
3D-Use

Plugin Visibilité dans la ville

Documentation

Cyril Briquet

Septembre 2015

Table des matières

I.	En gros comment ça marche	3
A.	Comment faire une analyse	3
B.	Fonctionnement interne d'une analyse	4
C.	Comment ajouter une nouvelle couche de données	4
II.	Données.....	5
A.	Boîte englobante	5
B.	Base de données de détection de belvédères	7
C.	Triangles	9
D.	Rayons	10
E.	Point de vue.....	10
III.	Algorithme	11
A.	Détection de toits plats	11
B.	Lancer de rayon	11
C.	Extrusion de fichier Shp.....	12
D.	Arbre d'alignement	13
E.	Donnée Lidar en CityGML : les parcs.....	14
F.	Visibilité	15
1.	Analyse Mono-Tuile.....	15
2.	Analyse Multi-Tuile.....	16
G.	Export des données	18
IV.	Interface	20
A.	DialogVisibilite	20

I. En gros comment ça marche

A. Comment faire une analyse

The screenshot shows the 'Visibilite' software interface with several fields and buttons. Blue boxes and numbers highlight specific areas:

- 1**: Points to the 'Tile Directory' field containing 'C:\VCityData\Tile' and the 'Search for directory' button.
- 2**: Points to the 'Capture Category' field containing 'Test' and the 'Reset' button.
- 3**: Points to the 'Emblematic View' section, which includes sliders for Sky %, Building %, Remarquable building %, Vegetation %, Water %, and Terrain %.
- 4**: Points to the 'Camera Settings' section, which includes fields for Position and Direction, and buttons for 'Add to Batch', 'Get Parameter', and 'Apply Parameter'.
- 5**: Points to the 'Resolution' and 'Field of view' fields.
- 6**: Points to the 'Basic Multi-Tile Analysis' button, which is highlighted with a blue border.

Other visible elements include 'Delta Distance' (1,00), 'Export Top' (10), 'Basic Analysis', 'Batch Multi-Tile Analysis', 'Basic Panorama', 'Cascade count' (4), 'Cascade increment' (5,00), 'Cascade Analysis', 'Cascade Multi-Tile Analysis', 'Cascade Panorama', 'Vegetation Tools' (Alignment Tree Generation, Lidar to CityGML), and 'Other' (Extrude Shp, Flat Roof Detection, Rebuild AABBs).

Les étapes suivantes font référence au champ numéroté dans l'image ci-dessus.

1 : Spécifier le répertoire des tuiles CityGML

2 : Renseigner éventuellement un label pour la base de données de recherche de belvédère (ne rien mettre pour ne pas lancer la recherche de belvédères). Pour rappel la recherche de belvédères

consiste à regarder après plusieurs analyse, les endroits la ville le plus vue. Ces endroits peuvent donc être considérés comme des belvédères.

3 : Définir éventuellement une vue emblématique. C'est quels pourcentages de chaque couche de données doivent être présents dans la vue pour qu'elle soit considéré emblématique.

4 : Positionner la caméra dans 3D-Use comme on le souhaite, puis récupérer ces informations avec le bouton "Get Parameter"

5 : Définir la résolution de l'analyse (qui sera aussi la résolution de l'image résultat), ainsi que les champs de vision horizontaux et verticaux.

6 : Appuyer sur un bouton d'analyse pour la lancer. A la fin, les résultats seront exportés sous forme d'image, de tableau csv et de fichiers Shp.

B. Fonctionnement interne d'une analyse

1 : Tout d'abord, on va charger la liste des boites englobantes de chaque tuile présentes dans le dossier des tuiles CityGML, il y a une liste par couche de données. *Données -> Boîte englobante -> Fonction LoadAABB*

2 : A partir des caractéristique de la caméra définis par l'utilisateur on va créer une liste de rayon que l'on va utiliser pour faire du lancer de rayon. *Données -> Rayons -> Structure RayCollection -> Fonction BuildCollection*

3 : On va ensuite garder uniquement les boites de la liste dont les rayons passent à travers. On va ensuite trier cette liste suivant l'ordre dans lequel les boites sont traversés par les rayons, les boites les plus proches seront en premier dans la liste. Cela nous servira lors du lancer de rayon, cet ordonnancement nous garantis que si un rayon touche une tuile correspondant à une boîte, il n'y aura pas d'élément 3D entre la caméra et le point touché. *Algorithme -> Visibilité -> Analyse Multi-Tuile -> Fonction Setup*

4 : Pour chaque boîte dans la liste.

