour,...),
- Des conditions météorologiques puisque les conditions de circulation (en véhicules comme à pieds) ne sont pas les mêmes en cas de verglas ou de pluies torrentielles et les rendus d'affichage changent avec les conditions nuageuses (moins de luminosité générale, changement de la gestion des ombres,...).

Tous ces éléments, dont la liste n'est pas exhaustive, ont donc un impact sur la simulation du trafic urbain et l'une des perspectives de ce travail serait la gestion de ces événements afin d'assurer une modélisation réaliste quelque soit l'heure et la saison d'étude, et ce en fonction de la ville (moussons en Asie, chutes de neige en Amérique du Nord,...).

Nous pourrions également prendre en compte les directions de flux. En effet, les flux sont souvent dans les mêmes directions vu que les logements sont situés généralement dans les mêmes zones géographiques, idem pour les espaces de travail. Prenons le cas de la région parisienne où le flux commence aux départements voisins pour se diriger vers le centre de la capitale.

La circulation "douce"

Depuis quelques années on observe un intérêt croissant des villes, collectivités et entreprises à proposer pour tous les citoyens des outils de calculs d'itinéraires de plus en plus performants. Si la circulation en voiture est guidée depuis déjà un certain temps, les GPS piétons, quant à eux, en sont à leur balbutiement. En effet ces derniers, permettent d'identifier son itinéraire entre un point A et un point B, mais pas d'emprunter des chemins avec des restrictions. Ainsi il semble primordial de proposer des produits et services innovants pour favoriser la mobilité des personnes en situation de handicap, permanent ou temporaire, mais également aux personnes dont la mobilité est "réduite" (parents avec des poussettes, personnes âgées,...). Pour ce faire, il est alors important de prendre en compte un grand nombre de nouveaux critères sur les itinéraires comme par exemple emprunter des bateaux, éviter les rues avec de forts dévers, les trottoirs en mauvais état ou trop étroits,... Ce calcul d'itinéraire peut alors être intégré à la simulation réalisée pour permettre de faire circuler (en prenant alors compte de la vitesse de l'individu) simultanément des individus très divers mais représentatifs de la population générale.

15.2.6 Géoréférencement de l'utilisateur et interactions

Une autre perspective envisageable, directement liée à la fois aux technologies modernes sur le géoréférencement et la géomatique, est la prise en compte des coordonnées de l'utilisateur. En effet considérer les coordonnées GPS de la personne visualisant le trafic routier peut permettre

une meilleure immersion dans l'espace simulé, en particulier via la Réalité Virtuelle proposée par les casques RV. Associé à la modification simultanée de la donnée afin de créer de nouveaux scénarios (travaux sur une voie, accident virtuel,...), ce mode géoréfencé serait un atout majeur pour la prise de décision.

Un inconvénient de la simulation actuelle, quelle soit sérieuse ou orientée vers la ludotique, l'interaction se fait via le clavier et la souris. Malgré le fait que leur utilisation est acquise depuis quelques années par un grand nombre de personnes, ils deviennent encombrants et peu intuitifs pour la simulation en réalité virtuelle. Nous pourrions ajouter un côté vidéo ludique et apporter une facilité d'accès à la simulation en utilisant un volant pour la conduite en mode immersif. Un autre objet encore plus intéressant pourrait être mis en place pour nous affranchir totalement de l'appareil de contrôle : le LEAP motion (voir figure102).

FIGURE 102 – Utilisation du Leap Motion avec Google Earth, Source : shashgear.com

Il s'agit d'un capteur pouvant reconnaître les mouvements des mains lorsqu'elles sont placées au dessus. Google l'utilise déjà pour la navigation dans Google Earth. ArcGis l'a intégré aussi dans sa dernière version. Des travaux sont déjà réalisés sur cet outil par l'université Sophia Antipolis de Nice pour se déplacer dans des modèles 3D [50]. Il pourrait être intéressant de l'intégrer dans la réalité virtuelle avec des gestes simples tels que le pincement de doigt pour s'approcher ou se reculer de la scène 3D.

15.2.7 Aller plus loin dans la prise de décision

Faire une méthodologie aboutissant sur un logiciel de modélisation de trafic urbain réaliste en trois dimensions a divers intérêts. Mais il en est un qui propose un nouvel usage : la prédiction. En effet ce logiciel pourrait permettre de modéliser, de façon prédictive, les changements de flux circulatoires en cas de modifications de la voirie (temporaires : lors de travaux, ou permanentes : mise en place de nouveaux équipements : pistes cyclables, ronds points,...). De nombreuses villes ont déjà eu affaire à des bouchons engendrés par des travaux sur la voirie. Souvent, aucun modèle n'a été fait pour les prévoir et encore moins pour les atténuer. Les répercussions de travaux peuvent être majeurs si ces derniers ne prennent pas en compte les flux routiers. Lors de la modélisation de ces scénarios, des solutions pourraient être prises pour avoir moins d'impact sur la circulation. Pour aller encore plus loin, la construction de voies de bus pourrait être prise en compte pour modéliser les nouveaux déplacements en ville, et peut-être pour optimiser le flux des piétons. Il en est de même pour la mise en place de pistes cyclables. Ces ajouts transformeraient ce modèle en un outil d'aide à la décision puissant pour divers corps de métier (municipalités, entreprises dédiées aux travaux sur la voirie, compagnies de bus/tramway,..).

15.2.8 Utilisation des données sources OSM mises à jour en direct

L'intérêt d'utiliser les données Open Street Map a été décrit dans le chapitre 6. Mais l'un des avantages qui pourrait être encore plus mis en avant dans ce travail de recherche est l'apport de l'utilisation de données collaboratives en temps réel, c'est-à-dire dès qu'une nouvelle contribution a été publiée. Cela impliquerait de connecter la base de données OSM directement au moteurs de simulations. Voici les points sur lesquels nous devons travailler pour permettre les diverses simulations :

— Garder une version hors ligne pour les personnes n'ayant pas la volonté de créer des simulations sur les données à jour,
— Avoir une base de données dédiée à la conversion 3D. Nous pouvons imaginer que des tâches planifiées mettront à jour les conversions toutes les semaines ou tous les mois afin de ne pas avoir de conversions à faire en temps réel. Cela permettra d'optimiser l'affichage,
— Permettre la modification de la donnée sur ce serveur en mode local ou non. En mode local, la donnée ne sera pas mise à jour sur la base de données OSM, et dans le cas contraire, la modification sera prise en compte,
— Avoir une version mobile allégée car les mémoires tampons dédiées aux navigateurs mobile Android ou Apple sont très restreintes,
— Faire le lien avec la donnée SRTM pour télécharger les informations topographiques.

Cet apport permettrait de nous affranchir de la phase d'import de la donnée, ce qui rendrait le modèle encore plus accessible.

15.2.9 Amélioration de la technique de l'hologramme

Si proposer une visualisation plus grande sur pyramide permettrait déjà une meilleure utilisation de cet outil, d'autres améliorations pourraient y être intégrées. Ainsi, afin d'améliorer ce principe il serait envisageable de diffuser cet hologramme non pas sur une pyramide renversée, tronquée ou non, mais sur une demi-sphère. Pour ce faire, il faudrait alors déformer l'image de la scène 3D entière pour pouvoir la plaquer sur une demi sphère transparente que l'on placerait au dessus de l'écran. Il n'y aurait plus d'arètes et donc la lecture de la simulation serait continue. Cette technique n'est pas sans rappeler les images obtenues par la méthode de l'anamorphose (voir figure 103).

FIGURE 103 – Exemple d'anamorphose, Source : scienceweek.lu

En effet, l'anamorphose repose sur la visualisation, via un système optique (généralement un miroir courbe) d'une image déformée. Un exemple est visible a grande échelle à Nantes avec la construction du Feydball, réalisé par l'agence nantaise Barré-Lambot associée au paysagiste Guillaume Sevin. Cet œuvre illustrée dans la figure 104 est composé d'un terrain de football

circulaire avec en son centre un mur de miroir lui même circulaire. Son reflet est un terrain rectangulaire non déformé.

FIGURE 104 – Le Feydball, terrain dessiné selon le principe d'anamorphose, Source : Jean-Philippe Defawe

Ainsi l'une des perspectives de ce travail serait de générer différents choix de visualisations "holographiques" selon la méthode choisie par l'utilisateur :

— La pyramide renversée,
— L'anamorphose cylindrique,
— L'anamorphose hémisphérique.

Conclusion

Conclusion

Au cours du temps la cartographie a bien évolué. Elle a été utilisée principalement pour les explorations, puis elle a été stratégique pendant les guerres, pour devenir un outil utilisé au quotidien. De nos jours, elle nous apparaît accessible et les évolutions technologiques vont changer, encore, la manière dont nous l'appréhendons. Ce travail de thèse a pour but d'explorer la visualisation cartographique sous plusieurs angles afin de comprendre en profondeur un élément majeur de notre société : les flux urbains.

La circulation citadine est souvent liée aux problèmes de l'urbanisation croissante que l'on constate tous les jours. Cette dernière engendre de multiples contraintes sur les flux, qu'ils soient routiers ou autres. En effet, l'urbanisation, telle que nous la connaissons depuis quelques années, prend rarement en compte l'impact qu'elle a sur les différents réseaux.

Les réseaux d'électricité sont de plus en plus étendus, agrandissant les zones impactées par de possibles pannes,

Les réseaux d'eau potable sont sous dimensionnés face à la demande, des baisses de pressions se ressentent au bout du circuit,

Les réseaux d'assainissement sont aussi trop petits face à la forte imperméabilisation des sols et à l'augmentation de la population,

Le réseau routier devient trop petit et sa circulation trop dense, ce qui augmente le risque d'accident,

etc ... .

Cette liste non-exhaustive nous montre l'impact de l'urbanisation qui nous entoure. Afin de mieux le comprendre, et de le gérer, des modèles analogiques ou numériques doivent être réalisés. Le principe de maquette 3D, reliant ces deux types de modélisation nous semble être une solution pour visualiser les phénomènes entourant les flux urbains.

La technologie qui nous entoure aujourd'hui, comme les casques de réalité améliorée ou virtuelle, est un excellent moyen d'avoir une bonne immersion dans ces simulations afin de mieux appréhender les problèmes qui peuvent se présenter. De plus, à l'heure où les licences ouvertes permettent l'accès à diverses données de bonne qualité, peu de logiciels de simulation de trafic urbain les utilisent. Nous nous sommes tournés vers des solutions libres, tant pour le développement du modèle que pour les données sources du fait du contexte actuel prônant l'accès à la donnée Open Source. Nous avons pris cette direction pour promouvoir cet aspect ainsi que pour démontrer que la construction de modèles précis peut être réalisée grâce cette donnée. Cela nous a même montré que certaines données libres pouvaient en devancer d'autres sous licences payantes.

Une des données utilisée est celle d'Open Street Map (OSM). Sa mise à disposition mondiale nous a orienté sur le fait de pouvoir simuler le trafic urbain partout où nous le souhaitons. Ainsi, un point majeur de ce travail de thèse s'est donc mis en place : l'automatisation. La plupart

des modèles, qu'ils soient sur les flux urbains, l'écoulement de fluides, les inondations, etc... sont créés pour un seul et même emplacement à la demande d'un client. Nous avons trouvé intéressant de rendre la création de modèles accessible à tous, sans contrainte de localisation. Pour cela nous avons dû penser à l'automatisation des traitements. Cela a été réalisé pour la conversion de la donnée en 3D pour la visualisation ainsi que pour la mise en place de la simulation à l'intérieur du modèle auto-généré. Pour arriver à ce résultat des choix ont été faits. Par exemple, nous avons considéré que le trafic urbain utilisait les règles de conduites identiques à la France, il en est de même pour le marquage au sol. Des différences existent entre les pays, et leurs prises en compte seront sûrement présentes dans une future version de ce modèle.

Cependant, afin de ne pas contraindre son utilisation, le modèle créé peut être alimenté par des données différentes que celles d'OSM. C'est pour cela que nous avons mis en place un calcul automatique de la topologie du réseau routier. Cette dernière peut être existante dans des données sources mais nous la mettons en place dans tous les cas lors de l'import des fichiers sources. Elle nous permet de définir des trajets auto-générés pour les simulations. Ces trajets sont définis selon une position de départ et suivent des règles pour leur mise en place (nombre de tronçons aléatoires, règles sur les voies sans issue,..).

Le génération de scénarios de façon automatique est régie par trois moteur de simulations :

1. **Le moteur 3D** pour la gestion de la visualisation 3D,
2. **Le moteur de simulation continue** pour la gestion des mouvements des mobiles entre autres,
3. **Le moteur de simulation discrète** pour la gestion des évènements.

Le moteur 3D permet d'afficher le rendu 3D de façon fluide. Nous avons fixé le taux de rafraîchissement à 24 images par seconde, limite pour l'œil humain à percevoir le scintillement des images s'affichant les unes après les autres. Cette valeur fixe notre pas de temps de calcul pour la simulation continue, cela veut dire que toutes les $24^{\text{ème}}$ de secondes, une procédure est lancée pour le déplacement des véhicules ou alors pour gérer l'état des feux de signalisation (rouge ou vert). Cette simulation peut être accélérée ou ralentie grâce à une variable modifiant son pas de temps. Ce processus n'influence par l'affichage mais permet, dans le cas d'une accélération de la vitesse de simulation, d'avoir cette variable dans les calculs de déplacement par exemple. La simulation discrète est gérée par la simulation continue. Pour chaque pas de temps, elle est appelée pour connaître l'état des divers éléments de la modélisation. Ainsi, elle indiquera l'occupation des tronçons par un autre véhicule, ou alors la présence d'un piéton sur un passage pour piétons.

Toutes les règles pour la circulation sont gérées par une Intelligence Artificielle appelée Système Expert (SE). Il permet d'automatiser les prises de décisions selon des règles établies au préalable. Son fonctionnement est dirigé par le moteur d'inférence qui choisit dans chaque cas les règles à appliquer pour résoudre la problématique. Par exemple, lors d'un dépassement, il choisira les règles concernant les distances minimales de sécurité et celles concernant la vitesse. Il sera établi de dépasser ou non, avec une accélération ou non, voire un dépassement de vitesse ou non. Sur ce dernier point, le Système Expert, tel qu'il est créé aujourd'hui, ne prend pas en compte les comportements dangereux mais une version future pourrait le faire afin de modéliser les réactions des autres conducteurs virtuels.

La visualisation de trafic routier est souvent liée à une vision aérienne afin d'avoir en globalité l'information. Nous avons décidé de garder cette vue 2D pour la visualisation de la donnée source mais nous nous sommes penchés sur la 3D pour l'affichage de la simulation. Le modèle 3D peut ainsi être appréhendé via un mode hélicoptère, permettant de se déplacer au dessus de la ville,

ou alors par un mode terrestre, proposant une vue libre ou en voiture, permettant de suivre un véhicule pendant son trajet. Ces différentes vues placent l'utilisateur au niveau de détail qui lui convient. Certains préférerons avoir une vision dans sa globalité tandis que d'autres visualiseront mieux les scénarios avec une vue à une intersection.

Par ailleurs, d'autres vues 3D ont été développées afin d'amorcer une réflexion sur le devenir de la cartographie numérique. Doit-elle rester sur nos écrans d'ordinateurs ou nos smartphones ? Nous avons donc mis en place une vue anaglyphique (de type cyan et rouge) permettant de visualiser en relief la scène 3D. Ce principe basé sur la stéréoscopie est limitée par la perception des couleurs, c'est pour cela que nous avons appliqué la vue stéréoscopique binoculaire. Ce principe, fortement développé lors des deux grandes guerres, permet de s'affranchir des lunettes cyan et rouge pour afficher une vue 3D colorisée via un casque binoculaire à lentilles. Cette technique est utilisée, de nos jours, dans les casques de réalité virtuelle. Ces casques permettent à l'utilisateur d'avoir un sentiment d'immersion dans un monde tridimensionnel. Une autre piste est celle de l'holographie qui permet de ne pas restreindre l'utilisation à une seule personne. L'holographie est appliquée via quatre écrans semi-transparents disposés en pyramide au-dessus d'un autre écran où le modèle 3D est représenté quatre fois. La réflexion des quatre représentations sur les faces de la pyramide apportent une dimension ludique au modèle. De plus, contrairement à ce que l'on trouve sur internet, nous avons décidé de projeter une vue différente sur chacune des faces, ce qui permet en tournant autour de cette représentation de visualiser la scène 3D dans sa globalité.

Afin de valider ce modèle, des comparaisons pourraient être faites avec les flux routiers des villes simulées. Des données existent via les caméras de surveillance. Cela permettrait d'appliquer ce modèle sur divers scénarios tels que la circulation des pompiers ou des ambulances lors d'un accident dans une zone à forte circulation. Cet outil pourrait aussi aider à la planification de travaux pour limiter leurs impacts sur la circulation, ou alors pour désengorger une avenue en créant des trajets alternatifs.

Glossaire et Acronymes

Glossaire

Anaglyphe Image imprimée pour être vue en relief, à l'aide de deux filtres de couleurs différentes (lunettes 3D) disposés devant chacun des yeux de l'observateur. Ce principe est fondé sur la notion de stéréoscopie qui permet à notre cerveau d'utiliser le décalage entre nos deux yeux pour percevoir le relief.. 106

Classe Élément du modèle HBDS : Ensemble d'objets homogène. 21

Graphe Ensemble de sommets (ou points) et d'arcs (ou lignes orientées) ou d'arêtes (ou lignes non orientées) liant certains couples de points.. 19

Holographie Procédé d'enregistrement de la phase et de l'amplitude de l'onde diffractée par un objet permettant de restituer ultérieurement une image en trois dimensions de l'objet.. 106

Hyperclasse Élément du modèle HBDS : Ensemble homogène. 21

Hypergraphe Les hypergraphes généralisent la notion de graphe non orienté dans le sens où les arêtes ne relient plus un ou deux sommets, mais un nombre quelconque de sommets [51].. 21

LIP6 Unité Mixte de Recherche de l'Université Pierre et Marie Curie et du Centre National de la Recherche. Elle est un laboratoire de recherche en informatique se consacrant à la modélisation et la résolution de problèmes fondamentaux motivés par les applications, ainsi qu'à la mise en oeuvre et la validation des solutions au travers de partenariats académiques et industriels.. 10

Multithreading Le multithreading sert à s'affranchir de la simple file d'exécution. Cela permet ainsi la création d'une nouvelle file d'exécution concourante de la première.. 65

Nycthéméral Terme de physiologie correspondant à un cycle jour nuit de 24h.. 71

Objet Élément du modèle HBDS : Élément. 21

Peer to peer Modèle de réseau informatique proche du modèle client-serveur mais où chaque client est aussi un serveur.. 25

Portulan est une sorte de carte de navigation, utilisée du xiiie siècle au xviiie siècle, servant essentiellement à repérer les ports et connaître les dangers qui peuvent les entourer : courants, hauts-fonds. 7

Signal Message envoyé par un widget lorsqu'un évènement se produit.. 56

Skybox Principe utilisé dans les scènes 3D afin de simuler un ciel et un horizon en plaquant une texture sur un cube englobant l'environnement affiché.. 98

Skydome Principe utilisé dans les scènes 3D afin de simuler un ciel et un horizon en plaquant une texture sur un dôme situé au dessus de l'environnement affiché.. 98

Slot Fonction qui est appelée lorsqu'un évènement s'est produit. On dit que le signal appelle le slot. 56

Stéréoscopie Ensemble des techniques mises en œuvre pour reproduire une perception du relief à partir de deux images planes.. 106

Streaming Principe utilisé principalement pour l'envoi de contenu en "direct" .. 114

Talweg Un talweg (ou thalweg) correspond à la ligne qui rejoint les points les plus bas soit d'une vallée1, soit du lit d'un cours d'eau.. 40

Widget Éléments de base permettant de créer une interface Qt : boutons, fenêtre, liste déroulante, ... 56

Acronymes

ADL Algorithm Description Language. 29
AGPL Affero General Public License. 26
API Applications Programming Interface. 30
ASSIMP Open Asset Import Library. 27

BDD Base De Données. 15
BRGM Bureau de Recherches Géologiques et Minières. 13
BSD Berkeley Software Distribution. 26

CAO Conception Assistée par Ordinateur. 14
CEA Commissariat à l'Energie atomique. 133
CeCILL CEA CNRS INRIA Logiciel. 26
CIME Cartographie Intelligente en MiliEu montagnard. 79
CNRS Centre National de la Recherche Scientifique. 133

DAVD Divers Augmented Vision Display. 109
DEM Digital Elevation Model. 36
DXF Drawing eXchange Format. 17

EGPL Exception General Public Licence. 27

GCC GNU Compilers Collection. 30
GFDL GNU Free Documentation License. 26
GML Geography Markup Language. 15
GNU GNU's not Unix. 26
GPL General Public License. 26
GPU Graphical Unit Process. 17
GUI Graphical User Interface. 30

HBDS Hypergraph Based Data Structure. 21

IA Intelligence Artificielle. 87
IASIG Informatique Appliquée aux Systèmes d'Information Géographique. 29
IDE Integrated Development Environment. 30
IDW Inverse Distance Weighted. 40
IGN Institut national de l'information géographique et forestière. 31
INRIA Institut National de Recherche en Informatique et en Automatique. 133

KML Keyhole Markup Language. 17

LGPL Lesser General Public License. 26
LICIT Laboratoire d'Ingénierie Circulation Transport. 10

LSC Local Sequence Counter. 65

MNT Modèle Numérique de Terrain. 15
MPL Mozilla Public License. 26

NC Non-Commercial. 25
ND No Derivatives. 25

ODBL Open DataBase License. 27
OSM OpenStreetMap. 32

POI Point Of Interest. 32

SA Share-Alike. 25
Sachem Système d'Aide à la Conduite des Hauts fourneaux En Marche. 79
SDK Software Development Kit. 15
SDL Simple Directmedia Layer. 27
Slerp SphErical Linear Interpolation. 94
SQL Structured Query Language. 15
SQS SeQuencing Set. 64
SRTM Shuttle Radar Topography Mission. 36

USGS United States Geological Survey. 14
UTM Universal Transverse Mercator. 8

VR Virtual Reality. 108

WMS Web Map Service. 15

Tables

Table des figures

Liste des tableaux

Bibliographie

Bibliographie

[1] M. Lacroix, *Méthodes pour la reconstruction, l'analyse et l'exploitation de réseaux tridimensionnels en milieu urbain.* PhD thesis, Paris 6, 2017.

[2] F. Bouillé, "Cartographie et système d'information géographique (sig)," *Géologues*, vol. 193, pp. 4–11, 2017.

[3] J. P. Snyder, *Flattening the earth : two thousand years of map projections.* University of Chicago Press, 1997.

[4] J. P. Goode, "The homolosine projection : A new device for portraying the earth's surface entire," *Annals of the Association of American Geographers*, vol. 15, no. 3, pp. 119–125, 1925.

[5] F. Bonnefoi, J. Soula, L. Leclercq, and C. Bécarie, "Simulation multi-échelle du trafic routier."

[6] W. Chaker and B. Moulin, "Forecasting travel supply and demand by modeling multiscale urban environments," *Transportation Research Record : Journal of the Transportation Research Board*, no. 2064, pp. 65–72, 2008.

[7] L. Euler, "Leonhard euler and the königsberg bridges," *Scientific American*, vol. 189, no. 1, pp. 66–70, 1953.

[8] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, vol. 8, pp. 128–140, 01 1736.

[9] F. Bouillé, "The hypergraph-based data structure : A new approach to data base modelling and application," in *GI—7. Jahrestagung*, pp. 37–55, Springer, 1977.

[10] S. B. Aruoba and J. Fernández-Villaverde, "A comparison of programming languages in economics," tech. rep., National Bureau of Economic Research, 2014.

[11] L. Kelty, P. Beckett, and L. Zalcman, "Desktop simulation," in *Advancing simulation technology and training, proceedings of SimTecT99, The simulation technology and training conference*, 1999.

[12] www.ovum.com, "Location platform index : Mapping and navigation, 1h16," tech. rep., Ovum, 2016.

[13] P. Neis, D. Zielstra, and A. Zipf, "The street network evolution of crowdsourced maps : Openstreetmap in germany 2007–2011," *Future Internet*, vol. 4, no. 1, pp. 1–21, 2011.

[14] M. Haklay and P. Weber, "Openstreetmap : User-generated street maps," *IEEE Pervasive Computing*, vol. 7, no. 4, pp. 12–18, 2008.

[15] H. I. Reuter, A. Nelson, and A. Jarvis, "An evaluation of void-filling interpolation methods for srtm data," *International Journal of Geographical Information Science*, vol. 2Prévision et analyse du trafic routier par des méthodes statistiques1, no. 9, pp. 983–1008, 2007.

[16] D. Shepard, "A two-dimensional interpolation function for irregularly-spaced data," in *Proceedings of the 1968 23rd ACM national conference*, pp. 517–524, ACM, 1968.

[17] B. Delaunay, "Sur la sphere vide," *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[18] M. Laporte, "Elaboration rapide de cartes gravimetriques deduites de l'anomalie de bouguer a l'aide d'une calculatrice electronique," *Geophysical Prospecting*, vol. 10, no. 3, pp. 238–257, 1962.

[19] J. Vignes, *Algorithmes numériques, analyse et mise en oeuvre : Équations et systèmes non linéaires*, vol. 2. Editions Technip, 1980.

[20] J. Richard, "Présentation : Utilisation de données gratuites open source pour la simulation de trafic dans un environnement 4d," 2015.

[21] J. Richard, "Modelisation of car traffic inside 3d cities," *Intercarto*, 2015.

[22] J. Richard, "Présentation : Simulation 3d de trafic routier à partir de la donnée osm," 2016.

[23] K. Nygaard and O.-J. Dahl, *The development of the SIMULA languages*. ACM, 1978.

[24] O.-J. Dahl, B. Myhrhaug, and K. Nygaard, "Simula 67 common base language," 1967.

[25] J. Vaucher, *La programmation de simulation*. Université de Montréal, Département d'informatique et de recherche opérationnelle, 1972.

[26] A. Wang and O.-J. Dahl, "Coroutine sequencing in a block structured environment," *BIT Numerical Mathematics*, vol. 11, no. 4, pp. 425–449, 1971.

[27] E. C. Russell, *Simulating with Processes and Resources in SIMSCRIPT II. 5 : User's Manual*. CACI, Incorporated-Federal, 1979.

[28] F. Bouillé, "Simulation discrète à processus persistants - application aux sig-4d, master-2 iasig," 2012.

[29] F. Bouillé, "Présentation : La simulation discrète à tad persistants."

[30] G. Brams, *Réseaux de Petri : théorie et pratique*, vol. 2. Masson, 1983.

[31] G. Allain, *Prévision et analyse du trafic routier par des méthodes statistiques*. PhD thesis, Université de Toulouse, France, 2008.

[32] J. Mint Moustapha, *Mathematical modelling and simulation of the road traffic : statistical analysis of merging models and probabilistic simulation of a kinetic model*. Theses, Université Paris-Est, Nov. 2014.

[33] C. Buisson and J. Lesort, *Comprendre le trafic routier. Méthodes et calculs*. Coll. Références, CERTU, Jan. 2010.

[34] C. Mering, D. Blamont, J. Ganascia, and F. Monjanel, "Cime : une application des systèmes experts à la télédétection," 1988.

[35] Y. Kubera, P. Mathieu, and S. Picault, "La complexité dans les Simulations Multi-Agents," in *Actes des 15e Journées Francophones sur les Systèmes Multi-Agents (JFSMA'2007)* (V. Camps and P. Mathieu, eds.), (undef, France), pp. 139–148, Cépaduès, 2007.

[36] W. R. Hamilton, *Elements of quaternions*. Longmans, Green, & Company, 1866.

[37] W. Alan and W. Mark, "Advanced animation and rendering techniques : Theory and practice," 1992.

[38] F. Béatrice and C. Cyril, "Projet quaternions," tech. rep., Université de Technologie de Belfort-Montbéliard, 2003.

[39] S. W. Shepperd, "Quaternion from rotation matrix.[four-parameter representation of coordinate transformation matrix]," 1978.

[40] K. Shoemake, "Animating rotation with quaternion curves," *ACM SIGGRAPH computer graphics*, vol. 19, no. 3, pp. 245–254, 1985.

[41] G. Riva, "Virtual reality in psychotherapy," *Cyberpsychology & behavior*, vol. 8, no. 3, pp. 220–230, 2005.

[42] L. A. Cushman, K. Stein, and C. J. Duffy, "Detecting navigational deficits in cognitive aging and alzheimer disease using virtual reality," *Neurology*, vol. 71, no. 12, pp. 888–895, 2008.

[43] C. A. Linte, J. Moore, A. D. Wiles, C. Wedlake, and T. M. Peters, "Virtual reality-enhanced ultrasound guidance : A novel technique for intracardiac interventions," *Computer Aided Surgery*, vol. 13, no. 2, pp. 82–94, 2008.

[44] W. Rollmann, "Zwei neue stereoskopische methoden," *Annalen der Physik*, vol. 166, no. 9, pp. 186–187, 1853.

[45] C. Wheatstone, "On binocular vision : and on the stereoscope, an instrument for illustrating its phenomena," *Report to the Brittish Association for the Advancement of Science Pt*, pp. 16–17, 1838.

[46] D. Brewster, "Lxiv. on the law of visible position in single and binocular vision, and the representation of solid figures by the union dissimilar plane pictures on the retina," *Philosophical Magazine Series 3*, vol. 24, no. 161, pp. 439–455, 1844.

[47] O. W. Holmes, "The stereoscope and the stereograph [1859]," *Photography : Essays and Images, London : Seeker & Warburg*, p. 54, 1980.

[48] M. Zyda, "From visual simulation to virtual reality to games," *Computer*, vol. 38, no. 9, pp. 25–32, 2005.

[49] F. Larceneux, "Ii. la disparition de l'agent immobilier : utopies et réalités de l'intermédiation," *Repères*, pp. 17–30, 2014.

[50] E. Bailly, B. Maureille, G. Perrot, and K. Rocher, "Leap motion et visualisation 3d vrml," 2013.

[51] C. Berge, "La theorie des graphes," *Paris, France*, 1958.

Publications et présentations

Julien Richard - **Simulation 3D de trafic routier à partir de la donnée OSM**, FIG 2016 - Concours de géo-visualisation lors du Festival International de Géographie, 2016.

Julien Richard - **Utilisation de données gratuites Open source pour la simulation de trafic dans un environnement 4D**, FIG 2015 - Concours de géo-visualisation lors du Festival International de Géographie - Prix du public, 2015.

Julien Richard - **Logiciel open source pour la création automatique de villes en 3D à partir de données SIG standard**, Revue Mappemonde, 2015.

Julien Richard, Ioulia Tchiguirinskaia et Daniel Schertzer, **3D visualisation of hydrological model outputs for a better understanding of multi-scale phenomena**, AGU 2014 - American Geosciences Union, 2014.

Julien Richard, Agathe Giangola-Murzyn, Ioulia Tchiguirinskaia et Daniel Schertzer, **MH3DV : visualisation3D de résultats hydrologiques**, FIG 2014 - Concours de géo-visualisation lors du Festival International de Géographie - 2éme place, 2014

Julien Richard, Agathe Giangola-Murzyn, Auguste Gires, Ioulia Tchiguirinskaia et Daniel Schertzer, **Open source 3D visualisation and interaction dedicated to hydrological models**, EGU 2014 - European Geosciences Union, 2014.

Julien Richard, **Conversion d'un modèle hydrologique distribué en un outil d'aide à la décision**. JDHU 2014 - Journées Doctorales en Hydrologie Urbaine, 2014.

Julien Richard, Agathe Giangola-Murzyn, Daniel Schertzer et Ioulia Tchiguirinskaia, **MH-AssimTool : An assimilation tool dedicated to a fully distributed model**, ICFR 2013 - International Conference on Flood Resilience, 2013.

Julien Richard, Agathe Giangola-Murzyn, Daniel Schertzer, Ioulia Tchiguirinskaia, **Open source assimilation tool for a distributed hydrological model**, EGU 2013 - European Geosciences Union, 2013.

Julien Richard, Agathe Giangola-Murzyn, Auguste Gires, Ioulia Tchiguirinskaia et Daniel Schertzer, **Advanced GIS data assimilation interface for evaluation of flood resilient systems**, EGU 2012 - European Geosciences Union, 2012

Julien Richard, Ioulia Tchiguirinskaia et Daniel Schertzer, **Advanced GIS data assimilation interface for ditributed hydrological models**, HIC 2012 - Hydrological Informatics Conference, 2012

Aspect utilisateur

Annexe A

Interface

L'interface graphique générée par Qt est primordiale pour utiliser le logiciel. Elle est découpée en une fenêtre principale pour l'affichage et d'autres, secondaires, pour l'import de données.

A.1 Fenêtre Principale

Elle est composée d'une partie affichage de données et d'une partie interaction directe avec l'utilisateur. La partie affichage est en position centrale avec deux onglets nommés "vue 2D" et "vue 3D".

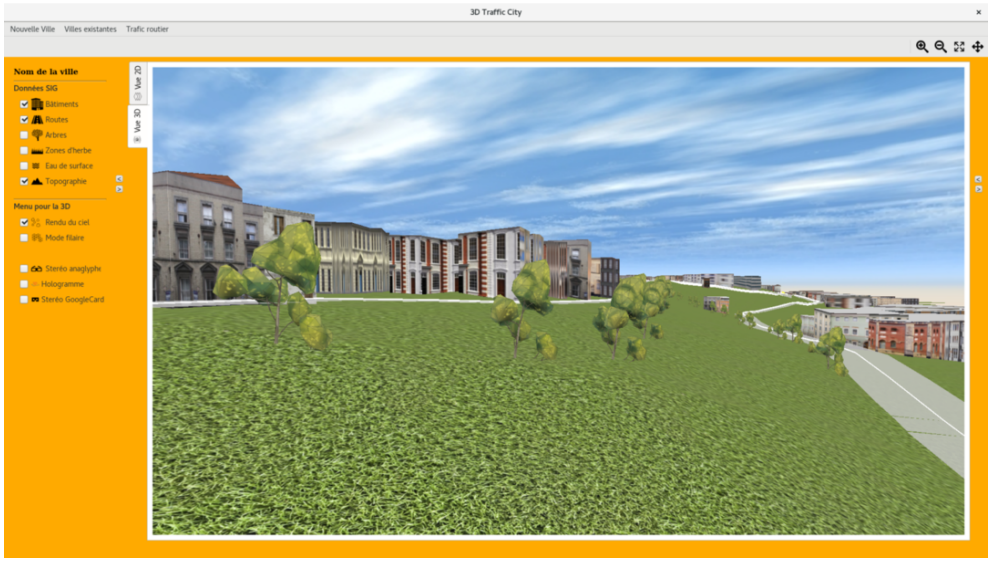


FIGURE 105 – Capture de la fenêtre principale

Comme leurs noms l'indiquent, cela permet de changer de la vue à plat à la vue convertie en 3D. La partie 2D est un affichage classique de SIG, celui-là même intégré à QGIS. Au démarrage ces deux onglets sont vides car aucune ville n'a été chargée. Pour en charger une, il faut soit importer de nouvelles données (voir partie A.3) soit charger une ville déjà traitée. Pour lancer des simulations, nous avons choisi aléatoirement quelques villes dont les données ont déjà été créées par l'outil de conversion. Cela permet de ne pas relancer les calculs de pré-processing.

Pour cela, nous stockons la donnée convertie, c'est à dire les triangles 3D trouvés lors de la conversion expliquée dans le chapitre 7 dans un fichier texte, propre à chaque donnée. Ainsi lors d'un deuxième affichage de cette ville, aucune conversion n'est à réaliser, seule la lecture de fichier permettra de remplir les tableaux de triangles nécessaires à l'affichage géré par OpenGL. Ces fichiers sont automatiquement créés lors d'une conversion, donc si l'utilisateur souhaite travailler sur cette même ville une autre fois, il aura juste à indiquer le fichier global pour importer à nouveau la vue 3D.

En plus de cette fenêtre principale, deux panneaux supplémentaires permettent des interactions entre l'utilisateur et le logiciel créé à partir des méthodologies expliquées au cours de ce manuscrit.

Panneau pour interagir avec la simulation

Le panneau de simulation permet de modifier les attributs de la modélisation du trafic avec le choix :

— de mettre en place un éclairage ou non,
— du mois de l'année ou prendra place la simulation,
— l'heure de début de la simulation,
— la vitesse de simulation.

Ce panneau est visible seulement une fois une simulation lancée par la fenêtre de simulation.La vitesse de simulation peut être modifiée au cours de la simulation pour accélérer ou diminuer les différentes interactions du trafic urbain modélisé.

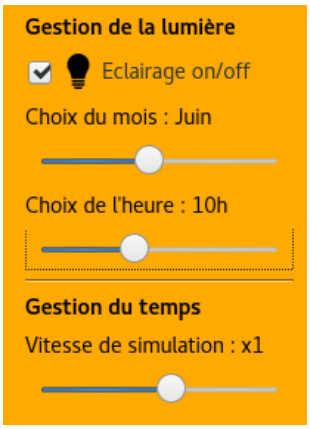


FIGURE 106 – Capture du panneau d'interaction avec la simulation

Panneau pour interagir avec la visualisation

Le second panneau est celui qui traite de la visualisation de la scène (figure 107). Il est séparé en deux parties, la première liée à l'affichage ou non des couches SIG. Il permet à l'utilisateur de définir quels sont les éléments qu'il souhaite afficher, que ce soit pour les données SIG :

— les bâtiments,
— les routes,
— la végétation (arbres et/ou surfaces herbacées),
— les zones d'eau (lacs, rivières,...),
— la topographie.

Ces cases à cocher interagissent aussi avec la vue 3D, c'est-à-dire que lorsque les routes sont affichées en 2D, la vue 3D se met aussi à jour.

La deuxième partie concerne uniquement la vue 3D, et d'ailleurs ne s'affiche que lorsque cette dernière est à l'écran. Cette partie permet l'accès à une visualisation filaire et avec ou sans le rendu du ciel donné par le skydôme dont le concept a été abordé dans le chapitre 13. De plus c'est grâce à ce panneau que le mode de rendu peut être choisi parmi la vue anaglyphique, la vision pour casque de réalité virtuelle ou la vue holographique.

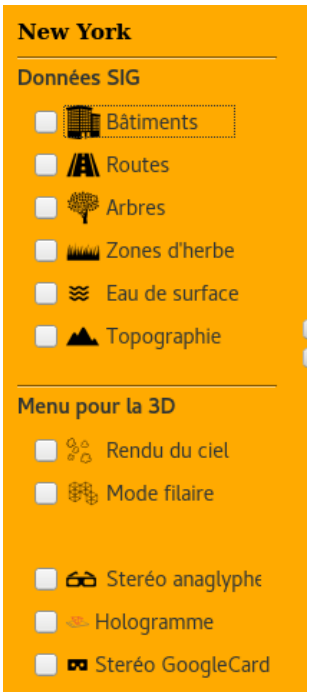


FIGURE 107 – Capture du panneau d'interaction avec la visualisation

A.2 Fenêtre pour la simulation

La fenêtre dédiée au lancement d'une simulation s'affiche via le menu "Trafic routier". Elle permet de paramétrer la simulation en choisissant un nombre voulu de véhicules ou de prendre en compte des collisions qui peuvent exister entre ces objets par exemple.

Ces paramètres de simulation sont remis à zéro à chaque lancement de cette fenêtre. La simulation en cours sera donc arrêtée et ré-initialisée. Parmi ces paramètres, nous avons la vitesse de simulation qui peut être définie pour le départ de celle-ci.

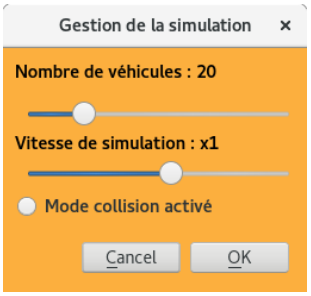


FIGURE 108 – Fenêtre pour le lancement d'une simulation

A.3 Fenêtres dédiées aux imports

La fenêtre dédiée aux imports permet, quant à elle, de faire le lien entre le logiciel et les données dont dispose l'utilisateur. C'est ici qu'il pourra y renseigner les informations nécessaires à la simulation dont la liste est équivalente à celle présentée dans les choix de visualisation dans le panneau qui s'y rapporte, c'est-à-dire :

— les bâtiments,
— les routes,
— la végétation (arbres et/ou surfaces herbacées),
— les zones d'eau (lacs, rivières,...),
— la topographie.

Pour chacune des données, les fichiers sont vérifiés pour la conversion future en 3D. En effet, chaque donnée SIG est attendue avec un certain type d'objet géométrique : les bâtiments, la végétation et les surfaces d'eau sont attendus avec des éléments de type polygone alors que les routes doivent être en ligne. La topographie doit être en format raster.

FIGURE 109 – Capture de la fenêtre d'import des fichiers sources

Annexe B

Notice d'utilisation

Proposer un logiciel OpenSource implique alors de s'intéresser à la façon de diffuser les informations nécessaires à son utilisation (via une interface intuitive, cf. annexe A) et également à son adaptabilité et sa lisibilité (avec la création d'une notice). Cette notice doit à la fois répertorier les différentes classes générées, mais également leurs relations entre elles, comme par exemple l'hérédité. De plus cette notice doit directement s'intégrer au code développé pour pouvoir être modifiée rapidement (voire automatiquement) selon les changements réalisés lors du développement. Au cours de cette étude nous avons choisi d'utiliser la documentation Doxygen. Cette documentation en version html se trouve en ligne à l'url suivante :

http://juliorichard.free.fr/TrafficCity3D/index.html

Elle est également consultable en scannant ce QR code :

B.1 Doxygen

Le logiciel Doxygen est un outil développé principalement par M. Dimitri van Heesch et proposé sous licence libre. Il est capable de générer une documentation logicielle en s'intégrant directement au code source du programme développé en prenant en compte un certain nombre d'éléments comme par exemple certains commentaires ou encore la grammaire générale du langage de programmation utilisé. De plus il est possible d'utiliser un certain nombre de tags supplémentaires pour renseigner diverses informations complémentaires comme par exemple :

— **struct** pour documenter une structure en C,
— **fn** pour documenter une fonction,
— **param** pour référencer les paramètres d'une fonction,
— **return** pour mettre en avant l'argument en sortie d'une fonction,

— **warning** pour créer une alerte,...

Cette documentation, via cet outil, peut être faite à partir d'un grand nombre de langages très divers et en particulier le C++. De plus elle peut être proposée sous divers formats, permettant alors une meilleure diffusion de l'information, ce qui est un avantage lors du développement d'un logiciel OpenSource comme le nôtre. On peut noter par exemple une diffusion HTML, LaTeX ou encore PDF pour ne citer que les plus utilisés.

B.2 Quelques exemples

Afin de simplifier la lecture de ce rapport, il ne sera présenté, dans les paragraphes suivants, qu'un certain nombre de cas d'intérêt notés dans cette notice : la description d'une classe et un graphe d'appel. Cependant l'intégralité de la documentation, représentant plusieurs centaines de pages, est accessible dans le tome 2 de ce présent manuscrit (tome intitulé "Manuel Utilisateur").

FIGURE 110 – Page d'accueil de la notice HTML

Exemples de descriptions de classes

Pour chacune des classes et fonctions réalisées, il a été nécessaire d'y intégrer un ensemble de descriptions directement intégrables à la notice Doxygen. Ci-après sont présentées quelques unes de ces descriptions (Figure 111).

Exemples de graphes d'appels

De plus, il existe un certains nombre de modules qui font appel à d'autres classes pour pouvoir fonctionner. Ce sont ces relations qui sont mises en avant par la réalisation de graphes d'appels. L'outil de documentation utilisé permet de générer automatiquement ces graphes d'appels à partir de ces dépendances et certains de ces schémas sont présentés dans la figure 112.

Les graphes ont plusieurs fonctionnalités dans cette documentation :

— Créer une représentation des relations directes et indirectes entre les classes,
— Montre les relations d'interdépendance des classes et structures (ex : un membre de classe A est du type classe B),
— Créer les diagrammes représentant la hiérarchie des classes,
— Créer les graphes de dépendances des fonctions,
— Créer les graphes des fonctions dont dépendent directement et indirectement les fonctions documentées,
— Créer les graphes d'appel direct et indirect des fonctions
— Créer les graphes des fonctions appelées directement et indirectement par les fonctions documentées.

8.1 Référence de la classe caculPositionSoleil

`#include <calculpositionsoleil.h>`

Fonctions membres publiques

— caculPositionSoleil ()

Fonctions membres publiques statiques

— static void calculJourJulien (long double jour, long double mois, long double annee, long double heure, long double minute, long double seconde)

(a)

8.40.2.23 triangulationRoad()

```
tools::triangulationRoad (
            QVector< segmentOfRoad > listOfSegments,
            float pathWayDepth )  [static]
```

Creer les triangles 3D des tronçons de route. 2 triangles sont créés pour chacun des triangles. Ils sont calculé selon la largeur de la route et l'angle avec les tronçons précédent et suivant. Si le tronçon est en bout de réseau, l'angle suivant est déclaré à 90°.

Paramètres

listOfSegments	Liste des tronçons en mode ligne.
pathWayDepth	Largeur par défaut de la route. Elle est utilisée lorsque cette information n'est pas présente dans la donnée source.

Renvoie

Liste de triangles composant les routes 3D.

(b)

Référence de la structure openGlDisplay::quaternion

Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacement d'objet, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en termes de calculs. \ param x Composante x \ param y Composante y \ param z Composante z \ param w Composante w. Plus de détails...

`#include <opengldisplay.h>`

Attributs publics

double x
double y
double z
double w

Description détaillée

Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacement d'objet, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en termes de calculs. \ param x Composante x \ param y Composante y \ param z Composante z \ param w Composante w.

