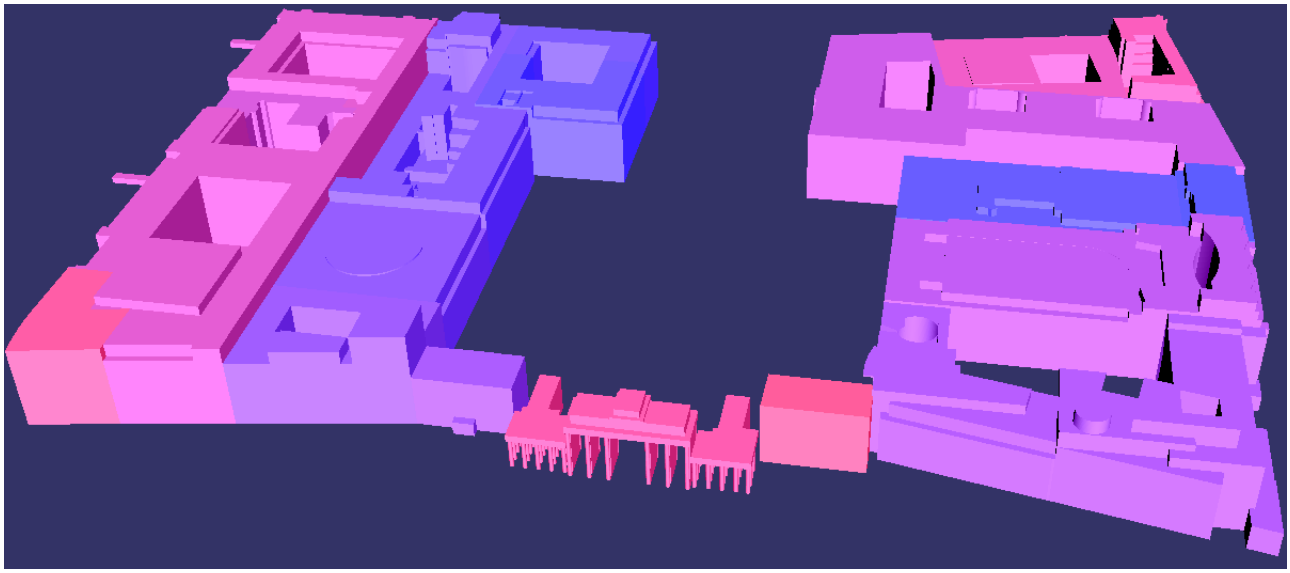
PGROU - VisDU3D

Rapport final

15/12/10



Alice Lan
Toinon Vigier
Elsa Arrou-Vignod
Florent Buisson
Robin Kervadec
Michael Lumbroso

Tuteur enseignant :
Guillaume Moreau

Table des matières

1 Présentation du projet Visdu3D

1.1 Objectifs et destinataires du document

Le projet est réalisé pour le CERMA, le Centre de Recherche Méthodologique d'Architecture, avec qui travaille en partenariat une équipe d'enseignants-chercheurs de l'Ecole Centrale de Nantes. Le projet Visdu3D a pour but de fournir un environnement de visualisation de données géographiques et des métadonnées (voir glossaire).

La problématique du projet est double : tout d'abord, permettre un affichage simultané de données géographiques 2D et 3D ; ensuite, proposer des solutions d'affichage des métadonnées de ces fichiers de façon ergonomique et lisible pour l'utilisateur. Le but du projet est de mener une réflexion sur des affichages adaptés au point de vue de la caméra, aux couleurs d'arrière-plan, à la taille des données, au type de données sémantiques représentées, etc.

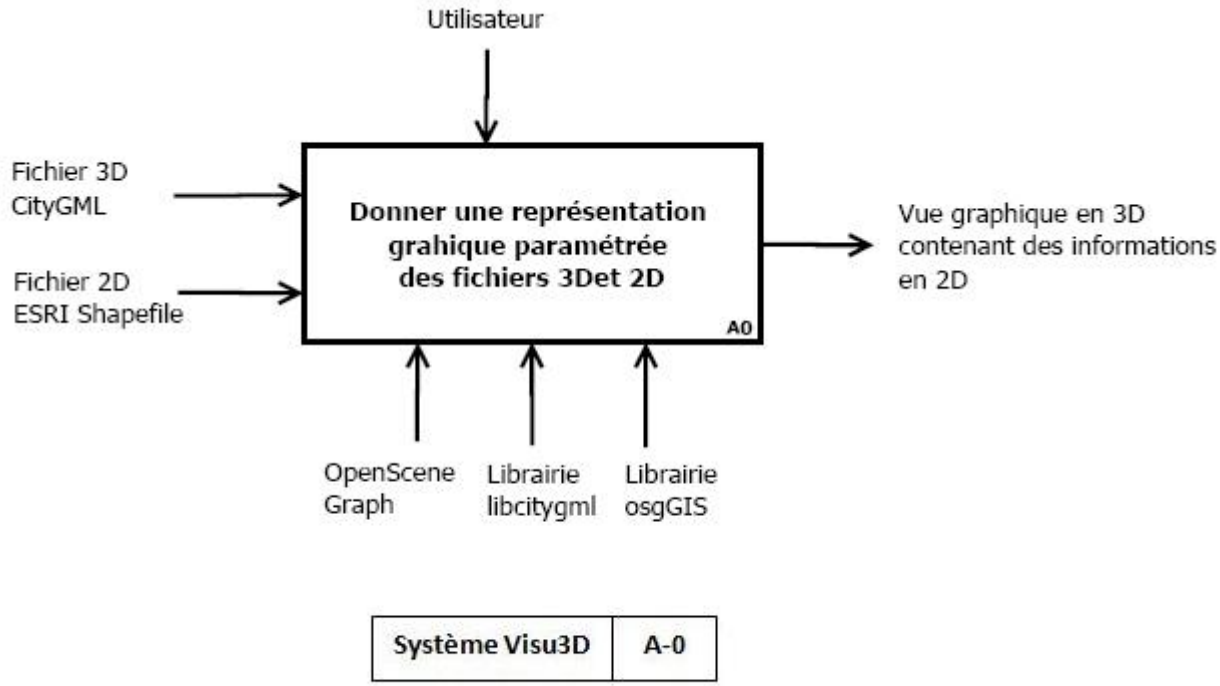


Fig. 1 – *Diagramme SADT au niveau 0 du projet Visdu3D.*

Les formats de fichiers à afficher sont des fichiers ESRI Shapefile pour la 2D, et CityGML pour la 3D. Une description plus détaillée de ces fichiers est disponible dans l'état de l'art.

1.2 Environnement du logiciel

Le développement du programme se fera en langage C++, en utilisant les systèmes d'exploitation Mac OS et Linux. Nous utiliserons Ubuntu pour les systèmes Windows. L'affichage en trois dimensions se fera à l'aide d'une bibliothèque C++ : Open Scene Graph. Le chargement des fichiers CityGML et ESRI Shapefile se fera respectivement grâce à deux bibliothèques : libcitygml et osgGIS.

1.3 Cahier des charges

Ce programme est destiné à être utilisé dans un système d'exploitation Linux ou Mac. Il pourra être complété et amélioré pour obtenir un plus large choix d'affichages. Il pourra être éventuellement repris pour créer une meilleure interface utilisateur.