4.1 : On charge les triangles de la tuile CityGML correspondant à la boîte. *Données -> Triangles -> Fonction BuildTriangleList*

4.2 : On créer une liste de rayons temporaires avec uniquement les rayons qui traverse cette boîte et qui n'ont pas déjà touché une autre tuile.

4.3 : On lance l'algorithme de lancer de rayon sur les triangles de la tuile. Le lancer de rayon est multi-threadé. *Algorithme -> Lancer de rayon -> Fonction RayTracing*

5 : Les résultats du lancer de rayon sur les tuiles CityGML est stocké dans une variable de type ViewPoint qui permet pour chaque pixel de l'image résultant, savoir si le rayon a touché une tuile, et dans le cas échéant avoir les informations sur ce qu'il a touché (identifiant, type, ... du cityobject). *Données -> Point de vue -> ViewPoint*

6 : A partir de notre variable ViewPoint, on va exporter les résultats dans des images, tableaux csv et fichier Shp. *Algorithme -> Export*

C. Comment ajouter une nouvelle couche de données

1 : Ajouter le type de cityobject à charger dans, *Triangles.cpp -> Fonction BuildTriangleList -> Marqueur #CityObjectType*

2 : Ajouter une liste de boite englobante dans, [AABB.hpp -> Structure AABBCollection -> Marqueur #AABBNewDataSet](#)

3 : Faire en sorte de pouvoir générer la liste des boites englobantes pour cette nouvelle couche de données. Il suffit de se servir des exemples en commentaires. [AABB.cpp -> Fonction BuildAABB](#)

4 : Faire en sorte de pouvoir charger la liste des boites englobantes pour cette nouvelle couche de données. Il suffit de se servir des exemples en commentaires. [AABB.cpp -> Fonction LoadAABB](#)

5 : Modifier l'algorithme d'analyse de façon à ce qu'il prenne en compte la nouvelle couche de données. Il suffit de se servir des exemples en commentaires. [MultiTileAnalysis.cpp -> Fonction MultiTileBasicAnalyse](#)

6 : Modifier la fenêtre Visibilité de façon à rajouter un champ pour la nouvelle couche de données dans la vue emblématique. Rajouter aussi cette nouvelle couche de données dans le code. [Export.hpp -> Structure EmblematicView; dialogVisibilite.cpp -> Fonction SetupEmblematicViewExportParameter](#)

7 : Ne pas oublier de compléter les fonctions d'export pour prendre en compte la nouvelle couche de données. [Export.cpp -> Marqueur #NewDataSetExport](#)

II. Données

A. Boite englobante

AABB.H – AABB.CPP

Struct AABB

C'est une boite englobante pour une tuile, on stocke le point minimum et maximum de la boite. Le nom de la boite correspond au chemin vers la tuile correspondante dans le dossier des tuiles.

On considère que deux boites sont identiques si elles ont le même nom.

Struct RayBoxHit

Représente ce que touche un rayon, on garde en mémoire juste la boîte touchée et à quelle distance de l'origine du rayon la boîte a été touchée.

Pour comparer deux touchés on va utiliser l'opérateur inférieur, on considère qu'un touché est inférieur à un autre si il est plus proche de l'origine du rayon.

Struct BoxOrder

Cela va nous servir à réordonner la boite dans un certain ordre. On garde uniquement le nom de la boite ainsi que son ordre. La comparaison se fait suivant l'ordre.

Struct AABBCollection

C'est une collection de plusieurs listes de boite. Il y a une liste pour chaque couche de données différente.

Function BuildAABB

C'est cette fonction qu'il faut appeler pour construire les boites englobantes d'un répertoire de tuile.