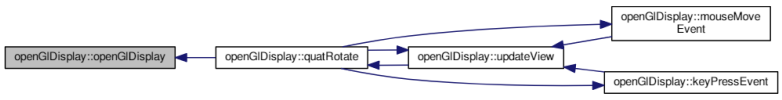
(c)

FIGURE 111 – Exemples de documentation sur les descriptions de classes réalisées au cours de cette étude

8.31.2.1 openGlDisplay() [1/2]

```
openGlDisplay::openGlDisplay (
            int view,
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe des appelants de cette fonction :



9.41.2.24 triangulationRoad()

```
tools::triangulationRoad (
            QVector< segmentOfRoad > listOfSegments,
            float pathWayDepth )  [static]
```

Creer les triangles 3D des tronçons de route. 2 triangles sont créés pour chacun des triangles. Ils sont calculés selon la largeur de la route et l'angle avec les tronçons précédent(s) et suivant(s). Si le tronçon est en bout de réseau, l'angle suivant est déclaré à 90°.

Paramètres

listOfSegments	Liste des tronçons en mode ligne.
pathWayDepth	Largeur par défaut de la route. Elle est utilisée lorsque cette information n'est pas présente dans la donnée source.

Renvoie

Liste de triangles composant les routes 3D.

9.37 Référence de la structure openGlDisplay : :quaternion

Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacements d'objets, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en terme de calculs.

```
#include <opengldisplay.h>
```

Attributs publics

- double x
- double y
- double z
- double w

9.37.1 Description détaillée

Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacements d'objets, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en terme de calculs.

Paramètres

x	Composante x
y	Composante y
z	Composante z
w	Composante w

FIGURE 112 – Exemples de documentation sur les graphes d'appels de classes programmées au cours de ces travaux

Annexes Algorithmes

Annexe C

Algorithme G3D

L'algorithme G3D a pour but de scanner la grille topographique à l'emplacement du point étudié. Il recherche 16 points minimum aux alentours et les envoi à la routine INTERXY qui créé une interpolation.

G3D(↓MAT, ↓DX, ↓DY, ↓NCOL, ↓NROW, ↓X0, ↓Y0, ↓X, ↓Y, ↑Z) :[[©*Je recherche la cellule contenant mon point dans la grille topographique*© i ← (Y – Y0) ÷ DY ; j ← (X – X0) ÷ DX ;

{$_{h:0}^{3}$ Imin ← i – h ; Imin < 1 ? Imin ← 1 |¿ ; Imax ← i + h ; Imax > NCOL ?

Imax ← NCOL |¿ ; Jmin ← j – h ; Jmin < 1 ? Jmin ← 1 | ¿ ; Jmax ← j + h ;

Jmax > NROW ? Jmax ← NROW | ¿ ; }$_{h}$

©*Nous avons maintenant la limite de petite grille autour de notre point de départ.*©

NP ← 0 ; {$_{m:Imin}^{Imax}$ {$_{n:Jmin}^{Jmax}$ ©*Je stocke les coordonnées xyz dans une table*©

NP ← NP + 1 ; TX_{NP} ← Y0 + m*DY ; TY_{NP} ← X0 + n*DX ; TZ_{NP} ← $MAT_{n,m}$; }$_{n}$ }$_{m}$

INTERXY(↓TX, ↓TY, ↓TZ, ↓NP, ↓X, ↓Y, ↑Z) ;]]

FIGURE 113 – Algorithme de Bouillé permettant de rechercher les points environnants la position à interpoler

Annexe D

Algorithme INTERXY

.

L'algorithme INTERXY est lancé une fois que la routine G3D a trouvé au minimum 16 points. Ces derniers sont interpolés selon le principe du paraboloïde hyperbolique. Il permet de gader une continuité dans les reliefs tels que les crêtes ou les talwegs.

```
INTERXY(↓TX, ↓TY, ↓TZ, ↓N, ↓XC, ↓YC, ↑ZC) :[[ ZC ← 1^10 ; ©s'il n'y a pas 16 points autour du point
étudié, on detectera cette valeur© N<16 ? !| ¿ {i:1 42} Bi ← 0}i ; V2=(XC²+Y²)^-20 ;
{i:1 N} XR ← Txi – XC ; YR ← TYi – YC ; ZR ← TZi ; D2 ← XR² + YR² ;
D2 < V2 ? ZC ← TZi ! | ¿ ; P ← 1 ÷ D2² ;
W1 ← P*XR ; W2 ← P*YR ; W3 ← W1*XR ; W4 ← W1*XR ; W5 ← W2*YR ;
W6 ← W3*XR ; W7 ← W5*XR ; W8 ← W5*YR ;
B1 ← B1 + W6*XR ; B2 ← B2 + W6*YR ; B3 ← B3 + W7*XR ; B4 ← B4 + W6 ;
B5 ← B5 + W4*XR ; B6 ← B6 + W3 ; B7 ← B7+W3*ZR ; B10 ← B10 + W8*XR ;
B12 ← B12 + W5*XR ; B13 ← B13 + W4 ; B14 ← B14 + W4*ZR ; B14 ← B14 + W4*ZR ;
B17 ← B17 + W6*YR ; B19 ← B19 + W8 ; B20 ← B20 + W5 ; B21 ← B21 + W5*ZR ;
B27 ← B27 + W1 ; B28 ← B28 + W1*ZR ; B34 ← B34 + W2 ; B35 ← B35 + W2*ZR ;
B41 ← B41 + P ; B1 ← B42 + P*ZR ;}i
B9 ← B3 ; B11 ← B5 ; B18 ← B12 ; B25 ← B6 ; B27 ← B13 ; B33 ← B20 ;
©Résolution du système©
B1=0 ? !| ¿ ; {i:2 7} BRi ← Bi ÷ B1}i ; L7 ← 0 ;
{L:2 6} LM1 ← L – 1 ; L7 ← L7 + 7 ; {JJ:L 7} J7 ← L7 + JJ ; KL ← L –7 ; KJ ← JJ – 7 ;
{K1:1 LM1} KL ← KL + 7 ; KJ ← KJ + 7 ; BJ7 ← BJ7 – BKL*BRKJ }K1 }JJ ;
KL ← L7 + L ; BKL = 0 ? !| ¿ ; J1X ← KL + 1 ; J2X ← L7 + 7 ;
{J3:J1X J2X} BRJ3 ← BJ3 ÷ BKL }J3 }L ZC ← BR42! ]]
```

FIGURE 114 – Transcription en ADL du code source de Laporte en Fortran CDC (de l'IFP)

Annexe E

Algorithme Création de triangles

Cet algorithme a pour but de convertir un donnée filaire en triangles. J'ai volontairement omis le coté 3D dans cet algorithme afin de le rendre moins lourd à la lecture. Il n'empêche que cette troisième dimension est prise en compte au niveau du code.

CREER TRIANGLE ROUTE(ListeSegments↓, NSegments↓, Largeur↓, ListeTriangles↑, Ntriangles↑) : [[

{ $_{i:0}^{NSegments}$ Xpoint1 ← $ListeSegments_{point1.x}$; Ypoint1 ← $ListeSegments_{point1.x}$;

Xpoint2 ← $ListeSegments_{point2.x}$; Ypoint2 ← $ListeSegments_{point2.x}$;

©*On récupère les angles avec les tronçons précedent et successeur.*℗

anglePred ← $ListeSegments_{angle1}$ ÷ 2; ©*s'il n'y a pas de pred (angle=0), on met l'angle à 90°*℗

anglePred = 0 ? anglePred ← 90 ;|¿

angleSucc ← $ListeSegments_{angle2}$ ÷ 2 ; ©*s'il n'y a pas de succ (angle=0), on met l'angle à 90°*℗

angleSucc = 0 ? angleSucc ← 90 ;|¿

©*Calcul des points projetés*℗

XPt1_Proj ← Xpoint1 + largeur * cos (anglePred + 3π/2) ;

YPt2_Proj ← Ypoint1 + largeur * sin (anglePred + 3π/2) ;

XPt2_Proj ← Xpoint2 + largeur * cos (anglePred + 3π/2) ;

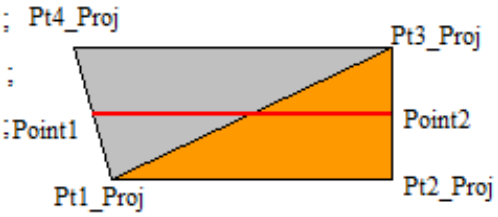
YPt2_Proj ← Ypoint2 + largeur * sin (anglePred + 3π/2) ;

XPt3_Proj ← Xpoint2 + largeur * cos (anglePred) ;

YPt3_Proj ← Ypoint2 + largeur * sin (anglePred) ;

XPt4_Proj ← Xpoint1 + largeur * cos (anglePred) ;

YPt4_Proj ← Ypoint1 + largeur * sin (anglePred) ;



©*Création de la liste des triangles*℗

NTriangles ← NTriangles +1 ;

©*Triangle 1 (en orange sur la figure)*℗

$ListeTriangles_{Ntriangles, point1.x}$ ← XPt1_Proj ; $ListeTriangles_{Ntriangles, point1.y}$ ← YPt1_Proj ;

$ListeTriangles_{Ntriangles, point2.x}$ ← XPt2_Proj ; $ListeTriangles_{Ntriangles, point2.y}$ ← Ypt2_Proj ;

$ListeTriangles_{Ntriangles, point3.x}$ ← XPt3_Proj ; $ListeTriangles_{Ntriangles, point3.y}$ ← YPt3_Proj ;

NTriangles ← NTriangles +1 ;

©*Triangle 2 (en gris sur la figure)*℗

$ListeTriangles_{Ntriangles, point1.x}$ ← XPt1_Proj ; $ListeTriangles_{Ntriangles, point1.y}$ ← YPt1_Proj ;

$ListeTriangles_{Ntriangles, point2.x}$ ← XPt3_Proj ; $ListeTriangles_{Ntriangles, point2.y}$ ← YPt3_Proj ;

$ListeTriangles_{Ntriangles, point3.x}$ ← XPt4_Proj ; $ListeTriangles_{Ntriangles, point3.y}$ ← YP4_Proj ;}$_{i}$]]

FIGURE 115 – Algorithme pour créer les triangles de route à partir d'un segment créé par Julien Richard

Annexe F

Algorithme Topologie

Cet algorithme a pour but de trouver les segments predecesseurs et successeurs d'un tronçon. Pour cela, chauqe tronçon a un vecteur de N tronçons successeurs possibles et un autre vecteur de N tronçons predecesseurs possibles. Je considére le point 1 de chaque tronçon comme un point d'entrée « predecesseurs » et donc le point 2 comme un point d'entrée « successeurs ».

CREER TOPOLOGIE(ListeSegmentsRoute, NSegments) : ⟦ $\{_{i:\,NSegments}^{0}$ ©*On regarde pour chaque tronçon si les coordonnées de chaque extrémité se trouve dans les autres tronçons*℗

XtestPoint1 ← $ListeSegmentsRoute_{i,\,point1.x}$; YtestPoint1 ← $ListeSegmentsRoute_{i,\,point1.y}$;

XtestPoint2 ← $ListeSegmentsRoute_{i,\,point2.x}$; YtestPoint2 ← $ListeSegmentsRoute_{i,\,point2.y}$;

$\{_{j:0}^{NSegments}$ i = j ? ↱| ¿ XtestPoint1 = $ListeSegmentsRoute_{j,\,point1.x}$ ⋃ YtestPoint1 = $ListeSegmentsRoute_{j,\,point1.y}$? ©*On regarde si le tronçon trouvé est déjà connu dans la table SIF du segment I* ℗

SegmentConnu ← ⊳ ; $\{_{k:0}^{NSegPred}$ $ListeSegmentsRoute_{i,\,k}$ = j ? SegmentConnu ← ¬ !| ¿ $\}_k$

SegmentConnu? ↱| NSegPred = NsegPred + 1 ; $ListeSegmentsRoute_{i,\,NSegPred}$ = j ¿ ©*On regarde si le tronçon trouvé est déjà connu dans la table SIF du segment j* ℗ SegmentConnu ← ⊳ ; $\{_{k:0}^{NSegPred}$ $ListeSegmentsRoute_{j,\,k}$ = i ? SegmentConnu ← ¬ !| ¿ $\}_k$; SegmentConnu? ↱| NSegPred = NsegPred + 1 ; $ListeSegmentsRoute_{j,\,NSegPred}$ = i ¿ |¿

©*On fait de meme avec l'autre extremité*℗

XtestPoint2 = $ListeSegmentsRoute_{j,\,point2.x}$ ⋃ YtestPoint2 = $ListeSegmentsRoute_{j,\,point2.y}$? ©*On regarde si le tronçon trouvé est déjà connu dans la table SIF du segment i* ℗

SegmentConnu ← ⊳ ; $\{_{k:0}^{NSegSucc}$ $ListeSegmentsRoute_{i,\,k}$ = j ? SegmentConnu ← ¬ !| ¿ $\}_k$

;SegmentConnu? ↱| NSegSucc = NsegSucc + 1 ; $ListeSegmentsRoute_{i,\,NSegSucc}$ = j ¿ ©*On regarde si le tronçon trouvé est déjà connu dans la table SIF du segment j* ℗ SegmentConnu ← ⊳ ; $\{_{k:0}^{NSegSucc}$ $ListeSegmentsRoute_{j,\,k}$ = i ? SegmentConnu ← ¬ !| ¿ $\}_k$; SegmentConnu? ↱| NSegSucc = NsegSucc + 1 ; $ListeSegmentsRoute_{j,\,NSegSucc}$ = i ¿ |¿ $\}_j$ $\}_i$ ⟧

FIGURE 116 – Algorithme appliquant une topologie sur un réseau de tronçon créé par Julien Richard

Annexe G

Algorithme AppartenanceTriangle

```
AppartenanceTriangle(↓PointA, ↓PointB, ↓PointC, ↓PointP, ↑Appartient) : [[
©Calcul des aires©
alpha ← aire(↓PointP, ↓PointB, ↓PointC, ↑valeur) / aire(↓PointA, ↓PointB, ↓PointC, ↑valeur) ;
beta ← aire(↓PointA, ↓PointP, ↓PointC, ↑valeur) / aire(↓PointA, ↓PointB, ↓PointC, ↑valeur) ;
gamma ← aire(↓PointP, ↓PointB, ↓PointP, ↑valeur) / aire(↓PointA, ↓PointB, ↓PointC, ↑valeur) ;
©Test d'appartenance©
? alpha+beta+gamma =1 ; Appartient ← ٦ | Appartient ← ⊧ ¿ ]]

aire(↓PointP, ↓PointB, ↓PointP, ↑valeur) : [[valeur ← 0,5 x abs(PointB.x – PointA.x)(PointC.y –
PointA.y) – (PointC.x – PointA.x)(PointB.y – PointA.y) ]]
```

FIGURE 117 – Algorithme pour connaître l'appartenance d'un point P à un triangle ABC

Annexe H

Algorithme Intersection

FINDING INTERSECTION (MAT↓, DX↓, DY↓, X0↓, Y0↓, NCOL↓, NROW↓, TXYZ↓, NTXYZ↓, NEW_TXYZ↑, NEW_NTXYZ↑): [[©*We have a table called TXYZ filled with the points which delimited the polygon. They are classed by order.*© NEW_NTXYZ ← 0 ; NEXT_I ← 0 ; NEXT_J ← 0 ;

{ $_{i:0}$ ©*We take the 2 following points*© X1 ← $TXYZ_{i,1}$; Y1 ← $TXYZ_{i,2}$; **G3D**(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X1↓, Y1↓, Z↑) ; Z1 ← Z ; i ← i+1 ; X2 ← $TXYZ_{i,1}$; Y2 ← $TXYZ_{i,2}$; **G3D**(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X2↓, Y2↓, Z↑) ; Z2 ← Z ;

©*We extract (and keep) the first point)*©

$NEW\_TXYZ_{NEW\_NTXYZ,1}$ ← X1;

$NEW\_TXYZ_{NEW\_NTXYZ,2}$ ← Y1 ;

$NEW\_TXYZ_{NEW\_NTXYZ,3}$ ← Z1 ;

NEW_NTXYZ ← NEW_NTXYZ +1;

©*We find the I J cell of the first point*©

$$J \leftarrow \frac{X1 - X0}{DX} ; I \leftarrow \frac{Y1 - Y0}{DY} ;$$

©*We find the I J cell of the second point*©

$$J\_END \leftarrow \frac{X2 - X0}{DX} ; I\_END \leftarrow \frac{Y2 - Y0}{DY} ;$$

©*we calculate the a and b value of the line equation y = ax + b*©

$$a \leftarrow \frac{Y2 - Y1}{X2 - X1} ; b \leftarrow Y1 - a*X1;$$

©*We have a loop while we did not reach the last cell*©

{ $_{k:0}$ ©*If we have a different cell to study*©

NEXT_I ≠ 0 ? J ← NEXT_J ; I ← NEXT_I ; X1 ← NEXT_X ; Y1 ← NEXT_Y ; | ¿

©*test to know if the 2 points are in the same cell*© I = I_END ⋂ J = J_END ?

©*We calculate the line equation of the diagonal to know if the 2 points are in the same triangle*©

$$a\_diag \leftarrow \frac{((Y0 + DY * I + DY) - (Y0 + DY * I)}{(X0 + DX * J + DX) - (X0 + DX * J)} ;$$

b_diag ← (Y0 + DY*I +DY) – a_diag * (X0 + DX * J);

Y1 < a_diag * (X0 + DX * J) + b_diag ? ©***the first point*** *is in the first triangle* ▨ ©

Y2 < a_diag * (X0 + DX * J) + b_diag ? ©***the two points*** *are in the first triangle* ▨ ©

©*We add this new point to the table*©

$NEW\_TXYZ_{NEW\_NTXYZ,1}$ ← X2;

$NEW\_TXYZ_{NEW\_NTXYZ,2}$ ← Y2 ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z2 ;

NEW_NTXYZ ← NEW_NTXYZ +1 ; ↱2

| ©***the second point*** *is in the second triangle, we have to know the coordinate of the* ***intersection point*** *between the diagonal and the line* ☺

$$X\_INTER \leftarrow \frac{b\_diag - b}{a - a\_diag} ; Y\_INTER \leftarrow a*X\_INTER + b ;$$

©*We add this new point to the table*☺

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1;

©*we add the last point in the table*☺

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X2;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y2 ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z2 ;

NEW_NTXYZ ← NEW_NTXYZ +1 ↱2 ¿ | ©***the first point*** *is in the second triangle* ☺

Y2 > a_diag * (X0 + DX * J) + b_diag ?

©***the two points*** *are in the second triangle* ☺ ©*We add this new point to the table*☺

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X2;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y2 ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z2 ;

NEW_NTXYZ ← NEW_NTXYZ +1 ; ↱2 | ©***the second point*** *is in the first triangle, we have to know the coordinate of* ***the intersection point*** *between the diagonal and the line* ☺

$$X\_INTER \leftarrow \frac{b\_diag - b}{a - a\_diag} ; Y\_INTER \leftarrow a*X\_INTER + b ;$$

©*We add this new point to the table*☺

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑)

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1;

©*we add the last point in the table*☺

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X2;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y2 ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z2 ;

NEW_NTXYZ ← NEW_NTXYZ +1 ↱2 ¿ ¿ | ©***the 2 points are not in the same cell***ↄ

©*we test the position of the 1rst point from the diagonal (upper or lower)*ↄ

$$a\_diag \leftarrow \frac{((Y0 + DY * I + DY) - (Y0 + DY * I)}{(X0 + DX * J + DX) - (X0 + DX * J)} ;$$

b_diag ← (Y0 + DY*I +DY) – a_diag * (X0 + DX * J) ;

Y1 < a_diag * (X0 + DX * J) + b_diag ? ©*the 1rst point is lower than the diagonal*ↄ ©*we test the position of the 2nd point from the diagonal (upper or lower)*ↄ

Y2 < a_diag * (X0 + DX * J) + b_diag ? ©*the 2nd point is lower than the diagonal*ↄ

©***We have only 2 solutions:*** ❑ ***and*** ❑ ◎

©*First side* ❑ ↄ

X1_CELL ← X0 + DX*I ;

Y1_CELL ← Y0 + DY*J ;

X2_CELL ← X0 + DX*I ;

Y2_CELL ← Y0 + DY*J + DY ;

$$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL} ; b1 \leftarrow Y1\_CELL - a1*X1\_CELL ;$$

©*Calculation of the intersection*ↄ

$$X\_INTER \leftarrow \frac{b1 - b}{a - a1} ; Y\_INTER \leftarrow a*X\_INTER + b ;$$

©*if this intersection is well in the I J cell, it is our intersection!*ↄ

$$I\_TEST \leftarrow \frac{X\_INTER - X0}{DX} ; J\_TEST \leftarrow \frac{Y\_INTER - Y0}{DY} ; I\_TEST = I \bigcap J\_TEST = J \ ?$$

©*We can interpolate the z value and add this new point to the table*

G3D(↓MAT, ↓DX, ↓DY, ↓NCOL, ↓NROW, ↓X0, ↓Y0, ↓X_INTER, ↓Y_INTER, ↑Z) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I; NEXT_J ← J – 1; NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER |

©*at this point, we are in the other side of the cell* ❑ ↄ

X1_CELL ← X0 + DX*I ;

Y1_CELL ← Y0 + DY*J ;

X2_CELL ← X0 + DX*I + DX ;

Y2_CELL ← Y0 + DY*J ;

$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*☺

$X\_INTER \leftarrow \frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*We can interpolate the z value and add this new point to the table*☺

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑) ;

$NEW\_TXYZ_{NEW\_NTXYZ,1}$ ← X_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,2}$ ← Y_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I – 1 ; NEXT_J ← J ; NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER ¿ |

©*the 2nd point is upper than the diagonal*☺

©*We have an intersection with the diagonal + one side of the cell*☺

$X\_INTER \leftarrow \frac{b\_diag - b}{a - a\_diag}$; Y_INTER ← a*X_INTER + b ;

©*We add this new point to the table*☺

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑)

$NEW\_TXYZ_{NEW\_NTXYZ,1}$ ← X_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,2}$ ← Y_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1;

©*test to know which side the polygone intersects (* ❑ ***or*** ❑ *)*☺

©*First side* ❑ ☺

X1_CELL ← X0 + DX*I + DX ;

Y1_CELL ← Y0 + DY*J ;

X2_CELL ← X0 + DX*I + DX ;

Y2_CELL ← Y0 + DY*J + DY ;

$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*☺

$X\_INTER \leftarrow \frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*if this intersection is well in the I J cell, it is our intersection!*☺

I_TEST ← $\frac{X\_INTER - X0}{DX}$; J_TEST ← $\frac{Y\_INTER - Y0}{DY}$; I_TEST = I ⋂ J_TEST = J ?

©*We can interpolate the z value and add this new point to the table*

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I; NEXT_J ← J + 1 ; NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER |

© *Second side* ❐ ◎

X1_CELL ← X0 + DX*I ;

Y1_CELL ← Y0 + DY*J + DY ;

X2_CELL ← X0 + DX*I;

Y2_CELL ← Y0 + DY*J + DY ;

a1 ← $\frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*◎

X_INTER ← $\frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*if this intersection is well in the I J cell, it is our intersection!*◎

©*We can interpolate the z value and add this new point to the table*

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I + 1 ; NEXT_J ← J; NEXT_X ← X_INTER ; NEXT-Y← Y_INTER ¿¿ |

©*the 1rst point is upper than the diagonal*◎

©***we test the position of the 2nd point from the diagonal (upper or lower)***◎

Y2 > a_diag * (X0 + DX * J) + b_diag ? ©*the 2nd point is upper than the diagonal*◎

©***We have only 2 solutions:*** ❐ ***and*** ❐ ◎

©*First side* ❐ ◎

X1_CELL ← X0 + DX*I + DX ;

Y1_CELL ← Y0 + DY*J ;

X2_CELL ← X0 + DX*I + DX ;

Y2_CELL ← Y0 + DY*J + DY ;

$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*☺

$X\_INTER \leftarrow \frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*if this intersection is well in the I J cell, it is our intersection!*☺

$I\_TEST \leftarrow \frac{X\_INTER - X0}{DX}$; $J\_TEST \leftarrow \frac{Y\_INTER - Y0}{DY}$; I_TEST = I ⋂ J_TEST = J ?

©*We can interpolate the z value and add this new point to the table*

G3D(↓MAT, ↓DX, ↓DY, ↓NCOL, ↓NROW, ↓X0, ↓Y0, ↓X_INTER, ↓Y_INTER, ↑Z) ;

$NEW\_TXYZ_{NEW\_NTXYZ,1}$ ← X_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,2}$ ← Y_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I; NEXT_J ← J + 1 ; NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER |

© *Second side* ☐ ☺

X1_CELL ← X0 + DX*I ;

Y1_CELL ← Y0 + DY*J + DY ;

X2_CELL ← X0 + DX*I;

Y2_CELL ← Y0 + DY*J + DY ;

$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*☺ $X\_INTER \leftarrow \frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*We can interpolate the z value and add this new point to the table*

G3D(↓MAT, ↓DX, ↓DY, ↓NCOL, ↓NROW, ↓X0, ↓Y0, ↓X_INTER, ↓Y_INTER, ↑Z) ;

$NEW\_TXYZ_{NEW\_NTXYZ,1}$ ← X_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,2}$ ← Y_INTER ;

$NEW\_TXYZ_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I + 1; NEXT_J ← J; NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER ¿ |

©*the 2nd point is lower than the diagonal*☺

©*We have an intersection with the diagonal + one side of the cell*☺

$X\_INTER \leftarrow \frac{b\_diag - b}{a - a\_diag}$; Y_INTER ← a*X_INTER + b ;

©*We add this new point to the table*©

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1;

©*test to know which side the polygone intersects (* ☐ ***and*** ☐ *)*©

©*First side* ☐ ©

X1_CELL ← X0 + DX*I ;

Y1_CELL ← Y0 + DY*J ;

X2_CELL ← X0 + DX*I ;

Y2_CELL ← Y0 + DY*J + DY ;

$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*© X_INTER ← $\frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*if this intersection is well in the I J cell, it is our intersection!*©

$I\_TEST \leftarrow \frac{X\_INTER - X0}{DX}$; $J\_TEST \leftarrow \frac{Y\_INTER - Y0}{DY}$; I_TEST = I ⋂ J_TEST = J ?

©*We can interpolate the z value and add this new point to the table*

G3D(MAT↓, DX↓, DY↓, NCOL↓, NROW↓, X0↓, Y0↓, X_INTER↓, Y_INTER↓, Z↑) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I; NEXT_J ← J – 1 ; NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER

©*at this point, we are in the other side of the cell* ☐ ©

X1_CELL ← X0 + DX*I ;

Y1_CELL ← Y0 + DY*J ;

X2_CELL ← X0 + DX*I + DX ;

Y2_CELL ← Y0 + DY*J ;

$a1 \leftarrow \frac{Y2\_CELL - Y1\_CELL}{X2\_CELL - X1\_CELL}$; b1 ← Y1_CELL – a1*X1_CELL ;

©*Calculation of the intersection*©

X_INTER ← $\frac{b1 - b}{a - a1}$; Y_INTER ← a*X_INTER + b ;

©*We can interpolate the z value and add this new point to the table*☺

G3D(MAT, DX, DY, NCOL, NROW, X0, Y0, X_INTER, Y_INTER, Z) ;

NEW_TXYZ $_{NEW\_NTXYZ,1}$ ← X_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,2}$ ← Y_INTER ;

NEW_TXYZ $_{NEW\_NTXYZ,3}$ ← Z ;

NEW_NTXYZ ← NEW_NTXYZ +1 ;

NEXT_I ← I – 1 ; NEXT_J ← J NEXT_X1 ← X_INTER ; NEXT_Y1← Y_INTER; ¿

¿ ©*end of the test to know the position of the second point*☺

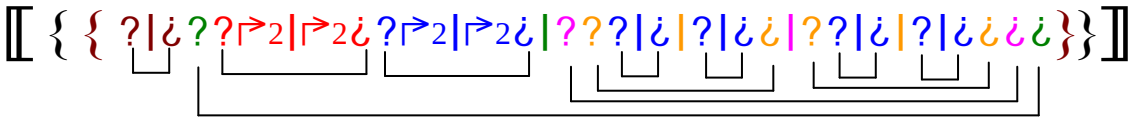
¿ ©*end of the test to know the position of the first point*☺

¿ ©*end of the test to know if the 2 points are in the same cell*☺

}$_{k}$ / NEXT_I = I_END ∩ NEXT_J = J_END

}$_{i}$ / i<NTXYZ]]

Skeleton:



Tome 2 :
Traffic city3D créé par Julien Richard

v1.0

Manuel utilisateur

Un CD contenant une présentation vidéo est disponible avec ce manuel

Table des matières

Chapitre 1

Documentation du logiciel Traffic City 3D

1.1 Introduction

Ce programme est développé dans le cadre de la thèse de Julien Richard. Il a pour but de gérer des données SIG standard et de les convertir en 3D pour un affichage virtuel de villes. Un module de trafic est intégré pour afficher la circulation des voitures dans la ville modélisée. Pour développer ce programme, Qt Creator a été utilisé pour compiler le projet ainsi que pour la création de l'interface homme-machine. La partie fonction est codée en C++. Le rendu 3D est géré par OpenGl et la gestion de données SIG par l'API Qgis. Pour développer ce code il est nécessaire de compiler Qgis (voir partie suivante) et d'ajouter les paquets SDL et SDL image nécessaires pour la gestion respective de la souris et des textures. La bibliothèque Assimp est utilisée pour la gestion des objets 3D. Ce programme est développé sous débian (Linux) 32bits et 64bits et est sous la licence GPL.

Pour la complilation de Qgis, la documentation se trouve ici : `https://github.com/qgis/QGIS/blob/master/INSTALL`

Chapitre 2

Index des espaces de nommage

2.1 Liste des espaces de nommage

Liste de tous les espaces de nommage avec une brève description :

Chapitre 3

Index hiérarchique

3.1 Hiérarchie des classes

Cette liste d'héritage est classée approximativement par ordre alphabétique :

Chapitre 4

Index des classes

4.1 Liste des classes

Liste des classes, structures, unions et interfaces avec une brève description :

Chapitre 5

Index des fichiers

5.1 Liste des fichiers

Liste de tous les fichiers avec une brève description :

Chapitre 6

Documentation des espaces de nommage

6.1 Référence de l'espace de nommage Ui

Classes

— class chooseField
— class cityNameChoice
— class fieldDataBuilding
— class fieldDataRoad
— class gestionSimulation
— class gis4DWindow
— class helpBuilding
— class helpGrass
— class helpRoad
— class helpTopo
— class helpTree
— class helpWater
— class hologramme
— class holoView
— class newCity
— class stereoDisplay

Chapitre 7

Documentation des classes

7.1 Référence de la classe calculPositionSoleil

`#include <calculpositionsoleil.h>`

Fonctions membres publiques

— caculPositionSoleil ()

Fonctions membres publiques statiques

— static void calculJourJulien (long double jour, long double mois, long double annee, long double heure, long double minute, long double seconde)

7.1.1 Documentation des constructeurs et destructeur

7.1.1.1 caculPositionSoleil()

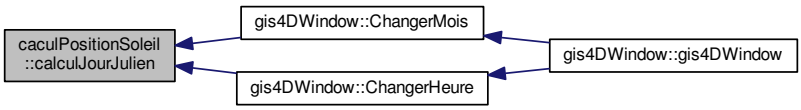
`caculPositionSoleil::caculPositionSoleil ( )`

7.1.2 Documentation des fonctions membres

7.1.2.1 calculJourJulien()

```
static void caculPositionSoleil::calculJourJulien (
            long double jour,
            long double mois,
            long double annee,
            long double heure,
            long double minute,
            long double seconde )  [inline], [static]
```

Voici le graphe des appelants de cette fonction :



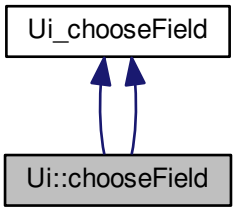
La documentation de cette classe a été générée à partir des fichiers suivants :

- calculpositionsoleil.h
- calculpositionsoleil.cpp

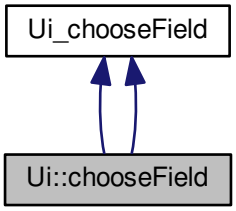
7.2 Référence de la classe Ui : :chooseField

```
#include <ui_choosefield.h>
```

Graphe d'héritage de Ui : :chooseField :



Graphe de collaboration de Ui : :chooseField :

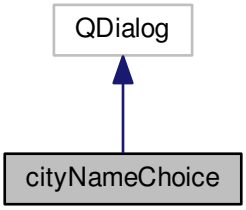


Membres hérités additionnels

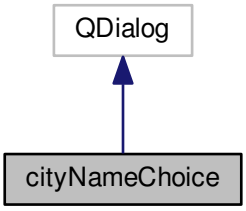
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_choosefield.h

7.3 Référence de la classe cityNameChoice

`#include <citynamechoice.h>`
Graphe d'héritage de cityNameChoice :



Graphe de collaboration de cityNameChoice :



Connecteurs publics

— void changeCityName ()

Fonctions membres publiques

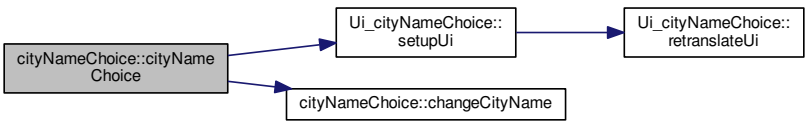
— cityNameChoice (QWidget *parent=0)
— ~cityNameChoice ()

7.3.1 Documentation des constructeurs et destructeur

7.3.1.1 cityNameChoice()

```
cityNameChoice::cityNameChoice (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.3.1.2 ~cityNameChoice()

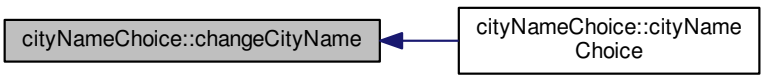
```
cityNameChoice::~cityNameChoice ( )
```

7.3.2 Documentation des fonctions membres

7.3.2.1 changeCityName

```
void cityNameChoice::changeCityName ( )  [slot]
```

Voici le graphe des appelants de cette fonction :

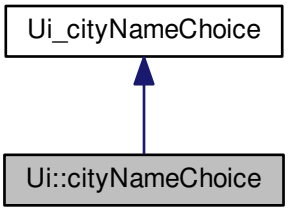


La documentation de cette classe a été générée à partir des fichiers suivants :
— citynamechoice.h
— citynamechoice.cpp

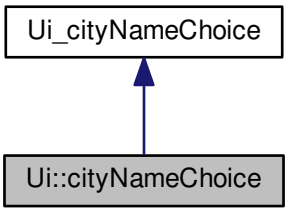
7.4 Référence de la classe Ui : :cityNameChoice

`#include <ui_citynamechoice.h>`

Graphe d'héritage de Ui : :cityNameChoice :



Graphe de collaboration de Ui : :cityNameChoice :



Membres hérités additionnels

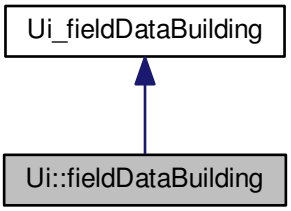
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_citynamechoice.h

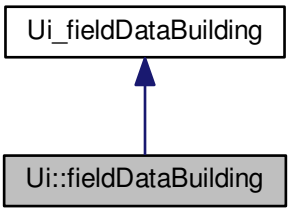
7.5 Référence de la classe Ui : :fieldDataBuilding

`#include <ui_fielddatabuilding.h>`

Graphe d'héritage de Ui : :fieldDataBuilding :



Graphe de collaboration de Ui : :fieldDataBuilding :



Membres hérités additionnels

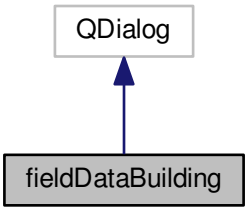
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_fielddatabuilding.h

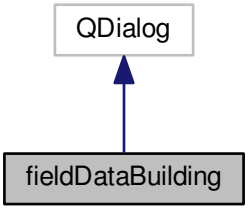
7.6 Référence de la classe fieldDataBuilding

`#include <fielddatabuilding.h>`

Graphe d'héritage de fieldDataBuilding :



Graphe de collaboration de fieldDataBuilding :



Connecteurs publics

— void okButton ()

Fonctions membres publiques

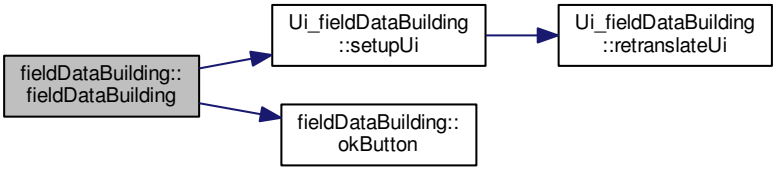
— fieldDataBuilding (QWidget *parent=0)
— ~fieldDataBuilding ()

7.6.1 Documentation des constructeurs et destructeur

7.6.1.1 fieldDataBuilding()

```
fieldDataBuilding::fieldDataBuilding (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.6.1.2 ~fieldDataBuilding()

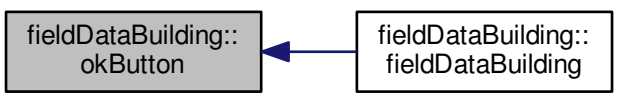
```
fieldDataBuilding::~fieldDataBuilding ( )
```

7.6.2 Documentation des fonctions membres

7.6.2.1 okButton

```
void fieldDataBuilding::okButton ( )  [slot]
```

Voici le graphe des appelants de cette fonction :



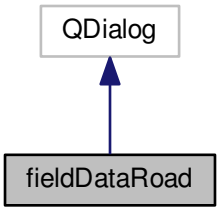
La documentation de cette classe a été générée à partir des fichiers suivants :

- fielddatabuilding.h
- fielddatabuilding.cpp

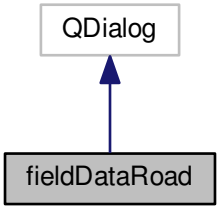
7.7 Référence de la classe fieldDataRoad

`#include <fielddataroad.h>`

Graphe d'héritage de fieldDataRoad :



Graphe de collaboration de fieldDataRoad :



Connecteurs publics

— void okButton ()

Fonctions membres publiques

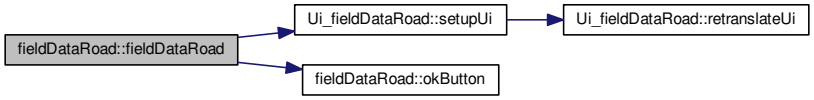
— fieldDataRoad (QWidget *parent=0)
— ~fieldDataRoad ()

7.7.1 Documentation des constructeurs et destructeur

7.7.1.1 fieldDataRoad()

```
fieldDataRoad::fieldDataRoad (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.7.1.2 ~fieldDataRoad()

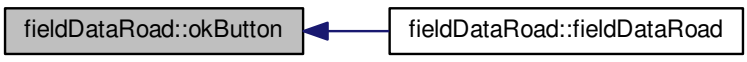
```
fieldDataRoad::~fieldDataRoad ( )
```

7.7.2 Documentation des fonctions membres

7.7.2.1 okButton

```
void fieldDataRoad::okButton ( )  [slot]
```

Voici le graphe des appelants de cette fonction :



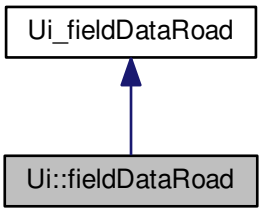
La documentation de cette classe a été générée à partir des fichiers suivants :

- fielddataroad.h
- fielddataroad.cpp

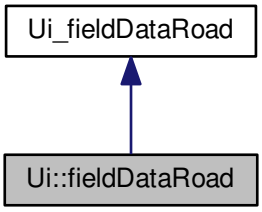
7.8 Référence de la classe Ui : :fieldDataRoad

`#include <ui_fielddataroad.h>`

Graphe d'héritage de Ui : :fieldDataRoad :



Graphe de collaboration de Ui : :fieldDataRoad :



Membres hérités additionnels

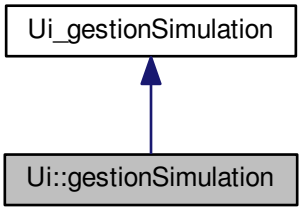
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_fielddataroad.h

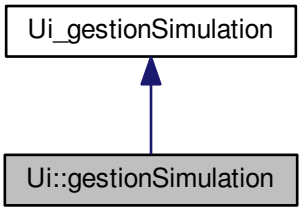
7.9 Référence de la classe Ui : :gestionSimulation

`#include <ui_gestionsimulation.h>`

Graphe d'héritage de Ui : :gestionSimulation :



Graphe de collaboration de Ui : :gestionSimulation :

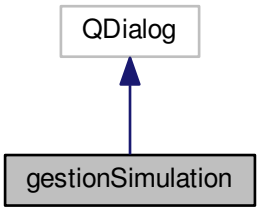


Membres hérités additionnels

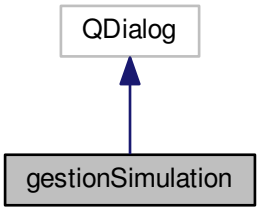
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_gestionsimulation.h

7.10 Référence de la classe gestionSimulation

`#include <gestionsimulation.h>`
Graphe d'héritage de gestionSimulation :



Graphe de collaboration de gestionSimulation :



Connecteurs publics

- void ChangerNbVoiture (int i)
  Fonction pour changer le nombre de voitures pour la simulation.
- void ChangerVitesse (int i)
  Fonction pour changer la vitesse de simulation.
- void ChangerModeCollision (bool active)
  Fonction pour déclencher le mode collision pour la simulation.
- void lancerSimulation ()
  Fonction pour lancer la simulation.