Le cahier des charges que nous avons établi après réunion avec Guillaume Moreau, est le suivant :

- fournir un état de l'art de l'existant sur l'affichage de données 2D/3D et la sémiologie graphique
- afficher simultanément des fichiers CityGML (3D) et ESRI Shapefile (2D)
- proposer le plus possible de solutions pour afficher des métadonnées de façon ergonomique
- fournir des résultats de test et une analyse des solutions proposées

1.4 Livrables

- **État de l'art :**
  - Intégration du texte dans un environnement 3D
  - Shapefile
  - Bibliothèques libcitygml, osgGIS
- **Tutoriaux :**
  Installation de l'environnement de programmation :
  - Linux
  - C++
  - OSG
  - libcitygml
  - osgGIS
- **Application :**
  - Diagramme de classes
  - Fichiers sources
  - Résultats de tests

2 Déroulement du projet

2.1 Plannings prévisionnel et effectué

Voici le planning des tâches que nous avions prévu durant la phase de lancement du projet, à savoir dans les deux premières semaines :

Dates	01/10	15/10	05/11	19/11	03/12	15/12
Jalons	RA1	RA2	RA3	RA4	RA5	Rapport final
État de l'art	■	■				■
Installations	■	■				
Affichage simultané			■	■		
Affichage d'informations			■	■	■	

Tab. 1 – *Planning prévisionnel de déroulement du projet. RA : rapport d'avancement.*

Le planning réel de déroulement du projet a été en fait le suivant :

Dates	01/10	15/10	05/11	19/11	03/12	15/12
Jalons	RA1	RA2	RA3	RA4	RA5	Rapport final
État de l'art	■	■				■
Installations	■	■	■	■	■	
Affichage simultané					■	■
Affichage d'informations					■	■

Tab. 2 – *Planning effectif du projet. RA : rapport d'avancement.*

Comme l'indiquent les rapports d'avancement, nous avons eu de nombreux problèmes liés à l'installation des différentes bibliothèques, ce qui a retardé considérablement le début des phases d'affichage simultané et d'affichage d'informations sémantiques. Cela se constate sur le planning réel : la phase d'installation a duré près de 10 semaines, dont les 6 premières ont été consacrées principalement à l'installation d'Open Scene Graph, bibliothèque indispensable pour réaliser le projet. De plus, bien que nous ayons pu par la suite faire quelques tests isolés d'affichage de texte, de formes 3D, de changements de couleur, etc., nous avons dû attendre d'être parvenus à installer également les plugins pour pouvoir appliquer nos tests à de véritables fichiers CityGML, puis plus tard à des fichiers Shapefile.

2.2 Répartition du travail

2.2.1 Les différentes tâches du projet

Lancement du projet

Cette phase comprend la prise de connaissance du sujet avec Guillaume Moreau, la définition du cahier des charges et la familiarisation avec les bibliothèques que Guillaume Moreau nous suggérait d'utiliser.

État de l'art

Notre état de l'art comprend une description des formats de fichiers CityGML et Shapefile, une description des bibliothèques utilisées et les règles classiques de la sémiologie graphique.

Installations

Avant de démarrer la phase de développement du projet, il nous a fallu installer les différentes bibliothèques sur les machines sous Mac et sous Linux. Cette phase d'installations a été particulièrement laborieuse et longue.

2.2.2 Répartition des tâches entre les membres de l'équipe

Tâches	Alice	Toinon	Elsa	Florent	Robin	Michael
Lancement du projet	6	6	6	6	6	6
État de l'art	4	6	2			8
Installations	4	20	10	35	10	8
Affichage simultané		10		3	5	
Affichage d'informations		15	20	25	2	
Rapports d'avancement	10	5	5	5	5	5
Rapport final	10	5	15	2	6	10
Réunions	9	9	9	9	9	9
Total	43	76	67	85	43	37

Tab. 3 – *Nombre d'heures passées sur chaque partie du projet.*

3 Etat de l'art

3.1 Les formats CityGML et ESRI Shapefile

3.1.1 CityGML

Le format CityGML est un format de représentation de données urbaines qui dérive du format GML, lui-même dérivé du langage XML. Il est conçu pour permettre de stocker des métadonnées très variées. Ce type de fichier est optimisé pour représenter des données urbaines, et propose des représentations génériques déjà implantées dans le Schema pour les objets urbains les plus courants (routes, végétation, tunnels, ...).

Voici un exemple de fichier GML :

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- File: cambridge.xml -->
<CityModel xmlns="http://www.opengis.net/examples" ... >

 <gml:name>Cambridge</gml:name>
 <gml:boundedBy>
  <gml:Box srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
   <gml:coord><gml:X>0.0</gml:X><gml:Y>0.0</gml:Y></gml:coord>
   <gml:coord><gml:X>100.0</gml:X><gml:Y>100.0</gml:Y></gml:coord>
  </gml:Box>
 </gml:boundedBy>