Pour cela on va parcourir tous les dossiers du répertoire de tuile. Pour savoir à quelle couche de données les tuiles d'un dossier appartiennent, on va regarder le suffixe du dossier. A l'heure actuelle il y a 4 couches de données :

Couche	Suffixe
Bâtiment	_BATI
Terrain	_MNT
Eau	_WATER
Végétation	_VEGET

On récupère pour chaque couche de données, l'ensemble de ces dossiers et on va construire une liste de boîte pour cette couche. La fonction pour construire cette liste est : *DoBuildAABB*, cette fonction prend une liste de dossier et un type d'objet CityGML. La valeur de retour est un dictionnaire donc la clé est le nom d'une boîte et la valeur est une paire correspondant à point minimum et maximum de la boîte.

Ensuite les listes sont sauvegarder par couches, en appelant la fonction *DoSaveAABB*, qui prends en paramètres le nom du fichier (<DataSuffixe>_AABB.dat) et une liste de boîte. Le format du fichier de sorti est décrit plus loin.

Function LoadAABB

Cette fonction permet de charger une AABBCollection depuis un répertoire de tuile.

Pour cela on va vérifier que les fichiers (<DataSuffixe>_AABB.dat) de chaque couche de données sont présents, ensuite pour chaque couche on va charger la liste de boîte correspondante grâce à la fonction *DoLoadAABB*, qui prends en paramètres le chemin vers le fichier de la couche.

Format du fichier des listes de boîtes

Ce fichier permet de stocké une liste de boîte englobante (AABB) sous format texte.

Le format est le suivant :

```
<Nombre de boîte>
<#AABB 1>
<#AABB 2>
...
<#AABB X>
```

#AABB :

```
<Nom de la boîte>
<Minimum X>
<Minimum Y>
<Minimum Z>
<Maximum X>
<Maximum Y>
<Maximum Z>
```

Exemple de fichier :

```
3
Lyon_WATER/LYON_1ER_WATER_2012.gml
1841001.270000
5175467.550000
160.936000
1843085.340000
5176506.546464
173.158000
Lyon_WATER/LYON_2EME_WATER_2012.gml
1841077.400000
5171217.040000
157.791000
1843099.450000
5175676.800000
165.021000
Lyon_WATER/LYON_3EME_WATER_2012.gml
1843009.130000
5174588.599410
160.985000
1843173.198186
5175377.740646
162.822000
```

B. Base de données de détection de belvédères

BELVEDEREDB.H – BELVEDEREDB.CPP

Cette fonctionnalité va nous permettre de détecter des belvédères au travers d'une série d'analyses de point de vue.

Structure GlobalData

Contient des données générales par rapport à notre série d'analyse, combien de capture on a effectué, la liste des positions de points de vue utilisés pour les analyses.

Structure PolygonData

Contient des données pour un polygone CityGML : depuis combien de points de vue on le voit, des données sur sa tuile et son cityobject ainsi que la liste des points de vue le voyant.

Class BelvedereDB

C'est la classe qui va manipuler et stocké les données issue des analyses de points de vue. C'est une classe singleton mais qui permet de gérer plusieurs bases en changeant un label au moment de la préparation. Il y a un fichier par tuile par label de base de données.

Avant de faire une analyse il faut préparer la base de données en appelant la fonction Setup, il faut fournir le répertoire des tuiles CityGML, le label de la base de données ainsi qu'une distance à laquelle un point de vue ne sera pas pris en compte s'il est trop proche d'autre point. A partir de ce

moment il est possible d'accéder à des données globales, le nombre d'analyse effectué ainsi que la liste des points de vue.

Une fois la base préparée il est possible d'exporter les données d'un point de vue après une analyse grâce à la fonction [ExportViewpointData](#).

Une fois la base de données remplie il est possible de récupérer les polygones les plus vues grâce à la fonction [GetTop](#).

Pour une autre exploitation de la base, il est possible de récupérer les données d'une tuile avec la fonction [GetTileData](#).

Il est possible de réinitialiser une base avec la fonction [ResetDB](#).

Format du fichier de données globales

Les nombres d'analyses effectuées et la liste des points de vue utilisés sont stockés. Ce fichier est situé à la racine du répertoire de tuile.

Nom du fichier :

<Label>_globalBVDB.dat

Format du fichier :

<Nombre d'analyse>

<#Point 1>

<#Point 2>

...