Fonctions membres publiques

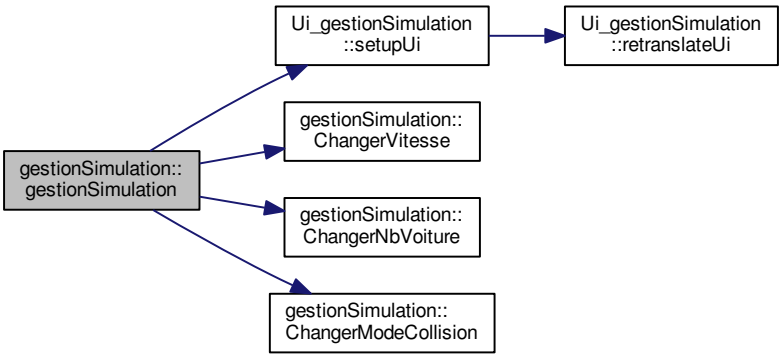
- gestionSimulation (QWidget *parent=0)
- ~gestionSimulation ()

7.10.1 Documentation des constructeurs et destructeur

7.10.1.1 gestionSimulation()

```
gestionSimulation::gestionSimulation (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.10.1.2 ~gestionSimulation()

```
gestionSimulation::~gestionSimulation ( )
```

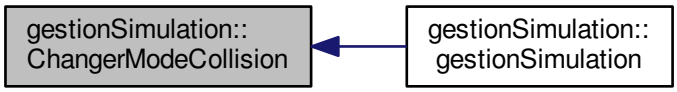
7.10.2 Documentation des fonctions membres

7.10.2.1 ChangerModeCollision

```
gestionSimulation::ChangerModeCollision (
            bool active ) [slot]
```

Fonction pour déclencher le mode collision pour la simulation.

Voici le graphe des appelants de cette fonction :



7.10.2.2 ChangerNbVoiture

```
gestionSimulation::ChangerNbVoiture (
            int i ) [slot]
```

Fonction pour changer le nombre de voitures pour la simulation.

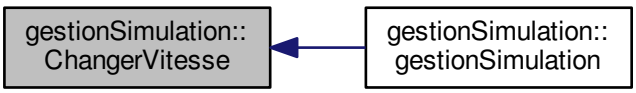
Voici le graphe des appelants de cette fonction :

7.10.2.3 ChangerVitesse

```
gestionSimulation::ChangerVitesse (
            int i ) [slot]
```

Fonction pour changer la vitesse de simulation.

Voici le graphe des appelants de cette fonction :



7.10.2.4 lancerSimulation

```
gestionSimulation::lancerSimulation ( ) [slot]
```

Fonction pour lancer la simulation.

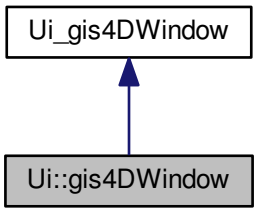
La documentation de cette classe a été générée à partir des fichiers suivants :

- gestionsimulation.h
- gestionsimulation.cpp

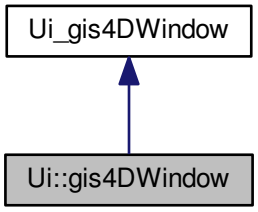
7.11 Référence de la classe Ui : :gis4DWindow

`#include <ui_gis4dwindow.h>`

Graphe d'héritage de Ui : :gis4DWindow :



Graphe de collaboration de Ui : :gis4DWindow :



Membres hérités additionnels

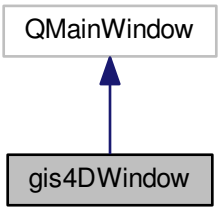
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_gis4dwindow.h

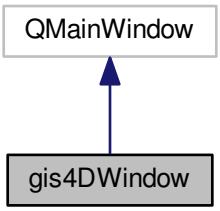
7.12 Référence de la classe gis4DWindow

`#include <gis4dwindow.h>`

Graphe d'héritage de gis4DWindow :



Graphe de collaboration de gis4DWindow :



Connecteurs publics

- void showHouse (bool disp)
  Fonction pour afficher la donnée de bâti.
- void showRoad (bool disp)
  Fonction pour afficher la donnée de routes.
- void showTree (bool disp)
  Fonction pour afficher la donnée d'arbres.
- void showGrass (bool disp)
  Fonction pour afficher la donnée de zones herbacées.
- void showWater (bool disp)
  Fonction pour afficher la donnée de surface d'eau.
- void showTopo (bool disp)
  Fonction pour afficher la donnée de topographie.
- void showHoloWindow (bool disp)
  Fonction pour afficher la vue holographique (4 vues 3D à 90° les unes des autres)
- void showStereo (bool disp)
  Fonction pour afficher la vue stéréoscopiqe cyan et rouge.
- void showWireMode (bool disp)
  Fonction pour afficher le mode filaire.
- void showSky3D (bool disp)
  Fonction pour afficher le ciel (skydome)

— void showLight (bool disp)
Fonction pour activer la vue avec les triangles éclairés.
— void showSteroCardBoard (bool disp)
Fonction pour afficher la vue stéréo sur 2 écrans séparés pour l'utilisation d'un casque de réalité virtuelle.
— void zoomInMode ()
Fonction gérant le zoom + sur la carte.
— void zoomOutMode ()
Fonction gérant le zoom - sur la carte.
— void panMode ()
Fonction gérant le déplacement de la carte.
— void fullExtendMode ()
Fonction pour afficher la carte en entier.
— void showCreateCity ()
Fonction pour lancer l'import d'une nouvelle ville.
— void loadCity1 ()
Fonction pour charger une ville déjà créée avec liens en dur.
— void loadCity3 ()
Fonction pour charger une ville déjà créée avec liens en dur.
— void loadCity4 ()
Fonction pour charger une ville déjà créée avec liens en dur.
— void loadCityProcessed ()
Fonction pour charger une ville déjà créée.
— void enableDisable2D3D (int i)
Fonction pour gérer le basculement entre le mode 2D et 3D.
— void ChangerMois (int i)
Fonction pour changer l'éclairage en fonction du mois.
— void ChangerVitesse (int i)
Fonction pour changer la vitesse de simulation.
— void ChangerHeure (int i)
Fonction pour changer l'éclairage en fonction de l'heure.
— void afficherMenuSimulation ()
Fonction pour afficher le menu dédié à la simulation.

Fonctions membres publiques

— gis4DWindow (QWidget *parent=0)
— ~gis4DWindow ()

Fonctions membres protégées

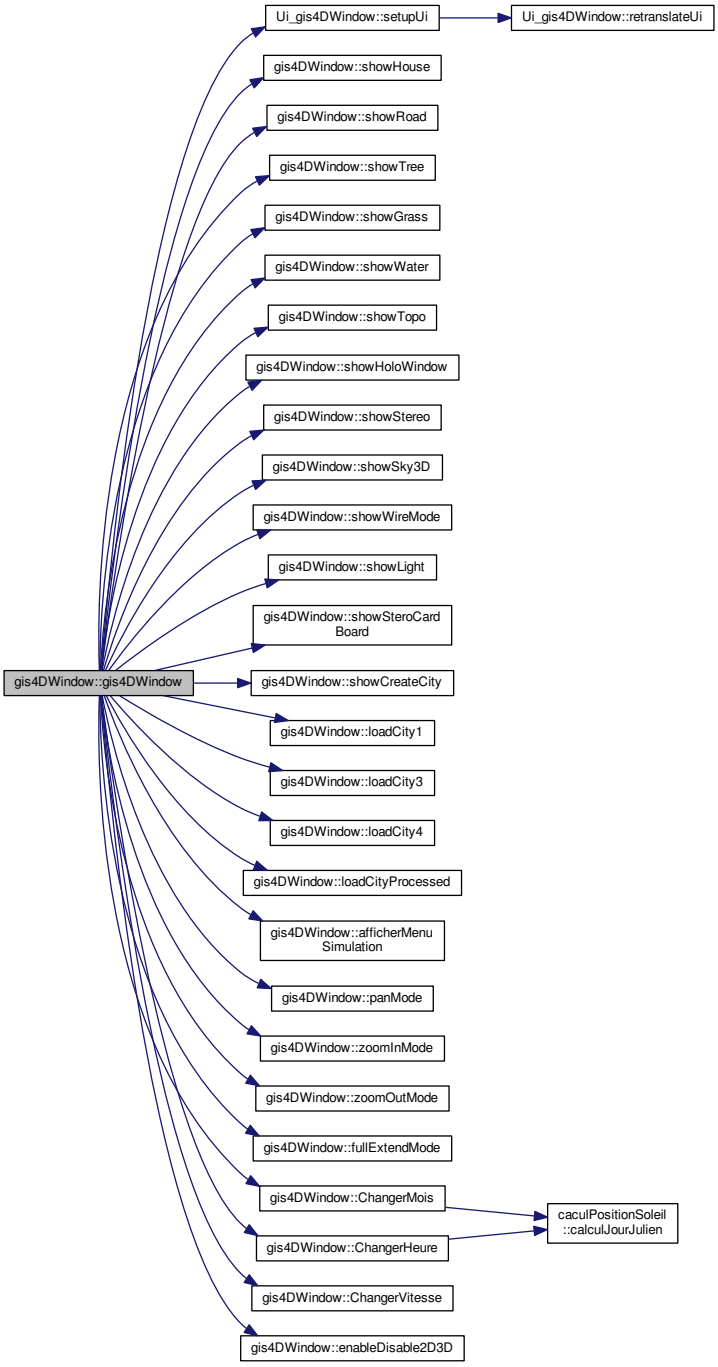
— void mouseMoveEvent (QMouseEvent *event)
Fonction pour collecter les évènements de la souris.

7.12.1 Documentation des constructeurs et destructeur

7.12.1.1 gis4DWindow()

```
gis4DWindow::gis4DWindow (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.12.1.2 ~gis4DWindow()

`gis4DWindow::~gis4DWindow ( )`

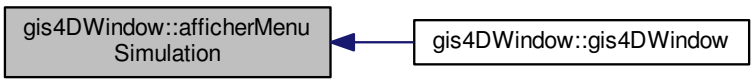
7.12.2 Documentation des fonctions membres

7.12.2.1 afficherMenuSimulation

`gis4DWindow::afficherMenuSimulation ( ) [slot]`

Fonction pour afficher le menu dédié à la simulation.

Voici le graphe des appelants de cette fonction :

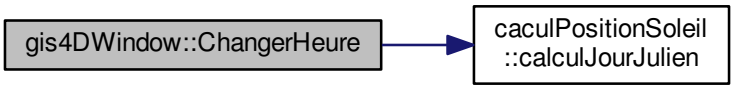


7.12.2.2 ChangerHeure

```
gis4DWindow::ChangerHeure (
            int i ) [slot]
```

Fonction pour changer l'éclairage en fonction de l'heure.

Voici le graphe d'appel pour cette fonction :



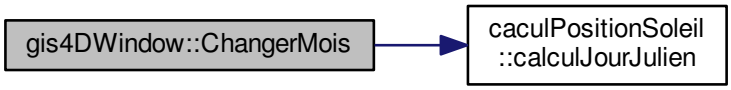
Voici le graphe des appelants de cette fonction :

7.12.2.3 ChangerMois

```
gis4DWindow::ChangerMois (
            int i ) [slot]
```

Fonction pour changer l'éclairage en fonction du mois.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.12.2.4 ChangerVitesse

```
gis4DWindow::ChangerVitesse (
            int i ) [slot]
```

Fonction pour changer la vitesse de simulation.

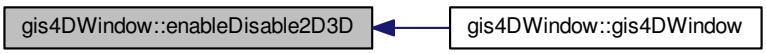
Voici le graphe des appelants de cette fonction :

7.12.2.5 enableDisable2D3D

```
gis4DWindow::enableDisable2D3D (
            int i ) [slot]
```

Fonction pour gérer le basculement entre le mode 2D et 3D.

Voici le graphe des appelants de cette fonction :



7.12.2.6 fullExtendMode

```
gis4DWindow::fullExtendMode ( ) [slot]
```

Fonction pour afficher la carte en entier.

Voici le graphe des appelants de cette fonction :

7.12.2.7 loadCity1

```
gis4DWindow::loadCity1 ( ) [slot]
```

Fonction pour charger une ville déjà créée avec liens en dur.

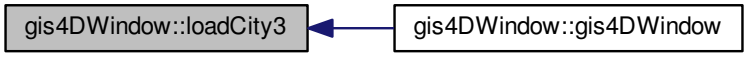
Voici le graphe des appelants de cette fonction :

7.12.2.8 loadCity3

`gis4DWindow::loadCity3 ( ) [slot]`

Fonction pour charger une ville déjà créée avec liens en dur.

Voici le graphe des appelants de cette fonction :



7.12.2.9 loadCity4

`gis4DWindow::loadCity4 ( ) [slot]`

Fonction pour charger une ville déjà créée avec liens en dur.

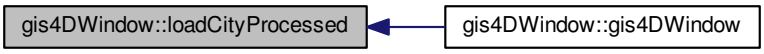
Voici le graphe des appelants de cette fonction :

7.12.2.10 loadCityProcessed

`gis4DWindow::loadCityProcessed ( ) [slot]`

Fonction pour charger une ville déjà créée.

Voici le graphe des appelants de cette fonction :



7.12.2.11 mouseMoveEvent()

```
gis4DWindow::mouseMoveEvent (
            QMouseEvent * event ) [protected]
```

Fonction pour collecter les évènements de la souris.

Paramètres

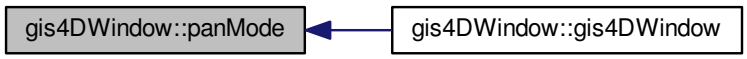
event	Evenements de la souris

7.12.2.12 panMode

`gis4DWindow::panMode ( ) [slot]`

Fonction gérant le déplacement de la carte.

Voici le graphe des appelants de cette fonction :



7.12.2.13 showCreateCity

`gis4DWindow::showCreateCity ( ) [slot]`

Fonction pour lancer l'import d'une nouvelle ville.

Voici le graphe des appelants de cette fonction :

7.12.2.14 showGrass

```
gis4DWindow::showGrass (
            bool disp ) [slot]
```

Fonction pour afficher la donnée de zones herbacées.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.15 showHoloWindow

```
gis4DWindow::showHoloWindow (
            bool disp ) [slot]
```

Fonction pour afficher la vue holographique (4 vues 3D à 90° les unes des autres)

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.16 showHouse

```
gis4DWindow::showHouse (
            bool disp ) [slot]
```

Fonction pour afficher la donnée de bâti.

Paramètres

disp	Booléen à bvrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.17 showLight

```
gis4DWindow::showLight (
            bool disp ) [slot]
```

Fonction pour activer la vue avec les triangles éclairés.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.18 showRoad

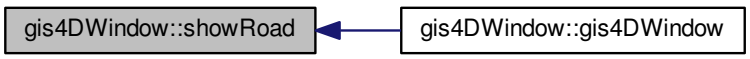
```
gis4DWindow::showRoad (
            bool disp ) [slot]
```

Fonction pour afficher la donnée de routes.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :



7.12.2.19 showSky3D

```
gis4DWindow::showSky3D (
            bool disp ) [slot]
```

Fonction pour afficher le ciel (skydome)

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.20 showStereo

```
gis4DWindow::showStereo (
            bool disp ) [slot]
```

Fonction pour afficher la vue stéréoscopiqe cyan et rouge.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.21 showSteroCardBoard

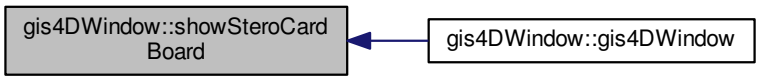
```
gis4DWindow::showSteroCardBoard (
            bool disp ) [slot]
```

Fonction pour afficher la vue stéréo sur 2 écrans séparés pour l'utilisation d'un casque de réalité virtuelle.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :



7.12.2.22 showTopo

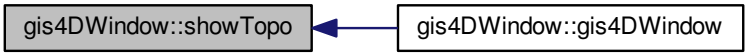
```
gis4DWindow::showTopo (
            bool disp ) [slot]
```

Fonction pour afficher la donnée de topographie.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :



7.12.2.23 showTree

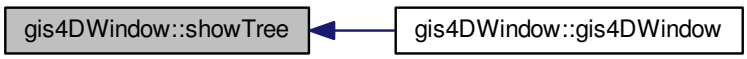
```
gis4DWindow::showTree (
            bool disp ) [slot]
```

Fonction pour afficher la donnée d'arbres.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :



7.12.2.24 showWater

```
gis4DWindow::showWater (
            bool disp ) [slot]
```

Fonction pour afficher la donnée de surface d'eau.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.25 showWireMode

```
gis4DWindow::showWireMode (
            bool disp ) [slot]
```

Fonction pour afficher le mode filaire.

Paramètres

disp	Booléen à vrai pour affichage

Voici le graphe des appelants de cette fonction :

7.12.2.26 zoomInMode

```
gis4DWindow::zoomInMode ( ) [slot]
```

Fonction gérant le zoom + sur la carte.

Voici le graphe des appelants de cette fonction :

7.12.2.27 zoomOutMode

```
gis4DWindow::zoomOutMode ( ) [slot]
```

Fonction gérant le zoom - sur la carte.

Voici le graphe des appelants de cette fonction :

La documentation de cette classe a été générée à partir des fichiers suivants :

- gis4dwindow.h
- gis4dwindow.cpp

7.13 Référence de la classe globalvariables

```
#include <globalvariables.h>
```

Fonctions membres publiques

- globalvariables ()

7.13.1 Documentation des constructeurs et destructeur

7.13.1.1 globalvariables()

```
globalvariables::globalvariables ( )
```

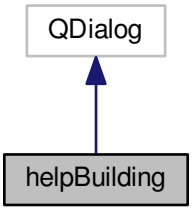
La documentation de cette classe a été générée à partir des fichiers suivants :

- globalvariables.h
- globalvariables.cpp

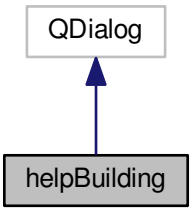
7.14 Référence de la classe helpBuilding

`#include <helpbuilding.h>`

Graphe d'héritage de helpBuilding :



Graphe de collaboration de helpBuilding :



Fonctions membres publiques

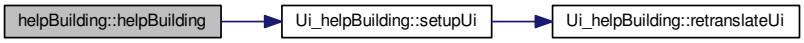
- helpBuilding (QWidget *parent=0)
- ~helpBuilding ()

7.14.1 Documentation des constructeurs et destructeur

7.14.1.1 helpBuilding()

```
helpBuilding::helpBuilding (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.14.1.2 ~helpBuilding()

`helpBuilding::~helpBuilding ( )`

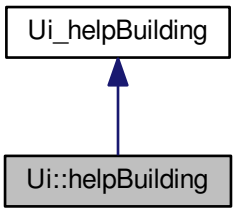
La documentation de cette classe a été générée à partir des fichiers suivants :

- helpbuilding.h
- helpbuilding.cpp

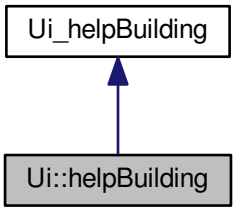
7.15 Référence de la classe Ui : :helpBuilding

`#include <ui_helpbuilding.h>`

Graphe d'héritage de Ui : :helpBuilding :



Graphe de collaboration de Ui : :helpBuilding :



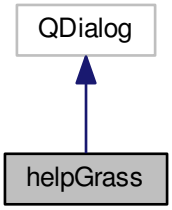
Membres hérités additionnels

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helpbuilding.h

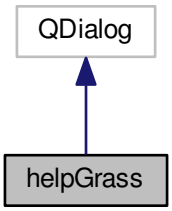
7.16 Référence de la classe helpGrass

`#include <helpgrass.h>`

Graphe d'héritage de helpGrass :



Graphe de collaboration de helpGrass :



Fonctions membres publiques

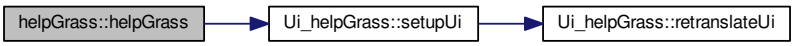
— helpGrass (QWidget *parent=0)
— ~helpGrass ()

7.16.1 Documentation des constructeurs et destructeur

7.16.1.1 helpGrass()

```
helpGrass::helpGrass (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.16.1.2 ~helpGrass()

```
helpGrass::~helpGrass ( )
```

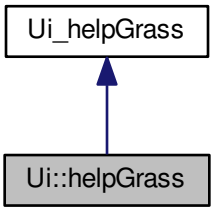
La documentation de cette classe a été générée à partir des fichiers suivants :

- helpgrass.h
- helpgrass.cpp

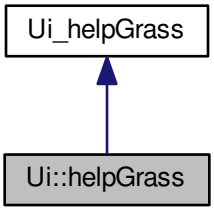
7.17 Référence de la classe Ui : :helpGrass

```
#include <ui_helpgrass.h>
```

Graphe d'héritage de Ui : :helpGrass :



Graphe de collaboration de Ui : :helpGrass :

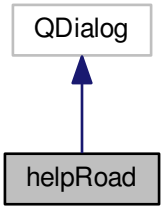


Membres hérités additionnels

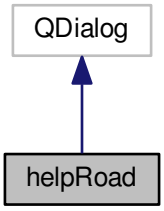
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helpgrass.h

7.18 Référence de la classe helpRoad

`#include <helproad.h>`
Graphe d'héritage de helpRoad :



Graphe de collaboration de helpRoad :



Fonctions membres publiques

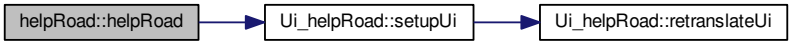
— helpRoad (QWidget *parent=0)
— ~helpRoad ()

7.18.1 Documentation des constructeurs et destructeur

7.18.1.1 helpRoad()

```
helpRoad::helpRoad (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.18.1.2 ~helpRoad()

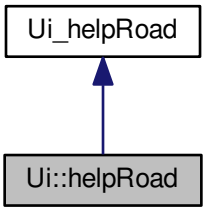
```
helpRoad::~helpRoad ( )
```

La documentation de cette classe a été générée à partir des fichiers suivants :
— helproad.h
— helproad.cpp

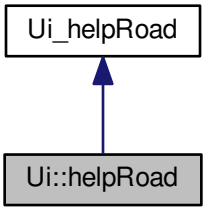
7.19 Référence de la classe Ui : :helpRoad

```
#include <ui_helproad.h>
```

Graphe d'héritage de Ui : :helpRoad :



Graphe de collaboration de Ui : :helpRoad :

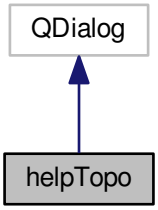


Membres hérités additionnels

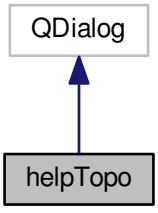
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helproad.h

7.20 Référence de la classe helpTopo

`#include <helptopo.h>`
Graphe d'héritage de helpTopo :



Graphe de collaboration de helpTopo :



Fonctions membres publiques

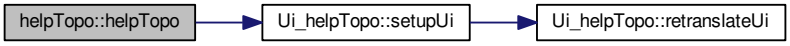
— helpTopo (QWidget *parent=0)
— ~helpTopo ()

7.20.1 Documentation des constructeurs et destructeur

7.20.1.1 helpTopo()

```
helpTopo::helpTopo (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.20.1.2 ~helpTopo()

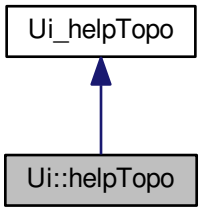
```
helpTopo::~helpTopo ( )
```

La documentation de cette classe a été générée à partir des fichiers suivants :
— helptopo.h
— helptopo.cpp

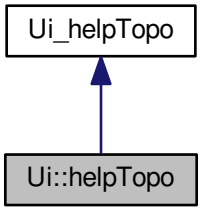
7.21 Référence de la classe Ui : :helpTopo

```
#include <ui_helptopo.h>
```

Graphe d'héritage de Ui : :helpTopo :



Graphe de collaboration de Ui : :helpTopo :

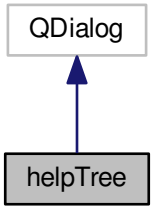


Membres hérités additionnels

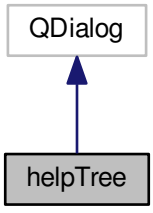
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helptopo.h

7.22 Référence de la classe helpTree

`#include <helptree.h>`
Graphe d'héritage de helpTree :



Graphe de collaboration de helpTree :



Fonctions membres publiques

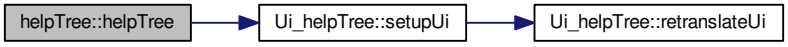
— helpTree (QWidget *parent=0)
— ~helpTree ()

7.22.1 Documentation des constructeurs et destructeur

7.22.1.1 helpTree()

```
helpTree::helpTree (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.22.1.2 ~helpTree()

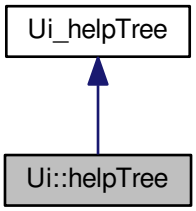
```
helpTree::~helpTree ( )
```

La documentation de cette classe a été générée à partir des fichiers suivants :
— helptree.h
— helptree.cpp

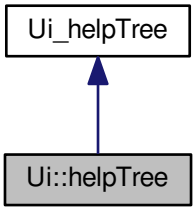
7.23 Référence de la classe Ui : :helpTree

```
#include <ui_helptree.h>
```

Graphe d'héritage de Ui : :helpTree :



Graphe de collaboration de Ui : :helpTree :

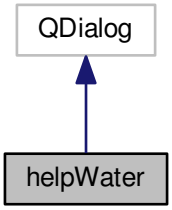


Membres hérités additionnels

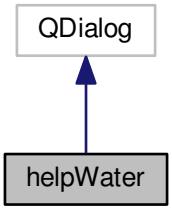
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helptree.h

7.24 Référence de la classe helpWater

`#include <helpwater.h>`
Graphe d'héritage de helpWater :



Graphe de collaboration de helpWater :



Fonctions membres publiques

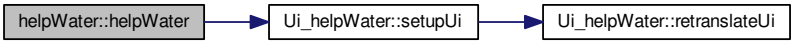
— helpWater (QWidget *parent=0)
— ~helpWater ()

7.24.1 Documentation des constructeurs et destructeur

7.24.1.1 helpWater()

```
helpWater::helpWater (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.24.1.2 ~helpWater()

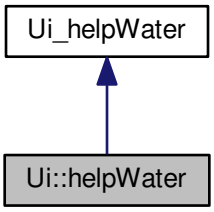
```
helpWater::~helpWater ( )
```

La documentation de cette classe a été générée à partir des fichiers suivants :
— helpwater.h
— helpwater.cpp

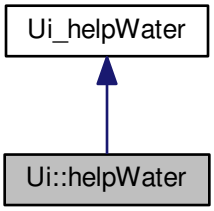
7.25 Référence de la classe Ui : :helpWater

```
#include <ui_helpwater.h>
```

Graphe d'héritage de Ui : :helpWater :



Graphe de collaboration de Ui : :helpWater :

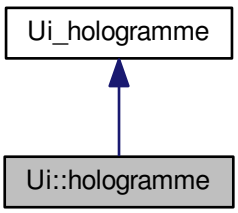


Membres hérités additionnels

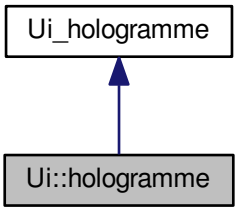
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helpwater.h

7.26 Référence de la classe Ui : :hologramme

`#include <ui_hologramme.h>`
Graphe d'héritage de Ui : :hologramme :



Graphe de collaboration de Ui : :hologramme :



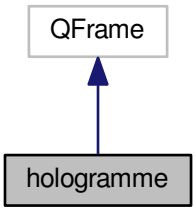
Membres hérités additionnels

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_hologramme.h

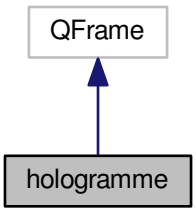
7.27 Référence de la classe hologramme

`#include <hologramme.h>`

Graphe d'héritage de hologramme :



Graphe de collaboration de hologramme :



Fonctions membres publiques

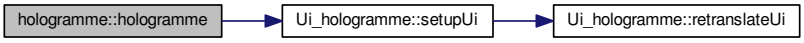
— hologramme (QWidget *parent=0)
— ~hologramme ()

7.27.1 Documentation des constructeurs et destructeur

7.27.1.1 hologramme()

```
hologramme::hologramme (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.27.1.2 ~hologramme()

```
hologramme::~hologramme ( )
```

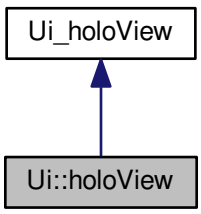
La documentation de cette classe a été générée à partir des fichiers suivants :

- hologramme.h
- hologramme.cpp

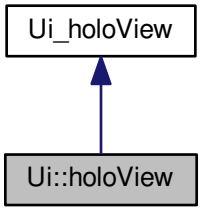
7.28 Référence de la classe Ui : :holoView

```
#include <ui_holoview.h>
```

Graphe d'héritage de Ui : :holoView :



Graphe de collaboration de Ui : :holoView :

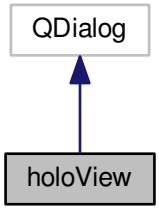


Membres hérités additionnels

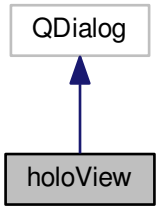
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_holoview.h

7.29 Référence de la classe holoView

`#include <holoview.h>`
Graphe d'héritage de holoView :



Graphe de collaboration de holoView :



Fonctions membres publiques

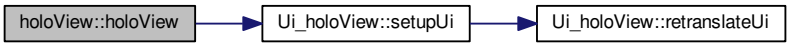
— holoView (QWidget *parent=0)
— ~holoView ()

7.29.1 Documentation des constructeurs et destructeur

7.29.1.1 holoView()

```
holoView::holoView (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



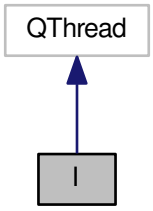
7.29.1.2 ~holoView()

```
holoView::~holoView ( )
```

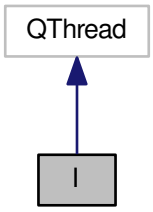
La documentation de cette classe a été générée à partir des fichiers suivants :
— holoview.h
— holoview.cpp

7.30 Référence de la classe I

Graphe d'héritage de I :



Graphe de collaboration de I :



Fonctions membres publiques statiques

— static void sleep (unsigned long secs)

7.30.1 Documentation des fonctions membres

7.30.1.1 sleep()

```
static void I::sleep (
            unsigned long secs ) [inline], [static]
```

Voici le graphe des appelants de cette fonction :

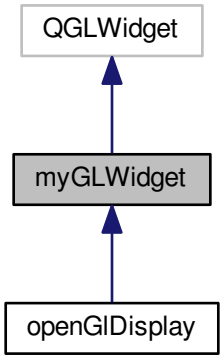
La documentation de cette classe a été générée à partir du fichier suivant :

— main.cpp

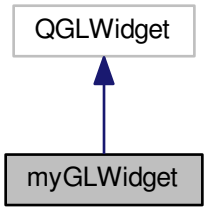
7.31 Référence de la classe myGLWidget

```
#include <myGLWidget.h>
```

Graphe d'héritage de myGLWidget :



Graphe de collaboration de myGLWidget :



Connecteurs publics

— virtual void timeOutSlot ()
Fonction pour mettre à jour le dessin.

Fonctions membres publiques

— myGLWidget (int framesPerSecond=0, QWidget *parent=0, char *name=0)
— virtual void initializeGL ()=0
Initialisation des paramètres OpenGl.
— virtual void resizeGL (int width, int height)=0
Initialisation des paramètres OpenGl.
— virtual void paintGL ()=0
La fonction draw contient tout les objets 3Dà dessiner.

7.31.1 Description détaillée

Cette classe initialise la vue 3D. OpenGl est activé une fois cette classe appellée.

7.31.2 Documentation des constructeurs et destructeur

7.31.2.1 myGLWidget()

```
myGLWidget::myGLWidget (
            int framesPerSecond = 0,
            QWidget * parent = 0,
            char * name = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :

7.31.3 Documentation des fonctions membres

7.31.3.1 initializeGL()

```
virtual void myGLWidget::initializeGL ( )  [pure virtual]
```

Initialisation des paramètres OpenGl.

Implémenté dans openGlDisplay.

7.31.3.2 paintGL()

```
virtual void myGLWidget::paintGL ( )  [pure virtual]
```

La fonction draw contient tout les objets 3D à dessiner.

Implémenté dans openGlDisplay.

7.31.3.3 resizeGL()

```
virtual void myGLWidget::resizeGL (
            int width,
            int height )  [pure virtual]
```

Initialisation des paramètres OpenGl.

Implémenté dans openGlDisplay.

7.31.3.4 timeOutSlot

```
void myGLWidget::timeOutSlot ( )  [virtual], [slot]
```

Fonction pour mettre à jour le dessin.

Voici le graphe des appelants de cette fonction :

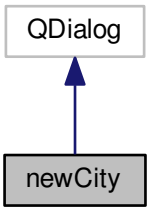
La documentation de cette classe a été générée à partir des fichiers suivants :

- myGLWidget.h
- myGLWidget.cpp

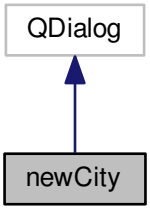
7.32 Référence de la classe newCity

```
#include <newcity.h>
```

Graphe d'héritage de newCity :



Graphe de collaboration de newCity :



Connecteurs publics

- void showBuildingUpload ()
  Affiche la fenêtre d'import des bâtiments.
- void showRoadUpload ()
  Affiche la fenêtre d'import des routes.
- void showTreeUpload ()
  Affiche la fenêtre d'import des arbres.
- void showGrassUpload ()
  Affiche la fenêtre d'import des zones herbacées.
- void showWaterUpload ()
  Affiche la fenêtre d'import des surfaces d'eau.
- void showTopoUpload ()
  Affiche la fenêtre d'import de la topographie.
- void showHelpBuilding ()
  Affiche la fenêtre d'aide sur l'import des bâtiments.
- void showHelpRoad ()
  Affiche la fenêtre d'aide sur l'import des routes.
- void showHelpTree ()
  Affiche la fenêtre d'aide sur l'import des arbres.
- void showHelpGrass ()
  Affiche la fenêtre d'aide sur l'import des zones herbacées.
- void showHelpWater ()
  Affiche la fenêtre d'aide sur l'import des surfaces d'eau.
- void showHelpTopo ()
  Affiche la fenêtre d'aide sur l'import de la topographie.
- void okButton ()

Fonctions membres publiques

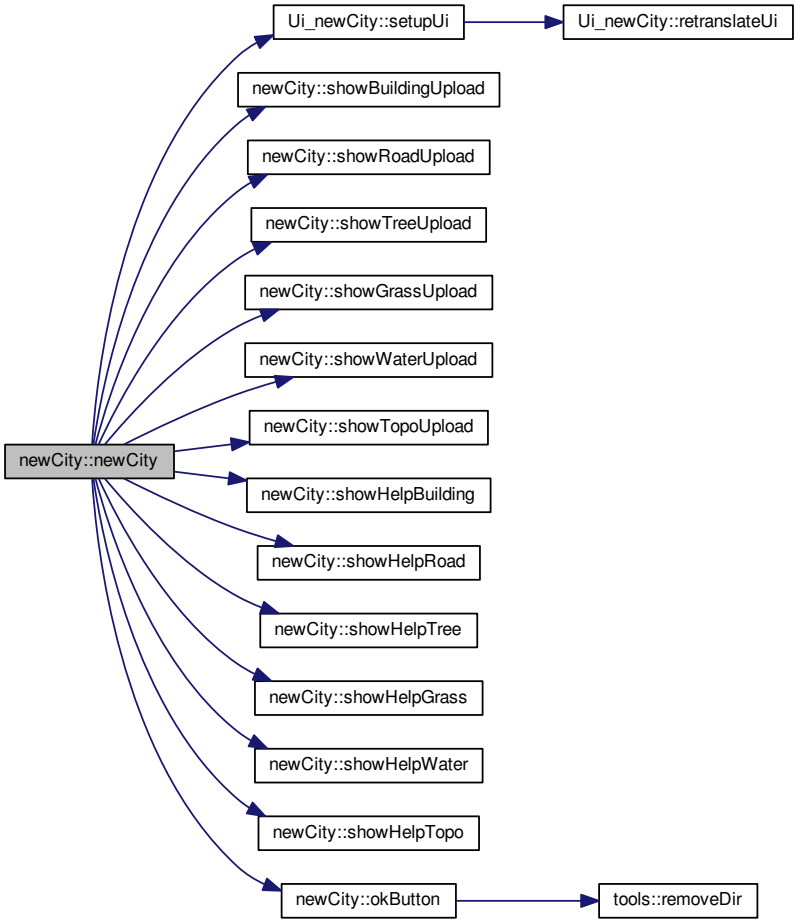
- newCity (QWidget *parent=0)
- ~newCity ()

7.32.1 Documentation des constructeurs et destructeur

7.32.1.1 newCity()

```
newCity::newCity (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :



7.32.1.2 ~newCity()

```
newCity::~newCity ( )
```

7.32.2 Documentation des fonctions membres

7.32.2.1 okButton

`void newCity::okButton ( ) [slot]`

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :

7.32.2.2 showBuildingUpload

`void newCity::showBuildingUpload ( ) [slot]`

Affiche la fenêtre d'import des bâtiments.

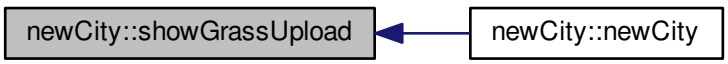
Voici le graphe des appelants de cette fonction :

7.32.2.3 showGrassUpload

`void newCity::showGrassUpload ( ) [slot]`

Affiche la fenêtre d'import des zones herbacées.

Voici le graphe des appelants de cette fonction :

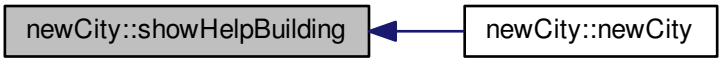


7.32.2.4 showHelpBuilding

`void newCity::showHelpBuilding ( ) [slot]`

Affiche la fenêtre d'aide sur l'import des bâtiments.

Voici le graphe des appelants de cette fonction :

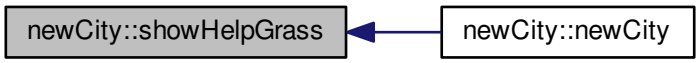


7.32.2.5 showHelpGrass

`void newCity::showHelpGrass ( ) [slot]`

Affiche la fenêtre d'aide sur l'import des zones herbacées.

Voici le graphe des appelants de cette fonction :

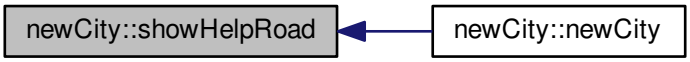


7.32.2.6 showHelpRoad

`void newCity::showHelpRoad ( ) [slot]`

Affiche la fenêtre d'aide sur l'import des routes.

Voici le graphe des appelants de cette fonction :

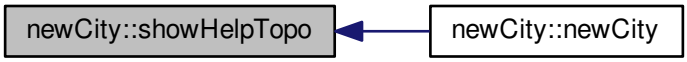


7.32.2.7 showHelpTopo

`void newCity::showHelpTopo ( ) [slot]`

Affiche la fenêtre d'aide sur l'import de la topographie.

Voici le graphe des appelants de cette fonction :

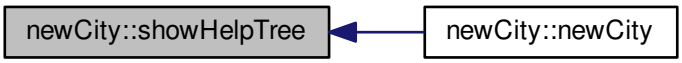


7.32.2.8 showHelpTree

`void newCity::showHelpTree ( ) [slot]`

Affiche la fenêtre d'aide sur l'import des arbres.

Voici le graphe des appelants de cette fonction :

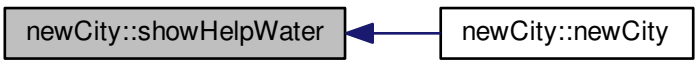


7.32.2.9 showHelpWater

`void newCity::showHelpWater ( ) [slot]`

Affiche la fenêtre d'aide sur l'import des surfaces d'eau.

Voici le graphe des appelants de cette fonction :



7.32.2.10 showRoadUpload

`void newCity::showRoadUpload ( ) [slot]`

Affiche la fenêtre d'import des routes.

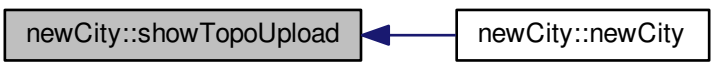
Voici le graphe des appelants de cette fonction :

7.32.2.11 showTopoUpload

`void newCity::showTopoUpload ( ) [slot]`

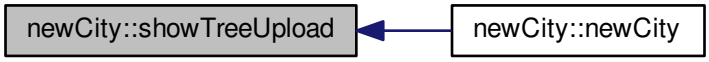
Affiche la fenêtre d'import de la topographie.

Voici le graphe des appelants de cette fonction :



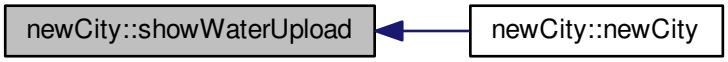
7.32.2.12 showTreeUpload

`void newCity::showTreeUpload ( ) [slot]`
Affiche la fenêtre d'import des arbres.
Voici le graphe des appelants de cette fonction :



7.32.2.13 showWaterUpload

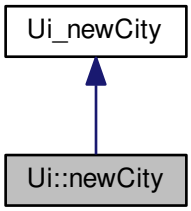
`void newCity::showWaterUpload ( ) [slot]`
Affiche la fenêtre d'import des surfaces d'eau.
Voici le graphe des appelants de cette fonction :



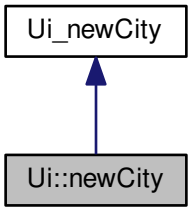
La documentation de cette classe a été générée à partir des fichiers suivants :
- newcity.h
- newcity.cpp

7.33 Référence de la classe Ui : :newCity

`#include <ui_newcity.h>`
Graphe d'héritage de Ui : :newCity :



Graphe de collaboration de Ui : :newCity :



Membres hérités additionnels

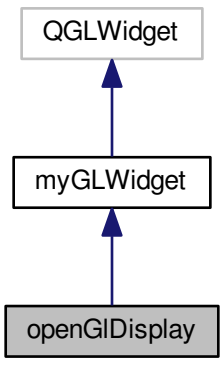
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_newcity.h

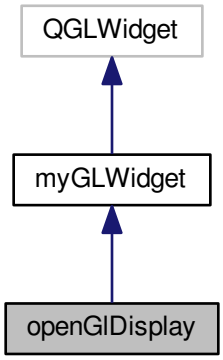
7.34 Référence de la classe openGlDisplay

`#include <opengldisplay.h>`

Graphe d'héritage de openGlDisplay :



Graphe de collaboration de openGlDisplay :



Classes

— struct quaternion
Déclaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacements d'objets, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en terme de calculs.

Fonctions membres publiques

— openGlDisplay (int view, QWidget *parent=0)
— double lengthVect (QVector3D vect)
Calcul d'une distance Elle est calculée selon cette formule :
$\sqrt{x^2 + y^2 + z^2}$.
— QVector3D normalizeVect (QVector3D vect)
Normalisation d'un vecteur Pour normaliser un vecteur, nous divisons chacun de ses composantes par sa longueur.
— QVector3D Difference (QVector3D vect1, QVector3D vect2)
Soustraction entre deux vecteurs. Chacune des composantes du vecteur est soustrait aux composantes d'un autre.
— QVector3D CrossProduct (QVector3D vect1, QVector3D vect2)
Quand a et b commencent à un point d'origine (0, 0, 0), le produit terminera à : cx = aybz - azby cy = azbx - axbz cz = axby - aybx La fonction pour calcule un produit de vecteur :
Pour la composante x : $x = (y1 * z2 - z1 * y2)$
Pour la composante y : $y = (z1 * x2 - x1 * z2)$
Pour la composante z : $z = (x1 * y2 - y1 * x2)$
.
— double length (quaternion quat)
Fonction pour avoir la longueur d'un quaternion On commence par calculer la longueur d'un quaternion pour ensuite le normaliser, ce qui est possible grâce la formule de la distance Euclidienne :
$|Q| = \sqrt{x^2 + y^2 + z^2 + w^2}$.
— quaternion normalizeQuat (quaternion quat)
— quaternion conjugate (quaternion quat)

Fonction pour avoir le conjugué d'un quaternion Pour avoir le conjugué d'un quaternion, on prend son opposé :
$Q' = [w, -[b]v[/b]]$.

— quaternion mult (quaternion A, quaternion B)
La multiplication d'un quaternion A par un quaternion B est faite selon :
$C = A * B$
Ce qui donne :
$C.x = A.w * B.x + A.x * B.w + A.y * B.z - A.z * B.y$
$C.y = A.w * B.y - A.x * B.z + A.y * B.w + A.z * B.x$
$C.z = A.w * B.z + A.x * B.y - A.y * B.x + A.z * B.w$
$C.w = A.w * B.w - A.x * B.x - A.y * B.y - A.z * B.z$
.

— QVector3D quatRotate (QVector3D v, quaternion q)
Rotation d'un quaternion depuis un vecteur. Création d'un nouveau quaternion avec un quaternion tampon avec le vecteur (la composante w est remplit à 0). Ensuite, on multiplie ce quaternion avec le conjugué du quaternion initial.

— openGlDisplay (QWidget *parent=0)

— void getData ()
Toutes les données d'import sont gérées via cette méthode. Cette méthode est appelée au chargement de la classe.

— void GestionLumiere ()
Gestion de la lumière (soleil)

— void GestionCamera (int numCamera)

— void AfficherTopographie ()
Affichage des triangles de topographie.

— void AfficherTopographieFilaire ()
Affichage de la topographie en mode filaire.

— void AfficherRoutes ()

— void AfficherRoutesFilaire ()

— void AfficherBatiments ()
Affichage des triangles de bâtiments.

— void AfficherSkyBox ()
Affichage de la skybox.

— void AfficherObj3D ()
Affichage des objets 3D.

— void AfficherTrajets ()

— void ChargerObjetsVoitures ()
Charge les objets voitures.

— void AfficherArbres ()

— void AfficherMarquage ()
Permet d'afficher les marquages au sol.

— void initializeGL ()
Initialisation des variables.

— void resizeGL (int width, int height)
Taille de la fenêtre OpenGl.

— void paintGL ()
Cette méthode est appelée à chaque image (24 fois par seconde). Toute la scène 3D est gérée ici. C'est là que la méthode PlaceSceneElements() est appelée.

— void PlaceSceneElements ()
Tous les triangles sont affichés ici.

— void toggleFullWindow ()
Méthode pour afficher la vue 3D en plein écran ou non.

— void updateView ()

— void ChangePitch (GLfloat degrees)
Change le tangage de la caméra (pitch) selon les mouvements de la sourirs.

— void ChangeHeading (GLfloat degrees)
Change l'angle d'inclinaison de la caméra (heading) selon les mouvements de la sourirs.

— void Move2D (int x, int y)
Déplacement de la caméra.
— const aiScene * ChargerOjb3D (QString chemin)

Fonctions membres publiques statiques

— static quaternion Q_from_AngAxis (double angle, QVector3D axis)
Fonction qui permet de passer d'un angle d'Euler en quaternion Les angles doivent être en degré pour ce calcul. La composante w d'un quaternion est calculée avec le cosinus de la moitié de l'angle
*La composante x d'un quaternion est calculée avec la composante x du vecteur * le sinus de l'angle*
*La composante y d'un quaternion est calculée avec la composante y du vecteur * le sinus de l'angle*
*La composante z d'un quaternion est calculée avec la composante z du vecteur * le sinus de l'angle*
.

Fonctions membres protégées

— void mouseMoveEvent (QMouseEvent *event)
Mouvements de la souris.
— void keyPressEvent (QKeyEvent *)
Evénements clavier.

Membres hérités additionnels

7.34.1 Description détaillée

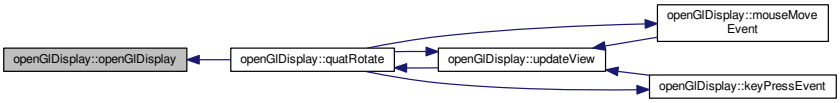
Cette calsse gère toute la partie 3D. on trouvera le chargement de la donnée, sa conversion, son affichage et la gestion de l'animation 4D.