 <cityMember>
  <River>
   <gml:description>The river that runs through Cambridge.</gml:description>
   <gml:name>Cam</gml:name>
   <gml:centerLineOf>
    <gml:LineString srsName="http://www.opengis.net/gml/srs/epsg.xml#4326">
     <gml:coord><gml:X>0</gml:X><gml:Y>50</gml:Y></gml:coord>
     <gml:coord><gml:X>70</gml:X><gml:Y>60</gml:Y></gml:coord>
     <gml:coord><gml:X>100</gml:X><gml:Y>50</gml:Y></gml:coord>
    </gml:LineString>
   </gml:centerLineOf>
  </River>
 </cityMember>
...
```

Fig. 2 – *Fichier gml*

Un fichier CityGML est composé, comme tous les fichiers XML, d'une racine (CityModel) et de noeuds fils (CityObjectMember) qui contiennent les informations géométriques et sémantiques sur les différents objets urbains.

Ce format permet de décrire des entités simples :
- Informations géographiques et descriptives
- Informations géométriques simples (points, lignes,...)

CityGML définit notamment plusieurs niveaux de détails (Level of Detail, LoD). Ces niveaux de détails sont étiquetés de 0 à 4. Voici un exemple de ce que l'on peut obtenir grâce à CityGML, en LoD 3 :

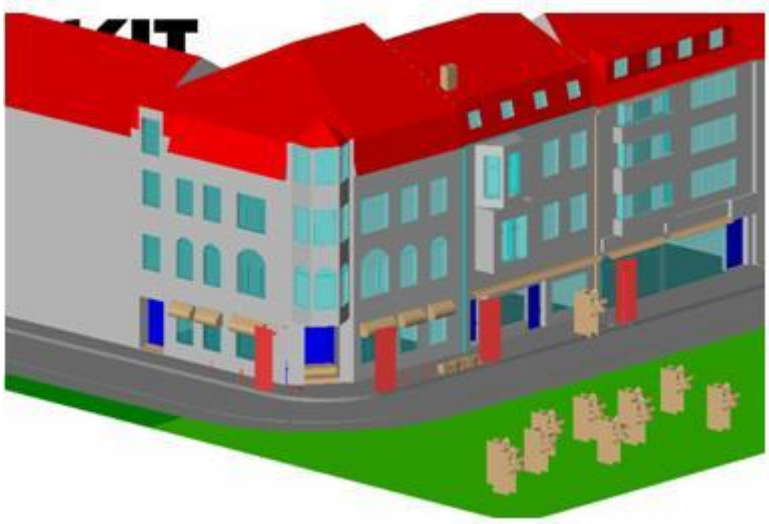


Fig. 3 – *Exemple de fichier CityGML en LoD 3*

3.1.2 ESRI Shapefile

Le format de stockage de données géospatiales Shapefile est un format originellement propriétaire créé par ESRI, un des leaders en développement de Systèmes d'Information Géograhiques (SIG). Développé au départ pour les logiciels commerciaux de ESRI, le Shapefile est devenu un standard pour le stockage des images de type vecteurs et est aujourd'hui utilisé par de nombreux logiciels libres [Wik].

Contenu :

Le Shapefile est composé au minimum de trois fichiers :
- Le fichier principal d'extension .shp qui contient la géométrie des différents objets. Chaque enregistrement est accessible directement et décrit par une liste de points.
- Le fichier index, d'extension .shx qui contient un index des différents enregistrements.
- Une dBase : table d'extension .dbf qui contient l'ensemble des données attributaires des objets géométriques.

D'autres fichiers peuvent également être fournis et associés à ces trois fichiers principaux. Dans le cas de notre projet, nous aurons également besoin de l'extension .prj qui contient le système de projection utilisé exprimé dans un langage à balise, le WKT (Well-Known Text).

Important : les fichiers doivent tous avoir le même nom devant l'extension.

Le Shapefile peut contenir un très grand nombre de géométries différentes définies dans le livre blanc ESRI Shapefile Technical Description [ESR98]. Ici, nous nous sommes restreints aux géométries 2D suivantes :

- Points
- Multipoints
- Lignes
- Polygones
- ...

3.2 Les outils d'affichage 3D

Pour réaliser ce projet, nous avons eu besoin de bibliothèques pour afficher et manipuler des objets 3D, parser des fichiers CityGML et ESRI Shapefile et permettre d'exploiter les informations géométriques et sémantiques de ces fichiers.

3.2.1 Open Scene Graph

Open Scene Graph (OSG) est une bibliothèque graphique open-source permettant le développement d'applications graphiquement exigeantes, comme des simulateurs de vols, ou, ce qui va nous intéresser ici, de la réalité virtuelle.

Open Scene Graph est en fait une surcouche d'OpenGL , écrite en C++ et orientée objet. Elle fournit notamment des outils permettant de gérer la caméra, l'éclairage d'une scène...

OSG dispose d'un système de gestion de plugins dynamiques, ce qui nous a permis d'utiliser un loader 3D pour les fichiers CityGML : libcitygml.

3.2.2 libcitygml

3.2.3 osgGIS