<#Point X>

#Point :

<Position X>

<Position Y>

<Position Z>

Exemple :

```
3
1842544.750000
5175949.000000
795.700012
1842609.500000
5175794.000000
215.529999
1842735.500000
5175960.500000
212.050003
```

Format des fichiers de données des tuiles

C'est dans ces fichiers qu'on va mettre les informations sur les polygones vus (lors des analyses) de chaque tuile. On va garder l'identifiant du polygone et de son cityobject ainsi que la liste des points de vue qui ont vu ce polygone

Nom du fichier :

<Label>_<Nom de la Tuile>.dat

Format du fichier :

<Nombre de Polygone>

<#Polygone 1>

<#Polygone 2>

...

<#Polygone X>

#Polygone :

<Identifiant Polygone>

<Identifiant CityObject du polygone>

<Nombre de fois vue>

<#Point 1>

<#Point 2>

...

<#Point X>

#Point :

<Position X>

<Position Y>

<Position Z>

Exemple :

2

UUID_6dadf6cf-d93e-4889-9089-a3f49450506b

LYON_1ER_00137

1

1842609.500000

5175794.000000

215.529999

UUID_c997d127-54f3-4e93-a861-6d0e19863db9

LYON_1ER_00137

1

1842609.500000

5175794.000000

215.529999

C. Triangles

TRIANGLE.HPP – TRIANGLE.CPP

Structure Triangle

Les tuiles CityGML chargée sont triangulées, on va stocker des informations pour chaque triangle en plus de ces coordonnées spatiales, l'identifiant du cityobject et du polygone auquel le triangle appartient, ainsi que le type et sous type du cityobject. On garde aussi le chemin vers la tuile citygml.

Le fait de garder des données brutes et non des pointeurs vers des objets CityGML vient du fait que l'on va charger une tuile, récupérer les données et décharger la tuile. On n'aura donc plus accès aux objets CityGML. Aussi une fois une analyse faite sur une tuile, on ne gardera que les triangles qui nous intéressent et on pourra gagner en utilisation mémoire.

Structure TriangleList

Nous permet de stocké une liste de pointeur vers des triangles, et ne pas avoir à s'occuper à delete tous les triangles.

Fonction BuildTriangleList

C'est cette fonction qui va nous permettre de charger une liste de triangles depuis une tuile CityGML. Elle prend en paramètre le chemin vers la tuile sur le disque ainsi que le type de cityobject que l'on souhaite charger (qui correspond à une couche de données).

D. Rayons

RAY.HPP – RAY.CPP – HIT.HPP

Structure Ray

Représente un rayon dont les caractéristiques sont : son point d'origine et sa direction, les coordonnées du pixel correspondant au rayon, sa direction inverse, les signes des trois composantes de sa direction ainsi que la collection à laquelle il appartient.

On garde aussi la liste des boites englobantes que ce rayon traverse, cet attribut n'est utilisé que lors de l'algorithme de visibilité multi-tuile. La liste est vide à tout autre moment.

On peut utiliser deux fonctions d'intersection (*Intersect*) : rayon -> triangle (Copié depuis la librairie Geometric Tools Engine, <http://www.geometrictools.com/>) et rayon -> boite englobantes (Copié depuis pbrt-v2, <https://github.com/mmp/pbrt-v2>).

Il est aussi possible de construire un rayon en spécifiant une coordonnée de pixel ainsi qu'une caméra (Algorithme : http://www.unknownroad.com/rtfm/graphics/rt_eyerays.html)

Structure RayCollection

Nous permet de stocké une liste de pointeur vers des rayons, et ne pas avoir à s'occuper à delete tous les rayons. On a aussi un pointeur vers le point de vue associé à cette collection. Il est possible de construire une collection de rayon à partir d'une caméra (*BuildCollection*).

E. Point de vue

VIEWPOINT.H – VIEWPOINT.CPP

Structure Skyline

La skyline d'un point de vue. On stocke la liste des pixels formant la skyline dans l'image d'un point de vue et la liste des points 3D correspondant (ces deux listes ont et doivent toujours avoir la même taille). On a aussi la position du point de vue.

Il y a aussi différentes variables statistiques de la skyline, que l'on peut calculer en appelant la fonction `ComputeData`. Ce qu'on va appeler rayon dans la suite de ce paragraphe est le rayon d'un cercle de centre le point de vue passant par un point de la skyline. Les variables statistiques sont, la liste des rayons de la skyline, le rayon minimum et maximum, ainsi que la moyenne et l'écart type des rayons.