7.34.2 Documentation des constructeurs et destructeur

7.34.2.1 openGlDisplay() [1/2]

```
openGlDisplay::openGlDisplay (
            int view,
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe des appelants de cette fonction :



7.34.2.2 openGlDisplay() [2/2]

```
openGlDisplay::openGlDisplay (
            QWidget * parent = 0 )  [explicit]
```

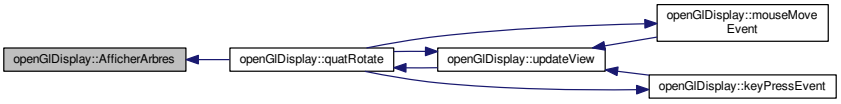
7.34.3 Documentation des fonctions membres

7.34.3.1 AfficherArbres()

```
void openGlDisplay::AfficherArbres ( )
```

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



7.34.3.2 AfficherBatiments()

```
openGlDisplay::AfficherBatiments ( )
```

Affichage des triangles de bâtiments.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :

7.34.3.3 AfficherMarquage()

`openGlDisplay::AfficherMarquage ( )`

Permet d'afficher les marquages au sol.

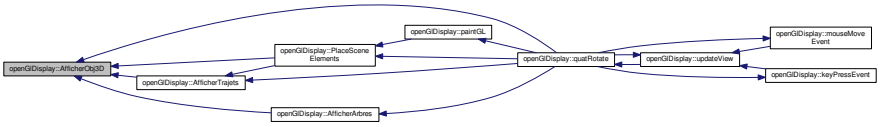
Voici le graphe des appelants de cette fonction :

7.34.3.4 AfficherObj3D()

`openGlDisplay::AfficherObj3D ( )`

Affichage des objets 3D.

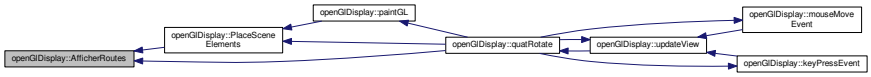
Voici le graphe des appelants de cette fonction :



7.34.3.5 AfficherRoutes()

`void openGlDisplay::AfficherRoutes ( )`

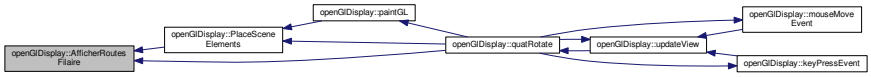
Voici le graphe des appelants de cette fonction :



7.34.3.6 AfficherRoutesFilaire()

`void openGlDisplay::AfficherRoutesFilaire ( )`

Voici le graphe des appelants de cette fonction :



7.34.3.7 AfficherSkyBox()

`openGlDisplay::AfficherSkyBox ( )`

Affichage de l if(tools : :ComparaisonCoordonnees(positionVoiture,coordPoint2))a skybox.

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :

7.34.3.8 AfficherTopographie()

`openGlDisplay::AfficherTopographie ( )`

Affichage des triangles de topographie.

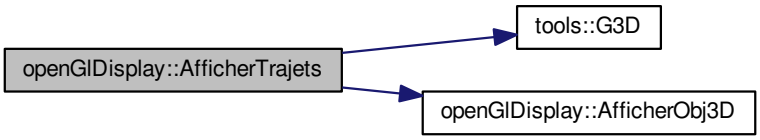
Voici le graphe des appelants de cette fonction :

7.34.3.9 AfficherTopographieFilaire()

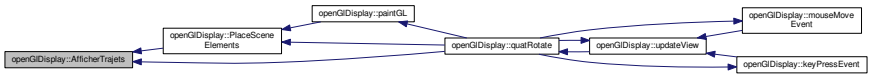
`openGlDisplay::AfficherTopographieFilaire ( )`
Affichage de la topographie en mode filaire.
Voici le graphe des appelants de cette fonction :

7.34.3.10 AfficherTrajets()

`void openGlDisplay::AfficherTrajets ( )`
Voici le graphe d'appel pour cette fonction :



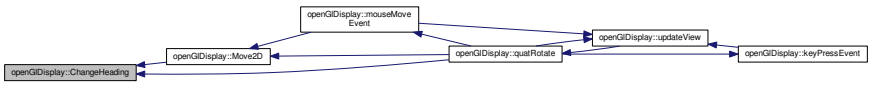
Voici le graphe des appelants de cette fonction :



7.34.3.11 ChangeHeading()

```
openGlDisplay::ChangeHeading (
            GLfloat degrees )
```

Change l'angle d'inclinaison de la caméra (heading) selon les mouvements de la sourirs.
Voici le graphe des appelants de cette fonction :

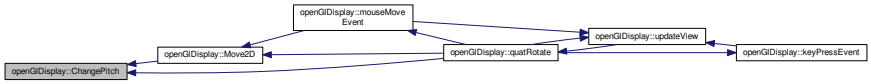


7.34.3.12 ChangePitch()

```
openGlDisplay::ChangePitch (
            GLfloat degrees )
```

Change le tangage de la caméra (pitch) selon les mouvements de la sourirs.

Voici le graphe des appelants de cette fonction :

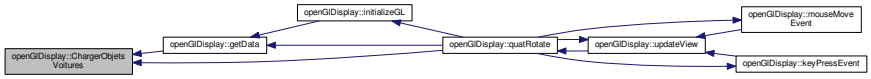


7.34.3.13 ChargerObjetsVoitures()

```
openGlDisplay::ChargerObjetsVoitures ( )
```

Charge les objets voitures.

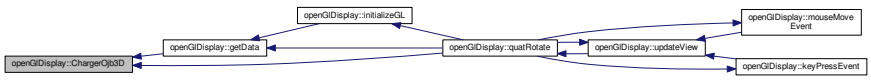
Voici le graphe des appelants de cette fonction :



7.34.3.14 ChargerOjb3D()

```
const aiScene * openGlDisplay::ChargerOjb3D (
            QString chemin )
```

Voici le graphe des appelants de cette fonction :



7.34.3.15 conjugate()

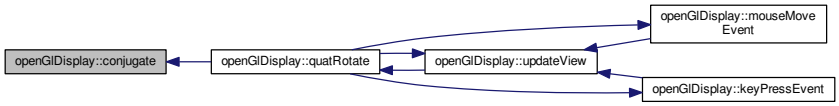
```
openGlDisplay::conjugate (
            quaternion quat ) [inline]
```

Fonction pour avoir le conjugué d'un quaternion Pour avoir le conjugé d'un quaternion, on prend simplement son opposé :
$Q' = [w, -[b]v[/b]]$.

Renvoie

Conjugué d'un quaternion

Voici le graphe des appelants de cette fonction :



7.34.3.16 CrossProduct()

```
openGlDisplay::CrossProduct (
            QVector3D vect1,
            QVector3D vect2 ) [inline]
```

Quand a et b commence à un point d'origine (0, 0, 0), le produit terminera à : cx = aybz - azby cy = azbx - axbz cz = axby - aybx La fonction pour calcule un produit de vecteur :
Pour la composante x : $x = (y1 * z2 - z1 * y2)$
Pour la composante y : $y = (z1 * x2 - x1 * z2)$
Pour la composante z : $z = (x1 * y2 - y1 * x2)$
.

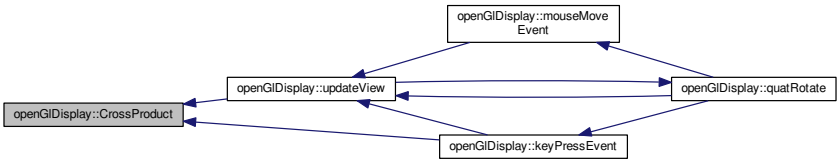
Paramètres

vect1	Premier vecteur
vect2	Deuxième vecteur

Renvoie

Produit des vecteurs 1 et 2

Voici le graphe des appelants de cette fonction :



7.34.3.17 Difference()

```
openGlDisplay::Difference (
            QVector3D vect1,
            QVector3D vect2 )  [inline]
```

Soustraction entre deux vecteurs. Chacun des composant du vecteur est soustrait aux composants d'un autre.

Paramètres

vect1	Premier vecteur.
vect2	Deuxième vecteur.

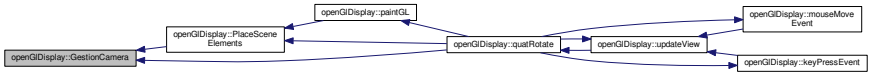
Renvoie

Soustraction du vecteur 2 par le vecteur 1.

7.34.3.18 GestionCamera()

```
void openGlDisplay::GestionCamera (
            int numCamera )
```

Voici le graphe des appelants de cette fonction :



7.34.3.19 GestionLumiere()

`openGlDisplay::GestionLumiere ( )`

Gestion de la lumière (soleil)

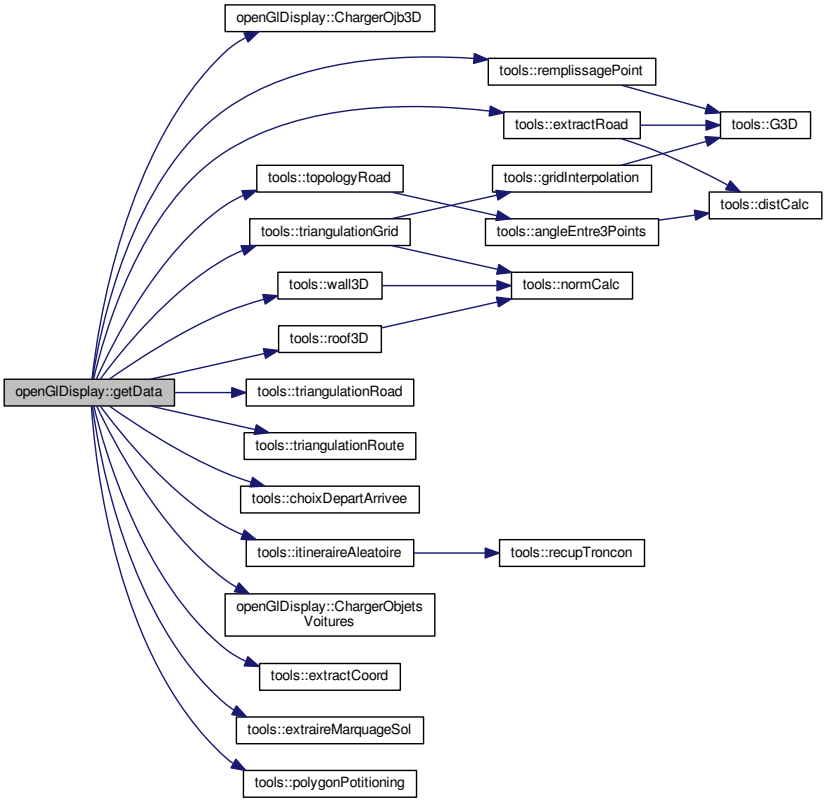
Voici le graphe des appelants de cette fonction :

7.34.3.20 getData()

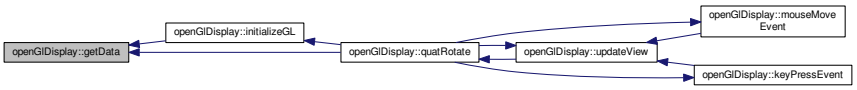
`openGlDisplay::getData ( )`

Toutes les données d'import sont gérées via cette méthode. Cette méthode est appelé au chargement de la classe.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



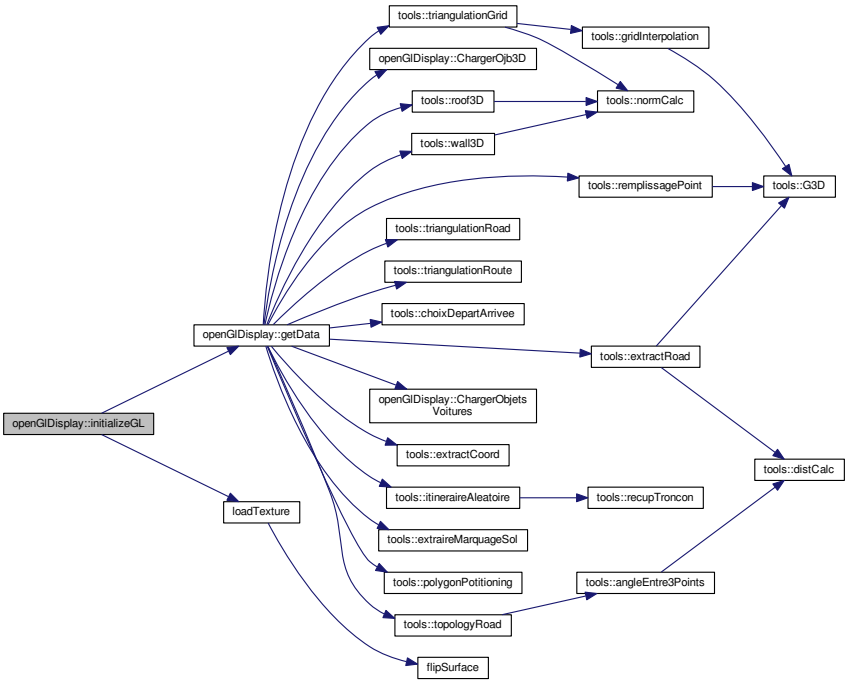
7.34.3.21 initializeGL()

`openGlDisplay::initializeGL ( ) [virtual]`

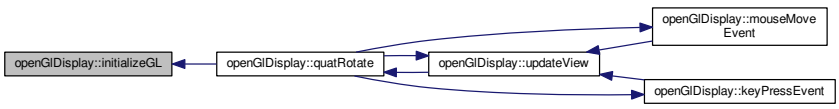
Initialisation des variables.

Implémente myGLWidget.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

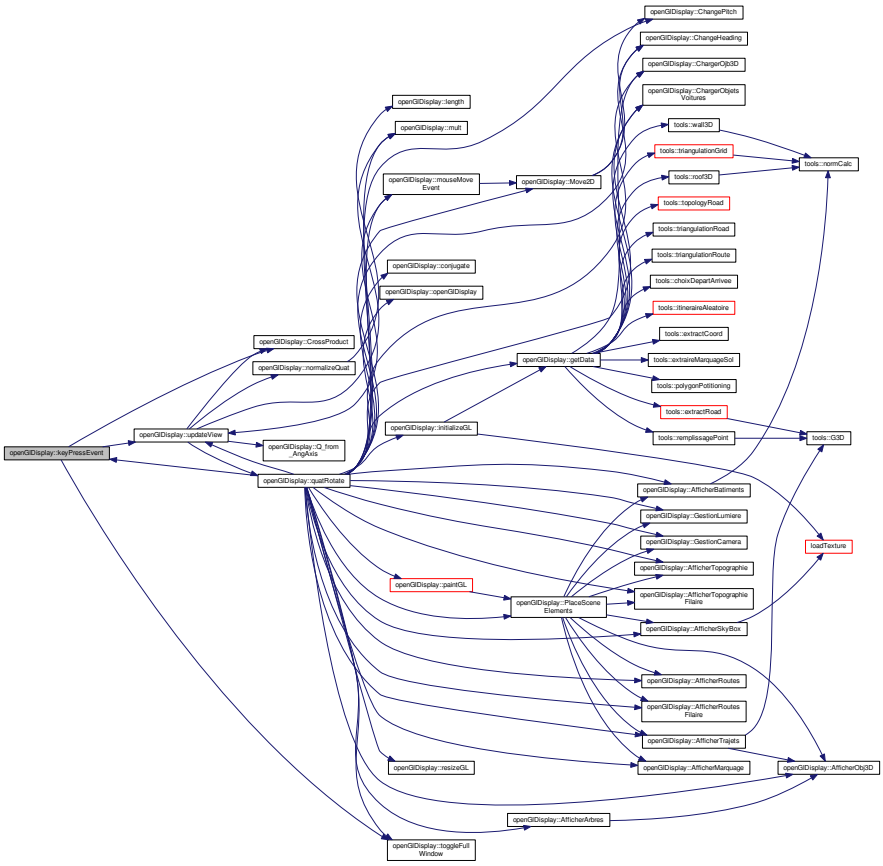


7.34.3.22 keyPressEvent()

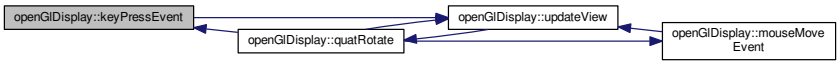
```
void openGlDisplay::keyPressEvent (
            QKeyEvent * keyEvent ) [protected]
```

Evénements clavier.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.34.3.23 length()

```
openGlDisplay::length (
            quaternion quat ) [inline]
```

Fonction pour avoir la longeur d'un quaternion On commence par calculer la longeur d'un quaternion pour ensuite le normaliser, ce qui est possible grâce la forumle de la distance Euclidienne

:
$|Q| = \sqrt{x^2 + y^2 + z^2 + w^2}$.

Fonction pour normaliser un quaternion Normaliser un quaternio est similaire à la normalisation d'un vecteur, on divise chaque composante par la longeur.

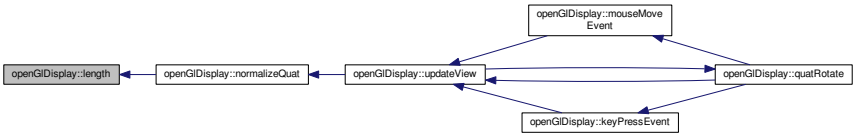
Paramètres

quat	Quaternion

Renvoie

Longeur du quaternion
Quaternion normalisé

Voici le graphe des appelants de cette fonction :



7.34.3.24 lengthVect()

```
openGlDisplay::lengthVect (
            QVector3D vect ) [inline]
```

Calcul d'une distance Elle est calculée selon cette formule :
$\sqrt{x^2 + y^2 + z^2}$.

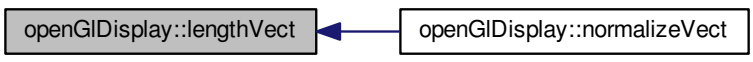
Paramètres

vect	Vecteur 3D.

Renvoie

Distance

Voici le graphe des appelants de cette fonction :

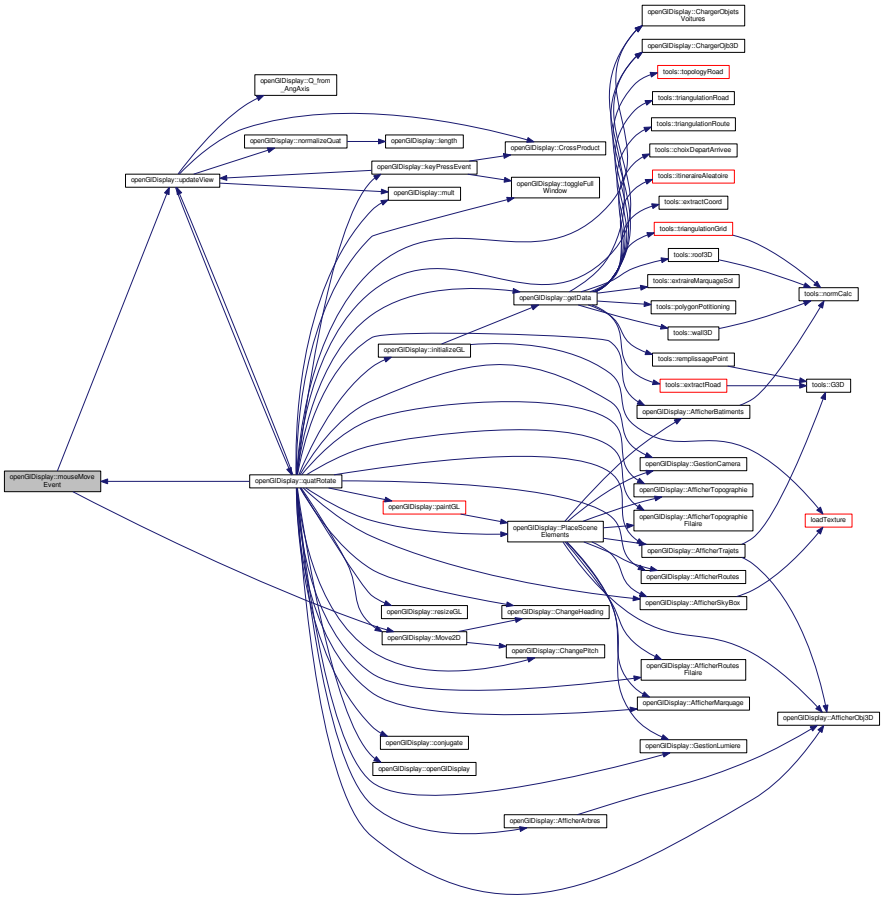


7.34.3.25 mouseMoveEvent()

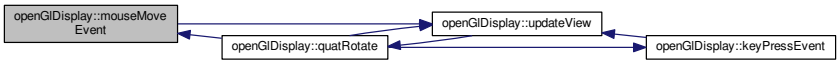
```
void openGlDisplay::mouseMoveEvent (
            QMouseEvent * event ) [protected]
```

Mouvements de la souris.

Voici le graphe d'appel pour cette fonction :



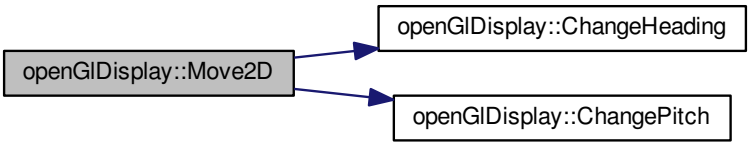
Voici le graphe des appelants de cette fonction :



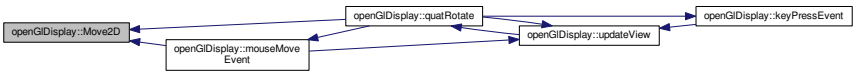
7.34.3.26 Move2D()

```
openGlDisplay::Move2D (
            int x,
            int y )
```

Déplacement de la caméra.
Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



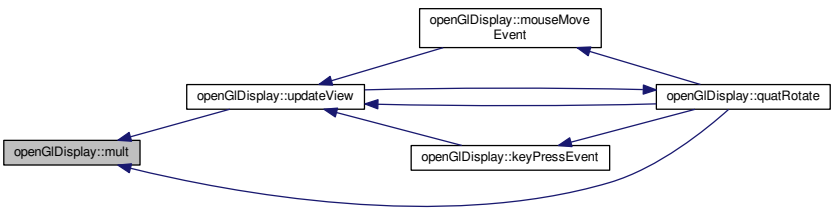
7.34.3.27 mult()

```
openGlDisplay::mult (
            quaternion A,
            quaternion B ) [inline]
```

La multiplication d'un quaternion A par un quaternion B est faite selon :
$C = A * B$
Ce qui donne :
$C.x = A.w * B.x + A.x * B.w + A.y * B.z - A.z * B.y$
$C.y = A.w * B.y - A.x * B.z + A.y * B.w + A.z * B.x$
$C.z = A.w * B.z + A.x * B.y - A.y * B.x + A.z * B.w$
$C.w = A.w * B.w - A.x * B.x - A.y * B.y - A.z * B.z$
.
Renvoie

Quaternion

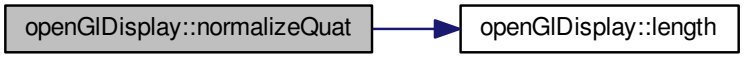
Voici le graphe des appelants de cette fonction :



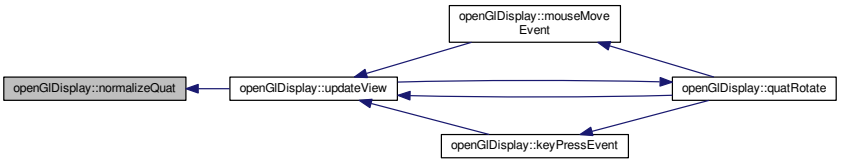
7.34.3.28 normalizeQuat()

```
quaternion openGlDisplay::normalizeQuat (
            quaternion quat ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.34.3.29 normalizeVect()

```
openGlDisplay::normalizeVect (
            QVector3D vect ) [inline]
```

Normalisataion d'un vecteur Pour normaliser un vecteur, nous divisons chacun de ses composantes par sa longeur.

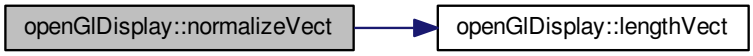
Paramètres

vect	Vecteur 3D

Renvoie

Vecteur normalisé

Voici le graphe d'appel pour cette fonction :



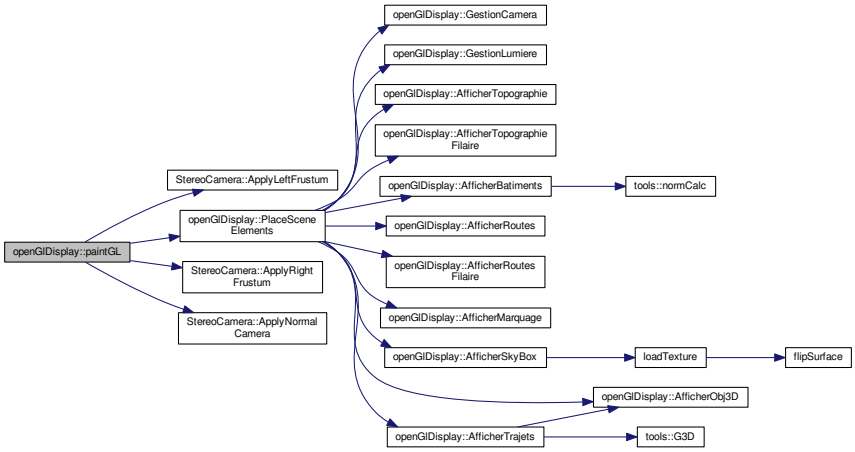
7.34.3.30 paintGL()

`openGlDisplay::paintGL ( ) [virtual]`

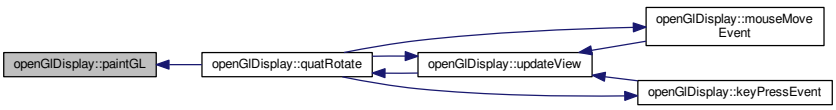
Cette méthode est appelée à toute les images par seconde. Toute la scene 3D est gérée ici. C'est là que la méthode PlaceSceneElements() est appelée.

Implémente myGLWidget.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

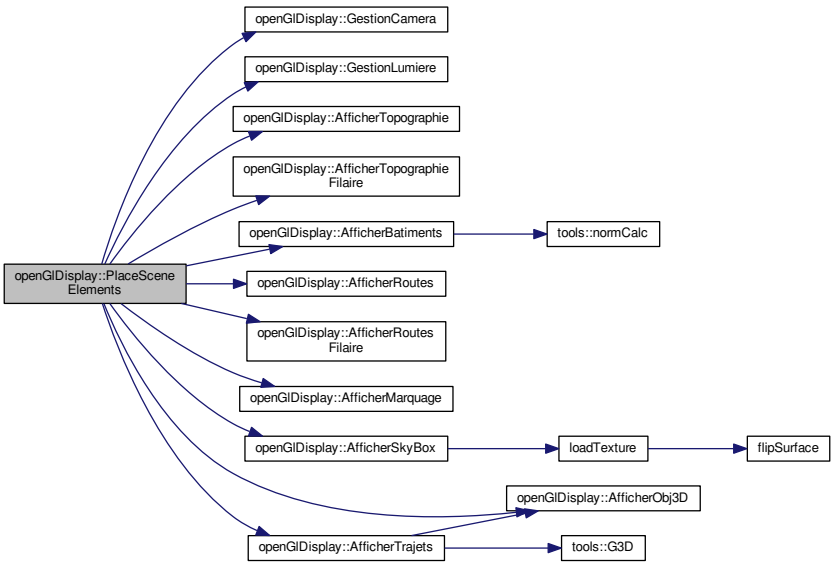


7.34.3.31 PlaceSceneElements()

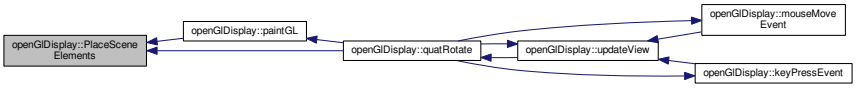
`openGlDisplay::PlaceSceneElements ( )`

Tous les triangles sont affichés ici.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.34.3.32 Q_from_AngAxis()

```
openGlDisplay::Q_from_AngAxis (
            double angle,
            QVector3D axis ) [inline], [static]
```

Fonction qui permet de passer d'un angle d'Euler en quaternion Les angles doivent être en degré pour ce calcul. Le composant w d'un quaternion est calculé avec le cosinus de la moitié de l'angle

Le composant x d'un quaternion est calculé avec la composante x du vecteur $*$ le sinus de l'angle

Le composant y d'un quaternion est calculé avec la composante y du vecteur $*$ le sinus de l'angle

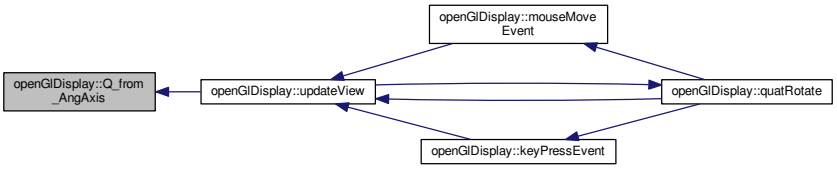
Le composant z d'un quaternion est calculé avec la composante z du vecteur $*$ le sinus de l'angle

.

Renvoie

Quaternion

Voici le graphe des appelants de cette fonction :



7.34.3.33 quatRotate()

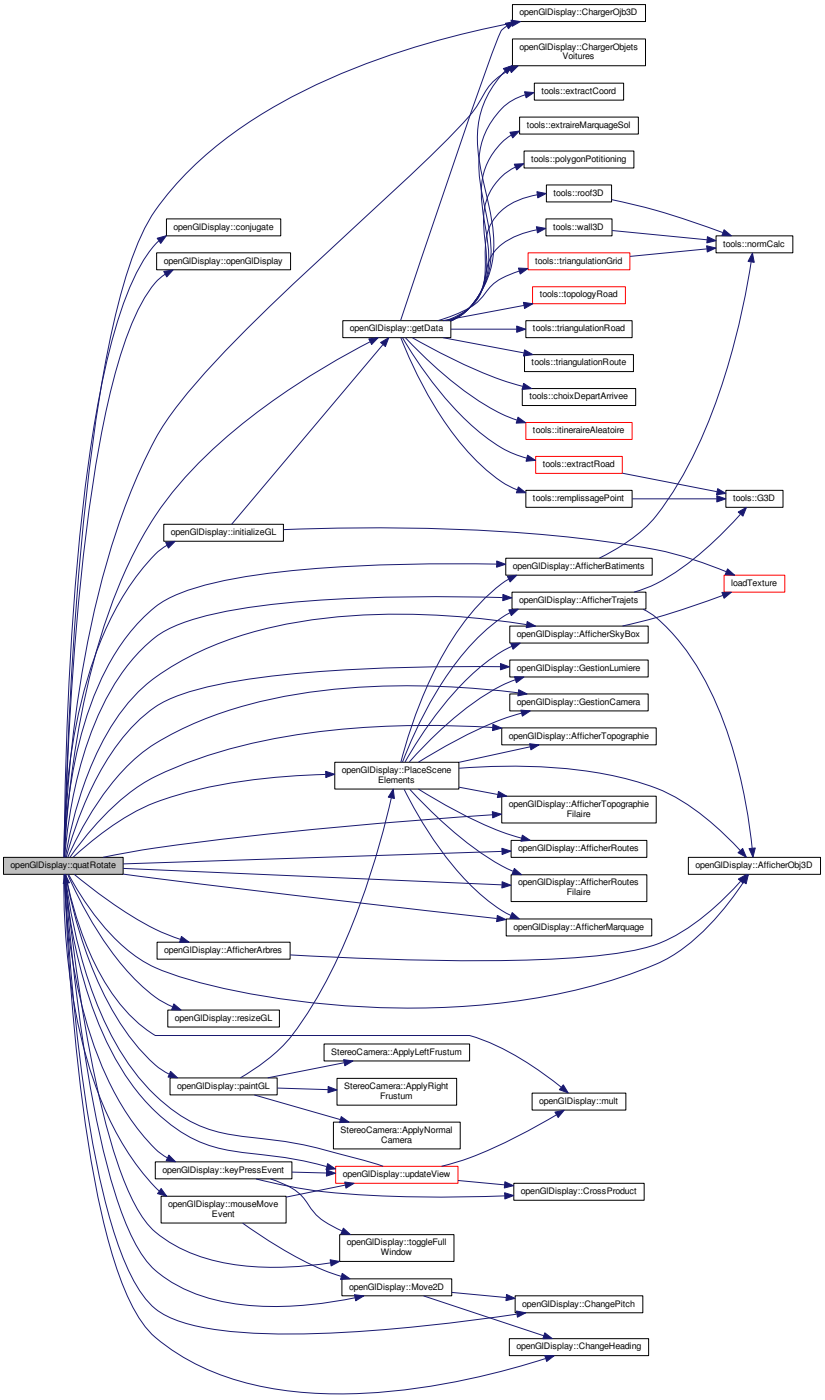
```
openGlDisplay::quatRotate (
            QVector3D v,
            quaternion q )  [inline]
```

Rotation d'un quaternion depuis un vecteur. Création d'un nouveau quaternion avec un quaternion tampon avec le vecteur (la composante w est remplit à 0). Ensuite, on multiplie ce quaternion avec le congugé du quaternion initial.

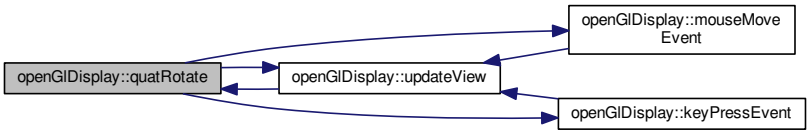
Renvoie

Le vecteur correspondant à la rotation (quaternion sans la composante w)

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.34.3.34 resizeGL()

```
openGlDisplay::resizeGL (
            int width,
            int height ) [virtual]
```

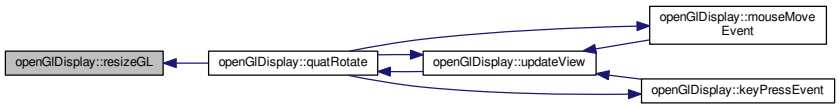
Taille de la fenêtre OpenGl.

Paramètres

width	Largeur de la fenêtre.
height	Hauteur de la fenêtre.

Implémente myGLWidget.

Voici le graphe des appelants de cette fonction :



7.34.3.35 toggleFullWindow()

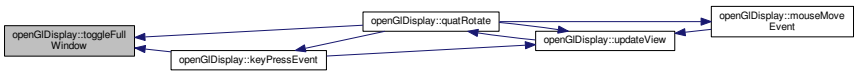
```
openGlDisplay::toggleFullWindow ( )
```

Méthode pour afficher la vue 3D en plein écran ou non.

Méthode appelée à chaque mouvement de souris Elle permet un rafraichissement (appel vers PlaceSceneElements())

Bogue Ne fonctionne pas pour l'instant

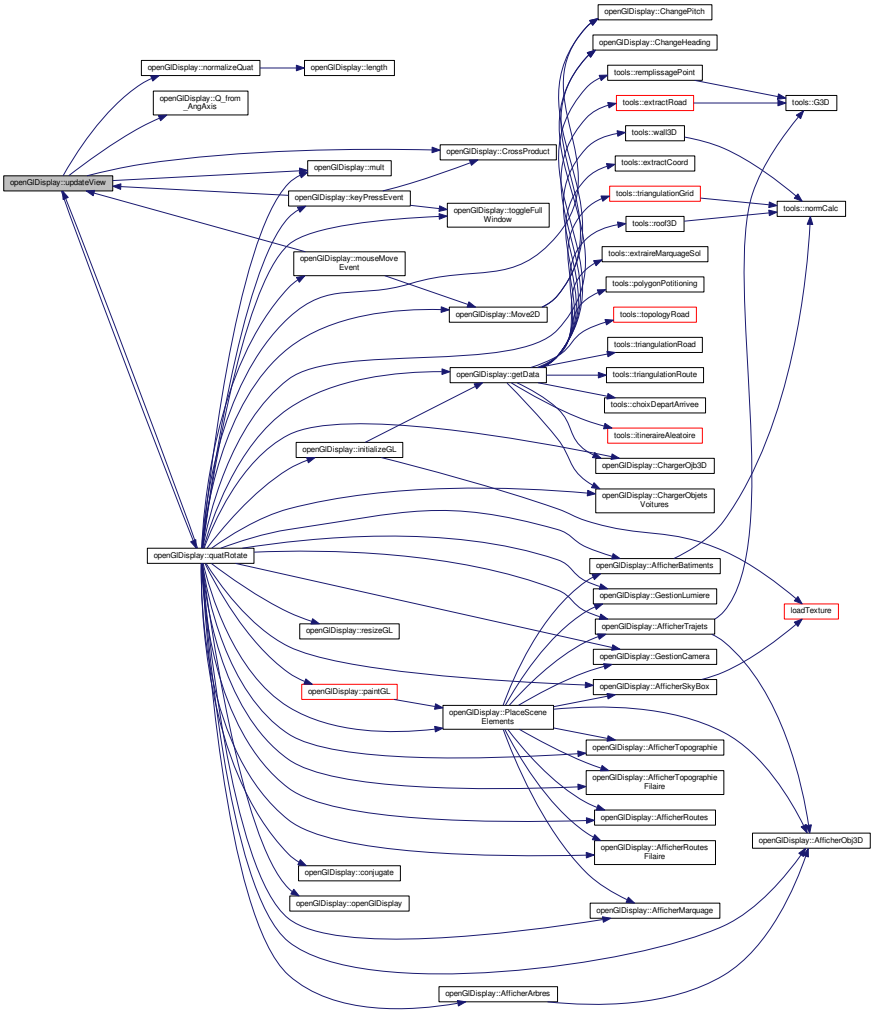
Voici le graphe des appelants de cette fonction :



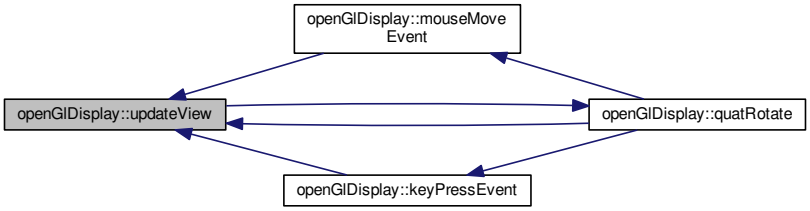
7.34.3.36 updateView()

```
void openGlDisplay::updateView ( )
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



La documentation de cette classe a été générée à partir des fichiers suivants :
- opengldisplay.h
- opengldisplay.cpp

7.35 Référence de la structure tools : :point3D

Coordonnées 3D d'un point. Les données sont des doubles.

`#include <tools.h>`

Attributs publics

- double x
- double y
- double z

7.35.1 Description détaillée

Coordonnées 3D d'un point. Les données sont des doubles.

7.35.2 Documentation des données membres

7.35.2.1 x

`double tools::point3D::x`

7.35.2.2 y

`double tools::point3D::y`

7.35.2.3 z

`double tools::point3D::z`

Coordonnées x, y et z d'un point.

La documentation de cette structure a été générée à partir du fichier suivant :
— tools.h

7.36 Référence de la structure tools : :pointInter

Point d'intersection Poibt d'intersection utilisé lors de la superposition entre la grille topo et un tronçon.

`#include <tools.h>`

Attributs publics

— double x
— double y
— bool intersection

7.36.1 Description détaillée

Point d'intersection utilisé lors de la superposition entre la grille topo et un tronçon.

7.36.2 Documentation des données membres

7.36.2.1 intersection

`bool tools::pointInter::intersection`

Booléen pour savoir s'il sagit bien d'une intersection.

7.36.2.2 x

`double tools::pointInter::x`

Position X.

7.36.2.3 y

`double tools::pointInter::y`

Position Y.

La documentation de cette structure a été générée à partir du fichier suivant :
— tools.h

7.37 Référence de la structure openGlDisplay : :quaternion

Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacements d'objets, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en terme de calculs.

`#include <opengldisplay.h>`

Attributs publics

- double x
- double y
- double z
- double w

7.37.1 Description détaillée

Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacements d'objets, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en terme de calculs.

Paramètres

x	Composante x
y	Composante y
z	Composante z
w	Composante w

7.37.2 Documentation des données membres

7.37.2.1 w

`double openGlDisplay::quaternion::w`

7.37.2.2 x

`double openGlDisplay::quaternion::x`

7.37.2.3 y

`double openGlDisplay::quaternion::y`

7.37.2.4 z

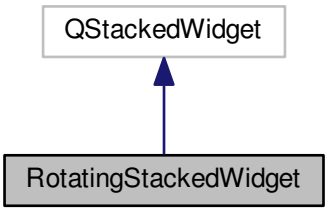
`double openGlDisplay::quaternion::z`

La documentation de cette structure a été générée à partir du fichier suivant :
— opengldisplay.h

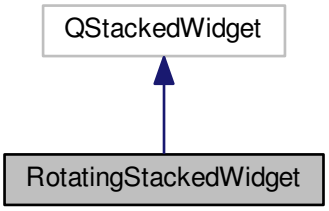
7.38 Référence de la classe RotatingStackedWidget

`#include <RotatingStackedWidget.h>`

Graphe d'héritage de RotatingStackedWidget :



Graphe de collaboration de RotatingStackedWidget :



Fonctions membres publiques

— RotatingStackedWidget (QWidget *parent=0)
— void paintEvent (QPaintEvent *)
— void rotate (int)
— float rotateVal ()
— void setRotateVal (float)

Propriétés

— float rotateVal

7.38.1 Documentation des constructeurs et destructeur

7.38.1.1 RotatingStackedWidget()

```
RotatingStackedWidget::RotatingStackedWidget (
            QWidget * parent = 0 )  [explicit]
```

7.38.2 Documentation des fonctions membres

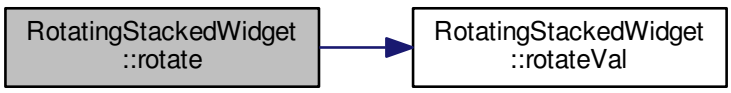
7.38.2.1 paintEvent()

```
void RotatingStackedWidget::paintEvent (
            QPaintEvent * event )
```

7.38.2.2 rotate()

```
void RotatingStackedWidget::rotate (
            int index )
```

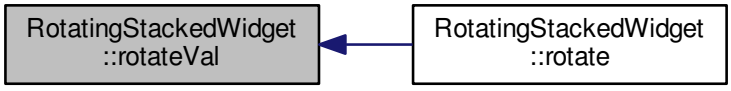
Voici le graphe d'appel pour cette fonction :



7.38.2.3 rotateVal()

```
float RotatingStackedWidget::rotateVal ( )
```

Voici le graphe des appelants de cette fonction :



7.38.2.4 setRotateVal()

```
void RotatingStackedWidget::setRotateVal (
            float fl )
```

7.38.3 Documentation des propriétés

7.38.3.1 rotateVal

```
float RotatingStackedWidget::rotateVal  [read], [write]
```

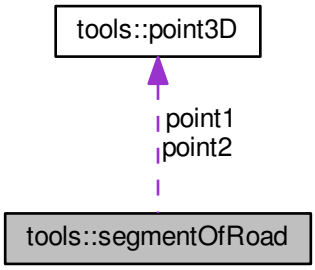
La documentation de cette classe a été générée à partir des fichiers suivants :
- RotatingStackedWidget.h
- RotatingStackedWidget.cpp

7.39 Référence de la structure tools : :segmentOfRoad

Définition d'un segment de route. Pour un segment de route, nous avons différents arguments pour le définir.

`#include <tools.h>`

Graphe de collaboration de tools : :segmentOfRoad :



Attributs publics

- QVector< int > previousSeg
- QVector< int > nextSeg
- point3D point1
- point3D point2
- bool uniqueWay
- QVector< double > previousAngle
- QVector< double > nextAngle
- int anglePrecedentUtilise = -1
- int angleSuivantUtilise = -1
- double width
- double distance
- double timeToDrive
- bool deleteMe
- bool newOne

7.39.1 Description détaillée

Définition d'un segment de route. Pour un segment de route, nous avons différents arguments pour le définir.

7.39.2 Documentation des données membres

7.39.2.1 anglePrecedentUtilise

```
int tools::segmentOfRoad::anglePrecedentUtilise = -1
```

Index de l'angle précedent utilisé pour calculer la triangulation.

7.39.2.2 angleSuivantUtilise

```
int tools::segmentOfRoad::angleSuivantUtilise = -1
```

Index de l'angle suivant utilisé pour calculer la triangulation.

7.39.2.3 deleteMe

```
bool tools::segmentOfRoad::deleteMe
```

Booléen pour savoir si ce tronçon doit être affiché.

7.39.2.4 distance

```
double tools::segmentOfRoad::distance
```

Longueur du tronçon.

7.39.2.5 newOne

```
bool tools::segmentOfRoad::newOne
```

Booléen pour savoir si ce tronçon est un tronçon créé ou pas.

7.39.2.6 nextAngle

```
QVector<double> tools::segmentOfRoad::nextAngle
```

Liste d'angles entre ce tronçon et les tronçons suivants.

7.39.2.7 nextSeg

```
QVector<int> tools::segmentOfRoad::nextSeg
```

Liste d'Id des tronçons suivants.

7.39.2.8 point1

point3D tools::segmentOfRoad::point1

7.39.2.9 point2

point3D tools::segmentOfRoad::point2

Coordonnées du tronçon.

7.39.2.10 previousAngle

QVector<double> tools::segmentOfRoad::previousAngle

Liste d'angles entre ce tronçon et les tronçons précédents.

7.39.2.11 previousSeg

QVector<int> tools::segmentOfRoad::previousSeg

Liste d'Id des tronçons précedents.

7.39.2.12 timeToDrive

double tools::segmentOfRoad::timeToDrive

Temps de circulation totale du tronçon.

7.39.2.13 uniqueWay

bool tools::segmentOfRoad::uniqueWay

Booléen pour savoir si le tronçon est à sens unique

7.39.2.14 width

double tools::segmentOfRoad::width

Largeur de la route

La documentation de cette structure a été générée à partir du fichier suivant :

— tools.h

7.40 Référence de la classe StereoCamera

#include <stereocamera.h>

Fonctions membres publiques

— StereoCamera (float Convergence, float EyeSeparation, float AspectRatio, float FOV, float NearClippingDistance, float FarClippingDistance)
— void ApplyLeftFrustum ()
— void ApplyRightFrustum ()
— void ApplyNormalCamera ()

7.40.1 Description détaillée

La stéréoscopie est défiine par la convergence, la séparation des yeux, le ratio, la distance entre l'objectif et le fond de l'affichahe et la focale

7.40.2 Documentation des constructeurs et destructeur

7.40.2.1 StereoCamera()