osgGIS est une boîte à outils permettant le traitement de données d'informations géographiques dans OpenSceneGraph. Cette boîte à outil qui s'utilise directement en lignes de commande permet de visualiser des images géospatiales mais aussi de les manipuler et de faire de requêtes à travers des applications Open Scene Graph.

3.2.4 GDAL - OGR

GDAL (Geospatial Data Abstract Library) est une bibliothèque open source développée en langage C++ qui permet de traiter différents formats de stockage d'images géographique. Une sous-bibliothèque de GDAL est OGR qui permet de travailler en lecture et de manière plus minime en écriture sur les images de type vecteur. OGR supporte la plupart des formats courants (qu'ils soient porpriétaires ou non) de formes vectorielles dont le format ESRI Shapefile mais aussi Oracle Spatial, PostGIS ou encore MapInfo.

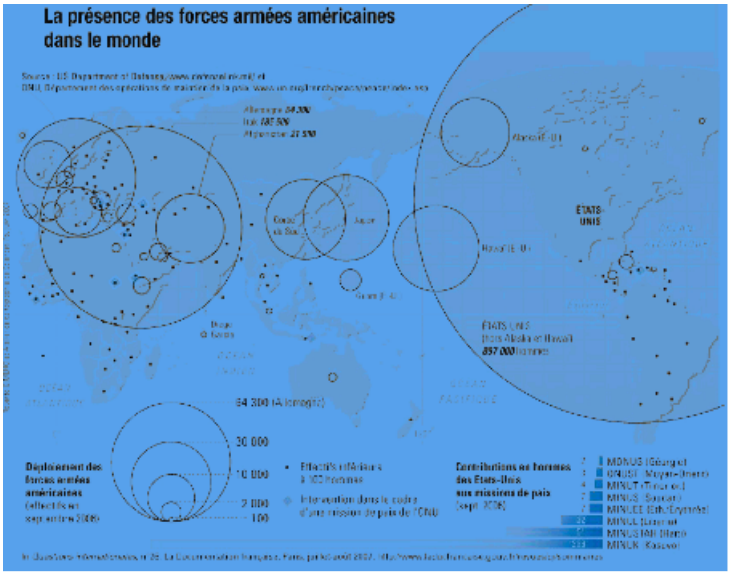
3.3 La sémiologie graphique

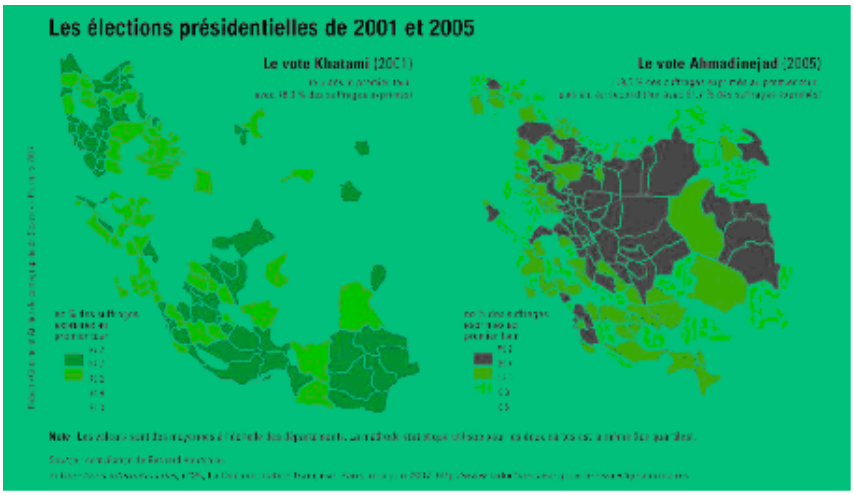
Il n'existe pas encore de norme pour la sémiologie graphique en 3D. Pour l'instant, l'affichage d'informations sémantiques se fait principalement par l'ajout de texte à côté des objets 3D. Nous nous sommes donc appuyés sur des règles d'affichage graphique en 2D pour la cartographie, que nous avons tenté d'appliquer à notre problème spécifique.

Tout d'abord, la sémiologie graphique distingue deux types de données : l'information qualitative (par exemple des différences de climat, la présence d'eau potable, ...), et quantitative (nombre d'habitants, taille d'une ville, pourcentage de femmes, ...). Voir le cours de Gwendall Petit : [Pet11].

Il existe différentes manières d'afficher des informations sémiologiques :

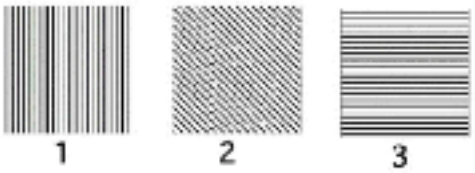
Il s'agit principalement de jouer sur les contrastes, que ce soit au niveau des tailles ou des couleurs.



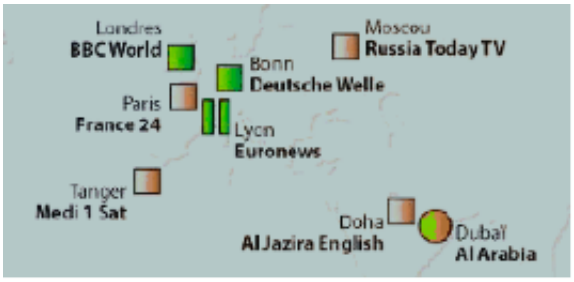


Enfin, d'autres variables visuelles rentrent en jeu, comme :

La variation de texture ou d'orientation s'il s'agit de hachures.



La variation de forme (ici combiné à une variation de couleurs pour plus de clarté) :



En combinant ces différentes techniques, on arrive à optimiser l'affichage de données.

4 La démarche appliquée au projet

4.1 Affichage de la géométrie et des informations des fichiers Shapefile

Pour pouvoir afficher la géométrie des formats ESRI Shapefile, nous nous avons utilisé l'outils osgGIS en particulier le programme `osggis\_viewer` que nous avons modifié afin de ne pas passer par le terminal. L'affichage n'est donc possible que si osgGIS a été installé et compilé sur la machine. Cette installation n'est pas forcément évidente, vous trouverez en Annexe un tutorial explicitant la marche à suivre.

L'affichage se fait grâce à la méthode transformShapefile qui retourne, à partir d'un fichier ESRI Shapefile, une pointeur de `osg::Node` qu'il est ensuite possible de visualiser grâce à la méthode `viewer()` de OSG. Cette implémentation permet de visualiser sur la même fenêtre des fichiers Shapefile et des fichiers CityGML.

L'utilisation d'osgGIS permet un affichage rapide et simplifié des formats Shapefile. Néanmoins, les codes sources étant opaques et la documentation pas très enrichie, il est difficile de réutiliser les méthodes de traitement sur les informations du Shapefile. Nous avons donc décidé de créer nos propres méthodes en utilisant la bibliothèque OGR, elle-même utilisée par osgGIS.

4.2 Diagramme de classes

Voir page suivante.

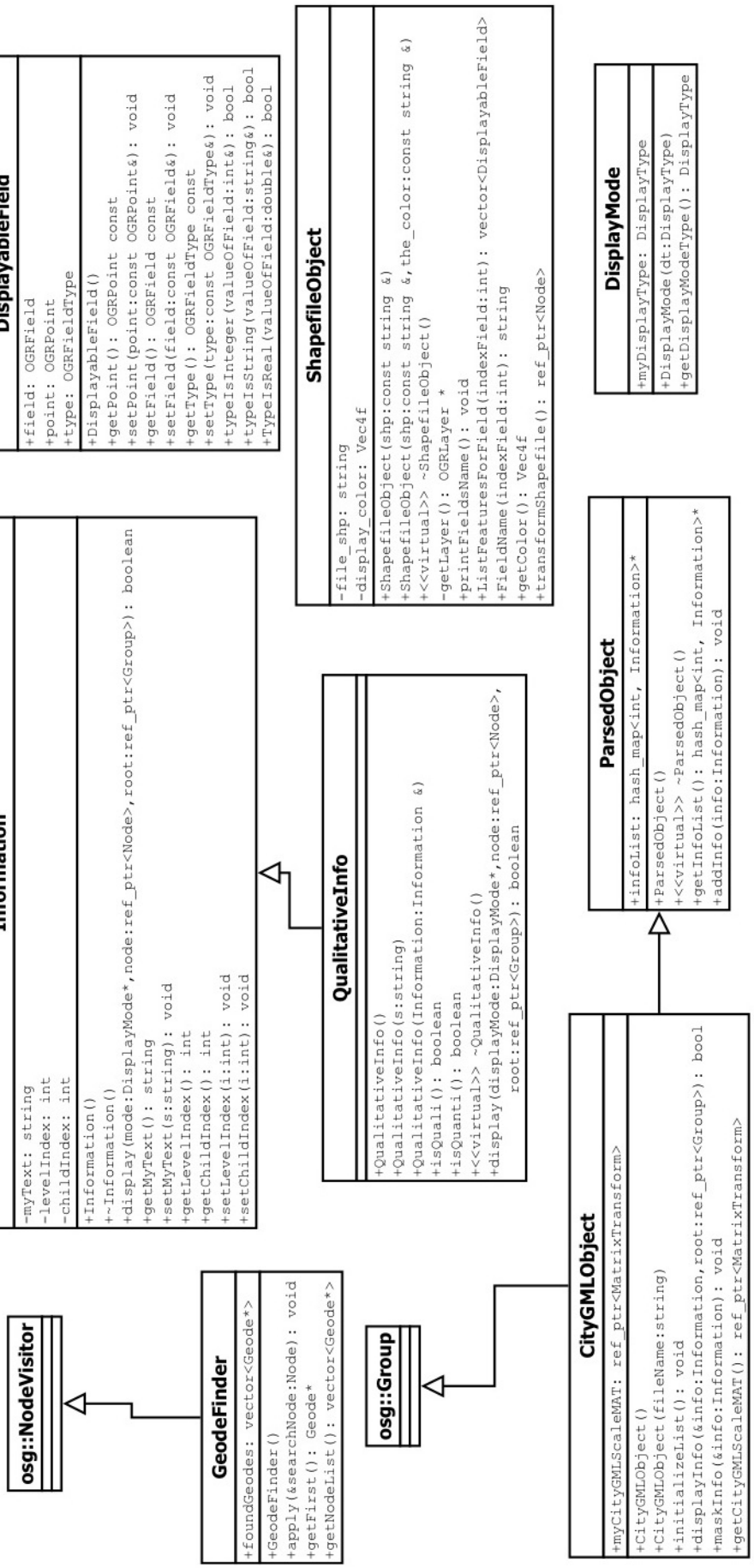


Fig. 4 – *Diagramme de classes de notre programme.*

4.3 Comment est construit le programme

Pour l'instant, notre programme prend en arguments l'emplacement d'un citygml et l'emplacement d'un shapefile.