Structure ViewPoint

C'est la structure qui va nous servir à stocker toutes les données d'une analyse de points de vue. On va stocker un tableau de `Hit` qui va correspondre à chaque pixel de notre image résultat, la taille de notre image est disponible en attribut. On va avoir la direction de la lumière si jamais une image colorée doit être générée. Il y a aussi les distances minimum et maximum auxquelles on va toucher un triangle dans la scène. On va aussi stocker la position du point de vue et la skyline de notre vue.

Fonction ComputeSkyline

C'est la fonction qui va calculer la skyline de notre point de vue. L'algorithme de freeman est utilisé pour ça, c'est une simple détection de contour.

Fonction ComputeMinMaxDistance

Cela va calculer les distances minimum et maximum auxquelles on va toucher un triangle dans la scène en parcourant notre tableau de Hit et en faisant une série de min/max.

Fonction Reset

Cette fonction va réinitialiser le tableau de `Hit` ainsi que les distances et la skyline.

III. Algorithme

A. Détection de toits plats

FLATROOF.HPP – FLATROOF.CPP

Fonction DetectionToitsPlats

Permet de détecter les toits plats d'une tuile CityGML. Pour cela il faut fournir le chemin vers la tuile, une aire minimum pour que le toit soit pris en compte, ainsi qu'une valeur pour la pente pour que le toit soit considéré plat.

Pour chaque bâtiment de la tuile, on va projeter en 2D son toit et calculer son aire. Ensuite si l'aire est supérieure au minimum souhaité on va regarder la normale (3D) du toit pour avoir sa valeur de pente. Si le toit n'est pas trop pentu on va l'ajouter dans un fichier Shp.

B. Lancer de rayon

RAYTRACING.H – RAYTRACING.CPP

Structure RayTracingData

Cette structure contient tout ce qu'il faut pour effectuer l'algorithme de lancer de rayon sur un thread, la liste des triangles du modèle 3D que l'on souhaite ainsi que la liste des rayons à utiliser.

Fonction RayTracing

C'est cette fonction qui va organiser le lancer de rayon d'un ensemble de rayon sur un ensemble de triangles.

En premier le nombre de thread disponible est récupéré, on en enlève un pour éviter de bloquer complètement l'ordinateur de l'utilisateur. Ensuite on va partager l'ensemble de rayon entre chaque thread équitablement. Pour finir on va lancer la fonction [RayLoop](#) sur chacun des threads disponibles avec pour chacun la liste de tous les triangles ainsi que les rayons qu'ils devront utilisés pour le ray tracing.

Fonction RayLoop

C'est cette fonction qui va effectuer le ray tracing d'un ensemble de rayon sur un ensemble de triangles. Le résultat sera stocké dans le point de vue référencé par chaque rayon.

Pour chaque rayon, on va parcourir tous les triangles et regarder si le rayon intersecte ce triangle. Dans le cas échéant on vérifie que ce triangle est plus proche du dernier triangle intersecté, s'il existe. On inscrit ensuite le résultat dans le point de vue référencé par le rayon.

C. Extrusion de fichier Shp

SHP EXTRUSION.H — SHP EXTRUSION.CPP

Typedef LRing

C'est une simple liste de vecteur 3D.

Fonction ShpExtrusion

Tout d'abord, l'utilisateur va choisir un fichier Shp à extruder, ensuite on va parcourir chacun des polygones de ce fichier Shp.

Pour chaque polygone on va récupérer sa hauteur, l'attribut HAUTEUR. Ensuite on va récupérer l'OGRLinearRing extérieur du polygone et le transformer en [LRing](#) grâce à la fonction [OGRLinearRingToLRing](#). Ce [LRing](#) étant à la hauteur zéro, on va utiliser la fonction [PutLRingOnTerrain](#) pour le placer sur le terrain. On va ensuite dédoubler ce [LRing](#) et affecter la hauteur récupérée depuis les attributs à ce nouveau [LRing](#) avec la fonction [GetLRingWidthHeight](#), on a maintenant le sol et le toit. On va répéter l'opération pour les OGRLinearRing intérieurs du polygone.