```
StereoCamera::StereoCamera (
            float Convergence,
            float EyeSeparation,
            float AspectRatio,
            float FOV,
            float NearClippingDistance,
            float FarClippingDistance ) [inline]
```

7.40.3 Documentation des fonctions membres

7.40.3.1 ApplyLeftFrustum()

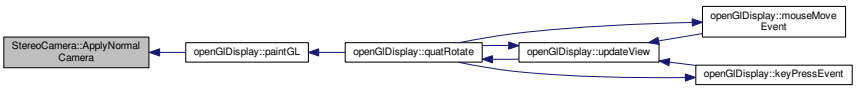
```
void StereoCamera::ApplyLeftFrustum ( ) [inline]
```

Voici le graphe des appelants de cette fonction :

7.40.3.2 ApplyNormalCamera()

`void StereoCamera::ApplyNormalCamera ( ) [inline]`

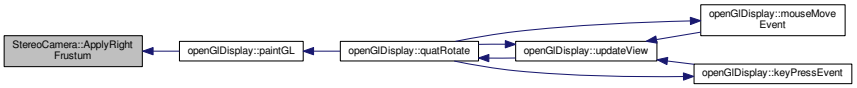
Voici le graphe des appelants de cette fonction :



7.40.3.3 ApplyRightFrustum()

`void StereoCamera::ApplyRightFrustum ( ) [inline]`

Voici le graphe des appelants de cette fonction :



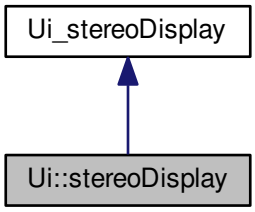
La documentation de cette classe a été générée à partir du fichier suivant :

— stereocamera.h

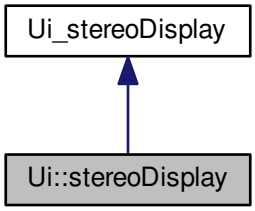
7.41 Référence de la classe Ui : :stereoDisplay

`#include <ui_stereodisplay.h>`

Graphe d'héritage de Ui : :stereoDisplay :



Graphe de collaboration de Ui : :stereoDisplay :

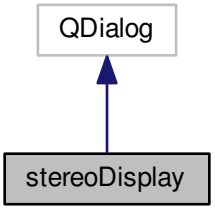


Membres hérités additionnels

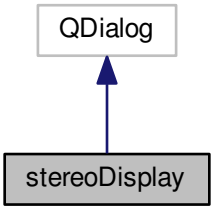
La documentation de cette classe a été générée à partir du fichier suivant :
— ui_stereodisplay.h

7.42 Référence de la classe stereoDisplay

`#include <stereodisplay.h>`
Graphe d'héritage de stereoDisplay :



Graphe de collaboration de stereoDisplay :



Fonctions membres publiques

— stereoDisplay (QWidget *parent=0)
— ~stereoDisplay ()

7.42.1 Documentation des constructeurs et destructeur

7.42.1.1 stereoDisplay()

```
stereoDisplay::stereoDisplay (
            QWidget * parent = 0 )  [explicit]
```

Voici le graphe d'appel pour cette fonction :

7.42.1.2 ~stereoDisplay()

```
stereoDisplay::~stereoDisplay ( )
```

La documentation de cette classe a été générée à partir des fichiers suivants :
— stereodisplay.h
— stereodisplay.cpp

7.43 Référence de la classe tools

```
#include <tools.h>
```

Classes

— struct point3D
Coordonnées 3D d'un point. Les données sont des doubles.
— struct pointInter
Point d'intersection utilisé lors de la superposition entre la grille topo et un tronçon.
— struct segmentOfRoad
Définition d'un segment de route. Pour un segment de route, nous avons différents arguments pour le définir.
— struct triangleToDisplay
Définition d'un triangle. Pour un triangle, nous avons différents arguments pour le définir.

Fonctions membres publiques

— tools ()

Fonctions membres publiques statiques

— static point3D normCalc (point3D pt1, point3D pt2, point3D pt3)
Calcul de la norme Calcul de la norme à partir de 3 points.
— static bool removeDir (const QString &dirName)
Supprime le dossier courant Suppression d'un répertoire selon son chemin.
— static QVector< triangleToDisplay > triangulationGrid (QString gridPath)
Triangulation d'une grille topographique
— static QVector< triangleToDisplay > wall3D (QVector< QVector< QVector< double > > > TXYZ)
Création de murs en 3D.
— static QVector< triangleToDisplay > roof3D (QVector< QVector< QVector< double > > > TXYZ, long heightRoof)
Création de toits en 3D. Cette méthode est utilisée seulement pour les bâtiments ayant 4 façades.
— static QVector< QVector< QVector< double > > > polygonPotitioning (QVector< QVector< float > > mat, double dX, double dY, double x0, double y0, QVector< Q↩ Vector< QVector< double > > > TXY, int h)
Positionnement des bâtiments sur la topographie Rattachement des bâtiments à la topographie avec attribution de la texture.
— static QVector< QVector< QVector< double > > > extractCoord (QgsVectorLayer *layer)
Extraction des coordonnées à partir d'un fichier de polygones Lecture d'un fichier .shp contenant des polygones.
— static double G3D (QVector< QVector< float > > mat, double dX, double dY, double x0, double y0, double x, double y)
Recherche de 16 points autour d'un point donné G3D permet de trouver les points autour d'un point donné afin de pouvoir lancer l'interpolation InterXY.
— static double interXY (QVector< double > TX, QVector< double > TY, QVector< double > TZ, double x, double y)
Interpolation selon le paraboloïde hyperbolique de Laporte. Ce calcul d'interpolation a été conçu par Laporte. Il permet l'interpolation de valeurs en prenant en compte les crêtes et les vallées et assure une continuité dans les pentes.
— static QVector< segmentOfRoad > extractRoad (QgsVectorLayer *layer)
Extraction des coordonnées à partir d'un fichier de lignes. Lecture d'un fichier .shp contenant des lignes pour en extraire les sommets et les informations importantes pour le module de simulation.
— static QVector< segmentOfRoad > extraireMarquageSol (QVector< segmentOfRoad > segment)
Extrait le marquage au sol selon les segments de route. Pour chaque troçon de route, un marque au sol est réalisé sur les bords des routes. Il est situé à 20cm du bord de la route de chaque côté.
— static QVector< triangleToDisplay > triangulationRoute (QVector< segmentOfRoad > segment)
Création des triangles de routes 2 triangles sont créés pour chacun des triangles. Ils sont calculés selon la largeur de la route et l'angle avec les tronçons précédent(s) et suivant(s). Si le tronçon est en bout de réseau, l'angle suivant est déclaré à 90°.
— static double distCalc (point3D pt1, point3D pt2)
Calcul de distance. Calcul basique de distance entre deux points.

— static QVector< segmentOfRoad > topologyRoad (QVector< tools : :segmentOfRoad > listOfSegments)

Création de la topologie. Création de la topologie en prenant en compte tous les tronçons chargés.

— static double angleEntre3Points (point3D pt1, point3D pointCentral, point3D pt3)

Calcul de l'angle entre 3 points. Calcul de l'angle entre deux tronçons prenant en compte le point en commun et les 2 extrêmes.

— static QVector< triangleToDisplay > triangulationRoad (QVector< segmentOfRoad > listOfSegments, float pathWayDepth)

Créer les triangles 3D des tronçons de route. 2 triangles sont créés pour chacun des triangles. Ils sont calculés selon la largeur de la route et l'angle avec les tronçons précédent(s) et suivant(s). Si le tronçon est en bout de réseau, l'angle suivant est déclaré à 90°.

— static QVector< QVector< float > > gridInterpolation (QVector< QVector< float > > mat, int newResolution)

Interpolation d'une grille topographique.

— static pointInter segIntersection (point3D pt1, point3D pt2, point3D pt3, point3D pt4)

Calcul de l'intersection entre deux segments.

— static QVector< triangleToDisplay > burnRoadsIntoTopo (QVector< triangleToDisplay > listTriangleTopo, QVector< triangleToDisplay > listTriangleRoads, double pathWayDepth)

Inclusion des triangles de routes dans la topographie avec prise en compte des trotoirs

— static QVector< segmentOfRoad > cutSegment (QVector< segmentOfRoad > listOfSegments, QVector< triangleToDisplay > listTriangleTopo, QVector< QVector< float > > mat)

Découpage des segments de routes selon la grille de topographie sous-jacente.

— static int choixDepartArrivee (QVector< segmentOfRoad > listOfSegment)

Choix d'un segment sur le réseau routier. Un segment est choisi aléatoirement sur le réseau.

— static QVector< int > itineraireAleatoire (QVector< segmentOfRoad > listOfSegment, int depart)

Trouve un chemin aléatoire sur le réseau. Un trajet aléatoire est trouvé à partir d'un point de départ. La longueur du trajet est fixée selon un nombre de tronçons définis.

— static int recupTroncon (QVector< segmentOfRoad > listOfSegment, QVector< int > listeItineraire, int tronconTest)

Récupère le tronçon suivant d'un itinéraire. Selon un tronçon donné, la méthode trouve un prochain tronçon. En premier, on regarde dans les tronçons suivants, si nous sommes dans une impasse, nous regardons dans les tronçons précédents. Dans le cas pas de tronçon n'est trouvé, l'itinéraire est stopé.

— static bool existDeja (QVector< int > listeItineraire, int tronconTest)

Test si un tronçon est déjà présent dans une liste. Scan de la liste pour savoir si le tronçon potentiel est déjà dans la liste d'itinéraire.

— static bool ComparaisonCoordonnees (point3D pt1, point3D pt2)

Comparaison de deux coordonnées.

— static QVector< tools : :point3D > remplissagePoint (QVector< QVector< QVector< double > > > listePolygone)

Rempli un vecteur de points3D à partir d'un vecteur de vecteur de points.

7.43.1 Documentation des constructeurs et destructeur

7.43.1.1 tools()

```
tools::tools ( )
```

7.43.2 Documentation des fonctions membres

7.43.2.1 angleEntre3Points()

```
tools::angleEntre3Points (
            point3D pt1,
            point3D pointCentral,
            point3D pt3 )  [static]
```

Calcul de l'angle entre 3 points. Calcul de l'angle entre deux tronçons prenant en compte le point en commun et les 2 extrêmes.

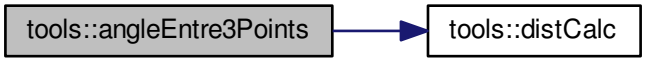
Paramètres

pt1	cordonnées du point extérieur précèdent.
pt2	cordonnées du point commun entre les deux tronçons.
pt3	cordonnées du point extérieur suivant.

Renvoie

Liste des segments de routes avec topologie.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.43.2.2 burnRoadsIntoTopo()

```
tools::burnRoadsIntoTopo (
            QVector< triangleToDisplay > listTriangleTopo,
            QVector< triangleToDisplay > listTriangleRoads,
            double pathWayDepth ) [static]
```

Inclusion des triangles de routes dans la topographie avec prise en compte des trottoirs

Incruste les routes à l'intérieur de la topographie. Les triangles de routes sont intégrés directement dans la topographie avec une profondeur correspondant à la hauteur des trottoirs.

Paramètres

listTriangleTopo	Liste de triangles de topographie.
listTriangleRoads	Liste de triangles de routes.
pathWayDepth	Profondeur de la route (= hauteur des troitoirs).

Renvoie

Liste de triangles.

7.43.2.3 choixDepartArrivee()

```
tools::choixDepartArrivee (
            QVector< segmentOfRoad > listOfSegment ) [static]
```

Choix d'un segment sur le réseau routier. Un segment est choisi aléatoirement sur le réseau.

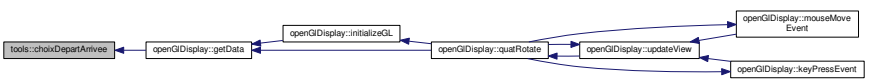
Paramètres

listOfSegment	Liste de segments de route.

Renvoie

Id du segment choisi.

Voici le graphe des appelants de cette fonction :



7.43.2.4 ComparaisonCoordonnees()

```
tools::ComparaisonCoordonnees (
            point3D pt1,
            point3D pt2 ) [static]
```

Comparaison de deux coordonnées.

Paramètres

pt1	cordonnées du premier point
pt2	cordonnées du deuxième point

Renvoie

Booleen à vrai si les deux points sont identiques.

7.43.2.5 cutSegment()

```
tools::cutSegment (
            QVector< segmentOfRoad > listOfSegments,
            QVector< triangleToDisplay > listTriangleTopo,
            QVector< QVector< float > > mat ) [static]
```

Découpage des segments de routes selon la grille de topographie sous-jacente.

Découpage des tronçons de route selon les triangles de la topographie. A chaque intersection entre les segments et les arrêtes de chaque triangles de la topographie, le segment est découpé. Plus la grille à une résolution petite, plus ce calcul est long.

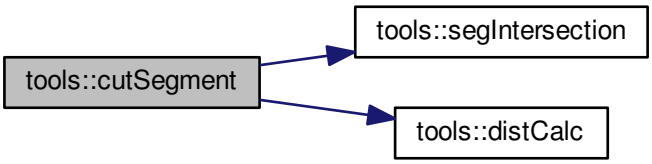
Paramètres

listOfSegments	Liste de segments de route.
listTriangleTopo	Liste de triangles de topographie.
mat	Matrice de topographie interpolée.

Renvoie

Liste de segments découpés.

Voici le graphe d'appel pour cette fonction :



7.43.2.6 distCalc()

```
tools::distCalc (
            point3D pt1,
            point3D pt2 )  [static]
```

Calcul de distance. Calcul basique de distance entre deux points.

Paramètres

pt1	Premier point.
pt2	Deuxième point.

Renvoie

Distance calculée.

Voici le graphe des appelants de cette fonction :

7.43.2.7 existDeja()

```
tools::existDeja (
            QVector< int > listeItineraire,
            int tronconTest )  [static]
```

Test si un tronçon est déjà présent dans une liste. Scan de la liste pour savoir si le tronçon potentiel est déjà dans la liste d'itinéraire.

Paramètres

listeItineraire	Pile d'id alimentant un itinéraire.
tronconTest	Id du tronçon courant

Renvoie

Booleen à vrai si le tronçon a déjà été rentré dans la liste

7.43.2.8 extractCoord()

```
tools::extractCoord (
            QgsVectorLayer * layer )  [static]
```

Extraction des coordonnées à partir d'un fichier de polygones. Lecture d'un fichier .shp contenant des polygones.

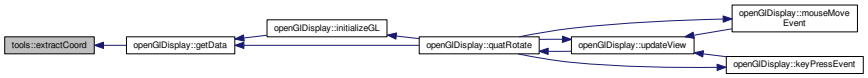
Paramètres

layer	Fichier .shp

Renvoie

Liste de de polygones

Voici le graphe des appelants de cette fonction :



7.43.2.9 extractRoad()

```
tools::extractRoad (
            QgsVectorLayer * layer ) [static]
```

Extraction des coordonnées à partir d'un fichier de lignes. Lecture d'un fichier .shp contenant des lignes pour en extraire les sommets et les informations importantes pour le module de simulation.

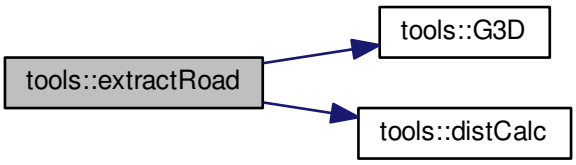
Paramètres

layer	Fichier .shp.

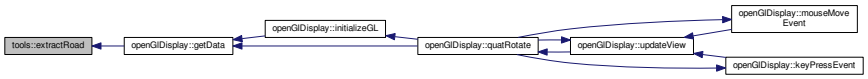
Renvoie

Liste de de segments.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.43.2.10 extraireMarquageSol()

```
tools::extraireMarquageSol (
            QVector< segmentOfRoad > segment ) [static]
```

Extrait le marquage au sol selon les segments de route ; Pour chaque tronçon de route, un marque au sol est réalisé sur les bords des routes. Il est situé à 20cm du bord de la route de chaque côté.

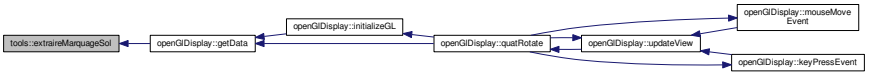
Paramètres

segment	Liste de route.

Renvoie

Liste de de segments.

Voici le graphe des appelants de cette fonction :



7.43.2.11 G3D()

```
tools::G3D (
            QVector< QVector< float > > mat,
            double dX,
            double dY,
            double x0,
            double y0,
            double x,
            double y ) [static]
```

Recherche de 16 points autour d'un point donné. G3D permet de trouver les points autour d'un point donné afin de pouvoir lancer l'interpolation InterXY.

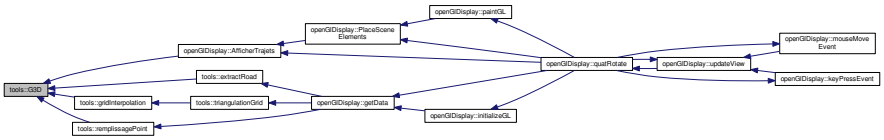
Paramètres

mat	Matrice contenant la topographie
dX	Taille de la grille en largeur
dY	Taille de la grille en hauteur
x0	position en x du coin inférieur gauche
x0	position en y du coin inférieur gauche
x	position en x du point étudié
y	position en y du point étudié

Renvoie

La valeur interpolée au centre des 16 points

Voici le graphe des appelants de cette fonction :



7.43.2.12 gridInterpolation()

```
tools::gridInterpolation (
            QVector< QVector< float > > mat,
            int newResolution ) [static]
```

Interpolation d'une grille topographique.

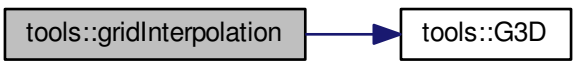
Paramètres

mat	Matrice contenant les valeur de la grille.
newResolution	Nouvelle résolution pour l'interpolation.

Renvoie

Tableau de tableau avec les valeurs de la nouvelle grille interpolée.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.43.2.13 interXY()

```
tools::interXY (
            QVector< double > TX,
            QVector< double > TY,
            QVector< double > TZ,
            double x,
            double y ) [static]
```

Interpolation selon le paraboloïde hyperbolique de Laporte. Ce calcul d'interpolation a été conçu par Laporte. Il permet l'interpolation de valeurs en prenant en compte les crêtes et les vallées et assure une continuité dans les pente.

Paramètres

TX	liste des coordonnées x/
TY	liste des coordonnées y.
TZ	liste des coordonnées z.
x	position en x du point étudié.
y	position en y du point étudié.

Renvoie

Valeur interpolée.

7.43.2.14 itineraireAleatoire()

```
tools::itineraireAleatoire (
            QVector< segmentOfRoad > listOfSegment,
            int depart ) [static]
```

Trouve un chemin aléatoire sur le réseau. Un trajet aléatoire est trouvé à partir d'un point de départ. La longueur du trajet est fixé selon un nombre de tronçons défini.

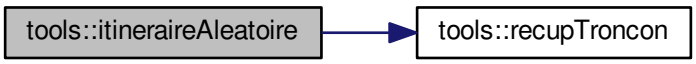
Paramètres

listOfSegment	Liste de segments de route.
depart	Id du tronçon de départ.

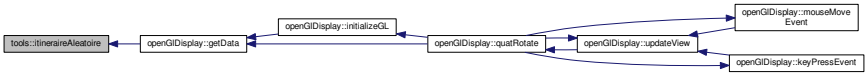
Renvoie

liste d'Id de segments trouvés.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.43.2.15 normCalc()

```
tools::normCalc (
            point3D pt1,
            point3D pt2,
            point3D pt3 )  [static]
```

Calcul de la norme à partir de 3 points.

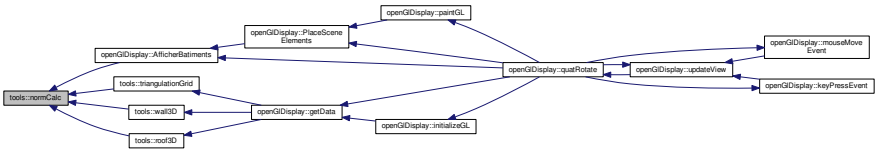
Paramètres

pt1	point 3D
pt2	point 3D
pt3	point 3D

Renvoie

Un point 3D correspondant à la norme de la surface du triangle

Voici le graphe des appelants de cette fonction :



7.43.2.16 polygonPotitioning()

```
tools::polygonPotitioning (
            QVector< QVector< float > > mat,
            double dX,
            double dY,
            double x0,
            double y0,
            QVector< QVector< QVector< double > > > TXY,
            int h ) [static]
```

Positionnement des bâtiments sur la topographie. Rattachement des bâtiments à la topographie avec attribution de la texture.

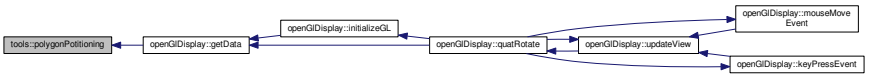
Paramètres

mat	Matrice contenant la topographie
dX	Taille de la grille en largeur
dY	Taille de la grille en hauteur
x0	position en x du coin inférieur gauche
x0	position en y du coin inférieur gauche
LTXY	Liste de bâtiments contenant pour chacun des façades, contenant elles mêmes des coordonnées x, y et z.
h	Hauteur par défaut pour les cas où la donnée n'est pas présente.

Renvoie

Liste de triangles de toits

Voici le graphe des appelants de cette fonction :



7.43.2.17 recupTroncon()

```
tools::recupTroncon (
            QVector< segmentOfRoad > listOfSegment,
            QVector< int > listeItineraire,
            int tronconTest ) [static]
```

Récupère le tronçon suivant d'un itinéraire. Selon un tronçon donné, la méthode trouve un prochain tronçon. En premier, on regarde dans les tronçons suivants, si nous sommes dans une impasse, nous regardons dans les tronçons précédents. Dans le cas où il n'y a pas de tronçon n'est trouvé, l'itinéraire est stoppé.

Paramètres

listOfSegment	Liste de segments de route.
listeItineraire	Pile d'id alimentant un itinéraire.
tronconTest	Id du tronçon courant

Renvoie

Id du tronçon trouvé

Voici le graphe des appelants de cette fonction :

7.43.2.18 removeDir()

```
tools::removeDir (
            const QString & dirName ) [static]
```

Suppression d'un répertoire selon son chemin.

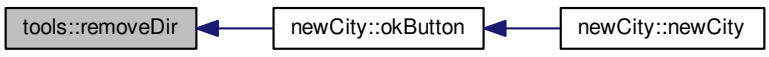
Paramètres

dirName	String contenant le nom du répertoire courant

Renvoie

booléen de reussite

Voici le graphe des appelants de cette fonction :



7.43.2.19 remplissagePoint()

```
tools::remplissagePoint (
            QVector< QVector< QVector< double > > > listePolygone ) [static]
```

Rempli un vecteur de points3D à partir d'un vecteur de vecteur de points.

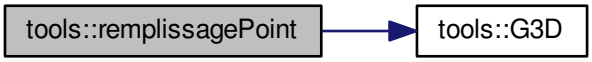
Paramètres

layer	fichier SIG
pt2	cordonnées du deuxième point

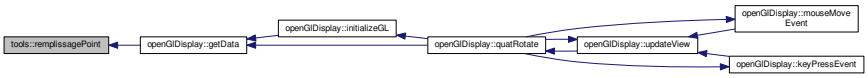
Renvoie

Liste de point3D

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.43.2.20 roof3D()

```
tools::roof3D (
            QVector< QVector< QVector< double > > > TXYZ,
            long heightRoof ) [static]
```

Création de toits en 3D. Cette méthode est utilisée seulement pour les bâtiments ayant 4 façades.

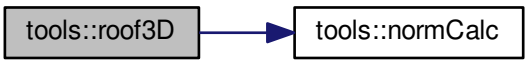
Paramètres

TXYZ	Liste de bâtiments contenant pour chacun des facades, contenant elles mêmes des coordonnées x, y et z.
heightRoof	Hauteur par défaut pour les cas où la donnée n'est pas présente.

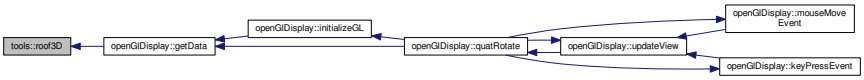
Renvoie

Liste de triangles de toits

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.43.2.21 segIntersection()

```
tools::segIntersection (
            point3D pt1,
            point3D pt2,
            point3D pt3,
            point3D pt4 )  [static]
```

Calcul de l'intersection entree deux segments.

Calcul de l'intersection entree deux segments. Calcul de base pour extraire les coordonnées d'un point d'intersection entre 2 segments.

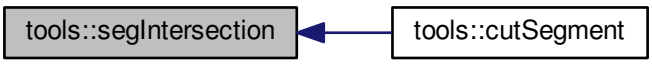
Paramètres

pt1	Premier point du premier segment.
pt2	Deuxième point du premier segment.
pt3	Premier point du deuxième segment.
pt4	Deuxième point du deuxième segment.

Renvoie

Point d'intersection.

Voici le graphe des appelants de cette fonction :



7.43.2.22 topologyRoad()

```
tools::topologyRoad (
            QVector< tools::segmentOfRoad > listOfSegments ) [static]
```

Création de la topologie. Création de la topologie en prenant en compte tous les tronçons chargés.

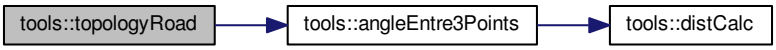
Paramètres

listOfSegments	Liste des segments de routes.

Renvoie

Liste des segments de routes avec topologie.

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.43.2.23 triangulationGrid()

```
tools::triangulationGrid (
            QString gridPath ) [static]
```

triangulation d'une grille Triangulation d'une grille topographique

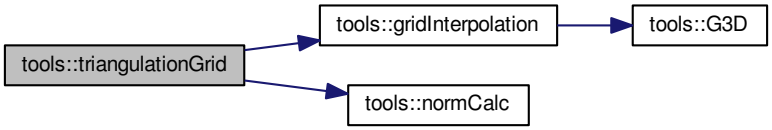
Paramètres

gridPath	String contenant le nom du fichier de topographie

Renvoie

Liste de triangles

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.43.2.24 triangulationRoad()

```
tools::triangulationRoad (
            QVector< segmentOfRoad > listOfSegments,
            float pathWayDepth ) [static]
```

Créé les triangles 3D des tronçons de route. 2 triangles sont créés pour chacun des triangles. Ils sont calculés selon la largeur de la route et l'angle avec les tronçons précédent(s) et suivant(s). Si le tronçon est en bout de réseau, l'angle suivant est déclaré à 90°.

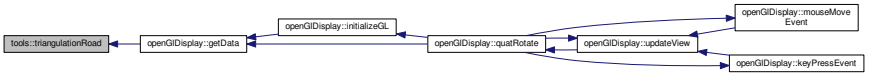
Paramètres

listOfSegments	Liste des tronçons en mode ligne.
pathWayDepth	Largeur par défaut de la route. Elle est utilisée lorsque cette information n'est pas présente dans la donnée source.

Renvoie

Liste de triangles composant les routes 3D.

Voici le graphe des appelants de cette fonction :



7.43.2.25 triangulationRoute()

```
tools::triangulationRoute (
            QVector< segmentOfRoad > segment ) [static]
```

Créé les triangles 3D des tronçons de route. 2 triangles sont créés pour chacun des triangles. Ils sont calculés selon la largeur de la route et l'angle avec les tronçons précédent(s) et suivant(s). Si le tronçon est en bout de réseau, l'angle suivant est déclaré à 90°.

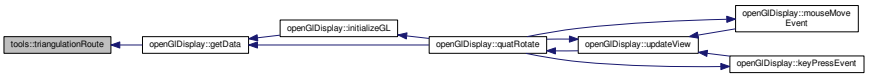
Paramètres

segment	Liste de route.

Renvoie

Liste de de segments.

Voici le graphe des appelants de cette fonction :



7.43.2.26 wall3D()

```
tools::wall3D (
            QVector< QVector< QVector< double > > > TXYZ ) [static]
```

Création de murs en 3D Création de murs en 3D.

Paramètres

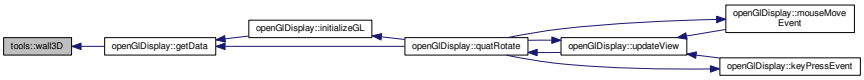
TXYZ	Liste de bâtiments contenant pour chacun des facades, contenant elles mêmes des coordonnées x, y et z

Renvoie

Liste de triangles de murs

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



La documentation de cette classe a été générée à partir des fichiers suivants :

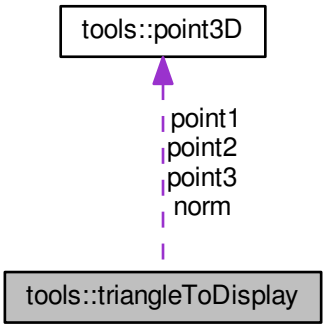
- tools.h
- tools.cpp

7.44 Référence de la structure tools : :triangleToDisplay

Définition d'un triangle. Pour un triangle, nous avons différents arguments pour le définir.

`#include <tools.h>`

Graphe de collaboration de tools : :triangleToDisplay :



Attributs publics

- point3D point1
- point3D point2
- point3D point3
- point3D norm
- int iGrid
- int jGrid
- int numTriangle
- double textureX
- double textureY
- int roofType
- bool deleteMe

7.44.1 Description détaillée

Définition d'un triangle. Pour un triangle, nous avons différents arguments pour le définir.

7.44.2 Documentation des données membres

7.44.2.1 deleteMe

```
bool tools::triangleToDisplay::deleteMe
```

Booléen pour savoir si ce triangle doit être affiché.

7.44.2.2 iGrid

```
int tools::triangleToDisplay::iGrid
```

7.44.2.3 jGrid

```
int tools::triangleToDisplay::jGrid
```

Position I, J dans la grille topo (non renseigné pour les triangles non concernés).

7.44.2.4 norm

```
point3D tools::triangleToDisplay::norm
```

Coordonnées des sommets suivi de la norme du triangle (utilisées pour l'affichage des ombres).

7.44.2.5 numTriangle

```
int tools::triangleToDisplay::numTriangle
```

L'id du triangle de topographie.

7.44.2.6 point1

point3D tools::triangleToDisplay::point1

7.44.2.7 point2

point3D tools::triangleToDisplay::point2

7.44.2.8 point3

point3D tools::triangleToDisplay::point3

7.44.2.9 roofType

int tools::triangleToDisplay::roofType

Identifiant correspondant au type de toit.

7.44.2.10 textureX

double tools::triangleToDisplay::textureX

7.44.2.11 textureY

double tools::triangleToDisplay::textureY

Position de sa texture.

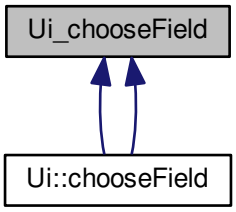
La documentation de cette structure a été générée à partir du fichier suivant :

— tools.h

7.45 Référence de la classe Ui_chooseField

#include <ui_choosefield.h>

Graphe d'héritage de Ui_chooseField :



Fonctions membres publiques

- void setupUi (QDialog *chooseField)
- void retranslateUi (QDialog *chooseField)
- void setupUi (QDialog *chooseField)
- void retranslateUi (QDialog *chooseField)

Attributs publics

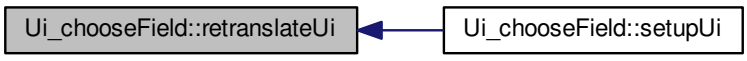
- QComboBox * fieldComboBox
- QRadioButton * fieldDataCheckbox
- QRadioButton * defaultDataCheckbox
- QDoubleSpinBox * defaultDataSpinbox
- QPushButton * okFieldValue
- QFrame * line

7.45.1 Documentation des fonctions membres

7.45.1.1 retranslateUi() [1/2]

```
void Ui_chooseField::retranslateUi (
            QDialog * chooseField )  [inline]
```

Voici le graphe des appelants de cette fonction :



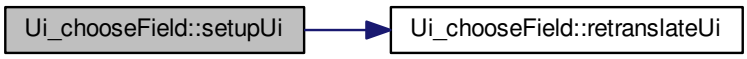
7.45.1.2 retranslateUi() [2/2]

```
void Ui_chooseField::retranslateUi (
            QDialog * chooseField )  [inline]
```

7.45.1.3 setupUi() [1/2]

```
void Ui_chooseField::setupUi (
            QDialog * chooseField ) [inline]
```

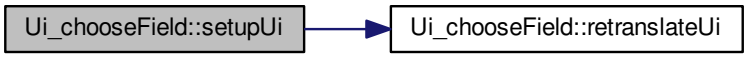
Voici le graphe d'appel pour cette fonction :



7.45.1.4 setupUi() [2/2]

```
void Ui_chooseField::setupUi (
            QDialog * chooseField ) [inline]
```

Voici le graphe d'appel pour cette fonction :



7.45.2 Documentation des données membres

7.45.2.1 defaultDataCheckbox

```
QRadioButton * Ui_chooseField::defaultDataCheckbox
```

7.45.2.2 defaultDataSpinbox

```
QDoubleSpinBox * Ui_chooseField::defaultDataSpinbox
```

7.45.2.3 fieldComboBox

```
QComboBox * Ui_chooseField::fieldComboBox
```

7.45.2.4 fieldDataCheckbox

```
QRadioButton * Ui_chooseField::fieldDataCheckbox
```

7.45.2.5 line

```
QFrame * Ui_chooseField::line
```

7.45.2.6 okFieldValue

```
QPushButton * Ui_chooseField::okFieldValue
```

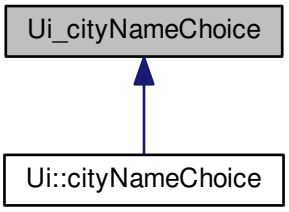
La documentation de cette classe a été générée à partir des fichiers suivants :

- ui_choosefield.h
- ui_choosefieldBuilding.h

7.46 Référence de la classe Ui_cityNameChoice

```
#include <ui_citynamechoice.h>
```

Graphe d'héritage de Ui_cityNameChoice :



Fonctions membres publiques

- void setupUi (QDialog *cityNameChoice)
- void retranslateUi (QDialog *cityNameChoice)

Attributs publics

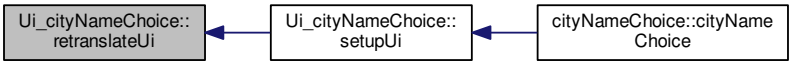
- QLineEdit * cityNameUser
- QLabel * label
- QPushButton * okButton

7.46.1 Documentation des fonctions membres

7.46.1.1 retranslateUi()

```
void Ui_cityNameChoice::retranslateUi (
            QDialog * cityNameChoice ) [inline]
```

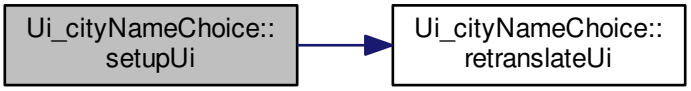
Voici le graphe des appelants de cette fonction :



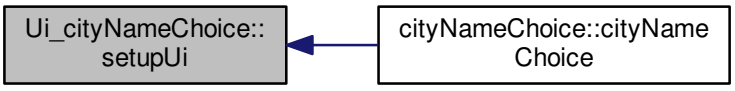
7.46.1.2 setupUi()

```
void Ui_cityNameChoice::setupUi (
            QDialog * cityNameChoice ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.46.2 Documentation des données membres

7.46.2.1 cityNameUser

```
QLineEdit* Ui_cityNameChoice::cityNameUser
```

7.46.2.2 label

```
QLabel* Ui_cityNameChoice::label
```

7.46.2.3 okButton

```
QPushButton* Ui_cityNameChoice::okButton
```

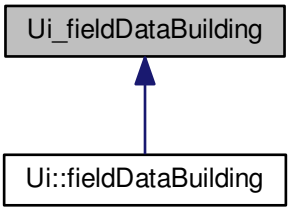
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_citynamechoice.h

7.47 Référence de la classe Ui_fieldDataBuilding

```
#include <ui_fielddatabuilding.h>
```

Graphe d'héritage de Ui_fieldDataBuilding :



Fonctions membres publiques

— void setupUi (QDialog *fieldDataBuilding)
— void retranslateUi (QDialog *fieldDataBuilding)

Attributs publics

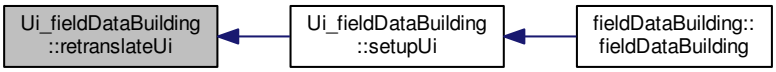
- QFrame * line
- QRadioButton * defaultDataCheckbox
- QComboBox * fieldComboBox
- QDoubleSpinBox * defaultDataSpinbox
- QPushButton * okFieldValue
- QRadioButton * fieldDataCheckbox
- QPushButton * closeButton

7.47.1 Documentation des fonctions membres

7.47.1.1 retranslateUi()

```
void Ui_fieldDataBuilding::retranslateUi (
            QDialog * fieldDataBuilding ) [inline]
```

Voici le graphe des appelants de cette fonction :

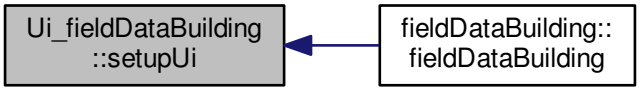


7.47.1.2 setupUi()

```
void Ui_fieldDataBuilding::setupUi (
            QDialog * fieldDataBuilding ) [inline]
```

Voici le graphe d'appel pour cette fonction :

Voici le graphe des appelants de cette fonction :



7.47.2 Documentation des données membres

7.47.2.1 closeButton

```
QPushButton* Ui_fieldDataBuilding::closeButton
```

7.47.2.2 defaultDataCheckbox

```
QRadioButton* Ui_fieldDataBuilding::defaultDataCheckbox
```

7.47.2.3 defaultDataSpinbox

```
QDoubleSpinBox* Ui_fieldDataBuilding::defaultDataSpinbox
```

7.47.2.4 fieldComboBox

```
QComboBox* Ui_fieldDataBuilding::fieldComboBox
```

7.47.2.5 fieldDataCheckbox

```
QRadioButton* Ui_fieldDataBuilding::fieldDataCheckbox
```

7.47.2.6 line

```
QFrame* Ui_fieldDataBuilding::line
```

7.47.2.7 okFieldValue

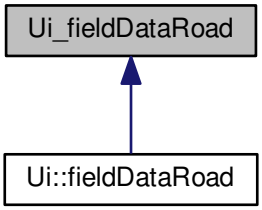
`QPushButton* Ui_fieldDataBuilding::okFieldValue`

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_fielddatabuilding.h

7.48 Référence de la classe Ui_fieldDataRoad

`#include <ui_fielddataroad.h>`

Graphe d'héritage de Ui_fieldDataRoad :



Fonctions membres publiques

— void setupUi (QDialog *fieldDataRoad)
— void retranslateUi (QDialog *fieldDataRoad)

Attributs publics

— QRadioButton * fieldDataCheckbox
— QPushButton * closeButton
— QRadioButton * defaultDataCheckbox
— QPushButton * okFieldValue
— QDoubleSpinBox * defaultDataSpinbox
— QFrame * line
— QComboBox * fieldComboBox

7.48.1 Documentation des fonctions membres

7.48.1.1 retranslateUi()

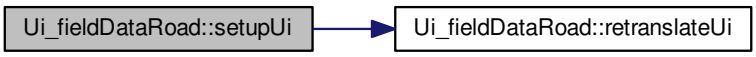
```
void Ui_fieldDataRoad::retranslateUi (
            QDialog * fieldDataRoad ) [inline]
```

Voici le graphe des appelants de cette fonction :

7.48.1.2 setupUi()

```
void Ui_fieldDataRoad::setupUi (
            QDialog * fieldDataRoad ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.48.2 Documentation des données membres

7.48.2.1 closeButton

```
QPushButton* Ui_fieldDataRoad::closeButton
```

7.48.2.2 defaultDataCheckbox

```
QRadioButton* Ui_fieldDataRoad::defaultDataCheckbox
```

7.48.2.3 defaultDataSpinbox

```
QDoubleSpinBox* Ui_fieldDataRoad::defaultDataSpinbox
```

7.48.2.4 fieldComboBox

```
QComboBox* Ui_fieldDataRoad::fieldComboBox
```

7.48.2.5 fieldDataCheckbox

```
QRadioButton* Ui_fieldDataRoad::fieldDataCheckbox
```

7.48.2.6 line

```
QFrame* Ui_fieldDataRoad::line
```

7.48.2.7 okFieldValue

```
QPushButton* Ui_fieldDataRoad::okFieldValue
```

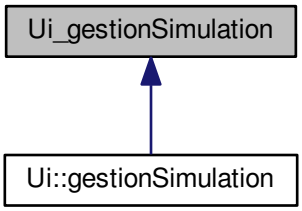
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_fielddataroad.h

7.49 Référence de la classe Ui_gestionSimulation

`#include <ui_gestionsimulation.h>`

Graphe d'héritage de Ui_gestionSimulation :



Fonctions membres publiques

- void setupUi (QDialog *gestionSimulation)
- void retranslateUi (QDialog *gestionSimulation)

Attributs publics

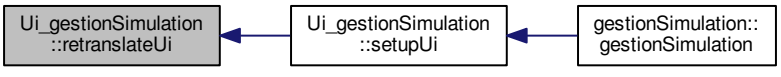
- QDialogButtonBox * buttonBox
- QSlider * nbVehiculeSlide
- QLabel * nbVehiculeLabel
- QLabel * VitesseLabel
- QSlider * vitesseSlide
- QRadioButton * collisionBox

7.49.1 Documentation des fonctions membres

7.49.1.1 retranslateUi()

```
void Ui_gestionSimulation::retranslateUi (
            QDialog * gestionSimulation ) [inline]
```

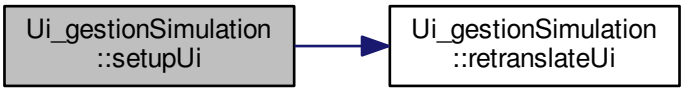
Voici le graphe des appelants de cette fonction :



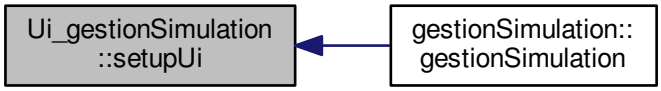
7.49.1.2 setupUi()

```
void Ui_gestionSimulation::setupUi (
            QDialog * gestionSimulation ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.49.2 Documentation des données membres

7.49.2.1 buttonBox

```
QDialogButtonBox* Ui_gestionSimulation::buttonBox
```

7.49.2.2 collisionBox

```
QRadioButton* Ui_gestionSimulation::collisionBox
```

7.49.2.3 nbVehiculeLabel

```
QLabel* Ui_gestionSimulation::nbVehiculeLabel
```

7.49.2.4 nbVehiculeSlide

```
QSlider* Ui_gestionSimulation::nbVehiculeSlide
```

7.49.2.5 VitesseLabel

```
QLabel* Ui_gestionSimulation::VitesseLabel
```

7.49.2.6 vitesseSlide

```
QSlider* Ui_gestionSimulation::vitesseSlide
```

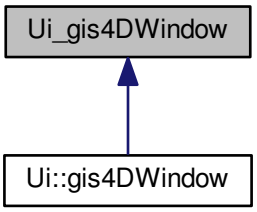
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_gestionsimulation.h

7.50 Référence de la classe Ui_gis4DWindow

```
#include <ui_gis4dwindow.h>
```

Graphe d'héritage de Ui_gis4DWindow :



Fonctions membres publiques

— void setupUi (QMainWindow *gis4DWindow)
— void retranslateUi (QMainWindow *gis4DWindow)

Attributs publics

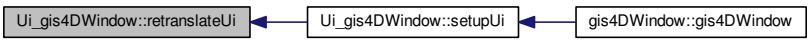
- QAction * actionCreate
- QAction * actionCity3
- QAction * actionCity1
- QAction * actionCity2
- QAction * actionAfficher_circulation
- QAction * actionZoomIn
- QAction * actionZoomOut
- QAction * actionFullExtent
- QAction * actionMoveMap
- QAction * actionCity4
- QAction * actionLoadCity
- QWidget * centralwidget
- QHBoxLayout * horizontalLayout_2
- QGridLayout * gridLayout
- QGroupBox * menuCity
- QLabel * cityName
- QFrame * line
- QGroupBox * gisDataGroup
- QCheckBox * buildingCheckbox
- QCheckBox * waterCheckbox
- QCheckBox * treeChexkbox
- QCheckBox * grassCheckbox
- QCheckBox * roadCheckbox
- QCheckBox * topoCheckbox
- QGroupBox * openglGroup
- QCheckBox * skyCheckbox
- QCheckBox * stereoCheckbox
- QCheckBox * filarCheckbox
- QCheckBox * holoCheckbox
- QCheckBox * cardBoardCheckbox
- QFrame * line_2
- QFrame * frame
- QPushButton * hideMenuButton
- QPushButton * showMenuButton
- QTabWidget * tabWidget
- QWidget * qgisTab
- QWidget * openglTab
- QFrame * frame_2
- QPushButton * MontrerMenuEclairage
- QPushButton * CacherMenuEclairage
- QFrame * GestionLumier
- QGridLayout * GestionLumiere
- QFrame * line_3
- QSlider * VitesseSlide
- QLabel * label
- QCheckBox * lightCheckbox
- QLabel * MoisLabel
- QSlider * HeureSlide
- QSlider * MoisSlide
- QLabel * HeureLabel
- QLabel * VitesseLabel
- QSpacerItem * verticalSpacer
- QLabel * label_2
- QMenuBar * menubar
- QMenu * menuNewCity
- QMenu * menuExistingCity
- QMenu * menuTraffic
- QStatusBar * statusBar
- QToolBar * toolBar

7.50.1 Documentation des fonctions membres

7.50.1.1 retranslateUi()

```
void Ui_gis4DWindow::retranslateUi (
            QMainWindow * gis4DWindow ) [inline]
```