```
./Visdu3D cheminCityGML cheminShapeFile
```

Il est possible de ne passer qu'un argument, qui doit alors être le cityGML. Il est également possible de spécifier une fois pour toutes les chemins dans le code si vous le manipulez. Nous utilisons un script bash permettant de définir les variables d'environnement nécessaire à OSG donc au fonctionnement correct de notre programme. Celui ci se charge d'appeler l'exécution de ce dernier :

```
#!/bin/bash

export
PATH="${PATH}:/Users/Flo/Dev/OpenSceneGraph-2.9.9/lib:/Users/Flo/Dev/OpenSceneGraph-2
export
DYLD_LIBRARY_PATH="/Users/Flo/Dev/OpenSceneGraph-2.9.9/lib:/Users/Flo/Dev/OpenSceneGr
export
OSG_FILE_PATH="/Users/Flo/Dev/OpenSceneGraph-2.9.9/../OpenSceneGraph-Data:/Users/Flo/
export DYLD_BIND_AT_LAUNCH=1

./Visdu3D ~/Dev/OpenSceneGraph-Data/Berlin_PariserPlatz.citygml
```

Nous n'avons bien sûr pas un programme capable de gérer n'importe quel type de données issues du shapeFile mais nos sources permettent d'en faciliter au maximum l'extraction et l'affichage. Dans cette optique nous avons défini différents types d'informations, qualitative en quantitative, pouvant être représentés par différentes types d'affichages 3D :

- changement de couleur,
- changement d'opacité,
- ajout d'une forme 3D (cylindre, sphère, pastille indicatrice smarties),
- ajout de texte

L'affichage peut se changer très facilement car tous les modes d'affichage sont redéfinis pour chaque type d'information avec les règles correspondantes. Cela est permi par une définition très simple des modes d'affichage grâce à une centralisation des options d'affichage dans la classe DisplayMode. Il est donc aussi plutôt aisé d'ajouter un nouveau type d'affichage fondé sur des transformations / ajouts déjà existants. Nous l'avons par exemple implémenté pour l'affichage simultané de formes 3D (cylindres) et de texte (valeur numérique associée).

Nous avons également incorporé un gestionnaire d'événements clavier permettant de configurer des touches pour pouvoir configurer rapidement la configuration des informations.

Grâce à celui-ci nous avons configuré des touches permettant de parcourir les bâtiments du CityGML et de leur affecter un type arbitraire d'affichage d'information. Pour l'instant il est par exemple possible d'affecter la couleur bleu ou rouge à un bâtiment puis de parcourir l'ensemble des bâtiments pour régler la couleur affichée par bâtiment. Grâce à un code similaire il est alors possible de parcourir la liste des informations et d'affecter un type d'affichage à chaque information. Ce code devrait être implémenté d'ici à la soutenance, avec cependant des informations arbitraire puisque nous manquons d'informations concrètes à extraire du Shapefile.

5 Résultats

5.1 Affichage simultané d'un fichier Shapefile et d'un fichier CityGML

Affichage simultané L'affichage simultané des deux types de fichiers a été possible grâce aux bibliothèques libcitygml et osgGIS. Nous avons au préalable dû reprojeter les fichiers shapefile qui étaient en Lambert II étendu alors que le fichier CityGML était en Lambert 93. Pour cela, nous avons utilisé la version binaire de OGR qui permet de traiter directement des fichiers à partir du terminal de la manière suivante :