Enfin, à partir de chacun des [LRing](#), on va créer le polygone CityGML avec la fonction [BuildPolygon](#) et ajouté les polygones résultant au model CityGML d'export. Pour finir on va relier les [LRing](#) de sol et toit ensemble pour créer les murs avec la fonction [GetWall](#).

Le model CityGML est ensuite exporter.

Fonction GetLRingWidthHeight

Cette fonction prend en paramètre un [LRing](#) et une hauteur. On va copier le vecteur passer en paramètre et ajouté la hauteur voulu à sa composante Z.

Fonction LRingToCityRing

Cette fonction va convertir un [LRing](#) en un CityGML LinearRing, en lui donnant un nom passé en paramètre ainsi qu'en précisant si c'est un anneau intérieur ou extérieur.

Fonction BuildPolygon

Cette fonction prend en paramètre un [LRing](#) correspondant à l'anneau extérieur ainsi qu'une liste d'anneau intérieur. Un polygon CityGML est créé et retourné en résultat. Pour convertir les [LRing](#) en CityGML LinearRing on va utiliser la fonction [LRingToCityRing](#).

Fonction GetWall

A partir d'un [LRing](#) de sol et d'un autre de toit, on va relier chacun des segments du sol avec ceux des toits pour former un rectangle. On va ensuite retourner cet ensemble de rectangle.

Fonction OGRLinearRingToLRing

Parcours tous les points d'un OGRLinearRing et les ajoutes dans un [LRing](#), retourne ensuite ce [LRing](#).

Fonction PutLRingOnTerrain

C'est cette fonction qui va nous permettre de placer un [LRing](#) à la hauteur du terrain.

Tout d'abord on va charger la collection de boites englobantes et ne garder que celle dont des points du [LRing](#) sont dedans.

Ensuite pour chacun des points du [LRing](#), on va charger la tuile qui correspond à la boite où est ce point et lancer un rayon depuis le point vers le haut. On va avoir un point d'intersection avec le terrain, et on va modifier la hauteur de notre point dans le [LRing](#).

D. [Arbre d'alignement](#)

ALIGNEMENTTREE.HPP – ALIGNEMENTTREE.CPP

Fonction ExtrudeAlignementTree

Grâce à cette fonction, on va pouvoir charger un fichier Shp d'arbre d'alignement et en générer les modèles 3D.

Tout d'abord on va demander à l'utilisateur de charger un fichier Shp. Ensuite on va récupérer la liste des points et features du Shp puis utiliser la fonction [PutLRingOnTerrain](#) (disponible dans la rubrique Extrusion de Shp) pour placer tous les points sur le terrain.

Pour chaque point on va extraire des attributs de la feature et créer un modèle 3D CityGml d'un arbre avec la fonction [GenCylindre](#) pour pouvoir exporter un modèle CityGML avec tous les arbres dedans.

Fonction GenCylindre

Cette fonction prendre en paramètre, un point 3D qui sera la base de l'arbre, une OGRFeature qui contiendra les informations sur l'arbre et un modèle CityGML ou l'on va exporter le modèle 3D de l'arbre.

Les informations extraites de la feature sont :

- L'identifiant de l'arbre : Attribut "identifian"
- Le rayon du fut : en fonction de la circonférence (Attribut "circonfere").
- La hauteur du fut : Attribut "hauteurfut"
- La taille de la couronne, en fonction de l'attribut "hauteurtot" et ce la hauteur du fut
- Le rayon de la couronne : Attribut "rayoncouro"

A partir de ces informations on va générer deux pavés avec la fonction BuildCube qui correspondront au fut et à la couronne de l'arbre. Ces deux pavés seront ajoutés à un CityObject qui lui-même sera ajouté au modèle CityGML d'export.

Fonction BuildCube

Cette fonction prend en paramètre un nom d'objet, une position 3D pour la base du cube (le point au centre du carré du bas), la hauteur et le rayon du cube (distance du point centrale jusqu'à un point d'un coin du carré de la base).

Avec ces paramètres on va générer une géométrie CityGML correspondant au cube voulu.