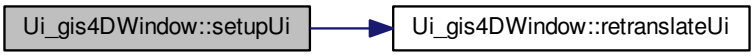
Voici le graphe des appelants de cette fonction :



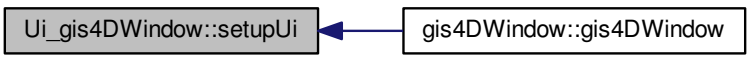
7.50.1.2 setupUi()

```
void Ui_gis4DWindow::setupUi (
            QMainWindow * gis4DWindow ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :



7.50.2 Documentation des données membres

7.50.2.1 actionAfficher_circulation

```
QAction* Ui_gis4DWindow::actionAfficher_circulation
```

7.50.2.2 actionCity1

```
QAction* Ui_gis4DWindow::actionCity1
```

7.50.2.3 actionCity2

```
QAction* Ui_gis4DWindow::actionCity2
```

7.50.2.4 actionCity3

```
QAction* Ui_gis4DWindow::actionCity3
```

7.50.2.5 actionCity4

```
QAction* Ui_gis4DWindow::actionCity4
```

7.50.2.6 actionCreate

```
QAction* Ui_gis4DWindow::actionCreate
```

7.50.2.7 actionFullExtent

```
QAction* Ui_gis4DWindow::actionFullExtent
```

7.50.2.8 actionLoadCity

```
QAction* Ui_gis4DWindow::actionLoadCity
```

7.50.2.9 actionMoveMap

QAction* Ui_gis4DWindow::actionMoveMap

7.50.2.10 actionZoomIn

QAction* Ui_gis4DWindow::actionZoomIn

7.50.2.11 actionZoomOut

QAction* Ui_gis4DWindow::actionZoomOut

7.50.2.12 buildingCheckbox

QCheckBox* Ui_gis4DWindow::buildingCheckbox

7.50.2.13 CacherMenuEclairage

QPushButton* Ui_gis4DWindow::CacherMenuEclairage

7.50.2.14 cardBoardCheckbox

QCheckBox* Ui_gis4DWindow::cardBoardCheckbox

7.50.2.15 centralwidget

QWidget* Ui_gis4DWindow::centralwidget

7.50.2.16 cityName

QLabel* Ui_gis4DWindow::cityName

7.50.2.17 filarCheckbox

QCheckBox* Ui_gis4DWindow::filarCheckbox

7.50.2.18 frame

QFrame* Ui_gis4DWindow::frame

7.50.2.19 frame_2

QFrame* Ui_gis4DWindow::frame_2

7.50.2.20 GestionLumier

QFrame* Ui_gis4DWindow::GestionLumier

7.50.2.21 GestionLumiere

QGridLayout* Ui_gis4DWindow::GestionLumiere

7.50.2.22 gisDataGroup

QGroupBox* Ui_gis4DWindow::gisDataGroup

7.50.2.23 grassCheckbox

QCheckBox* Ui_gis4DWindow::grassCheckbox

7.50.2.24 gridLayout

QGridLayout* Ui_gis4DWindow::gridLayout

7.50.2.25 HeureLabel

QLabel* Ui_gis4DWindow::HeureLabel

7.50.2.26 HeureSlide

QSlider* Ui_gis4DWindow::HeureSlide

7.50.2.27 hideMenuButton

```
QPushButton* Ui_gis4DWindow::hideMenuButton
```

7.50.2.28 holoCheckbox

```
QCheckBox* Ui_gis4DWindow::holoCheckbox
```

7.50.2.29 horizontalLayout_2

```
QHBoxLayout* Ui_gis4DWindow::horizontalLayout_2
```

7.50.2.30 label

```
QLabel* Ui_gis4DWindow::label
```

7.50.2.31 label_2

```
QLabel* Ui_gis4DWindow::label_2
```

7.50.2.32 lightCheckbox

```
QCheckBox* Ui_gis4DWindow::lightCheckbox
```

7.50.2.33 line

```
QFrame* Ui_gis4DWindow::line
```

7.50.2.34 line_2

```
QFrame* Ui_gis4DWindow::line_2
```

7.50.2.35 line_3

```
QFrame* Ui_gis4DWindow::line_3
```

7.50.2.36 menubar

QMenuBar* Ui_gis4DWindow::menubar

7.50.2.37 menuCity

QGroupBox* Ui_gis4DWindow::menuCity

7.50.2.38 menuExistingCity

QMenu* Ui_gis4DWindow::menuExistingCity

7.50.2.39 menuNewCity

QMenu* Ui_gis4DWindow::menuNewCity

7.50.2.40 menuTraffic

QMenu* Ui_gis4DWindow::menuTraffic

7.50.2.41 MoisLabel

QLabel* Ui_gis4DWindow::MoisLabel

7.50.2.42 MoisSlide

QSlider* Ui_gis4DWindow::MoisSlide

7.50.2.43 MontrerMenuEclairage

QPushButton* Ui_gis4DWindow::MontrerMenuEclairage

7.50.2.44 openglGroup

QGroupBox* Ui_gis4DWindow::openglGroup

7.50.2.45 openglTab

```
QWidget* Ui_gis4DWindow::openglTab
```

7.50.2.46 qgisTab

```
QWidget* Ui_gis4DWindow::qgisTab
```

7.50.2.47 roadCheckbox

```
QCheckBox* Ui_gis4DWindow::roadCheckbox
```

7.50.2.48 showMenuButton

```
QPushButton* Ui_gis4DWindow::showMenuButton
```

7.50.2.49 skyCheckbox

```
QCheckBox* Ui_gis4DWindow::skyCheckbox
```

7.50.2.50 statusBar

```
QStatusBar* Ui_gis4DWindow::statusBar
```

7.50.2.51 stereoCheckbox

```
QCheckBox* Ui_gis4DWindow::stereoCheckbox
```

7.50.2.52 tabWidget

```
QTabWidget* Ui_gis4DWindow::tabWidget
```

7.50.2.53 toolBar

```
QToolBar* Ui_gis4DWindow::toolBar
```

7.50.2.54 topoCheckbox

```
QCheckBox* Ui_gis4DWindow::topoCheckbox
```

7.50.2.55 treeChexkbox

```
QCheckBox* Ui_gis4DWindow::treeChexkbox
```

7.50.2.56 verticalSpacer

```
QSpacerItem* Ui_gis4DWindow::verticalSpacer
```

7.50.2.57 VitesseLabel

```
QLabel* Ui_gis4DWindow::VitesseLabel
```

7.50.2.58 VitesseSlide

```
QSlider* Ui_gis4DWindow::VitesseSlide
```

7.50.2.59 waterCheckbox

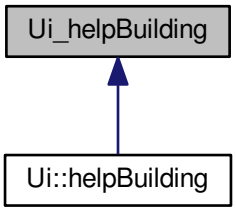
```
QCheckBox* Ui_gis4DWindow::waterCheckbox
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_gis4dwindow.h

7.51 Référence de la classe Ui_helpBuilding

```
#include <ui_helpbuilding.h>
```

Graphe d'héritage de Ui_helpBuilding :



Fonctions membres publiques

— void setupUi (QDialog *helpBuilding)
— void retranslateUi (QDialog *helpBuilding)

Attributs publics

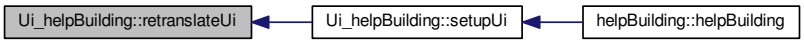
— QLabel * label
— QFrame * frame
— QPushButton * closeHelp

7.51.1 Documentation des fonctions membres

7.51.1.1 retranslateUi()

```
void Ui_helpBuilding::retranslateUi (
            QDialog * helpBuilding )  [inline]
```

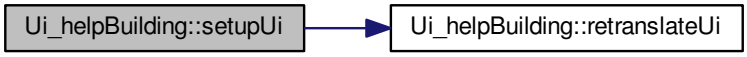
Voici le graphe des appelants de cette fonction :



7.51.1.2 setupUi()

```
void Ui_helpBuilding::setupUi (
            QDialog * helpBuilding )  [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.51.2 Documentation des données membres

7.51.2.1 closeHelp

```
QPushButton* Ui_helpBuilding::closeHelp
```

7.51.2.2 frame

```
QFrame* Ui_helpBuilding::frame
```

7.51.2.3 label

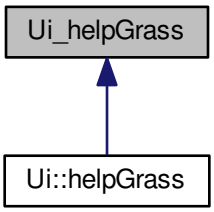
```
QLabel* Ui_helpBuilding::label
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helpbuilding.h

7.52 Référence de la classe Ui_helpGrass

```
#include <ui_helpgrass.h>
```

Graphe d'héritage de Ui_helpGrass :



Fonctions membres publiques

— void setupUi (QDialog *helpGrass)
— void retranslateUi (QDialog *helpGrass)

Attributs publics

— QFrame * frame
— QPushButton * closeHelp
— QLabel * label

7.52.1 Documentation des fonctions membres

7.52.1.1 retranslateUi()

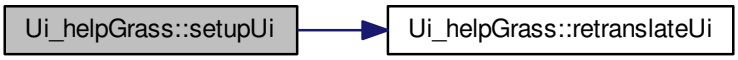
```
void Ui_helpGrass::retranslateUi (
            QDialog * helpGrass ) [inline]
```

Voici le graphe des appelants de cette fonction :

7.52.1.2 setupUi()

```
void Ui_helpGrass::setupUi (
            QDialog * helpGrass ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.52.2 Documentation des données membres

7.52.2.1 closeHelp

```
QPushButton* Ui_helpGrass::closeHelp
```

7.52.2.2 frame

```
QFrame* Ui_helpGrass::frame
```

7.52.2.3 label

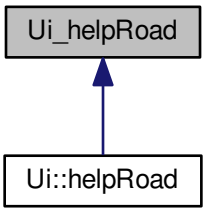
```
QLabel* Ui_helpGrass::label
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helpgrass.h

7.53 Référence de la classe Ui_helpRoad

```
#include <ui_helproad.h>
```

Graphe d'héritage de Ui_helpRoad :



Fonctions membres publiques

— void setupUi (QDialog *helpRoad)
— void retranslateUi (QDialog *helpRoad)

Attributs publics

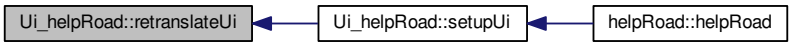
— QLabel * label
— QPushButton * closeHelp
— QFrame * frame

7.53.1 Documentation des fonctions membres

7.53.1.1 retranslateUi()

```
void Ui_helpRoad::retranslateUi (
            QDialog * helpRoad ) [inline]
```

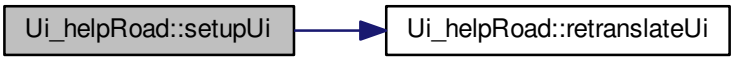
Voici le graphe des appelants de cette fonction :



7.53.1.2 setupUi()

```
void Ui_helpRoad::setupUi (
            QDialog * helpRoad ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.53.2 Documentation des données membres

7.53.2.1 closeHelp

```
QPushButton* Ui_helpRoad::closeHelp
```

7.53.2.2 frame

```
QFrame* Ui_helpRoad::frame
```

7.53.2.3 label

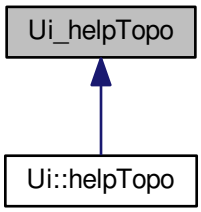
```
QLabel* Ui_helpRoad::label
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helproad.h

7.54 Référence de la classe Ui_helpTopo

```
#include <ui_helptopo.h>
```

Graphe d'héritage de Ui_helpTopo :



Fonctions membres publiques

— void setupUi (QDialog *helpTopo)
— void retranslateUi (QDialog *helpTopo)

Attributs publics

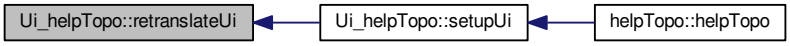
— QFrame * frame
— QPushButton * closeHelp
— QLabel * label

7.54.1 Documentation des fonctions membres

7.54.1.1 retranslateUi()

```
void Ui_helpTopo::retranslateUi (
            QDialog * helpTopo ) [inline]
```

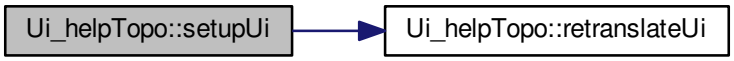
Voici le graphe des appelants de cette fonction :



7.54.1.2 setupUi()

```
void Ui_helpTopo::setupUi (
            QDialog * helpTopo ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.54.2 Documentation des données membres

7.54.2.1 closeHelp

```
QPushButton* Ui_helpTopo::closeHelp
```

7.54.2.2 frame

```
QFrame* Ui_helpTopo::frame
```

7.54.2.3 label

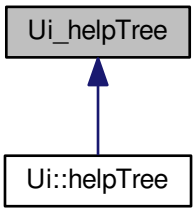
```
QLabel* Ui_helpTopo::label
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helptopo.h

7.55 Référence de la classe Ui_helpTree

```
#include <ui_helptree.h>
```

Graphe d'héritage de Ui_helpTree :



Fonctions membres publiques

— void setupUi (QDialog *helpTree)
— void retranslateUi (QDialog *helpTree)

Attributs publics

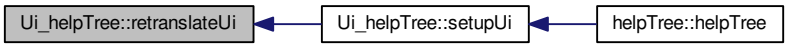
— QLabel * label
— QPushButton * closeHelp
— QFrame * frame

7.55.1 Documentation des fonctions membres

7.55.1.1 retranslateUi()

```
void Ui_helpTree::retranslateUi (
            QDialog * helpTree ) [inline]
```

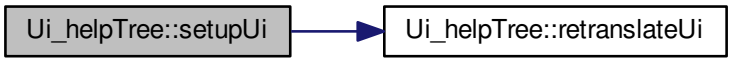
Voici le graphe des appelants de cette fonction :



7.55.1.2 setupUi()

```
void Ui_helpTree::setupUi (
            QDialog * helpTree ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.55.2 Documentation des données membres

7.55.2.1 closeHelp

```
QPushButton* Ui_helpTree::closeHelp
```

7.55.2.2 frame

```
QFrame* Ui_helpTree::frame
```

7.55.2.3 label

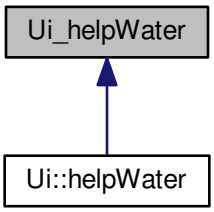
```
QLabel* Ui_helpTree::label
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_helptree.h

7.56 Référence de la classe Ui_helpWater

```
#include <ui_helpwater.h>
```

Graphe d'héritage de Ui_helpWater :



Fonctions membres publiques

— void setupUi (QDialog *helpWater)
— void retranslateUi (QDialog *helpWater)

Attributs publics

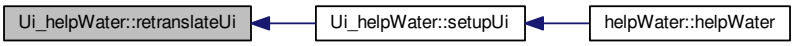
— QFrame * frame
— QPushButton * closeHelp
— QLabel * label

7.56.1 Documentation des fonctions membres

7.56.1.1 retranslateUi()

```
void Ui_helpWater::retranslateUi (
            QDialog * helpWater )  [inline]
```

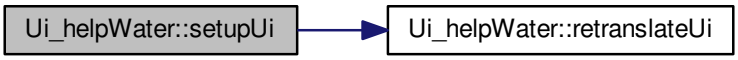
Voici le graphe des appelants de cette fonction :



7.56.1.2 setupUi()

```
void Ui_helpWater::setupUi (
            QDialog * helpWater )  [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.56.2 Documentation des données membres

7.56.2.1 closeHelp

```
QPushButton* Ui_helpWater::closeHelp
```

7.56.2.2 frame

```
QFrame* Ui_helpWater::frame
```

7.56.2.3 label

```
QLabel* Ui_helpWater::label
```

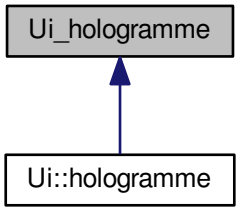
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_helpwater.h

7.57 Référence de la classe Ui_hologramme

```
#include <ui_hologramme.h>
```

Graphe d'héritage de Ui_hologramme :



Fonctions membres publiques

— void setupUi (QFrame *hologramme)
— void retranslateUi (QFrame *hologramme)

Attributs publics

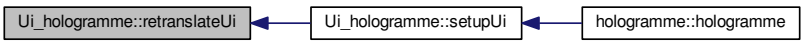
- QWidget * gridLayoutWidget_2
- QGridLayout * vueHautG
- QWidget * gridLayoutWidget_3
- QGridLayout * vueBasG
- QWidget * gridLayoutWidget
- QGridLayout * vueBasD
- QWidget * gridLayoutWidget_4
- QGridLayout * vueHautD

7.57.1 Documentation des fonctions membres

7.57.1.1 retranslateUi()

```
void Ui_hologramme::retranslateUi (
            QFrame * hologramme ) [inline]
```

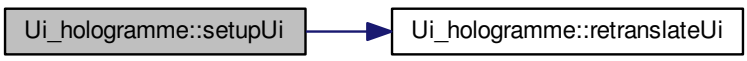
Voici le graphe des appelants de cette fonction :



7.57.1.2 setupUi()

```
void Ui_hologramme::setupUi (
            QFrame * hologramme ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.57.2 Documentation des données membres

7.57.2.1 gridLayoutWidget

```
QWidget* Ui_hologramme::gridLayoutWidget
```

7.57.2.2 gridLayoutWidget_2

```
QWidget* Ui_hologramme::gridLayoutWidget_2
```

7.57.2.3 gridLayoutWidget_3

```
QWidget* Ui_hologramme::gridLayoutWidget_3
```

7.57.2.4 gridLayoutWidget_4

```
QWidget* Ui_hologramme::gridLayoutWidget_4
```

7.57.2.5 vueBasD

```
QGridLayout* Ui_hologramme::vueBasD
```

7.57.2.6 vueBasG

```
QGridLayout* Ui_hologramme::vueBasG
```

7.57.2.7 vueHautD

```
QGridLayout* Ui_hologramme::vueHautD
```

7.57.2.8 vueHautG

```
QGridLayout* Ui_hologramme::vueHautG
```

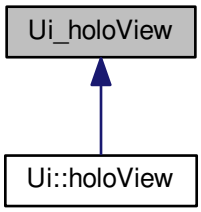
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_hologramme.h

7.58 Référence de la classe Ui_holoView

`#include <ui_holoview.h>`

Graphe d'héritage de Ui_holoView :



Fonctions membres publiques

— void setupUi (QDialog *holoView)
— void retranslateUi (QDialog *holoView)

Attributs publics

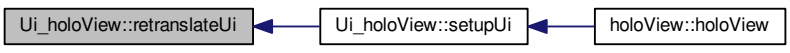
— QGridLayout * gridLayout

7.58.1 Documentation des fonctions membres

7.58.1.1 retranslateUi()

```
void Ui_holoView::retranslateUi (
            QDialog * holoView ) [inline]
```

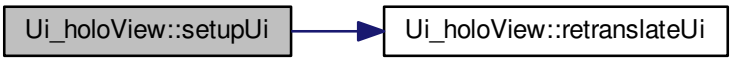
Voici le graphe des appelants de cette fonction :



7.58.1.2 setupUi()

```
void Ui_holoView::setupUi (
            QDialog * holoView ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.58.2 Documentation des données membres

7.58.2.1 gridLayout

```
QGridLayout* Ui_holoView::gridLayout
```

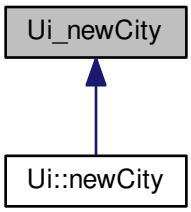
La documentation de cette classe a été générée à partir du fichier suivant :

— ui_holoview.h

7.59 Référence de la classe Ui_newCity

```
#include <ui_newcity.h>
```

Graphe d'héritage de Ui_newCity :



Fonctions membres publiques

— void setupUi (QDialog *newCity)
— void retranslateUi (QDialog *newCity)

Attributs publics

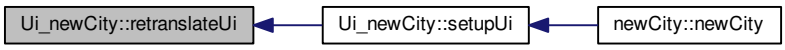
— QPushButton * loadBuildingButton
— QPushButton * loadRoadButton
— QPushButton * loadTreeButton
— QPushButton * loadGrassButton
— QPushButton * loadWaterButton
— QPushButton * loadTopoButton
— QLabel * label
— QFrame * line
— QLabel * label_2
— QFrame * line_2
— QPushButton * createCityButton
— QPushButton * helpBuildingButton
— QPushButton * helpRoadButton
— QPushButton * helpTreeButton
— QPushButton * helpGrassButton
— QPushButton * helpWaterButton
— QPushButton * helpTopoButton

7.59.1 Documentation des fonctions membres

7.59.1.1 retranslateUi()

```
void Ui_newCity::retranslateUi (
            QDialog * newCity )  [inline]
```

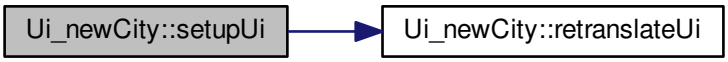
Voici le graphe des appelants de cette fonction :



7.59.1.2 setupUi()

```
void Ui_newCity::setupUi (
            QDialog * newCity )  [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.59.2 Documentation des données membres

7.59.2.1 createCityButton

```
QPushButton* Ui_newCity::createCityButton
```

7.59.2.2 helpBuildingButton

```
QPushButton* Ui_newCity::helpBuildingButton
```

7.59.2.3 helpGrassButton

```
QPushButton* Ui_newCity::helpGrassButton
```

7.59.2.4 helpRoadButton

```
QPushButton* Ui_newCity::helpRoadButton
```

7.59.2.5 helpTopoButton

```
QPushButton* Ui_newCity::helpTopoButton
```

7.59.2.6 helpTreeButton

```
QPushButton* Ui_newCity::helpTreeButton
```

7.59.2.7 helpWaterButton

```
QPushButton* Ui_newCity::helpWaterButton
```

7.59.2.8 label

```
QLabel* Ui_newCity::label
```

7.59.2.9 label_2

```
QLabel* Ui_newCity::label_2
```

7.59.2.10 line

```
QFrame* Ui_newCity::line
```

7.59.2.11 line_2

```
QFrame* Ui_newCity::line_2
```

7.59.2.12 loadBuildingButton

```
QPushButton* Ui_newCity::loadBuildingButton
```

7.59.2.13 loadGrassButton

```
QPushButton* Ui_newCity::loadGrassButton
```

7.59.2.14 loadRoadButton

```
QPushButton* Ui_newCity::loadRoadButton
```

7.59.2.15 loadTopoButton

```
QPushButton* Ui_newCity::loadTopoButton
```

7.59.2.16 loadTreeButton

```
QPushButton* Ui_newCity::loadTreeButton
```

7.59.2.17 loadWaterButton

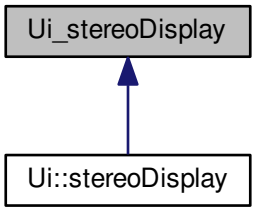
```
QPushButton* Ui_newCity::loadWaterButton
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_newcity.h

7.60 Référence de la classe Ui_stereoDisplay

```
#include <ui_stereodisplay.h>
```

Graphe d'héritage de Ui_stereoDisplay :



Fonctions membres publiques

— void setupUi (QDialog *stereoDisplay)
— void retranslateUi (QDialog *stereoDisplay)

Attributs publics

— QGridLayout * gridLayout
— QHBoxLayout * viewLeft
— QHBoxLayout * viewRight
— QSpacerItem * horizontalSpacer

7.60.1 Documentation des fonctions membres

7.60.1.1 retranslateUi()

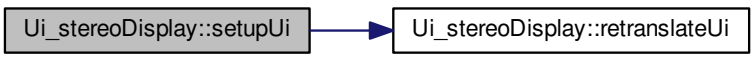
```
void Ui_stereoDisplay::retranslateUi (
            QDialog * stereoDisplay ) [inline]
```

Voici le graphe des appelants de cette fonction :

7.60.1.2 setupUi()

```
void Ui_stereoDisplay::setupUi (
            QDialog * stereoDisplay ) [inline]
```

Voici le graphe d'appel pour cette fonction :



Voici le graphe des appelants de cette fonction :

7.60.2 Documentation des données membres

7.60.2.1 gridLayout

```
QGridLayout* Ui_stereoDisplay::gridLayout
```

7.60.2.2 horizontalSpacer

```
QSpacerItem* Ui_stereoDisplay::horizontalSpacer
```

7.60.2.3 viewLeft

```
QHBoxLayout* Ui_stereoDisplay::viewLeft
```

7.60.2.4 viewRight

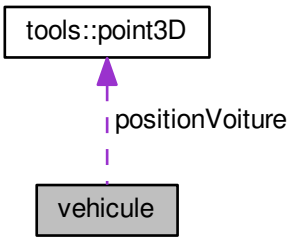
```
QHBoxLayout* Ui_stereoDisplay::viewRight
```

La documentation de cette classe a été générée à partir du fichier suivant :
— ui_stereodisplay.h

7.61 Référence de la classe vehicule

```
#include <vehicule.h>
```

Graphe de collaboration de vehicule :



Fonctions membres publiques

— vehicule ()

Attributs publics

- int numVehicule
- double vitesse
- double angle
- tools : :point3D positionVoiture
- int typeVoiture
- int indexTroncon
- double echelle
- int indexItineraire
- double anglePente

7.61.1 Documentation des constructeurs et destructeur

7.61.1.1 vehicule()

```
vehicule::vehicule ( )
```

7.61.2 Documentation des données membres

7.61.2.1 angle

```
double vehicule::angle
```

7.61.2.2 anglePente

```
double vehicule::anglePente
```

7.61.2.3 echelle

```
double vehicule::echelle
```

7.61.2.4 indexItineraire

```
int vehicule::indexItineraire
```

7.61.2.5 indexTroncon

```
int vehicule::indexTroncon
```

7.61.2.6 numVehicule

```
int vehicule::numVehicule
```

7.61.2.7 positionVoiture

```
tools::point3D vehicule::positionVoiture
```

7.61.2.8 typeVoiture

```
int vehicule::typeVoiture
```

7.61.2.9 vitesse

```
double vehicule::vitesse
```

La documentation de cette classe a été générée à partir des fichiers suivants :

- vehicule.h
- vehicule.cpp

Chapitre 8

Documentation des fichiers

8.1 Référence du fichier calculpositionsoleil.cpp

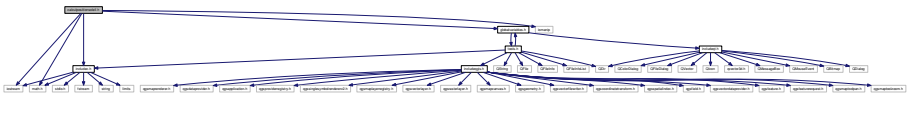
```
#include "calculpositionsoleil.h"
```

Graphe des dépendances par inclusion de calculpositionsoleil.cpp :

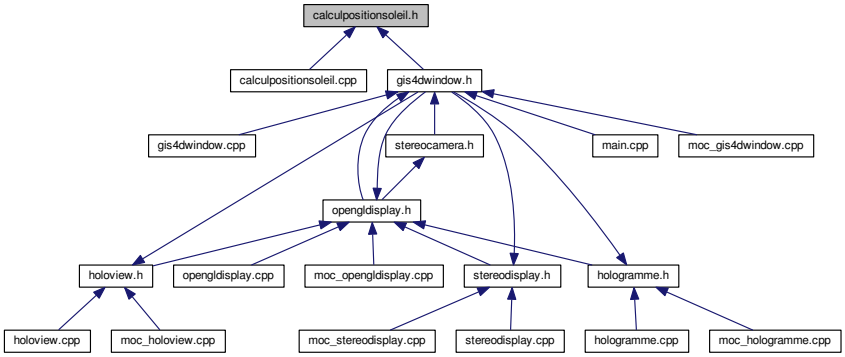
8.2 Référence du fichier calculpositionsoleil.h

```
#include <iostream>
#include <math.h>
#include <iomanip>
#include <globalvariables.h>
#include <includec.h>
```

Graphe des dépendances par inclusion de calculpositionsoleil.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class caculPositionSoleil

8.3 Référence du fichier citynamechoice.cpp

```
#include "citynamechoice.h"
#include "ui_citynamechoice.h"
```

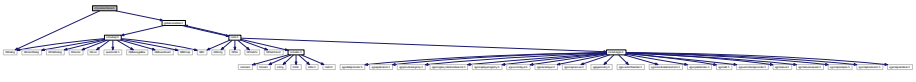
Graphe des dépendances par inclusion de citynamechoice.cpp :

8.4 Référence du fichier citynamechoice.h

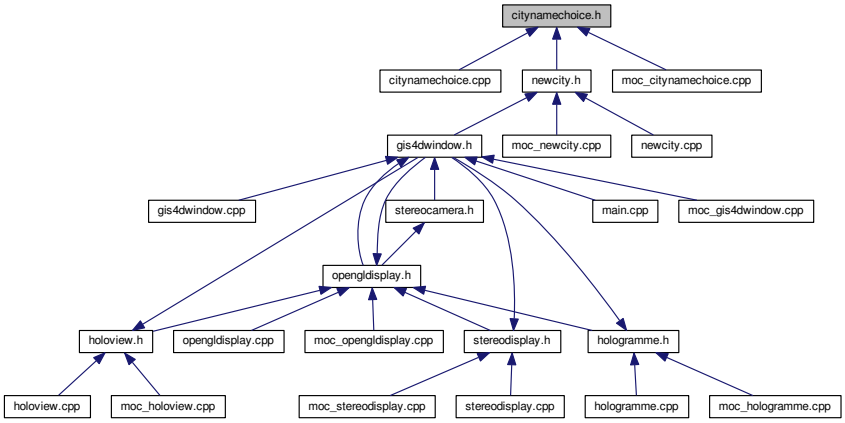
Fenêtre pour le choix du nom de ville.

```
#include <QDialog>
#include <globalvariables.h>
```

Graphe des dépendances par inclusion de citynamechoice.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class cityNameChoice

Espaces de nommage

— Ui

8.4.1 Description détaillée

Fenêtre pour le choix du nom de ville.

Auteur

Julien Richard

8.5 Référence du fichier fielddatabuilding.cpp

Fenêtre pour le choix de la donnée attributaire.

```
#include "fielddatabuilding.h"
#include "ui_fielddatabuilding.h"
```

Graphe des dépendances par inclusion de fielddatabuilding.cpp :

8.5.1 Description détaillée

Fenêtre pour le choix de la donnée attributaire.

Auteur

Julien Richard

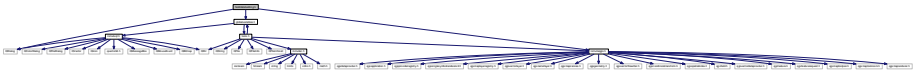
Les données de bâti et de route peuvent être extrudées grâce à une donnée attributaire de hauteur (pour les bâtiments). L'utilisateur peut avoir ou non ces données. Cette fenêtre lui permet de choisir entre une donnée existante dans la table attributaire ou alors une donnée par défaut. Cette dernière sera prise en compte pour un jeu de données aléatoires afin de ne pas avoir la même hauteur sur tous les bâtiments.

8.6 Référence du fichier fielddatabuilding.h

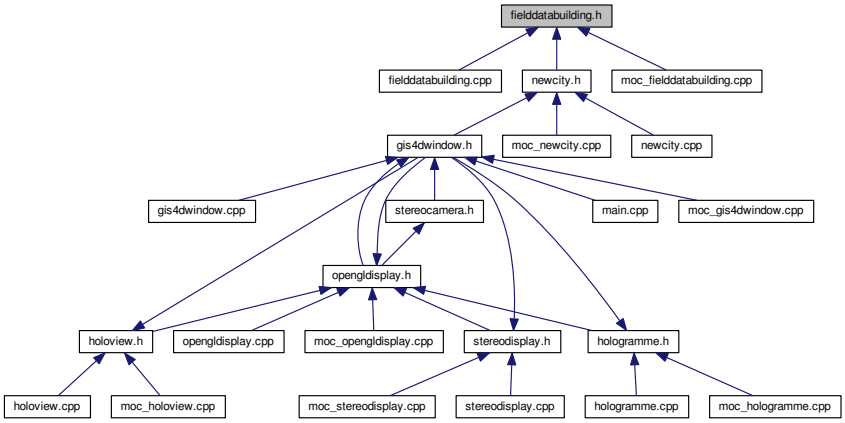
Fenêtre pour le choix du champs contenant la hauteur des bâtiments.

```
#include <QDialog>
#include <globalvariables.h>
#include <includeqgis.h>
```

Graphe des dépendances par inclusion de fielddatabuilding.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class fieldDataBuilding

Espaces de nommage

— Ui

8.6.1 Description détaillée

Fenêtre pour le choix du champs contenant la hauteur des bâtiments.

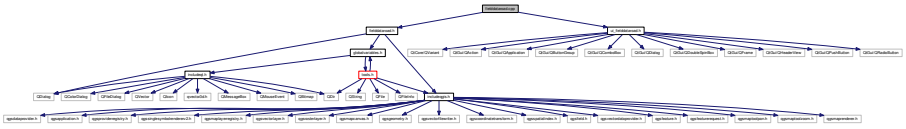
Auteur

Julien Richard

8.7 Référence du fichier fielddataroad.cpp

```
#include "fielddataroad.h"
#include "ui_fielddataroad.h"
```

Graphe des dépendances par inclusion de fielddataroad.cpp :

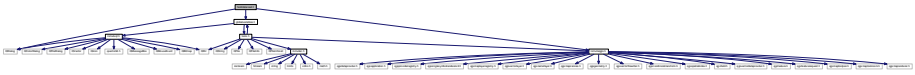


8.8 Référence du fichier fielddataroad.h

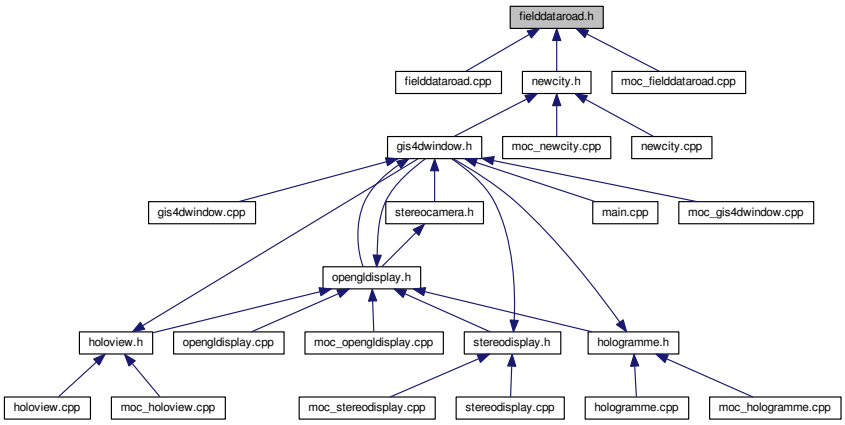
Fenêtre pour le choix du champs contenant la largeur des routes.

```
#include <QDialog>
#include <globalvariables.h>
#include <includeqgis.h>
```

Graphe des dépendances par inclusion de fielddataroad.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class fieldDataRoad

Espaces de nommage

— Ui

8.8.1 Description détaillée

Fenêtre pour le choix du champs contenant la largeur des routes.
Auteur

Julien Richard

8.9 Référence du fichier gestionsimulation.cpp

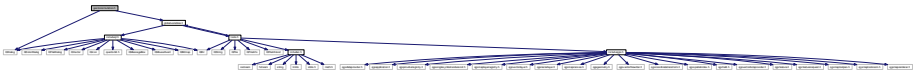
```
#include "gestionsimulation.h"
#include "ui_gestionsimulation.h"
```

Graphe des dépendances par inclusion de gestionsimulation.cpp :

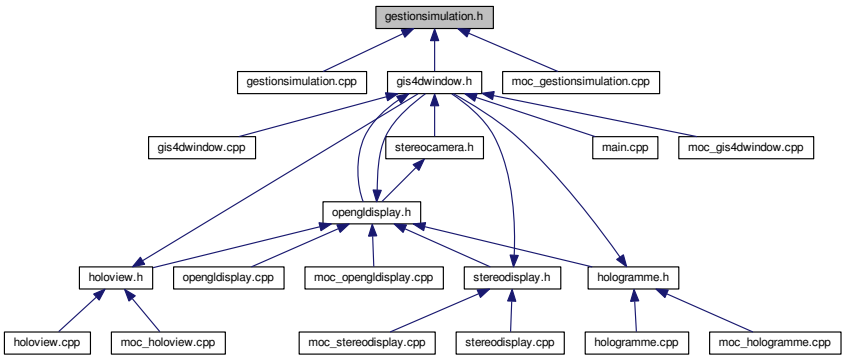
8.10 Référence du fichier gestionsimulation.h

```
#include <QDialog>
#include <globalvariables.h>
```

Graphe des dépendances par inclusion de gestionsimulation.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class gestionSimulation

Espaces de nommage

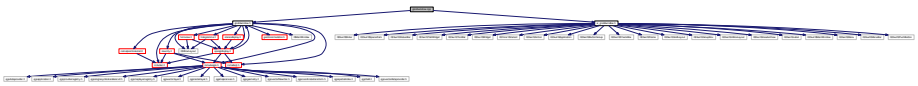
— Ui

8.11 Référence du fichier gis4dwindow.cpp

Fenêtre principale du logiciel.

```
#include "gis4dwindow.h"
#include "ui_gis4dwindow.h"
```

Graphe des dépendances par inclusion de gis4dwindow.cpp :



Variables

— QVBoxLayout * openGllayout
Cadre OpenGl à intégrer dans l'interface (ici dans un onglet de la fenêtre)

8.11.1 Description détaillée

Fenêtre principale du logiciel.

Auteur

Julien Richard

Cette interface se veut minimaliste pour interresser le plus grand nombre de personnes. Un menu est dédié à la création ou au chargement de ville 3D. Un autre est dédié à la gestion du trafic. Une barre d'outils dédiée à la partie SIG 2D (partie Qgis) est située sous le menu principal. Un barre latérale située à gauche de la fenêtre permet l'interaction avec la visualisation 2D ou 3D. Enfin, deux onglets permet de changer la vue 2D en 3D et inversement.

8.11.2 Documentation des variables

8.11.2.1 openGllayout

`QVBoxLayout* openGllayout`

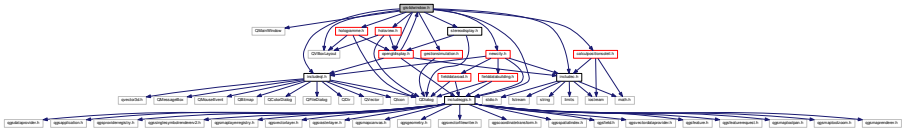
Cadre OpenGl à intégrer dans l'interface (ici dans un onglet de la fenêtre)

8.12 Référence du fichier gis4dwindow.h

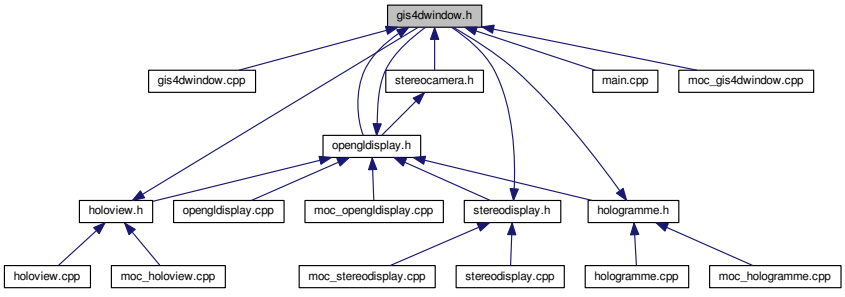
Fenêtre globale du logiciel Dans cette fenêtre se trouve toutes les possibilités d'affichage ou de calculs d'itinéraires. Un menu permet de choisir une ville à importer (nouvelle ou existante via les fichiers de suavegarde). Un panneau latéral à gauche permet de choisir les données à visualiser. Les vues stéréoscopique et holographique sont accessibles dans ce menu. Les simulations sont présentes dans la barre en haut de la fenêtre.

```
#include <QMainWindow>
#include <QVBoxLayout>
#include <includeqgis.h>
#include <includec.h>
#include <includeqt.h>
#include <newcity.h>
#include <opengldisplay.h>
#include <holoview.h>
#include <stereodisplay.h>
#include <calculpositionsoleil.h>
#include <hologramme.h>
#include <gestionsimulation.h>
```

Graphe des dépendances par inclusion de gis4dwindow.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class gis4DWindow

Espaces de nommage

— Ui

8.12.1 Description détaillée

Fenêtre globale du logiciel. Dans cette fenêtre se trouve toutes les possibilités d'affichage ou de calculs d'itinéraires. Un menu permet de choisir une ville à importer (nouvelle ou existante via les fichiers de sauvegarde). Un panneau latéral à gauche permet de choisir les données à visualiser. Les vues stéréoscopique et holographique sont accessibles dans ce menu. Les simulations sont présentes dans la barre en haut de la fenêtre.

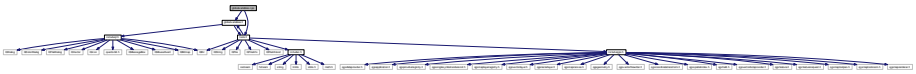
Auteur

Julien Richard

8.13 Référence du fichier globalvariables.cpp

```
#include "globalvariables.h"
#include <tools.h>
```

Graphe des dépendances par inclusion de globalvariables.cpp :



Variables

- QString messagePosQgis
  Message PosGis pour la barre de statut.
- bool cityLoaded =false
  Booléen pour savoir si la ville est chargée.
- bool existingCity = false
  Booléen pour savoir si la ville est une ville déjà existante dans la bdd.
- QString workFolder
  Chemin de travail global.
- QString pathBuilding
- QString pathRoad
- QString pathTree
  Chemin du fichier .shp des arbres.
- QString pathGrass
  Chemin du fichier .shp des zones herbacées.
- QString pathWater
  Chemin du fichier .shp des surface d'eau.
- QString pathTopo
  Chemin du fichier .asc de la topographie.
- QString save3DpathTopo
  Chemin du fichier .citytrafic3D contenant les triangles de la topographie.
- QString save3DpathWall
  Chemin du fichier .citytrafic3D contenant les triangles des façades des bâtiments.
- QString save3DpathRoof
  Chemin du fichier .citytrafic3D contenant les triangles de la route.
- int defaultOrNotBulding =0
  Valeur pour savoir si la donnée de hauteur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).
- int defaultOrNotRoad =0
  Valeur pour savoir si la donnée da largeur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).

— int fieldNumHeightBuilding =0
Id du champ contenant la hauteur de chaque bâtiment.
— int fieldNumWidthRoad =0
Id du champ contenant la largeur de chaque tronçon de route.
— int fieldUniqueRoad =0
Id du champ contenant le sens unique ou non de chaque tronçon de route.
— float defaultHeight = 0
Valeur par défaut de la hauteur des bâtiments.
— float defaultWidth =0
Valeur par défaut de la largeur des routes.
— QString cityName
Nom de la ville importée.
— QVector< QVector< float > > elevation3D
Grille d'élevation.
— float zMinimum
Valeur minimale pour l'élévation.
— float zMaximum
Valeur maximale pour l'élévation.
— double dXElev
Largeur du pixel de référence de la grille.
— double dYElev
Hauteur du pixel de référence de la grille.
— double x0Elev
Coordonnée en x du coin inférieur gauche de la grille.
— double y0Elev
Coordonnée en y du coin inférieur gauche de la grille.
— int nRowsElev
Nombre de lignes de la grille.
— int nColsElev
Nombre de colonnes de la grille.
— QVector3D View
QUATERNION CAMERA : Vecteur 3D pour le point de vue.
— QVector3D Position
QUATERNION CAMERA : Vecteur 3D pour la position de la camera.
— QVector3D Up
QUATERNION CAMERA : Vecteur 3D pour le pitch
— QVector3D Direction
QUATERNION CAMERA : Vecteur 3D pour la direction
— bool dispBuilding
Affichage ou non des bâtiments.
— bool dispRoad
Affichage ou non des routes.
— bool dispTopo
Affichage ou non de la topographie.
— bool dispStereo
Affichage ou non de la vue stérographique (cyan et rouge)
— bool dispSky
Affichage ou non du ciel.
— bool dispWireMode
Affichage ou non du mode filaire.
— bool dispLight
Affichage ou non de la lumière.
— double positionSoleilX
Position du soleil en X.
— double positionSoleilY
Position du soleil en Y.
— double positionSoleilZ
Position du soleil en Z.

— double positionSoleilAzimut
Position du soleil en azimut.
— QVector< tools : :triangleToDisplay > triangleTopography
Triangles fpour la topographie.
— QVector< tools : :triangleToDisplay > triangleTopography2
— QVector< tools : :triangleToDisplay > triangleBati
Triangles pour les bâtiments.
— QVector< tools : :triangleToDisplay > trianglesRoad
Triangle des routes créés depuis les segments de routes.
— QVector< QVector< QVector< double > > > polygonBati
Polygones des batiments.
— int nbVehicule
Nombre de véhicules.
— bool modeCollision
Mode collision activé ou non.
— int vitesse
Vitesse de simulation.

8.13.1 Documentation des variables

8.13.1.1 cityLoaded

```
bool cityLoaded =false
```

Booléen pour savoir si la ville est chargée.

8.13.1.2 cityName

```
QString cityName
```

Nom de la ville importée.

8.13.1.3 defaultHeight

```
float defaultHeight = 0
```

Valeur par défaut de la hauteur des bâtiments.

8.13.1.4 defaultOrNotBulding

```
int defaultOrNotBulding =0
```

Valeur pour savoir si la donnée de hauteur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).

8.13.1.5 defaultOrNotRoad

```
int defaultOrNotRoad =0
```

Valeur pour savoir si la donnée da largeur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).

8.13.1.6 defaultWidth

```
float defaultWidth =0
```

Valeur par défaut de la largeur des routes.

8.13.1.7 Direction

```
QVector3D Direction
```

Vecteur 3D pour la direction de la caméra.

8.13.1.8 dispBuilding

```
bool dispBuilding
```

Affichage ou non des bâtiments.

8.13.1.9 dispLight

```
bool dispLight
```

Affichage ou non de la lumière.

8.13.1.10 dispRoad

```
bool dispRoad
```

Affichage ou non des routes.

8.13.1.11 dispSky

```
bool dispSky
```

Affichage ou non du ciel.

8.13.1.12 dispStereo

`bool dispStereo`

Affichage ou non de la vue stérographique (cyan et rouge)

8.13.1.13 dispTopo

`bool dispTopo`

Affichage ou non de la topographie.

8.13.1.14 dispWireMode

`bool dispWireMode`

Affichage ou non du mode filaire.

8.13.1.15 dXElev

`double dXElev`

Largeur du pixel de référence de la grille.

8.13.1.16 dYElev

`double dYElev`

Hauteur du pixel de référence de la grille.

8.13.1.17 elevation3D

`QVector<QVector <float> > elevation3D`

Grille d'élevation.

8.13.1.18 existingCity

`bool existingCity = false`

Booléen pour savoir si la ville est une ville déjà existante dans la bdd.

8.13.1.19 fieldNumHeightBuilding

```
int fieldNumHeightBuilding =0
```

Id du champ contenant la hauteur de chaque bâtiment.

8.13.1.20 fieldNumWidthRoad

```
int fieldNumWidthRoad =0
```

Id du champ contenant la largeur de chaque tronçon de route.

8.13.1.21 fieldUniqueRoad

```
int fieldUniqueRoad =0
```

Id du champ contenant le sens unique ou non de chaque tronçon de route.

8.13.1.22 messagePosQgis

```
QString messagePosQgis
```

Message PosGis pour la barre de statut.

8.13.1.23 modeCollision

```
bool modeCollision
```

Mode collision activé ou non.

8.13.1.24 nbVehicule

```
int nbVehicule
```

Nombre de véhicules.

8.13.1.25 nColsElev

```
int nColsElev
```

Nombre de colonnes de la grille.

8.13.1.26 nRowsElev

`int nRowsElev`

Nombre de lignes de la grille.

8.13.1.27 pathBuilding

`QString pathBuilding`

du fichier .shp des bâtiments.

8.13.1.28 pathGrass

`QString pathGrass`

Chemin du fichier .shp des zones herbacées.

8.13.1.29 pathRoad

`QString pathRoad`

du fichier .shp des routes.

8.13.1.30 pathTopo

`QString pathTopo`

Chemin du fichier .asc de la topographie.

8.13.1.31 pathTree

`QString pathTree`

Chemin du fichier .shp des arbres.

8.13.1.32 pathWater

`QString pathWater`

Chemin du fichier .shp des surface d'eau.

8.13.1.33 polygonBati

`QVector<QVector <QVector <double> > > polygonBati`

Délimitation des bâtiments

8.13.1.34 Position

`QVector3D Position`

Vecteur 3D pour la position de la caméra.

8.13.1.35 positionSoleilAzimut

`double positionSoleilAzimut`

Position du soleil en azimut.

8.13.1.36 positionSoleilX

`double positionSoleilX`

Position du soleil en X.

8.13.1.37 positionSoleilY

`double positionSoleilY`

Position du soleil en Y.

8.13.1.38 positionSoleilZ

`double positionSoleilZ`

Position du soleil en Z.

8.13.1.39 save3DpathRoof

`QString save3DpathRoof`

Chemin du fichier .citytrafic3D contenant les triangles de la route.

8.13.1.40 save3DpathTopo

`QString save3DpathTopo`

Chemin du fichier .citytrafic3D contenant les triangles de la topographie.

8.13.1.41 save3DpathWall

`QString save3DpathWall`

Chemin du fichier .citytrafic3D contenant les triangles des façades des bâtiments.

8.13.1.42 triangleBati

`QVector<tools::triangleToDisplay> triangleBati`

Triangles 3D de bâtiments.

8.13.1.43 trianglesRoad

`QVector<tools::triangleToDisplay> trianglesRoad`

Triangles 3D de routes.

8.13.1.44 triangleTopography

`QVector<tools::triangleToDisplay> triangleTopography`

Triangles 3D de topographie.

8.13.1.45 triangleTopography2

`QVector<tools::triangleToDisplay> triangleTopography2`

Copie de triangles 3D de topographie.

8.13.1.46 Up

`QVector3D Up`

Vecteur 3D pour la position en haut de la caméra.

8.13.1.47 View

`QVector3D View`

Vecteur 3D pour la vue de la caméra (où l'on pointe)

8.13.1.48 vitesse

`int vitesse`

Vitesse de simulation.

8.13.1.49 workFolder

`QString workFolder`

Chemin de travail global.

8.13.1.50 x0Elev

`double x0Elev`

Coordonnée en x du coin inférieur gauche de la grille.

8.13.1.51 y0Elev

`double y0Elev`

Coordonnée en y du coin inférieur gauche de la grille.

8.13.1.52 zMaximum

`float zMaximum`

Valeur maximale pour l'élévation.

8.13.1.53 zMinimum

`float zMinimum`

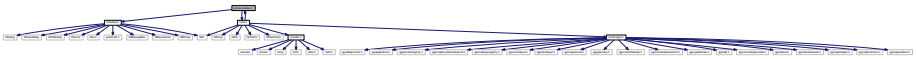
Valeur minimale pour l'élévation.

8.14 Référence du fichier globalvariables.h

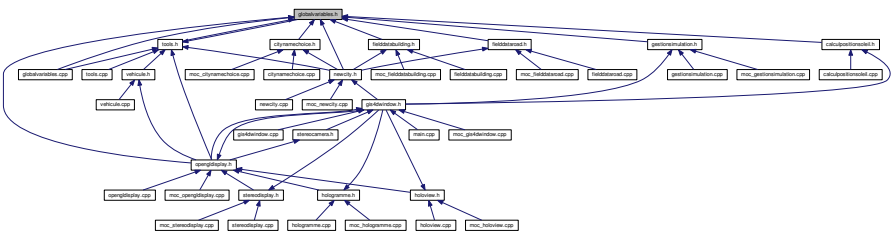
Variables globales pour le fonctionnement du logiciel.