```
ogr2ogr -t_srs "+init=IGNF:LAMB93" -s_srs "+init=IGNF:LAMBE +wktext"
    bati_indifferencie_ile_nantes.shp
    ~/Developpement/Bati_ile_nantes_l2e/bati_indifferencie_ile_nantes.shp
```

où `-t_srs` désigne le système de projection final et `-s_srs` désigne le système de projection initial.

Nous avons néanmoins toujours un problème dans la covisualisation : les couleurs des couches se superposant, on aperçoit aléatoirement l'une ou l'autre.

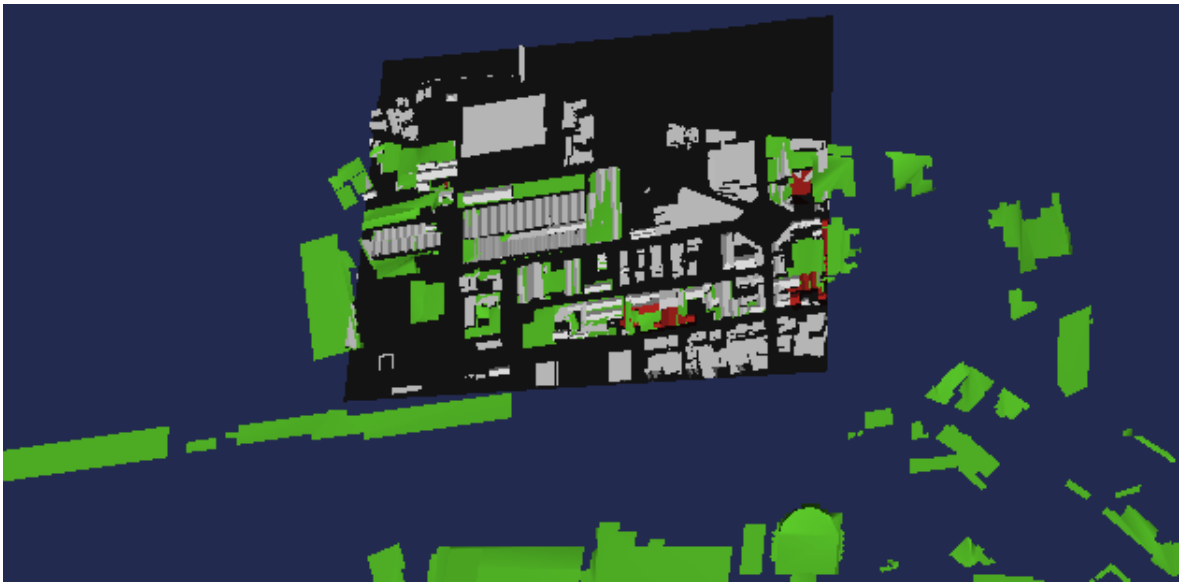


Fig. 5 – *Affichage d'un fichier Shapefile (en vert) et un fichier CityGML (encadré en noir).*

Cependant, nous n'avons pas réussi à installer osgGIS sur les machines Mac : de fait, l'affichage simultané est possible seulement sous Linux pour l'instant. De plus, l'installation a nécessité quelques manipulations : en effet, osgGIS est un projet qui n'a pas été

mis à jour depuis 2008, et l'installation est complexe et source d'erreurs de compilation. Il nous a notamment fallu écrire un script pour modifier les fichiers sources d'osgGIS car depuis 2008 une classe d'Open Scene Graph avait changé de nom.

5.2 Récupération des métadonnées dans les fichiers

5.2.1 Dans les fichiers ESRI Shapefile

La récupération des informations sémantiques dans les fichiers Shapefile se fait avec la bibliothèque OGR.

Nous avons procédé de la manière suivante :
- affichage des champs présnets dans la table dBase ;
- choix du champ c'est-à-dire de l'information que l'on souhaite afficher ;
- création du classe DisplayableField qui contient un point géométrique, une valeur de champ et le type de la valeur ;
- renvoie d'un tableau de DisplayableField : tableau à deux entrées qui stocke pour chaque enregistrement (ie objet géométrique), un point géométrique correspondant et la valeur (et le type) de l'information (ie le champ) demandé.

L'information peut être de type entier, réel ou chaine de caractères. Il existe des informations de type différent comme des tableaux numériques ou de chaines de caractères, ou encore des binaires mais nous avons décidé de ne pas les prendre en compte pour le projet pour les raisons suivantes : - ces types d'information étaient peu présents dans les fichiers sur lesquels nous avons travaillé ; - il est plus difficile d'y associer un sens.

Le point géométrique associé à un objet dépend du type de l'objet :
- Pour un objet de type Point nous renvoyons le point lui-même.
- Pour un objet de type LineString "ouvert", nous renvoyons le point médian sur la ligne.
- Pour un objet de type LinearRing (ligne brisée fermée), nous renvoyons l'isobarycentre.
- Pour un objet de type Polygon, nous renvoyons l'isobarycentre de la LinearRing extérieur.

Le point géométrique nous semble être la meilleure solution pour afficher les informations avec Open Scene Graph. En effet, il pourra être alors possible de créer un objet centré sur cette coordonnée qui contiendra l'information du bâtiment en dessous. Néanmoins, l'affichage des informations récupérées dans le fichier CityGML a pour l'instant été fait en liant l'objet contenant l'information au bâtiment. Afin d'éviter de réécrire toutes les méthodes d'affichage des informations pour les appliquer aux données attributaires récupérées dans le Shapefile, il serait plus intéressant de réussir à obtenir le bâtiment correspondant au point géométrique renvoyé puis d'appliquer les méthodes déjà définies. Une autre solution serait de renvoyer la géométrie complète de l'objet Shapefile et de l'identifier à un objet CityGML, mais cela serait sans doute très lourd à la fois en mémoire et en processus d'identification.

Les choix en matière de type d'informations et de géométries sont cependant discutables selon les fichiers utilisés et la portée du projet, il est toujours possible d'ajouter facilement de nouveaux types dans les codes sources que nous avons produits.

5.2.2 Dans les fichiers CityGML

Le site qui présente la bibliothèque libcitygml indique que celle-ci permet de parser un fichier CityGML et récupérer les données géométriques, et précise que les métadonnées du fichier sont également traitées et stockées dans les Node créés. Cependant, après avoir parcouru les messages log du site et examiné soigneusement les fichier C++ de cette bibliothèque, nous nous sommes rendu compte que la récupération des métadonnées n'était pas encore totalement implantée.