E. Donnée Lidar en CityGML : les parcs

VEGETOOL.HPP – VEGETOOL.CPP

Fonction ProcessLasShpVeget

Cette fonction va prendre deux fichiers choisis par l'utilisateur, un fichier Shp correspondant à tous les parcs souhaités et un fichier las contenant des données lidar.

Ensuite on va parcourir le fichier Shp pour récupérer tous les polygones et les stockés dans un vector, on en profite pour récupérer l'identifiant du polygone (qui est celui du parc) avec l'attribut "gid".

Ensuite pour chacun des points du lidar, on va parcourir notre vector de polygone et vérifier si ce point appartient au parc. Dans le cas échéant on va stocker ce point dans un dictionnaire, dont la clé est le polygone et la valeur, un vector contenant tous les points appartenant à ce polygone.

Une fois cela effectué on va exporter ces données intermédiaires dans un fichier texte.

Fonction ProcessCL

Cette fonction prends en paramètre un dictionnaire de données intermédiaires, la clé étant un identifiant de polygone et la valeur, l'ensemble des points de ce polygone. Ici c'est un faux dictionnaire, c'est un vector contenant des paires, identifiant de polygones vers un ensemble de points. On doit aussi fournir le nom du fichier CityGML que l'on va exporter.

Ensuite pour chacun des polygones on va appeler la fonction [VegToCityObject](#) qui va transformer notre dictionnaire de polygones/points en un cityobject. On va ajouter ce cityobject à un modèle CityGML et l'exporter.

Il existe une surcharge à cette fonction qui prend en paramètre un chemin vers un fichier texte de données intermédiaire et qui va lire ce fichier pour ensuite appeler la fonction [ProcessCL](#) de base.

Fonction VegToCityObject

Cette fonction va prendre en paramètre une paire polygones/ensemble de points, ainsi qu'un compteur pour pouvoir générer des identifiants unique, et va générer un cityobject. Le but va être de plonger les points dans une grille pour générer un modèle 3D.

Tout d'abords, on va récupérer les valeurs minimum et maximum des points fournis en paramètre pour avoir une boîte. Le but va être de générer une grille à partir de cette boîte. Ensuite on va récupérer la longueur lg de la grille en fonction des points min et max. On va utiliser cette longueur ainsi qu'un facteur f pour trouver la taille t d'une cellule de notre grille. La formule est la suivante : $t = lg/(lg/f)$. Plus f est grand, plus les cellules de la grille seront grandes. La valeur par default est 2.0 et

permet d'avoir assez de précisions dans avoir de trous dans le modèles 3D résultant de l'opération. On fait la même chose avec la largeur de la grille.

On va ensuite créer notre grille à partir de la longueur et largeur calculées. Dans notre implémentation il y a deux grilles, l'un pour stocké une somme de hauteur et l'autre pour compter le nombre de points présent dans une case de notre grille.

On va ensuite parcourir tous les points du polygone, calculer ses coordonnées dans la grille et mettre à jours les valeurs, ajouté la hauteur dans la grille correspondante et incrémenté le compteur de l'autre grille.

Pour finir on va parcourir notre grille et créer un maillage triangulé de cette manière : pour un point de la grille (i,j) on va créer deux triangles correspondant au carré formé des quatre points $(i,j), (i+1,j), (i+1,j+1), (i,j+1)$. Les hauteurs de chacun des points seront en fonction de la hauteur de chaque case de la grille pondérer par le nombre de points présents dans cette case.

On va ensuite retourner ce modèle.

Fichier d'export de données intermédiaires

C'est dans ce fichier que l'on va stocker les données intermédiaires issue du traitement de données lidar avec un fichier Shp

Format du fichier :

```
<Nombre de Polygone du Shp>
<#Polygon 1>
<#Polygon 2>
...
<#Polygon X>

#Polygon :
<Nombre de points dans ce polygone>
<Identifiant du polygone>
<#Point 1>
<#Point 2>
...
<#Point X>

#Point :
<Position X>
<Position Y>
<Position Z>
```

F. Visibilité

VISIBILITE.HPP

1. Analyse Mono-Tuile

MONOTILEANALYSIS.CPP

Fonction BasisAnalyse

Cette fonction prend simplement une liste de tuile, ainsi qu'une caméra en paramètre.

Pour effectuer l'analyse, on va juste appeler la fonction [DoMonoTileAnalysis](#) avec en paramètre la liste de tuile et la caméra.