```
#include <includeqt.h>
#include <tools.h>
```

Graphe des dépendances par inclusion de globalvariables.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class globalvariables

Variables

— QString messagePosQgis
Message PosGis pour la barre de statut.
— bool cityLoaded
Booléen pour savoir si la ville est chargée.
— bool existingCity
Booléen pour savoir si la ville est une ville déjà existante dans la bdd.
— QString workFolder
Chemin de travail global.
— QString pathBuilding
— QString pathRoad
— QString pathTree
Chemin du fichier .shp des arbres.
— QString pathGrass
Chemin du fichier .shp des zones herbacées.
— QString pathWater
Chemin du fichier .shp des surface d'eau.
— QString pathTopo
Chemin du fichier .asc de la topographie.
— QString save3DpathTopo
Chemin du fichier .citytrafic3D contenant les triangles de la topographie.
— QString save3DpathWall
Chemin du fichier .citytrafic3D contenant les triangles des façades des bâtiments.
— QString save3DpathRoof
Chemin du fichier .citytrafic3D contenant les triangles de la route.
— int defaultOrNotBulding
Valeur pour savoir si la donnée de hauteur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).
— int defaultOrNotRoad
Valeur pour savoir si la donnée de largeur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).
— int fieldNumHeightBuilding
Id du champ contenant la hauteur de chaque bâtiment.
— int fieldNumWidthRoad
Id du champ contenant la largeur de chaque tronçon de route.
— int fieldUniqueRoad
Id du champ contenant le sens unique ou non de chaque tronçon de route.
— float defaultHeight
Valeur par défaut de la hauteur des bâtiments.
— float defaultWidth
Valeur par défaut de la largeur des routes.
— QString cityName
Nom de la ville importée.
— QVector< QVector< float > > elevation3D
Grille d'élevation.
— float zMinimum
Valeur minimale pour l'élévation.
— float zMaximum
Valeur maximale pour l'élévation.
— double dXElev
Largeur du pixel de référence de la grille.
— double dYElev

Hauteur du pixel de référence de la grille.
— double x0Elev
Coordonnée en x du coin inférieur gauche de la grille.
— double y0Elev
Coordonnée en y du coin inférieur gauche de la grille.
— int nRowsElev
Nombre de lignes de la grille.
— int nColsElev
Nombre de colonnes de la grille.
— QVector3D View
Vecteur 3D pour la vue de la caméra (où l'on pointe)
— QVector3D Position
Vecteur 3D pour la position de la caméra.
— QVector3D Up
Vecteur 3D pour la position en haut de la caméra.
— QVector3D Direction
Vecteur 3D pour la direction de la caméra.
— bool dispBuilding
Affichage ou non des bâtiments.
— bool dispRoad
Affichage ou non des routes.
— bool dispTopo
Affichage ou non de la topographie.
— bool dispStereo
Affichage ou non de la vue stérographique (cyan et rouge)
— bool dispSky
Affichage ou non du ciel.
— bool dispWireMode
Affichage ou non du mode filaire.
— bool dispLight
Affichage ou non de la lumière.
— double positionSoleilX
Position du soleil en X.
— double positionSoleilY
Position du soleil en Y.
— double positionSoleilZ
Position du soleil en Z.
— double positionSoleilAzimut
Position du soleil en azimut.
— int nbVehicule
Nombre de véhicule.
— bool modeCollision
Mode collision activé ou non.
— int vitesse
Vitesse de simulation.

8.14.1 Description détaillée

Variables globales pour le fonctionnement du logiciel.

Auteur

Julien Richard

8.14.2 Documentation des variables

8.14.2.1 cityLoaded

```
bool cityLoaded
```

Booléen pour savoir si la ville est chargée.

8.14.2.2 cityName

```
QString cityName
```

Nom de la ville importée.

8.14.2.3 defaultHeight

```
float defaultHeight
```

Valeur par défaut de la hauteur des bâtiments.

8.14.2.4 defaultOrNotBulding

```
int defaultOrNotBulding
```

Valeur pour savoir si la donnée de hauteur est connue ou pas (1 : défaut, 2 : valeur du champs contenant la valeur).

8.14.2.5 defaultOrNotRoad

```
int defaultOrNotRoad
```

Valeur pour savoir si la donnée de largeur est connue ou pas (1 : défaut, 2 : valeur du champ contenant la valeur).

8.14.2.6 defaultWidth

```
float defaultWidth
```

Valeur par défaut de la largeur des routes.

8.14.2.7 Direction

`QVector3D Direction`

Vecteur 3D pour la direction de la caméra.

Vecteur 3D pour la direction de la caméra.

8.14.2.8 dispBuilding

`bool dispBuilding`

Affichage ou non des bâtiments.

8.14.2.9 dispLight

`bool dispLight`

Affichage ou non de la lumière.

8.14.2.10 dispRoad

`bool dispRoad`

Affichage ou non des routes.

8.14.2.11 dispSky

`bool dispSky`

Affichage ou non du ciel.

8.14.2.12 dispStereo

`bool dispStereo`

Affichage ou non de la vue stérographique (cyan et rouge)

8.14.2.13 dispTopo

`bool dispTopo`

Affichage ou non de la topographie.

8.14.2.14 dispWireMode

`bool dispWireMode`

Affichage ou non du mode filaire.

8.14.2.15 dXElev

`double dXElev`

Largeur du pixel de référence de la grille.

8.14.2.16 dYElev

`double dYElev`

Hauteur du pixel de référence de la grille.

8.14.2.17 elevation3D

`QVector<QVector <float> > elevation3D`

Grille d'élévation.

8.14.2.18 existingCity

`bool existingCity`

Booléen pour savoir si la ville est une ville déjà existante dans la bdd.

8.14.2.19 fieldNumHeightBuilding

`int fieldNumHeightBuilding`

Id du champ contenant la hauteur de chaque bâtiment.

8.14.2.20 fieldNumWidthRoad

`int fieldNumWidthRoad`

Id du champ contenant la largeur de chaque tronçon de route.

8.14.2.21 fieldUniqueRoad

`int fieldUniqueRoad`

Id du champ contenant le sens unique ou non de chaque tronçon de route.

8.14.2.22 messagePosQgis

`QString messagePosQgis`

Message PosGis pour la barre de statut.

8.14.2.23 modeCollision

`bool modeCollision`

Mode collision activé ou non.

8.14.2.24 nbVehicule

`int nbVehicule`

Nombre de véhicules.

8.14.2.25 nColsElev

`int nColsElev`

Nombre de colonnes de la grille.

8.14.2.26 nRowsElev

`int nRowsElev`

Nombre de lignes de la grille.

8.14.2.27 pathBuilding

`QString pathBuilding`

du fichier .shp des bâtiments.

8.14.2.28 pathGrass

`QString pathGrass`

Chemin du fichier .shp des zones herbacées.

8.14.2.29 pathRoad

`QString pathRoad`

du fichier .shp des routes.

8.14.2.30 pathTopo

`QString pathTopo`

Chemin du fichier .asc de la topographie.

8.14.2.31 pathTree

`QString pathTree`

Chemin du fichier .shp des arbres.

8.14.2.32 pathWater

`QString pathWater`

Chemin du fichier .shp des surface d'eau.

8.14.2.33 Position

`QVector3D Position`

Vecteur 3D pour la position de la caméra.

8.14.2.34 positionSoleilAzimut

`double positionSoleilAzimut`

Position du soleil en azimut.

8.14.2.35 positionSoleilX

`double positionSoleilX`

Position du soleil en X.

8.14.2.36 positionSoleilY

`double positionSoleilY`

Position du soleil en Y.

8.14.2.37 positionSoleilZ

`double positionSoleilZ`

Position du soleil en Z.

8.14.2.38 save3DpathRoof

`QString save3DpathRoof`

Chemin du fichier .citytrafic3D contenant les triangles de la route.

8.14.2.39 save3DpathTopo

`QString save3DpathTopo`

Chemin du fichier .citytrafic3D contenant les triangles de la topographie.

8.14.2.40 save3DpathWall

`QString save3DpathWall`

Chemin du fichier .citytrafic3D contenant les triangles des façades des bâtiments.

8.14.2.41 Up

`QVector3D Up`

Vecteur 3D pour la position en haut de la caméra.

8.14.2.42 View

`QVector3D View`

Vecteur 3D pour la vue de la caméra (où l'on pointe)

8.14.2.43 vitesse

`int vitesse`

Vitesse de simulation.

8.14.2.44 workFolder

`QString workFolder`

Chemin de travail global.

8.14.2.45 x0Elev

`double x0Elev`

Coordonnée en x du coin inférieur gauche de la grille.

8.14.2.46 y0Elev

`double y0Elev`

Coordonnée en y du coin inférieur gauche de la grille.

8.14.2.47 zMaximum

`float zMaximum`

Valeur maximale pour l'élévation.

8.14.2.48 zMinimum

`float zMinimum`

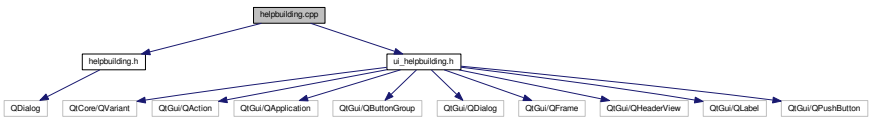
Valeur minimale pour l'élévation.

8.15 Référence du fichier helpbuilding.cpp

Fenêtre d'aide pour l'import des bâtiments.

```
#include "helpbuilding.h"
#include "ui_helpbuilding.h"
```

Graphe des dépendances par inclusion de helpbuilding.cpp :



8.15.1 Description détaillée

Fenêtre d'aide pour l'import des bâtiments.

Auteur

Julien Richard

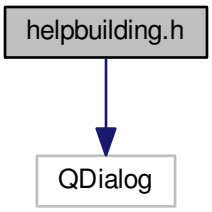
Les données de bâti se doivent d'être en format polygone avec si possible un attribut de hauteur. Si cet attribut n'existe pas l'utilisateur aura le choix d'une valeur par défaut.

8.16 Référence du fichier helpbuilding.h

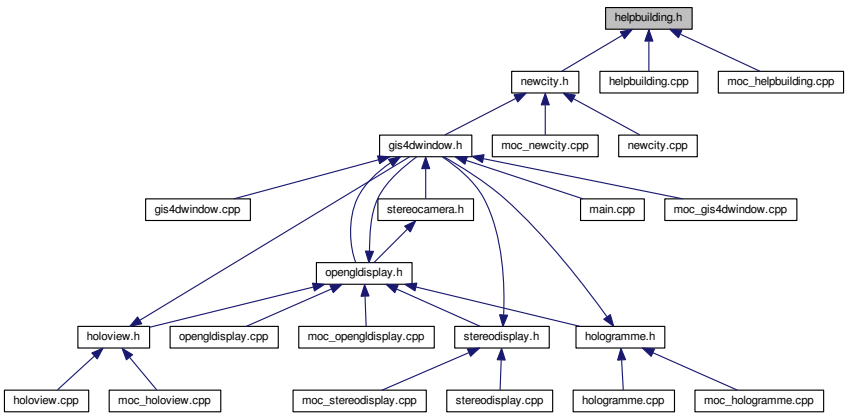
Fenêtre d'aide concernant l'import de la données des bâtiments.

```
#include <QDialog>
```

Graphe des dépendances par inclusion de helpbuilding.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class helpBuilding

Espaces de nommage

— Ui

8.16.1 Description détaillée

Fenêtre d'aide concernant l'import de la données des bâtiments.

Auteur

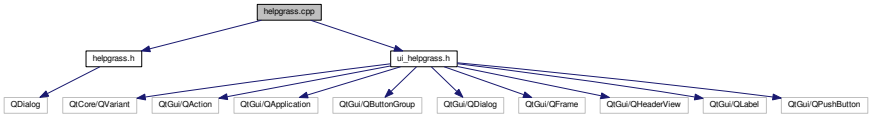
Julien Richard

8.17 Référence du fichier helpgrass.cpp

Fenêtre d'aide pour l'import des zones herbacées.

```
#include "helpgrass.h"
#include "ui_helpgrass.h"
```

Graphe des dépendances par inclusion de helpgrass.cpp :



8.17.1 Description détaillée

Fenêtre d'aide pour l'import des zones herbacées.

Auteur

Julien Richard

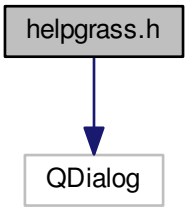
Les données de zones herbacées se doivent d'être en format polygone. Aucun attribut n'est nécessaire.

8.18 Référence du fichier helpgrass.h

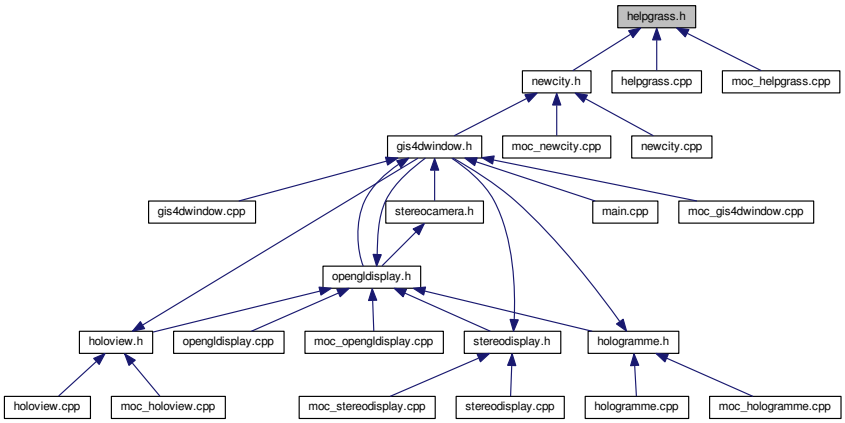
Fenêtre d'aide concernant l'import de la données des zones herbacées.

```
#include <QDialog>
```

Graphe des dépendances par inclusion de helpgrass.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class helpGrass

Espaces de nommage

— Ui

8.18.1 Description détaillée

Fenêtre d'aide concernant l'import de la données des zones herbacées.

Auteur

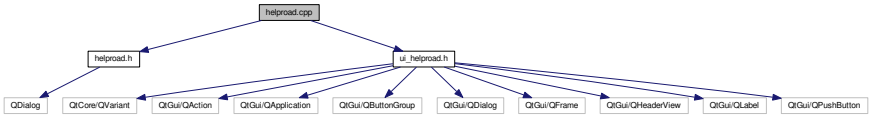
Julien Richard

8.19 Référence du fichier helproad.cpp

Fenêtre d'aide pour l'import des routes.

```
#include "helproad.h"
#include "ui_helproad.h"
```

Graphe des dépendances par inclusion de helproad.cpp :



8.19.1 Description détaillée

Fenêtre d'aide pour l'import des bâtiments.

Auteur

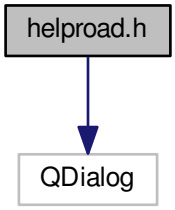
Julien Richard

Les données de routes se doivent d'être en format ligne avec si possible un attribut de largeur. Si cet attribut n'existe pas l'utilisateur aura le choix d'une valeur par défaut.

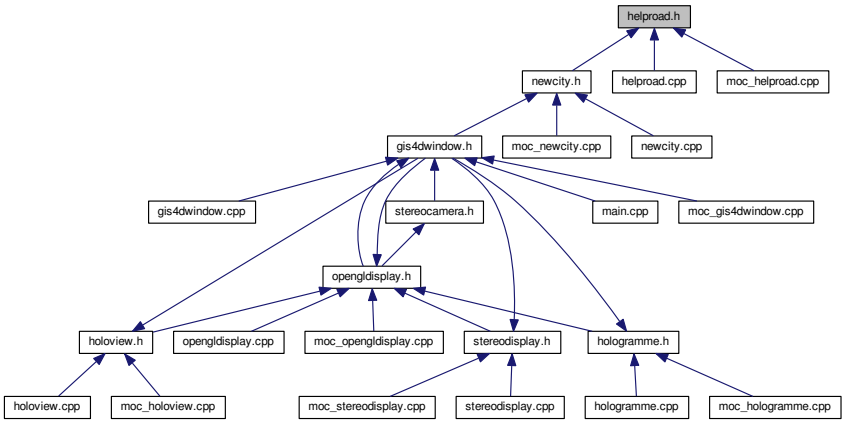
8.20 Référence du fichier helproad.h

Fenêtre d'aide concernant l'import de la données des routes.

`#include <QDialog>`
Graphe des dépendances par inclusion de helproad.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class helpRoad

Espaces de nommage

— Ui

8.20.1 Description détaillée

Fenêtre d'aide concernant l'import de la données des routes.
Auteur

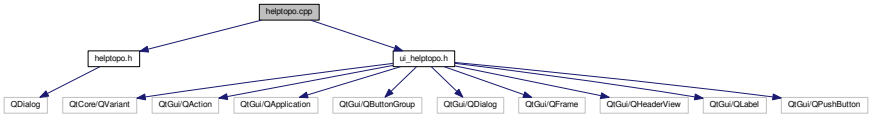
Julien Richard

8.21 Référence du fichier helptopo.cpp

Fenêtre d'aide pour l'import de la topographie.

```
#include "helptopo.h"
#include "ui_helptopo.h"
```

Graphe des dépendances par inclusion de helptopo.cpp :



8.21.1 Description détaillée

Fenêtre d'aide pour l'import de la topographie.
Auteur

Julien Richard

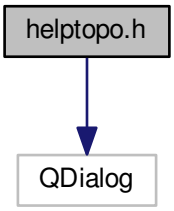
Les données d'élévation se doivent d'être en format grille (.asc).

8.22 Référence du fichier helptopo.h

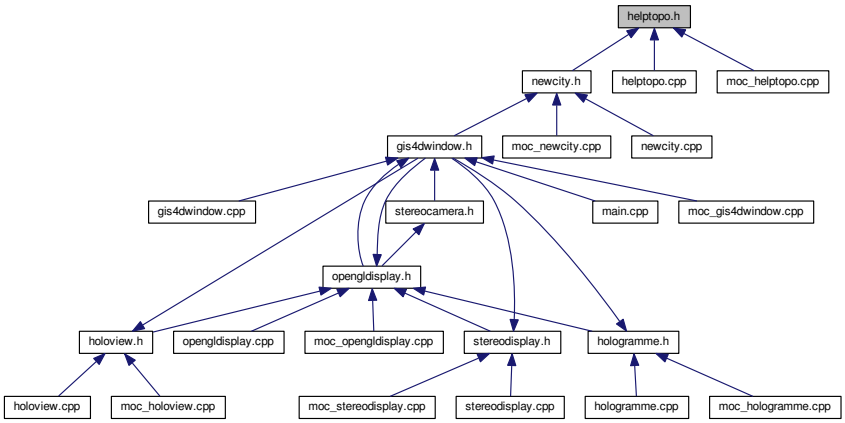
Fenêtre d'aide concernant l'import de la topographie.

```
#include <QDialog>
```

Graphe des dépendances par inclusion de helptopo.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class helpTopo

Espaces de nommage

— Ui

8.22.1 Description détaillée

Fenêtre d'aide concernant l'import de la topographie.

Auteur

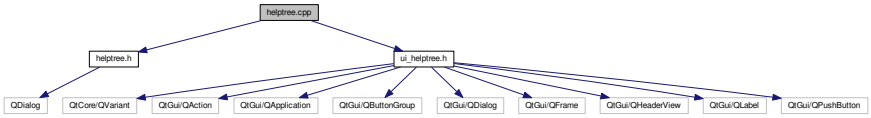
Julien Richard

8.23 Référence du fichier helptree.cpp

Fenêtre d'aide pour l'import des zones arborées.

```
#include "helptree.h"
#include "ui_helptree.h"
```

Graphe des dépendances par inclusion de helptree.cpp :



8.23.1 Description détaillée

Fenêtre d'aide pour l'import des zones arborées.

Auteur

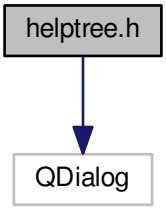
Julien Richard

Les données de zones arborées se doivent d'être en format polygone. Aucun attribut n'est nécessaire.

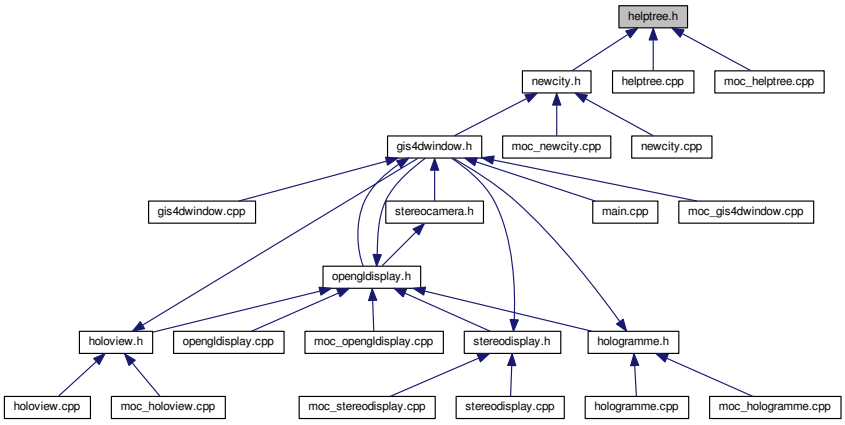
8.24 Référence du fichier helptree.h

Fenêtre d'aide concernant l'import de la données des arbres.

`#include <QDialog>`
Graphe des dépendances par inclusion de helptree.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class helpTree

Espaces de nommage

— Ui

8.24.1 Description détaillée

Fenêtre d'aide concernant l'import de la données des arbres.

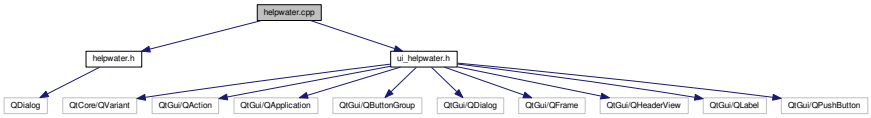
Auteur

Julien Richard

8.25 Référence du fichier helpwater.cpp

```
#include "helpwater.h"
#include "ui_helpwater.h"
```

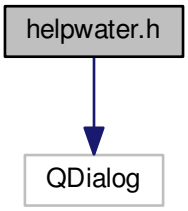
Graphe des dépendances par inclusion de helpwater.cpp :



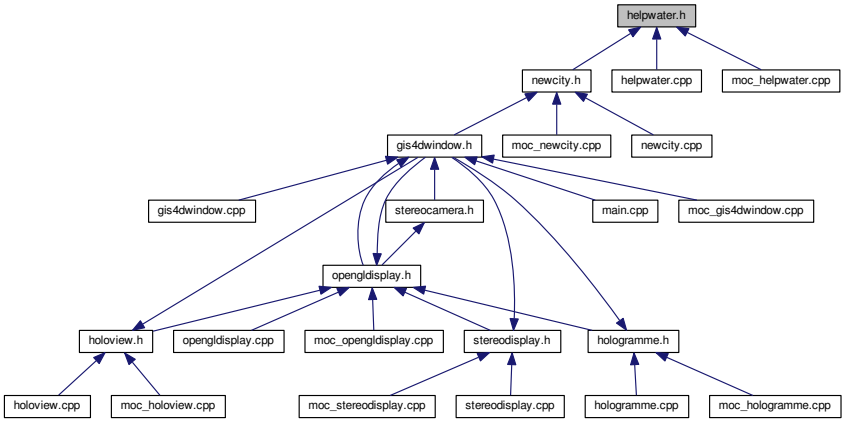
8.26 Référence du fichier helpwater.h

```
#include <QDialog>
```

Graphe des dépendances par inclusion de helpwater.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class helpWater

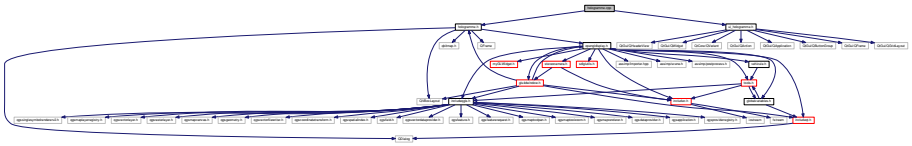
Espaces de nommage

— Ui

8.27 Référence du fichier hologramme.cpp

```
#include "hologramme.h"
#include "ui_hologramme.h"
```

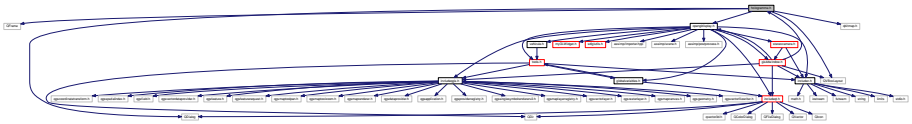
Graphe des dépendances par inclusion de hologramme.cpp :



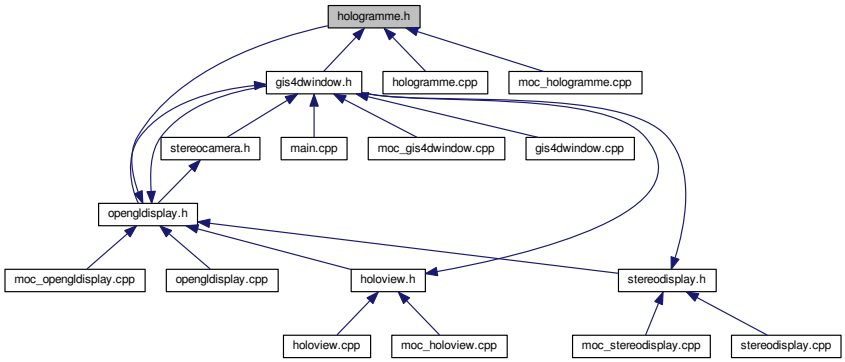
8.28 Référence du fichier hologramme.h

```
#include <QFrame>
#include <QDialog>
#include <QVBoxLayout>
#include <opengldisplay.h>
```

```
#include <qbitmap.h>
```
Graphe des dépendances par inclusion de hologramme.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class hologramme

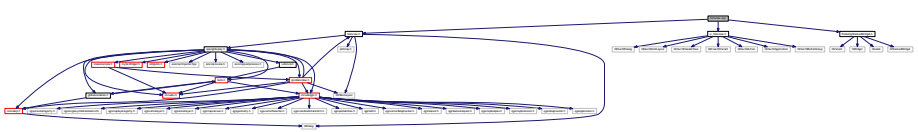
Espaces de nommage

— Ui

8.29 Référence du fichier holoview.cpp

Fenêtre avec 4 vues avec différentes rotations pour une vue de type holographique.

```
#include "holoview.h"
#include "ui_holoview.h"
#include "RotatingStackedWidget.h"
```
Graphe des dépendances par inclusion de holoview.cpp :



8.29.1 Description détaillée

Fenêtre avec 4 vues avec différentes rotations pour une vue de type holographique.

Auteur

Julien Richard

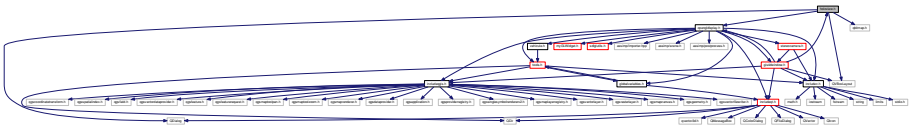
Les 4 vues perpendiculaires l'une de l'autre permettent de poser sur un écran à mis plat un tétraèdre transparent avec des pentes à 45° afin d'avoir un reflet sur chacune des faces

8.30 Référence du fichier holoview.h

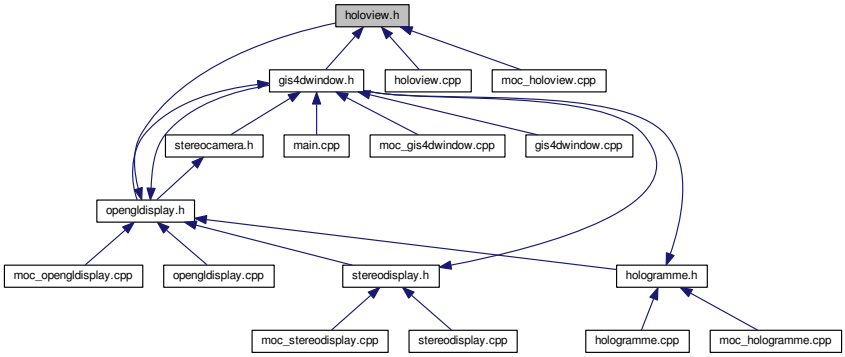
Fenêtre avec 4 vue 3D dessinées perpendiculairement les unes autres autres. Afin d'afficher la vue de façon holographique, il faut poser sur l'écran un tétraèdre tronqué à son sommet sur un écran posé à plat. Les vues 3D sont alors projetées par reflet sur les surface à 45° créant ainsi l'illusion d'hologramme.

```
#include <QDialog>
#include <QVBoxLayout>
#include <opengldisplay.h>
#include <qbitmap.h>
```

Graphe des dépendances par inclusion de holoview.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class holoView

Espaces de nommage

— Ui

8.30.1 Description détaillée

Fenêtre avec 4 vue 3D dessinées perpendiculairement les unes autres autres. Afin d'afficher la vue de façon holographique, il faut poser sur l'écran un tétraèdre tronque à son sommet sur un écran posé à plat. Les vues 3D sont alors projetées par reflet sur les surface à 45° créant ainsi l'illusion d'hologramme.

Auteur

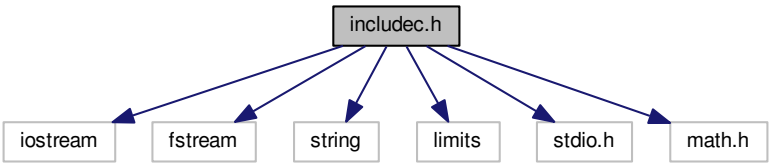
Julien Richard

8.31 Référence du fichier includec.h

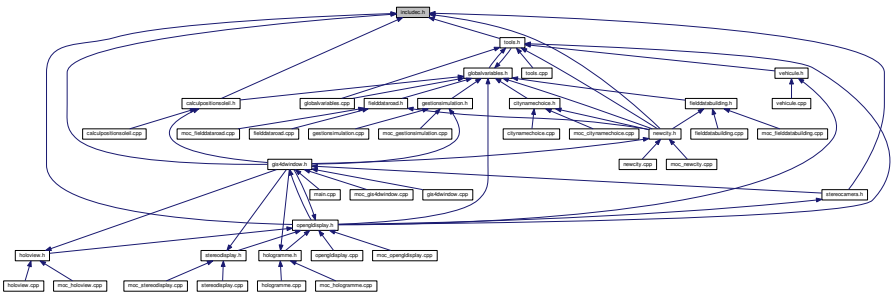
Liste d'include des fonctionnalités en C.

```
#include <iostream>
#include <fstream>
#include <string>
#include <limits>
#include <stdio.h>
#include <math.h>
```

Graphe des dépendances par inclusion de includec.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.31.1 Description détaillée

Liste d'include des fonctionnalités en C.

Auteur

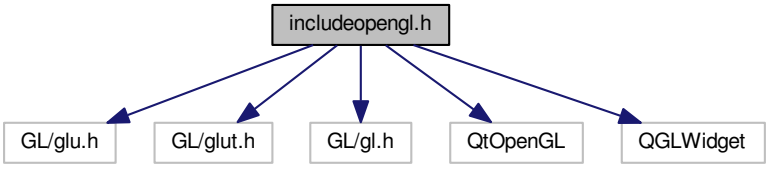
Julien Richard

8.32 Référence du fichier includeopengl.h

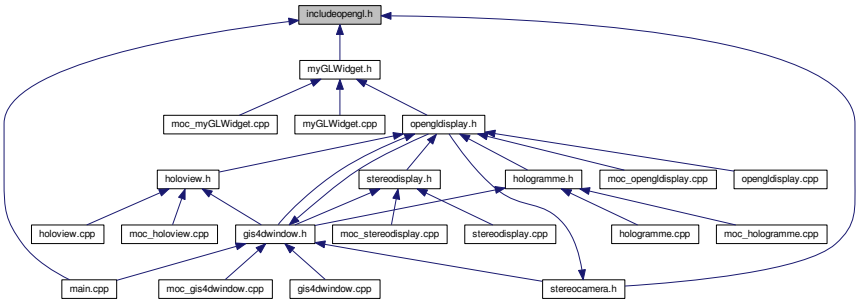
Liste d'include des fonctionnalités OpenGl.

```
#include <GL/glu.h>
#include <GL/glut.h>
#include <GL/gl.h>
#include <QtOpenGL>
#include <QGLWidget>
```

Graphe des dépendances par inclusion de includeopengl.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.32.1 Description détaillée

Liste d'include des fonctionnalités OpenGl.

Auteur

Julien Richard

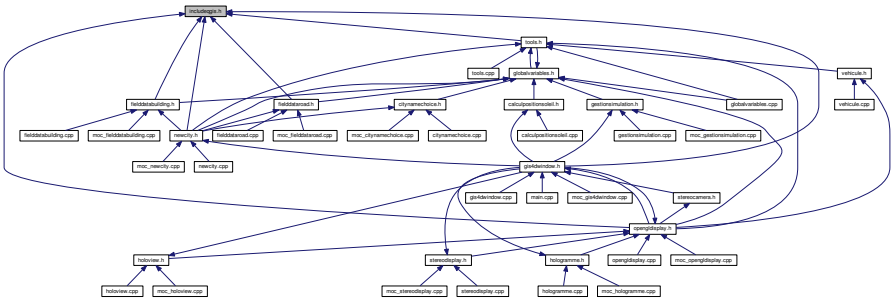
8.33 Référence du fichier includeqgis.h

Liste d'include des fonctionnalités QGIS.

```
#include <qgsdataprovider.h>
#include <qgsapplication.h>
#include <qgsproviderregistry.h>
#include <qgssinglesymbolrendererv2.h>
#include <qgsmaplayerregistry.h>
#include <qgsvectorlayer.h>
#include <qgsrasterlayer.h>
#include <qgsmapcanvas.h>
#include <qgsgeometry.h>
#include <qgsvectorfilewriter.h>
#include <qgscoordinatetransform.h>
#include <qgsspatialindex.h>
#include <qgsfield.h>
#include <qgsvectordataprovider.h>
#include <qgsfeature.h>
#include <qgsfeaturerequest.h>
#include "qgsmaptoolpan.h"
#include "qgsmaptoolzoom.h"
#include "qgsmaprenderer.h"
```

Graphe des dépendances par inclusion de includeqgis.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.33.1 Description détaillée

Liste d'include des fonctionnalités QGIS.

Auteur

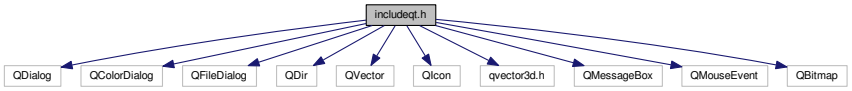
Julien Richard

8.34 Référence du fichier includeqt.h

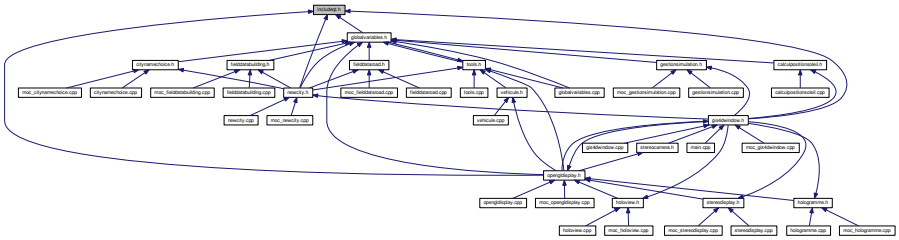
Liste d'include des fonctionnalités Qt.

```
#include <QDialog>
#include <QColorDialog>
#include <QFileDialog>
#include <QDir>
#include <QVector>
#include <QIcon>
#include <qvector3d.h>
#include <QMessageBox>
#include <QMouseEvent>
#include <QBitmap>
```

Graphe des dépendances par inclusion de includeqt.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



8.34.1 Description détaillée

Liste d'include des fonctionnalités Qt.

Auteur

Julien Richard

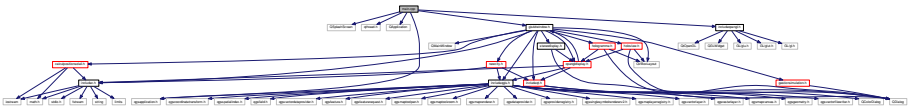
8.35 Référence du fichier main.cpp

Fichier de lancement du programme Traffic City 3D.