En effet, nous avons remarqué qu'il existe des variables prêtes à accueillir ce type de données, mais que nulle part ces métadonnées ne sont lues dans le fichier CityGML, ni transmises au Node principal que nous exploitons dans Open Scene Graph. Nous en avons conclu que cette fonctionnalité n'était pas encore totalement développée et qu'il nous était impossible de récupérer les métadonnées avec libcitygml.

Ayant constaté cela tard durant le projet, et un parser fait à la main étant long et difficile à mettre en place pour un format aussi riche que le CityGML, nous n'avons donc pas pu utiliser pour nos tests de métadonnées tirées de fichiers CityGML. Les informations affichées visibles sur les captures d'écran sont donc inventées pour les tests.

5.3 Affichage des métadonnées

Nous allons présenter ici des exemples d'affichage d'informations sémantiques sur des fichiers CityGML, Shapefile, et les deux formats simultanément. Nous avons fait le choix de présenter à la fois les exemples d'affichage qui semblent efficaces et ceux qui nous paraissent peu ergonomiques ou peu esthétiques, afin de montrer notre démarche. Les "mauvaises" propositions pourraient aussi inspirer d'autres utilisateurs et faire naître des solutions plus avantageuses.

Camaïeu de couleurs

Sur la fig. 8 nous avons affiché une information quantitative (par exemple, la quantité de pertes thermiques par bâtiment) sous la forme d'un camaïeu de couleurs continu (absence de paliers). Il serait également envisageable de catégoriser les valeurs par paliers afin de mieux distinguer les différences de valeurs.

À noter qu'un affichage similaire est possible avec une variation de transparence, cependant le résultat est beaucoup moins appréciable et peu ergonomique (fig. 9), car on distingue moins bien les nuances.

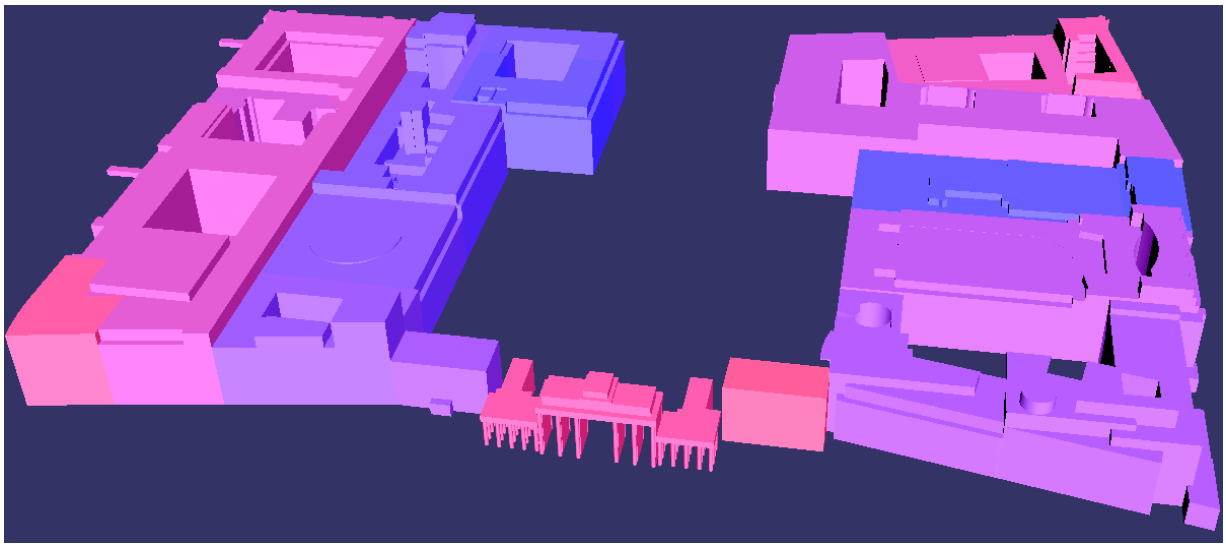


Fig. 6 – *Affichage d'une information quantitative sous forme d'un camaïeu de rouge et bleu.*

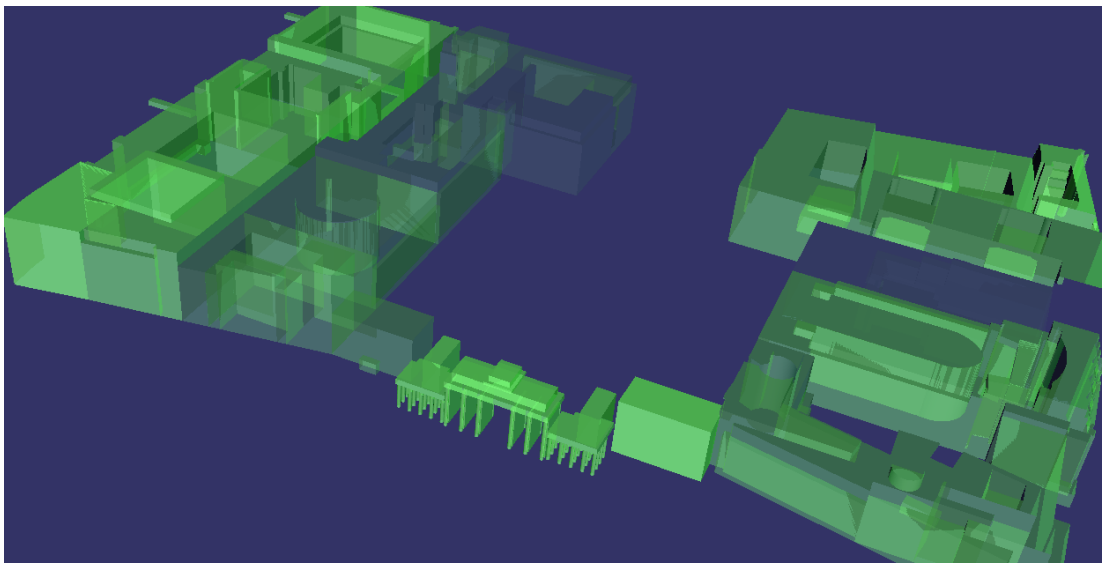


Fig. 7 – *Affichage d'une information quantitative sous forme d'une variation de transparence.*

Formes plus ou moins grandes

Une autre façon d'afficher des informations quantitatives est de placer des formes basiques dont la taille est l'élément qui va renseigner sur la valeur. Par exemple, pour deux bâtiments qui partagent un même type d'information, nous avons affiché deux cylindres de tailles différentes à la position des bâtiments qu'ils renseignent (fig. 10).

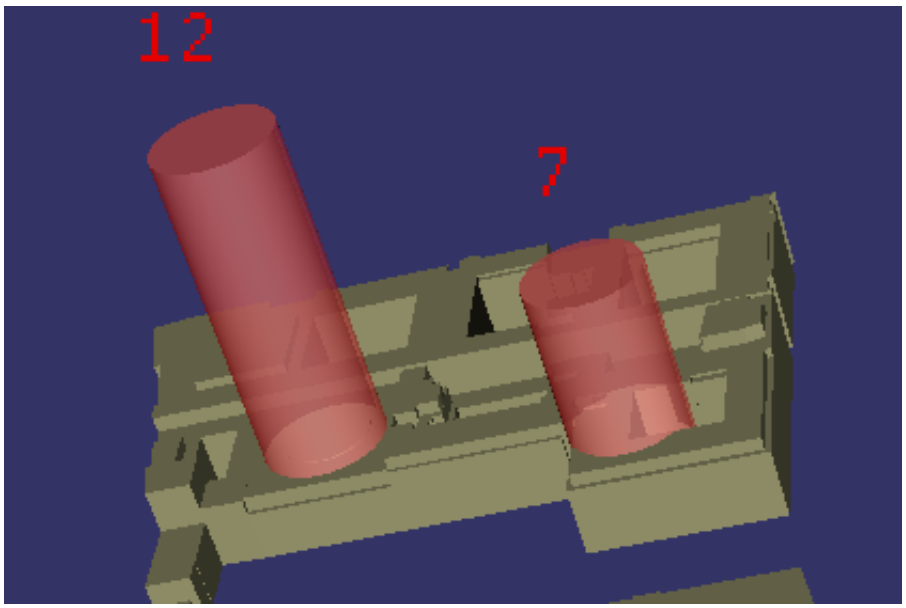


Fig. 8 – *Affichage d'une information quantitative sous forme de cylindres de taille variable.*

Les cylindres sont d'une couleur différente de celle des bâtiments pour pouvoir les distinguer et ne pas les confondre avec des informations de géométrie ; ils sont également transparents, afin de ne pas occulter complètement l'objet 3D de départ.

Il est important que l'utilisateur puisse lire les informations textuelles sous tous les angles de vue possibles : le texte affiché est donc toujours face à l'écran, comme sur la fig. 11.

Remarque : cette capture d'écran ne permet pas de bien distinguer les bâtiments les uns des autres. Si le contexte d'utilisation ne le tolérait pas, une solution serait de mixer ce type d'affichage avec le camaïeu de couleurs présenté plus haut, où l'information sémantique serait donnée par les cylindres et le camaïeu servirait à optimiser l'ergonomie de l'affichage. On peut aussi imaginer une solution consistant à repérer les arêtes des bâtiments et les afficher de façon plus marquée.

Nous avons fait un autre essai (fig. 12) avec des formes moins classiques que des cylindres, et qui conjugue variation de taille, affichage possible de texte ou dessin, de textures, et traits reliant l'objet signifiant à l'objet réel. Cela permettrait à un utilisateur averti d'Open Scene Graph de composer lui-même ses formes et de décider des éléments présents.

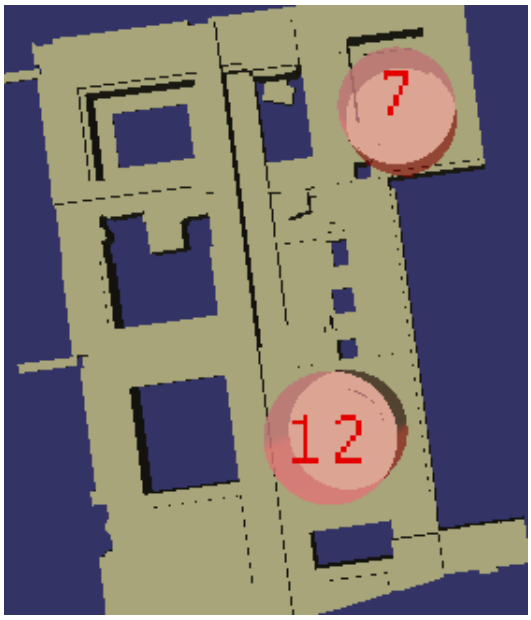


Fig. 9 – *Affichage d'une information quantitative sous forme de cylindres de taille variable.*

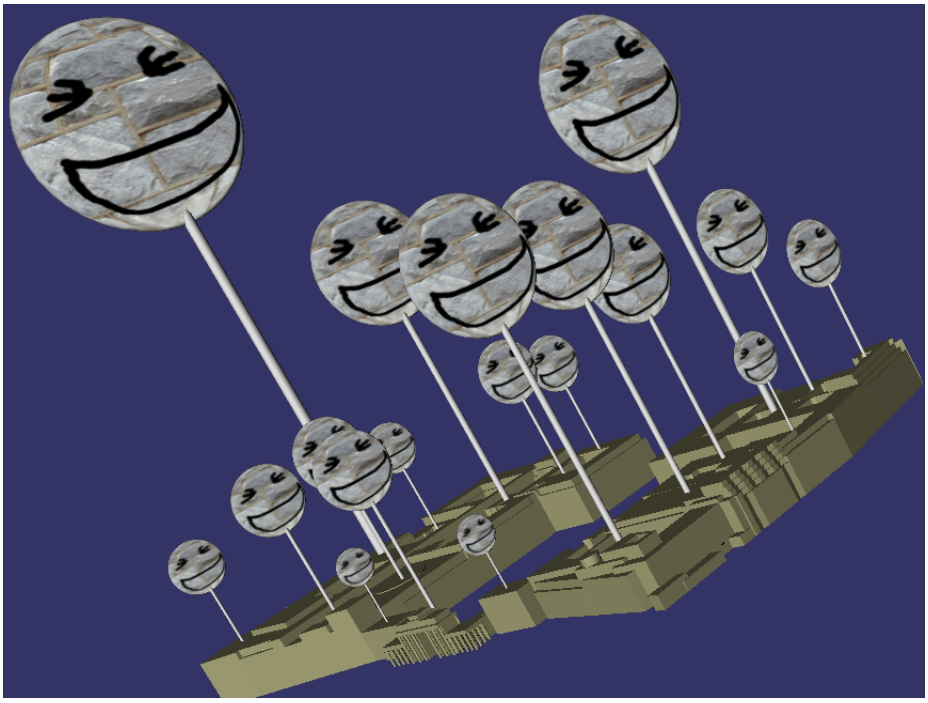


Fig. 10 – *Affichage d'une information quantitative avec des formes personnalisées conjuguant variation de taille, affichage d'un dessin, présence d'une texture.*

Distinction de quelques bâtiments parmi les autres

Pour signifier qu'un bâtiment a une caractéristique qui le distingue des autres, il faut qu'il soit distinctible à l'affichage également. Par exemple, on peut le marquer avec une couleur et/ou une transparence différente de celle des autres (fig. 13).

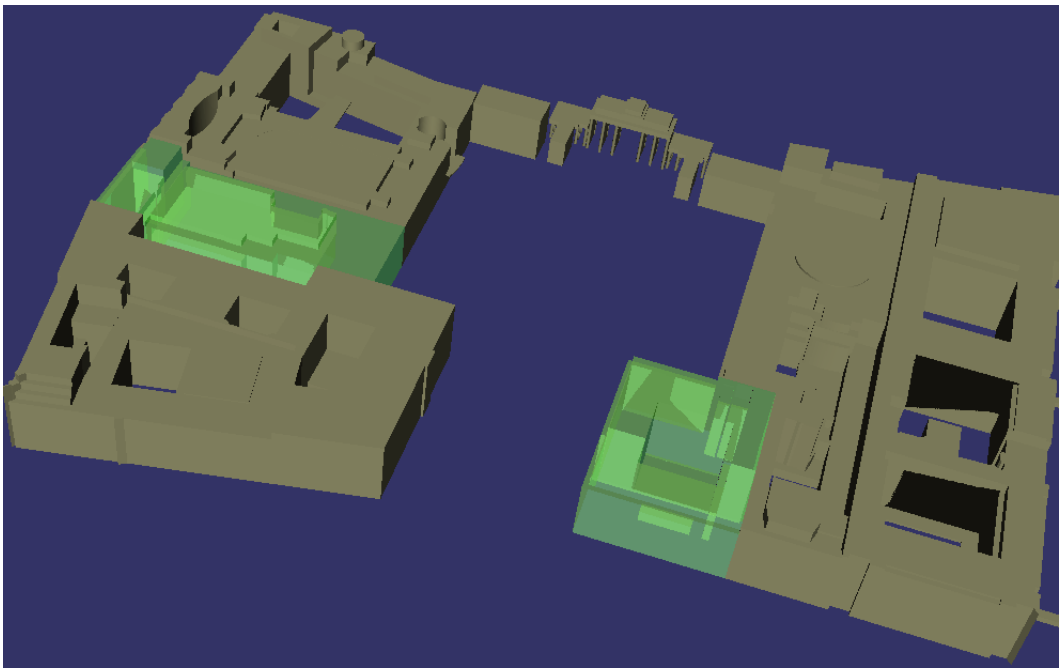


Fig. 11 – *Deux bâtiments se distinguent des autres par leur couleur et leur transparence.*

Autre exemple avec un fichier Shapefile. Les bâtiments en vert sont des bâtiments industriels. Les toits rouges sont dus à la coloration du fichier CityGML.

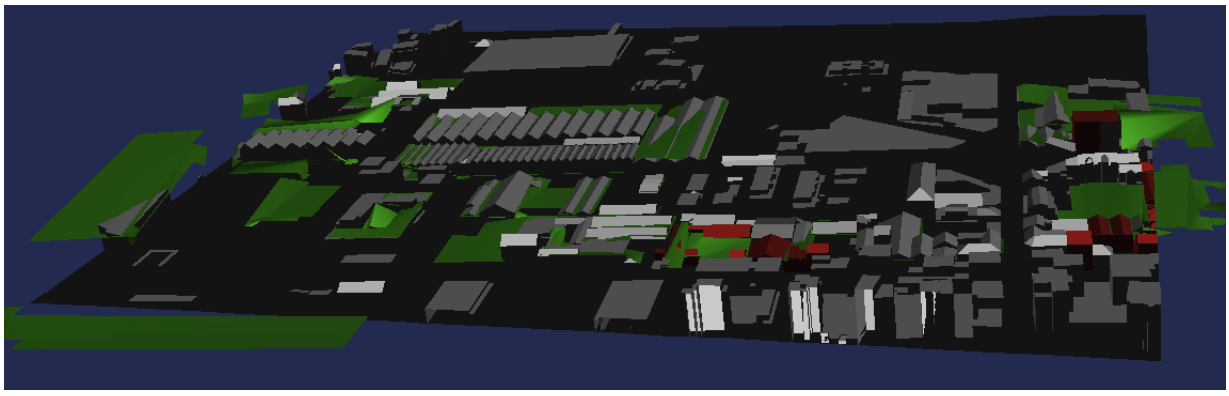


Fig. 12 – *Un fichier Shapefile et un fichier CityGML affichés simultanément. La coloration a été faite à partir d'informations sémantiques extraites du fichier Shapefile.*

Nous avons également imaginé placer des formes au-dessus des bâtiments pour le signaler :

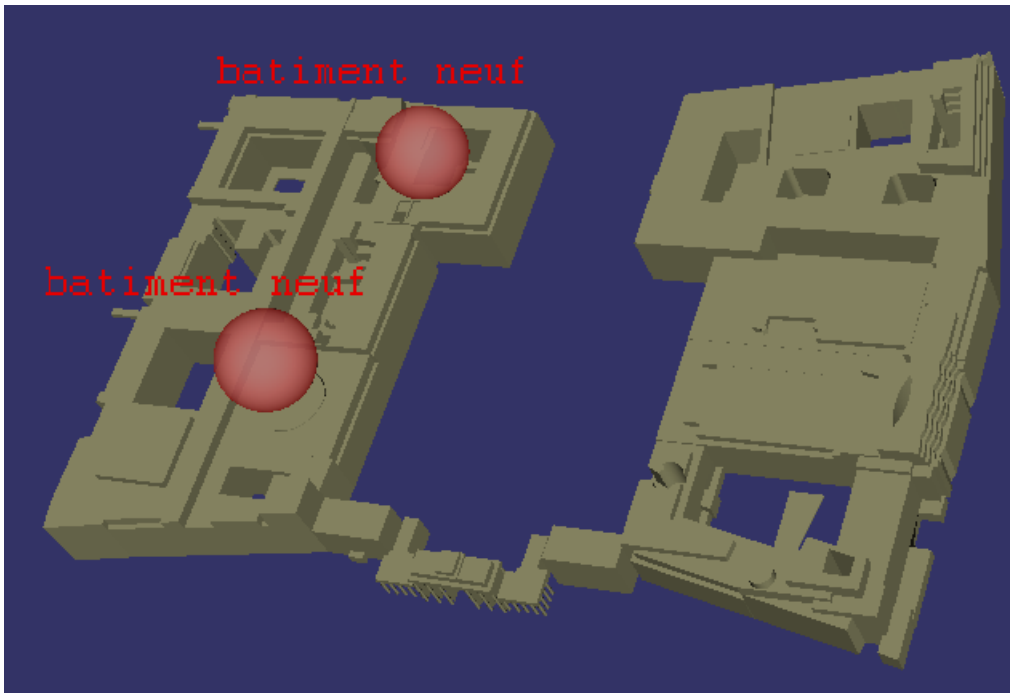


Fig. 13 – *Deux bâtiments se distinguent des autres par une sphère placée au-dessus d'eux et un texte.*

Ce type d'affichage n'est pas pleinement satisfaisant : en effet, il est difficile pour l'utilisateur de savoir quels sont les bâtiments qui sont marqués par cette sphère, d'autant plus sous cet angle de vue ; en outre, l'affichage n'est pas vraiment esthétique. Encore une fois, les limites de chaque bâtiment étant difficilement reconnaissables sur ce fichier, il faudrait conjuguer la présence des sphères avec un changement de couleur ou un affichage plus marqué des lignes de construction des objets 3D.

6 Conclusion

Au final notre production est moins un programme omnipotent qu'une base de travail pour développer facilement des modes d'affichage 3D ainsi que la récupération d'informations d'un fichier Shapefile. À ce titre notre rendu est constitué non seulement du présent rapport mais aussi d'un code source très fourni en commentaires afin d'en faciliter la reprise. Nous avons intégré divers types d'affichage qui nous semblaient pertinents mais il est très aisé d'en ajouter de nouveaux.

6.1 Perspectives

Il est important de penser à ce que le projet peut devenir. En effet, il a été conçu pour être modifié par des utilisateurs ultérieurs qui personnaliseront l'application de façon à la rendre la plus ergonomique possible. Voici donc des pistes d'approfondissement :

1. L'idéal serait alors de pouvoir écrire un fichier d'association informations / mode d'affichage afin de pouvoir conserver une telle visualisation.
2. Il est très peu probable que l'on puisse à l'avenir identifier directement le bâtiment CityGML associé à une information issue du Shapefile. Pour l'instant l'affichage au même endroit est possible donc l'association visuelle de l'information au bâtiment est réalisable. Cependant si l'on désire créer une association concrète entre ces deux données il va falloir essayer de localiser le bâtiment CityGML contenant spatialement le point issu du Shapefile. C'est dans cette optique qu'une classe `GeodeFinder` a été importée, c'est un `osg::NodeVisitor` qui permet de localiser les Geodes du fichier et donc d'effectuer des tests sur leurs informations géométriques. Se poseront alors des problèmes de convexité des bâtiments pouvant encercler un bâtiment central, etc...

6.2 Difficultés rencontrées

Comme nous l'avons précisé dans la partie Déroulement du projet, la partie laborieuse des installations a freiné considérablement le développement de l'application en elle-même. De plus, il nous a fallu appréhender le fonctionnement de trois bibliothèques différentes

7 Bibliographie

Références

[ESR98] ESRI. **ESRI Shapefile Technical Description**. 1998.

[Pet11] Petit G. **Cours de Master 2 STEU**. *IRSTV / Ecole Centrale de Nantes*, 2010–2011.

[Wik] Wikipédia. **http ://fr.wikipedia.org/wiki/Shapefile**.

8 Glossaire

Métadonnées : données sur la donnée. Dans le cas de notre projet, ce sont des informations sémantiques qualifiant les objets graphiques manipulés : par exemple, le nom des bâtiments, la densité de population d'un quartier, le fait qu'une rue soit à sens unique ou non, ...

Sémiologie graphique : ensemble de règles qui indiquent comment afficher des informations sur une carte à travers des éléments visuels, dans l'objectif d'en faciliter le plus possible la lecture et la compréhension.

9 Annexes