```
#include <QSplashScreen>
#include <qthread.h>
#include <QApplication>
#include <qgsapplication.h>
#include <includeopengl.h>
```

```
#include <gis4dwindow.h>
```

Graphe des dépendances par inclusion de main.cpp :



Classes

— class I

Fonctions

— int main (int argc, char **argv)

8.35.1 Description détaillée

Fichier de lancement du programme Traffic City 3D.

Auteur

Julien Richard

On trouvera ici le lancement de la fenêtre de chargement suivi de l'affichage de la fenêtre principale du logiciel gis4DWindow.

8.35.2 Documentation des fonctions

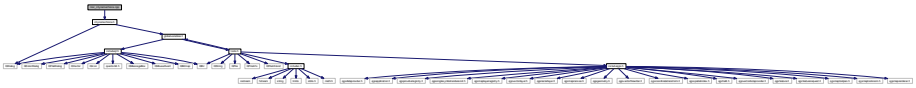
8.35.2.1 main()

```
int main (
            int argc,
            char ** argv )
```

Voici le graphe d'appel pour cette fonction :

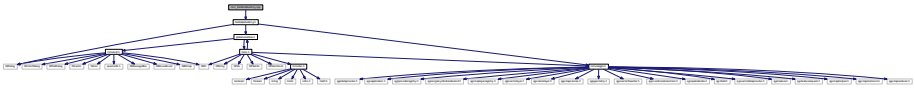
8.36 Référence du fichier moc_citynamechoice.cpp

`#include "citynamechoice.h"`
Graphe des dépendances par inclusion de moc_citynamechoice.cpp :



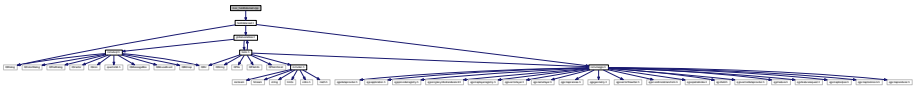
8.37 Référence du fichier moc_fielddatabuilding.cpp

`#include "fielddatabuilding.h"`
Graphe des dépendances par inclusion de moc_fielddatabuilding.cpp :



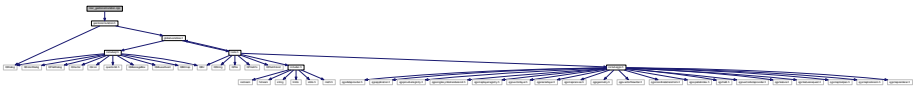
8.38 Référence du fichier moc_fielddataroad.cpp

`#include "fielddataroad.h"`
Graphe des dépendances par inclusion de moc_fielddataroad.cpp :



8.39 Référence du fichier moc_gestionsimulation.cpp

`#include "gestionsimulation.h"`
Graphe des dépendances par inclusion de moc_gestionsimulation.cpp :



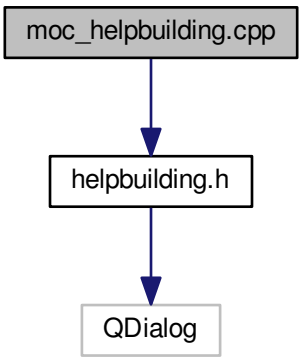
8.40 Référence du fichier moc_gis4dwindow.cpp

`#include "gis4dwindow.h"`
Graphe des dépendances par inclusion de moc_gis4dwindow.cpp :

8.41 Référence du fichier moc_helpbuilding.cpp

```
#include "helpbuilding.h"
```

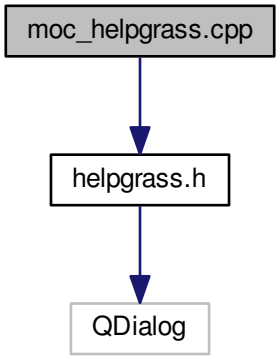
Graphe des dépendances par inclusion de moc_helpbuilding.cpp :



8.42 Référence du fichier moc_helpgrass.cpp

```
#include "helpgrass.h"
```

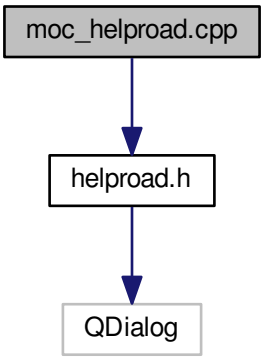
Graphe des dépendances par inclusion de moc_helpgrass.cpp :



8.43 Référence du fichier moc_helproad.cpp

```
#include "helproad.h"
```

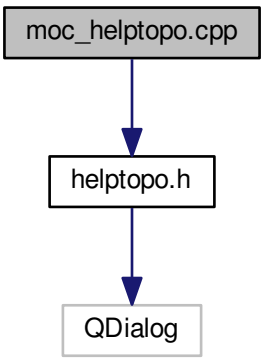
Graphe des dépendances par inclusion de moc_helproad.cpp :



8.44 Référence du fichier moc_helptopo.cpp

```
#include "helptopo.h"
```

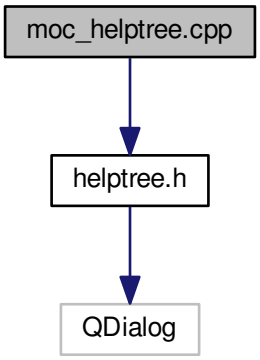
Graphe des dépendances par inclusion de moc_helptopo.cpp :



8.45 Référence du fichier moc_helptree.cpp

```
#include "helptree.h"
```

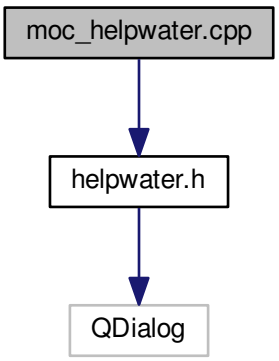
Graphe des dépendances par inclusion de moc_helptree.cpp :



8.46 Référence du fichier moc_helpwater.cpp

```
#include "helpwater.h"
```

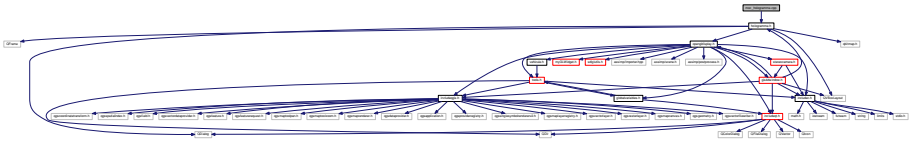
Graphe des dépendances par inclusion de moc_helpwater.cpp :



8.47 Référence du fichier moc_hologramme.cpp

```
#include "hologramme.h"
```

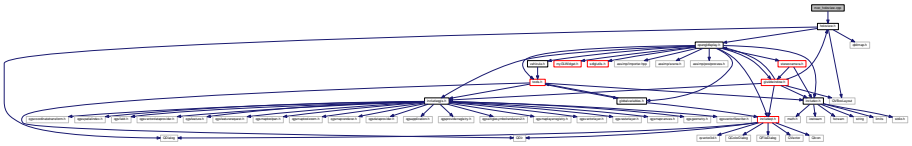
Graphe des dépendances par inclusion de moc_hologramme.cpp :



8.48 Référence du fichier moc_holoview.cpp

```
#include "holoview.h"
```

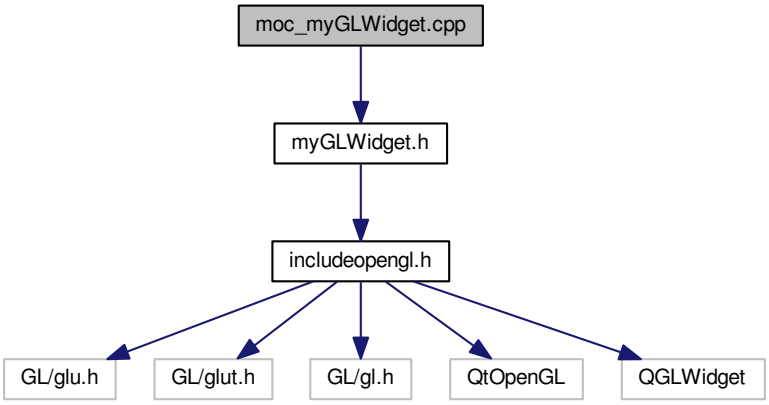
Graphe des dépendances par inclusion de moc_holoview.cpp :



8.49 Référence du fichier moc_myGLWidget.cpp

```
#include "myGLWidget.h"
```

Graphe des dépendances par inclusion de moc_myGLWidget.cpp :



8.50 Référence du fichier moc_newcity.cpp

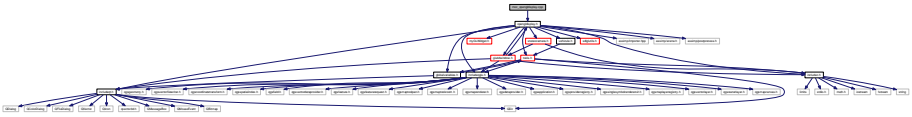
```
#include "newcity.h"
```

Graphe des dépendances par inclusion de moc_newcity.cpp :

8.51 Référence du fichier moc_opengldisplay.cpp

```
#include "opengldisplay.h"
```

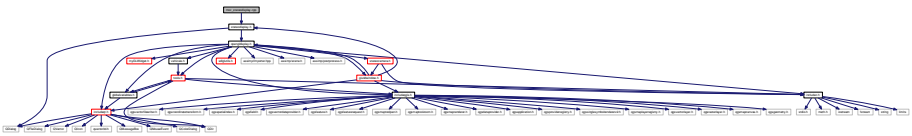
Graphe des dépendances par inclusion de moc_opengldisplay.cpp :



8.52 Référence du fichier moc_stereodisplay.cpp

```
#include "stereodisplay.h"
```

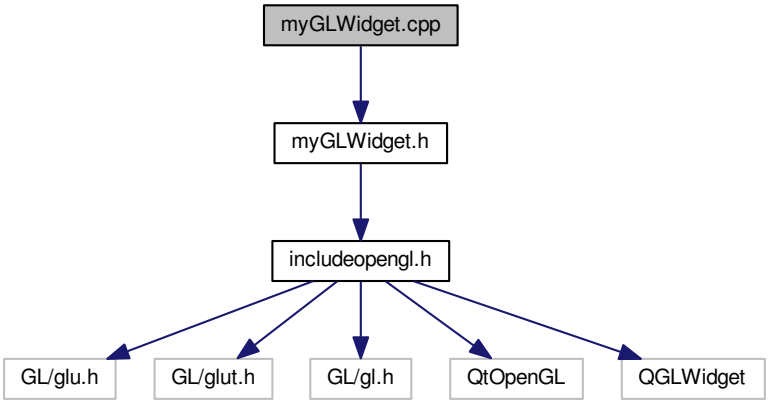
Graphe des dépendances par inclusion de moc_stereodisplay.cpp :



8.53 Référence du fichier myGLWidget.cpp

Classe pour l'activation d'Opengl dans la fenêtre 3D.

`#include "myGLWidget.h"`
Graphe des dépendances par inclusion de myGLWidget.cpp :



8.53.1 Description détaillée

Classe pour l'activation d'Opengl dans la fenêtre 3D.

Auteur

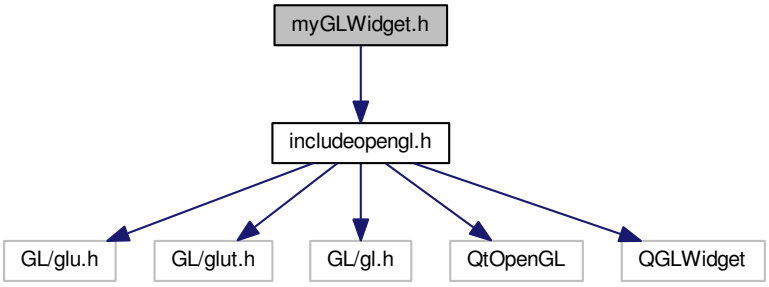
Julien Richard

Initialisation d'OpenGl.

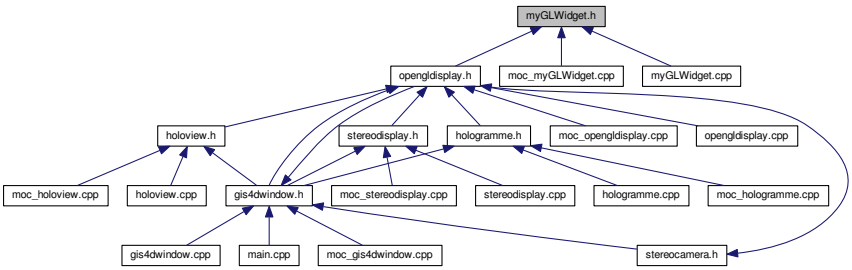
8.54 Référence du fichier myGLWidget.h

Fenêtre OpenGl.

`#include <includeopengl.h>`
Graphe des dépendances par inclusion de myGLWidget.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class myGLWidget

8.54.1 Description détaillée

Fenêtre OpenGl.

Auteur

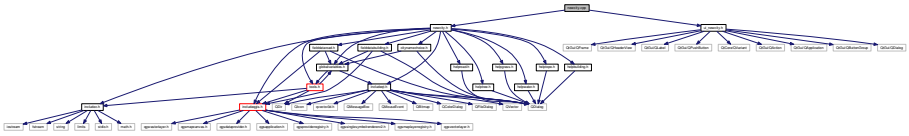
Julien RICHARD

8.55 Référence du fichier newcity.cpp

Fenêtre pour l'import de nouvelles données.

```
#include "newcity.h"
#include "ui_newcity.h"
```

Graphe des dépendances par inclusion de newcity.cpp :



8.55.1 Description détaillée

Fenêtre pour l'import de nouvelles données.

Auteur

Julien Richard

Cette fenêtre permet à un utlisateur d'importer de nouvelles données dans le but de créer une nouvelle ville en 3D. 6 boutons sont présents pour 6 données différentes : le bâti, les routes, les arbres, les zones d'herbes, les surface d'eaux, et la topographie. Pour chacune des catégories de donnée un bouton d'aide est disponible pour expliquer ce que le logiciel neccessite et aussi comment se procurer cette donnée.

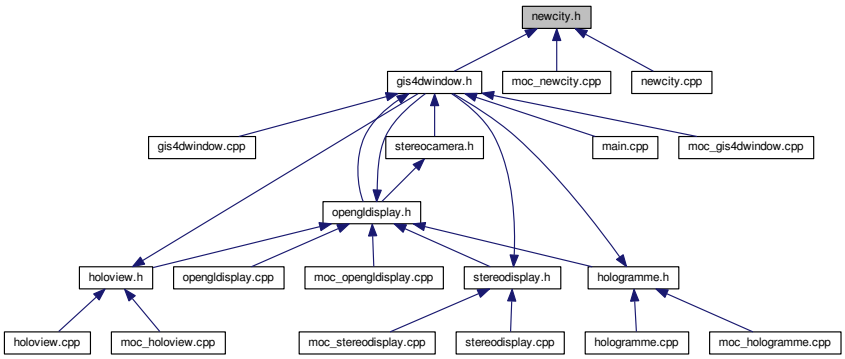
8.56 Référence du fichier newcity.h

Fenêtre pour l'import des données d'une nouvelle ville. L'utilisateur devra renseigner les chemins des données pour pouvoir avoir une génération de la ville en 3D.

```
#include <includeqt.h>
#include <includec.h>
#include <includeqgis.h>
#include <fielddatabuilding.h>
#include <fielddataroad.h>
#include <helpbuilding.h>
#include <helproad.h>
#include <helptree.h>
#include <helpgrass.h>
#include <helpwater.h>
#include <helptopo.h>
#include <globalvariables.h>
#include <citynamechoice.h>
#include <tools.h>
```

Graphe des dépendances par inclusion de newcity.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class newCity

Espaces de nommage

— Ui

8.56.1 Description détaillée

Fenêtre pour l'import des données d'une nouvelle ville. L'utilisateur devra renseigner les chemins des données pour pouvoir avoir une génération de la ville en 3D.

Auteur

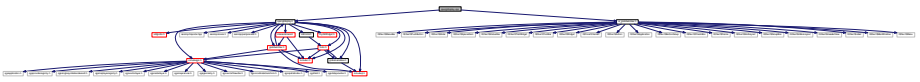
Julien Richard

8.57 Référence du fichier opengldisplay.cpp

Widget pour la vue 3D.

```
#include "opengldisplay.h"
#include "ui_gis4dwindow.h"
```

Graphe des dépendances par inclusion de opengldisplay.cpp :



Macros

— #define PI 3.14159265

Variables

— double unTiers = 1.0/3.0
— double deuxTiers = 2.0/3.0
— int MatSpec [4] = {1,1,1,1}

8.57.1 Description détaillée

Widget pour la vue 3D.

Auteur

Julien Richard

La vue 3D est générée ici. Il y a aussi toute la partie sur la gestion de la caméra et le chargement des données.

8.57.2 Documentation des macros

8.57.2.1 PI

```
#define PI 3.14159265
```

8.57.3 Documentation des variables

8.57.3.1 deuxTiers

```
double deuxTiers = 2.0/3.0
```

8.57.3.2 MatSpec

```
int MatSpec[4] = {1,1,1,1}
```

8.57.3.3 unTiers

```
double unTiers = 1.0/3.0
```

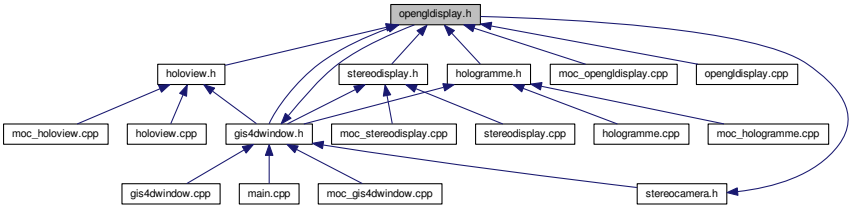
8.58 Référence du fichier opengldisplay.h

Fenêtre OpenGl et fonctions de conversions.

```
#include <includeqgis.h>
#include <includec.h>
#include <includeqt.h>
#include <myGLWidget.h>
#include <tools.h>
#include <gis4dwindow.h>
#include <stereocamera.h>
#include <sdlglutils.h>
#include <globalvariables.h>
#include <vehicule.h>
#include <assimp/Importer.hpp>
#include <assimp/scene.h>
#include <assimp/postprocess.h>
```

Graphe des dépendances par inclusion de opengldisplay.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class openGlDisplay
— struct openGlDisplay : :quaternion
Declaration d'un quaternion. Les quaternions sont utiles pour les rotations et les déplacements d'objets, ils nous permettent de nous affranchir des matrices de rotations qui sont bien plus gourmandes en terme de calculs.

8.58.1 Description détaillée

Fenêtre OpenGl et fonctions de conversions.

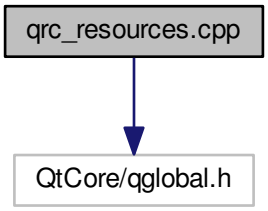
Auteur

Julien Richard

8.59 Référence du fichier qrc_resources.cpp

```
#include <QtCore/qglobal.h>
```

Graphe des dépendances par inclusion de qrc_resources.cpp :



Fonctions

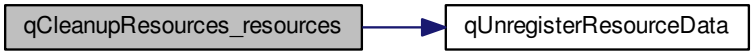
— QT_BEGIN_NAMESPACE Q_CORE_EXPORT bool qRegisterResourceData (int, const unsigned char *, const unsigned char *, const unsigned char *)
— Q_CORE_EXPORT bool qUnregisterResourceData (int, const unsigned char *, const unsigned char *, const unsigned char *)
— QT_END_NAMESPACE int QT_MANGLE_NAMESPACE() qInitResources_↩ resources ()
— int QT_MANGLE_NAMESPACE() qCleanupResources_resources ()

8.59.1 Documentation des fonctions

8.59.1.1 qCleanupResources_resources()

```
int QT_MANGLE_NAMESPACE() qCleanupResources_resources ( )
```

Voici le graphe d'appel pour cette fonction :



8.59.1.2 qInitResources_resources()

```
QT_END_NAMESPACE int QT_MANGLE_NAMESPACE() qInitResources_resources ( )
```

Voici le graphe d'appel pour cette fonction :

8.59.1.3 qRegisterResourceData()

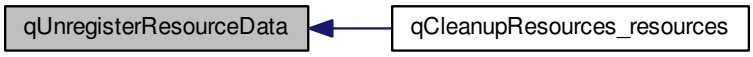
```
QT_BEGIN_NAMESPACE Q_CORE_EXPORT bool qRegisterResourceData (
            int ,
            const unsigned char * ,
            const unsigned char * ,
            const unsigned char *  )
```

Voici le graphe des appelants de cette fonction :

8.59.1.4 qUnregisterResourceData()

```
Q_CORE_EXPORT bool qUnregisterResourceData (
            int ,
            const unsigned char * ,
            const unsigned char * ,
            const unsigned char *  )
```

Voici le graphe des appelants de cette fonction :

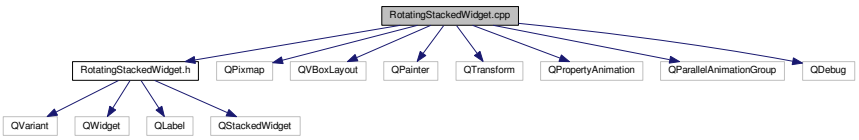


8.60 Référence du fichier README.md

8.61 Référence du fichier RotatingStackedWidget.cpp

```
#include "RotatingStackedWidget.h"
#include <QPixmap>
#include <QVBoxLayout>
#include <QPainter>
#include <QTransform>
#include <QPropertyAnimation>
#include <QParallelAnimationGroup>
#include <QDebug>
```

Graphe des dépendances par inclusion de RotatingStackedWidget.cpp :



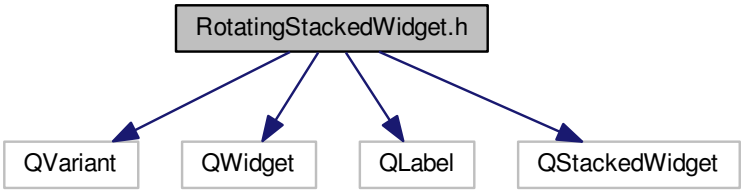
8.62 Référence du fichier RotatingStackedWidget.h

Gestion des rotations.

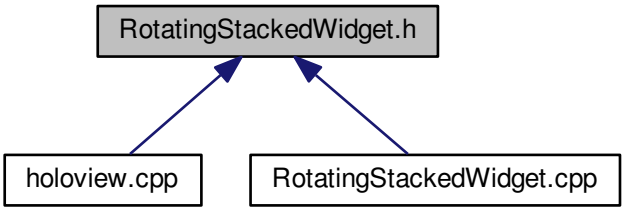
```
#include <QVariant>
#include <QWidget>
#include <QLabel>
```

```
#include <QStackedWidget>
```

Graphe des dépendances par inclusion de RotatingStackedWidget.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class RotatingStackedWidget

8.62.1 Description détaillée

Gestion des rotations.

Auteur

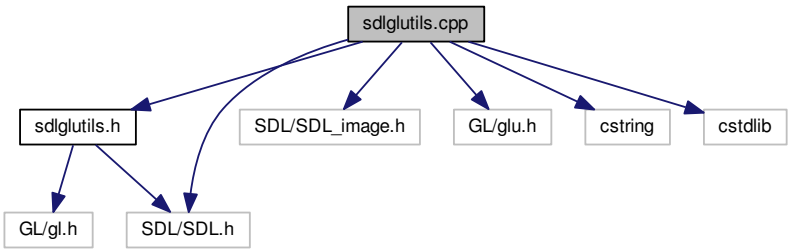
Abhijith Reddy

8.63 Référence du fichier sdlglutils.cpp

```
#include "sdlglutils.h"
#include <SDL/SDL.h>
#include <SDL/SDL_image.h>
#include <GL/glu.h>
#include <cstring>
```

```
#include <cstdlib>
```

Graphe des dépendances par inclusion de sdlglutils.cpp :



Fonctions

- SDL_Surface * flipSurface (SDL_Surface *surface)
- GLuint loadTexture (const char *filename, bool useMipMap)
- int takeScreenshot (const char *filename)
- void drawAxis (double scale)
- int initFullScreen (unsigned int *width, unsigned int *height)
- int XPMFromImage (const char *imagefile, const char *XPMfile)
- SDL_Cursor * cursorFromXPM (const char *xpm[])

8.63.1 Documentation des fonctions

8.63.1.1 cursorFromXPM()

```
SDL_Cursor* cursorFromXPM (
            const char * xpm[] )
```

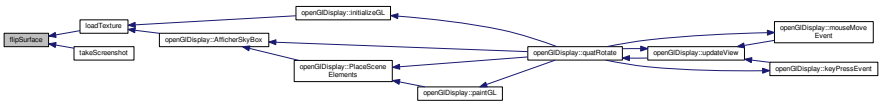
8.63.1.2 drawAxis()

```
void drawAxis (
            double scale )
```

8.63.1.3 flipSurface()

```
SDL_Surface * flipSurface (
            SDL_Surface * surface )
```

Voici le graphe des appelants de cette fonction :



8.63.1.4 initFullScreen()

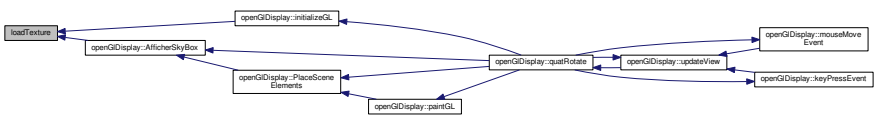
```
int initFullScreen (
            unsigned int * width,
            unsigned int * height )
```

8.63.1.5 loadTexture()

```
GLuint loadTexture (
            const char * filename,
            bool useMipMap )
```

Voici le graphe d'appel pour cette fonction :

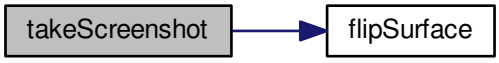
Voici le graphe des appelants de cette fonction :



8.63.1.6 takeScreenshot()

```
int takeScreenshot (
            const char * filename )
```

Voici le graphe d'appel pour cette fonction :



8.63.1.7 XPMFromImage()

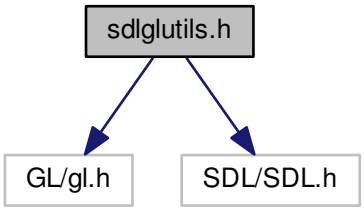
```
int XPMFromImage (
            const char * imagefile,
            const char * XPMfile )
```

8.64 Référence du fichier sdlglutils.h

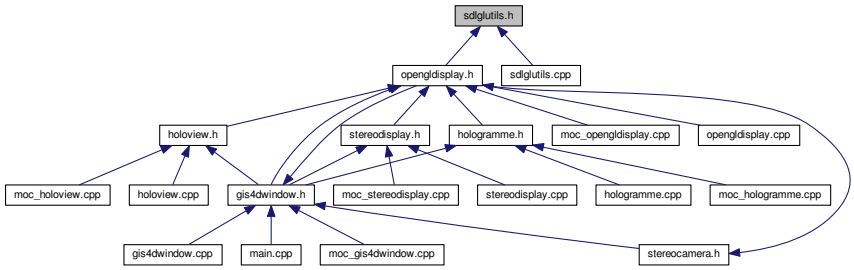
Outils pour la gestion des images/textures.

```
#include <GL/gl.h>
#include <SDL/SDL.h>
```

Graphe des dépendances par inclusion de sdlglutils.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Macros

— #define GL_CLAMP_TO_EDGE 0x812F

Fonctions

— GLuint loadTexture (const char *filename, bool useMipMap=true)
— int takeScreenshot (const char *filename)
— void drawAxis (double scale=1)
— int initFullScreen (unsigned int *width=NULL, unsigned int *height=NULL)
— int XPMFromImage (const char *imagefile, const char *XPMfile)
— SDL_Cursor * cursorFromXPM (const char *xpm[])

8.64.1 Description détaillée

Outils pour la gestion des images/textures.

Version

0.1

8.64.2 Documentation des macros

8.64.2.1 GL_CLAMP_TO_EDGE

```
#define GL_CLAMP_TO_EDGE 0x812F
```

8.64.3 Documentation des fonctions

8.64.3.1 cursorFromXPM()

```
SDL_Cursor* cursorFromXPM (
            const char * xpm[] )
```

8.64.3.2 drawAxis()

```
void drawAxis (
            double scale = 1 )
```

8.64.3.3 initFullScreen()

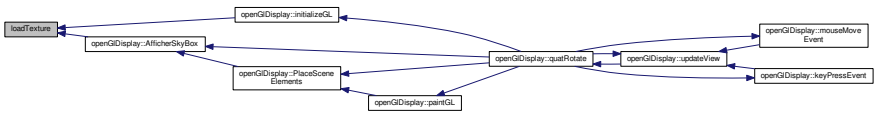
```
int initFullScreen (
            unsigned int * width = NULL,
            unsigned int * height = NULL )
```

8.64.3.4 loadTexture()

```
GLuint loadTexture (
            const char * filename,
            bool useMipMap = true )
```

Voici le graphe d'appel pour cette fonction :

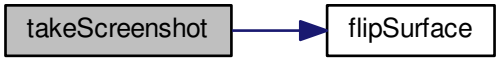
Voici le graphe des appelants de cette fonction :



8.64.3.5 takeScreenshot()

```
int takeScreenshot (
            const char * filename )
```

Voici le graphe d'appel pour cette fonction :



8.64.3.6 XPMFromImage()

```
int XPMFromImage (
            const char * imagefile,
            const char * XPMfile )
```

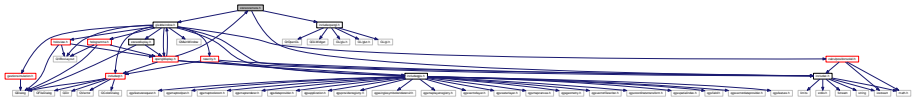
8.65 Référence du fichier stereocamera.cpp

8.66 Référence du fichier stereocamera.h

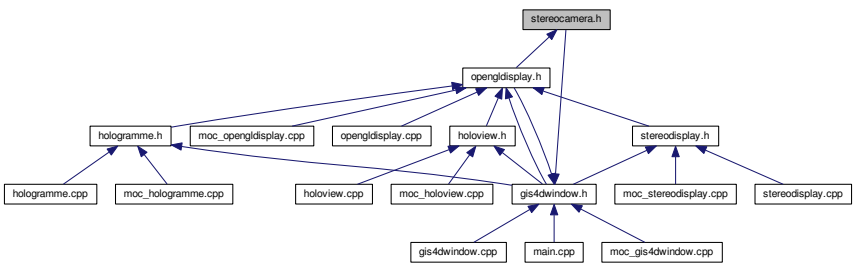
Calculate the left and right display for the steroscopic view.

```
#include <includec.h>
#include <includeopengl.h>
#include <gis4dwindow.h>
```

Graphe des dépendances par inclusion de stereocamera.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class StereoCamera

Macros

— #define PI 3.14159265

8.66.1 Description détaillée

Calcule la vue gauche et la vue droite pour la vue stéréoscopique.

Auteur

Julien RICHARD

8.66.2 Documentation des macros

8.66.2.1 PI

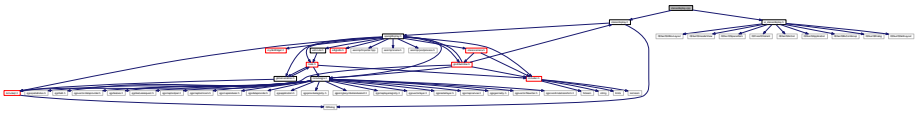
```
#define PI 3.14159265
```

8.67 Référence du fichier stereodisplay.cpp

Widget pour la vue stéréoscopique.

```
#include "stereodisplay.h"
#include "ui_stereodisplay.h"
```

Graphe des dépendances par inclusion de stereodisplay.cpp :



8.67.1 Description détaillée

Widget pour la vue stéréoscopique.

Auteur

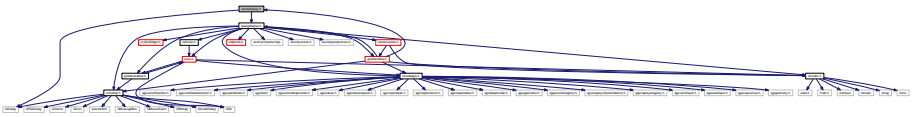
Julien Richard

Deux vues sont affichées avec un point de fuite légèrement décalé afin de créer un effet de profondeur lors de l'utilisation d'un stéréoscope.

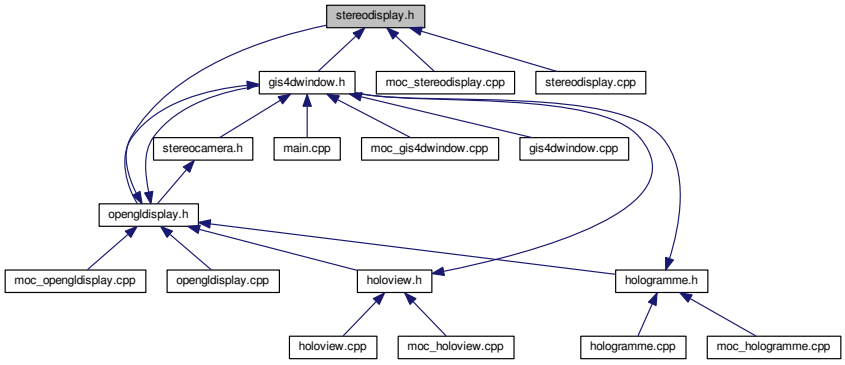
8.68 Référence du fichier stereodisplay.h

```
#include <QDialog>
#include <opengldisplay.h>
```

Graphe des dépendances par inclusion de stereodisplay.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class stereoDisplay

Espaces de nommage

— Ui

8.69 Référence du fichier tools.cpp

Liste d'outils pour les différents calculs.

```
#include "tools.h"
```

Graphe des dépendances par inclusion de tools.cpp :

Macros

— #define PI 3.14159265

8.69.1 Description détaillée

Liste d'outils pour les différents calculs.

Auteur

Julien Richard

Tous les outils sont présents ici pour convertir la donnée en 3D ainsi que pour la création de topologie.

8.69.2 Documentation des macros

8.69.2.1 PI

```
#define PI 3.14159265
```

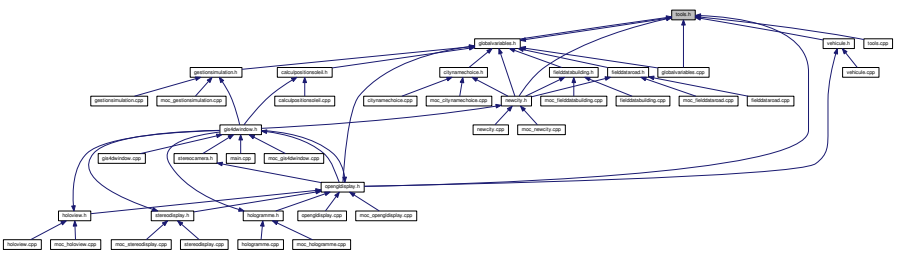
8.70 Référence du fichier tools.h

Outils servant à la conversion des données, à la simulation et à la vue 3d.

```
#include <QString>
#include <QDir>
#include <QFile>
#include <QFileInfo>
#include <QFileInfoList>
#include <includec.h>
#include <includeqgis.h>
#include <globalvariables.h>
```

Graphe des dépendances par inclusion de tools.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class tools
— struct tools : :point3D
Coordonnées 3D d'un point. Les données sont des doubles.
— struct tools : :triangleToDisplay
Définition d'un triangle. Pour un triangle, nous avons différents arguments pour le définir.
— struct tools : :segmentOfRoad
Définition d'un segment de route. Pour un segment de route, nous avons différents arguments pour le définir.
— struct tools : :pointInter
Point d'intersection. Point d'intersection utilisé lors de la superposition entre la grille topo et un tronçon.

Variables

— QVector< tools : :triangleToDisplay > triangleTopography
Triangles pour la topographie.
— QVector< tools : :triangleToDisplay > triangleTopography2
— QVector< tools : :triangleToDisplay > triangleBati
Triangles pour les bâtiments.
— QVector< tools : :triangleToDisplay > trianglesRoad
Triangle des routes créés depuis les segments de routes.
— QVector< QVector< QVector< double > > > polygonBati
Polygon des bati.

8.70.1 Description détaillée

Outils servant à la conversion des données, à la simulation et à la vue 3d.

Auteur

Julien RICHARD

8.70.2 Documentation des variables

8.70.2.1 polygonBati

`QVector<QVector <QVector <double> > > polygonBati`

Délimitation des bâtiments

8.70.2.2 triangleBati

`QVector<tools::triangleToDisplay> triangleBati`

Triangles 3D de bâtiments.

8.70.2.3 trianglesRoad

`QVector<tools::triangleToDisplay> trianglesRoad`

Triangles 3D de routes.

8.70.2.4 triangleTopography

`QVector<tools::triangleToDisplay> triangleTopography`

Triangles 3D de topographie.

8.70.2.5 triangleTopography2

`QVector<tools::triangleToDisplay> triangleTopography2`

Copie de triangles 3D de topographie.

8.71 Référence du fichier ui_choosefield.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QComboBox>
#include <QtGui/QDialog>
#include <QtGui/QDoubleSpinBox>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QPushButton>
#include <QtGui/QRadioButton>
```

Graphe des dépendances par inclusion de ui_choosefield.h :

Classes

— class Ui_chooseField
— class Ui::chooseField

Espaces de nommage

— Ui

8.72 Référence du fichier ui_choosefieldBuilding.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QComboBox>
```

```
#include <QtGui/QDialog>
#include <QtGui/QDoubleSpinBox>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QPushButton>
#include <QtGui/QRadioButton>
```

Graphe des dépendances par inclusion de ui_choosefieldBuilding.h :

Classes

— class Ui_chooseField
— class Ui : :chooseField

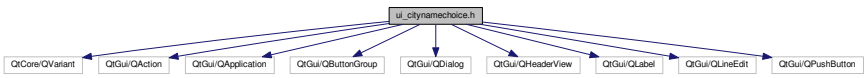
Espaces de nommage

— Ui

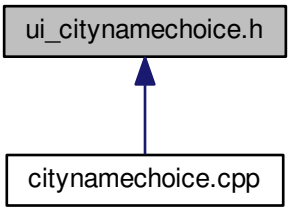
8.73 Référence du fichier ui_citynamechoice.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QLineEdit>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_citynamechoice.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_cityNameChoice
— class Ui : :cityNameChoice

Espaces de nommage

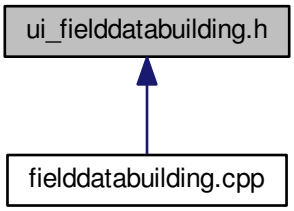
— Ui

8.74 Référence du fichier ui_fielddatabuilding.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QComboBox>
#include <QtGui/QDialog>
#include <QtGui/QDoubleSpinBox>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QPushButton>
#include <QtGui/QRadioButton>
```

Graphe des dépendances par inclusion de ui_fielddatabuilding.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_fieldDataBuilding
— class Ui : :fieldDataBuilding

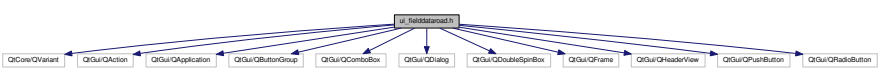
Espaces de nommage

— Ui

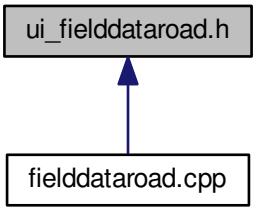
8.75 Référence du fichier ui_fielddataroad.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QComboBox>
#include <QtGui/QDialog>
#include <QtGui/QDoubleSpinBox>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QPushButton>
#include <QtGui/QRadioButton>
```

Graphe des dépendances par inclusion de ui_fielddataroad.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_fieldDataRoad
— class Ui::fieldDataRoad

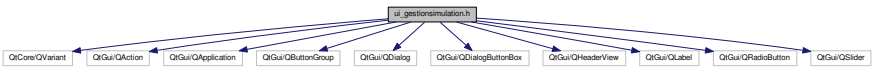
Espaces de nommage

— Ui

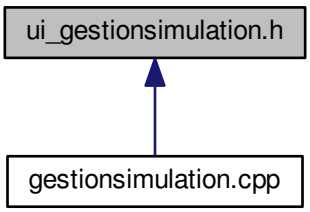
8.76 Référence du fichier ui_gestionsimulation.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QDialogButtonBox>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QRadioButton>
#include <QtGui/QSlider>
```

Graphe des dépendances par inclusion de ui_gestionsimulation.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_gestionSimulation
— class Ui : :gestionSimulation

Espaces de nommage

— Ui

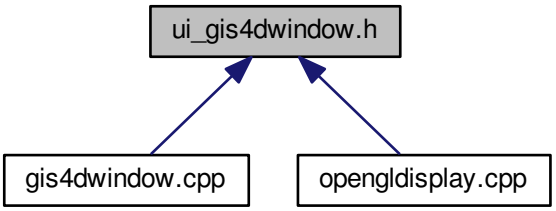
8.77 Référence du fichier ui_gis4dwindow.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QCheckBox>
#include <QtGui/QFrame>
#include <QtGui/QGridLayout>
#include <QtGui/QGroupBox>
#include <QtGui/QHBoxLayout>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QMainWindow>
#include <QtGui/QMenu>
#include <QtGui/QMenuBar>
#include <QtGui/QPushButton>
#include <QtGui/QSlider>
#include <QtGui/QSpacerItem>
#include <QtGui/QStatusBar>
#include <QtGui/QTabWidget>
#include <QtGui/QToolBar>
```

```
#include <QtGui/QWidget>
```

Graphe des dépendances par inclusion de ui_gis4dwindow.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_gis4DWindow
— class Ui : :gis4DWindow

Espaces de nommage

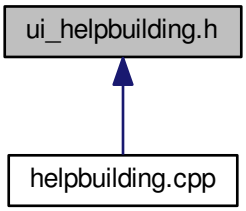
— Ui

8.78 Référence du fichier ui_helpbuilding.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_helpbuilding.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_helpBuilding
— class Ui : :helpBuilding

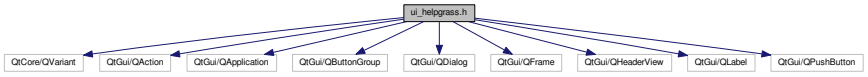
Espaces de nommage

— Ui

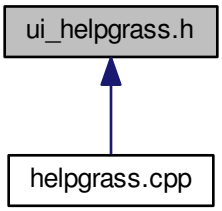
8.79 Référence du fichier ui_helpgrass.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_helpgrass.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_helpGrass
— class Ui : :helpGrass

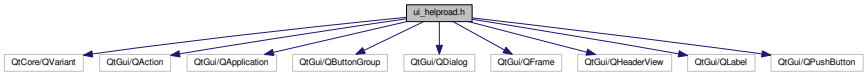
Espaces de nommage

— Ui

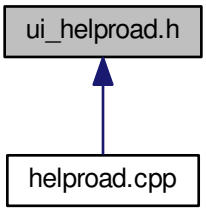
8.80 Référence du fichier ui_helproad.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_helproad.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_helpRoad
— class Ui : :helpRoad

Espaces de nommage

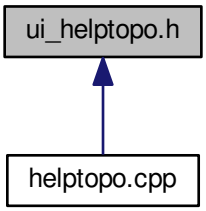
— Ui

8.81 Référence du fichier ui_helptopo.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_helptopo.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_helpTopo
— class Ui : :helpTopo

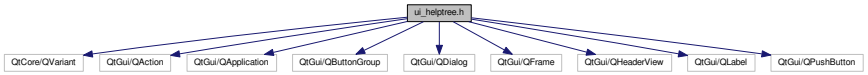
Espaces de nommage

— Ui

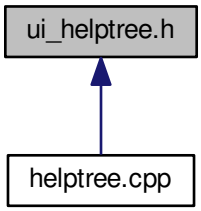
8.82 Référence du fichier ui_helptree.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_helptree.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_helpTree
— class Ui : :helpTree

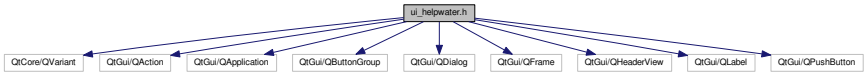
Espaces de nommage

— Ui

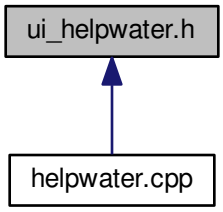
8.83 Référence du fichier ui_helpwater.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_helpwater.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_helpWater
— class Ui : :helpWater

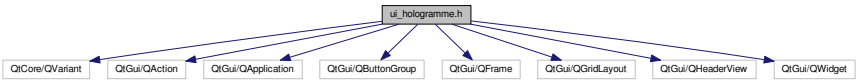
Espaces de nommage

— Ui

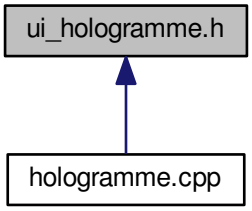
8.84 Référence du fichier ui_hologramme.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QFrame>
#include <QtGui/QGridLayout>
#include <QtGui/QHeaderView>
#include <QtGui/QWidget>
```

Graphe des dépendances par inclusion de ui_hologramme.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_hologramme
— class Ui : :hologramme

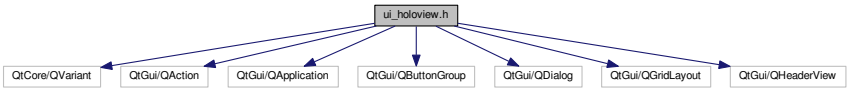
Espaces de nommage

— Ui

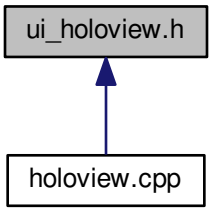
8.85 Référence du fichier ui_holoview.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QGridLayout>
#include <QtGui/QHeaderView>
```

Graphe des dépendances par inclusion de ui_holoview.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_holoView
— class Ui::holoView

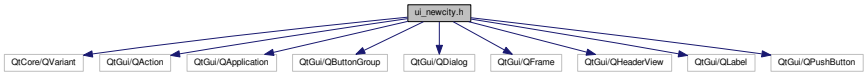
Espaces de nommage

— Ui

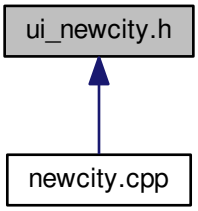
8.86 Référence du fichier ui_newcity.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QFrame>
#include <QtGui/QHeaderView>
#include <QtGui/QLabel>
#include <QtGui/QPushButton>
```

Graphe des dépendances par inclusion de ui_newcity.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_newCity
— class Ui : :newCity

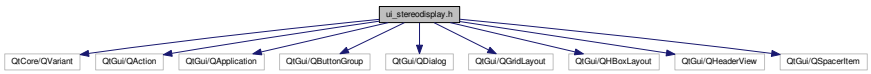
Espaces de nommage

— Ui

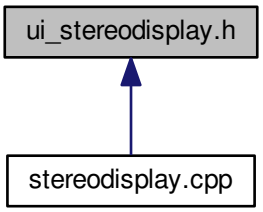
8.87 Référence du fichier ui_stereodisplay.h

```
#include <QtCore/QVariant>
#include <QtGui/QAction>
#include <QtGui/QApplication>
#include <QtGui/QButtonGroup>
#include <QtGui/QDialog>
#include <QtGui/QGridLayout>
#include <QtGui/QHBoxLayout>
#include <QtGui/QHeaderView>
#include <QtGui/QSpacerItem>
```

Graphe des dépendances par inclusion de ui_stereodisplay.h :



Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class Ui_stereoDisplay
— class Ui : :stereoDisplay

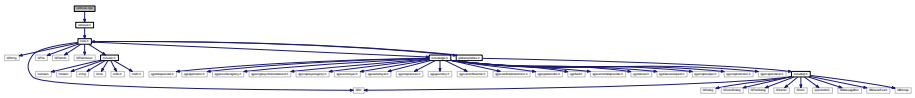
Espaces de nommage

— Ui

8.88 Référence du fichier vehicule.cpp

```
#include "vehicule.h"
```

Graphe des dépendances par inclusion de vehicule.cpp :

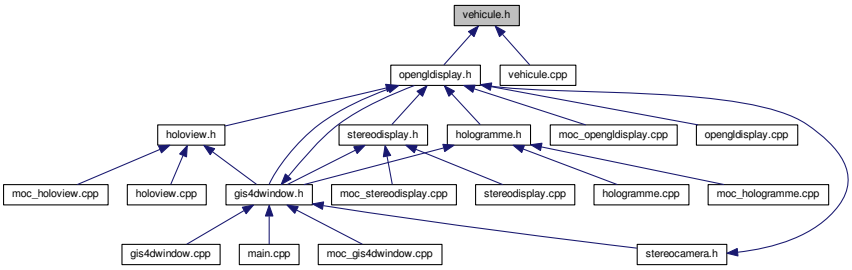


8.89 Référence du fichier vehicule.h

```
#include <tools.h>
```

Graphe des dépendances par inclusion de vehicule.h :

Ce graphe montre quels fichiers incluent directement ou indirectement ce fichier :



Classes

— class vehicule

Index