Fonction CascadeAnalyse

Cette fonction prend en paramètre une liste de tuile, une caméra modèle, le nombre de cascade à effectuer ainsi que la distance verticale en chacun cascade.

On va d'abord créer une liste de caméra basé sur la caméra modèle de taille voulu avec les hauteurs voulus, dont la composante Z de la position sera incrémenté suivant le paramètre fourni.

Enfin on va appeler la fonction [DoMonoTileAnalysis](#) avec la liste des tuiles et la liste de nos caméras.

Fonction MultiViewpointAnalyse

Cette fonction prend en paramètre une liste de tuile, une caméra modèle et une liste de couples position/direction.

On va d'abord créer une liste de caméra basé sur la caméra modèle voulu, donc la position et la direction seront suivant la liste fournie.

Enfin on va appeler la fonction [DoMonoTileAnalysis](#) avec la liste des tuiles et la liste de nos caméras.

Fonction DoMonoTileAnalysis

C'est cette fonction qui va effectuer l'analyse de points de vue.

Elle prend en paramètre une liste de caméra et une liste de tuile.

Tout d'abord on va parcourir toutes les caméras et créer un point de vue ([ViewPoint](#)) et une collection de rayon ([RayCollection](#)) par caméra.

Ensuite pour chacune des tuiles données en paramètre, on va charger les triangles de la tuile et les stocké. Enfin pour chacune des tuiles, on va effectuer un lancer de rayons sur les triangles de la tuile avec l'ensemble des rayons de toutes les caméras.

Pour finir, pour chacun des points de vue, on appeler les fonctions [ComputeSkyline](#) et [ComputeMinMaxDistance](#), exporter les résultats et les stocké dans un vector de retour.

2. Analyse Multi-Tuile

MULTITILEANALYSIS.CPP

Fonction MultiTileBasicAnalyse

C'est cette fonction qui va effectuer l'analyse de points de vue. Il faut lui fournir le chemin vers le répertoire des tuiles CityGML, la caméra à utiliser pour faire l'analyse, et un préfix pour les noms de fichiers d'export.

L'analyse va se faire en plusieurs étapes, premièrement on va faire une analyse partiel sur chacune des couches de données disponibles (actuellement, bâtiment, terrain, cours d'eau et végétation) en appelant la fonction [DoMultiTileAnalysis](#). Résultant de cela, on va avoir 4 point de vue que l'on va fusionner avec les fonctions [MergeViewpointTerrainOther](#) et [MergeViewpoint](#).

On va donc avoir un seul point de vue, dont on va pouvoir exporter les résultats.

Fonction MultiTileCascadeAnalyse

Cette fonction prend en paramètre le répertoire des tuiles CityGML, une caméra modèle, le nombre de cascade à effectuer ainsi que la distance verticale en chacun cascade.

On va d'abord créer une liste de caméra basé sur la caméra modèle de taille voulu avec les hauteurs voulus, dont la composante Z de la position sera incrémenté suivant le paramètre fourni.

Enfin on va appeler la fonction [MultiTileBasicAnalyse](#) pour chacune de nos caméras.

Fonction MultiTileMultiViewpointAnalyse

Cette fonction prend en paramètre le répertoire des tuiles CityGML, une caméra modèle et une liste de couples position/direction.

On va d'abord créer une liste de caméra basé sur la caméra modèle voulu, donc la position et la direction seront suivant la liste fournie.

Enfin on va appeler la fonction [MultiTileBasicAnalyse](#) pour chacune de nos caméras.

Fonction MultiTilePanoramaAnalyse

Cette fonction a des paramètres identiques à la fonction [MultiTileBasicAnalyse](#). Pour avoir un panorama, on va juste lancer 4 fois la fonction [MultiTileBasicAnalyse](#) en changeant la direction de la caméra à chaque fois, cela générera 4 points de vue correspondant à ce qu'on voit devant, à droite, derrière et à gauche.

Fonction MultiTileCascadePanoramaAnalyse

Cette fonction combine [MultiTileCascadeAnalyse](#) et [MultiTilePanoramaAnalyse](#).

Fonction DoMultiTileAnalysis

Cette fonction va effectuer l'analyse et retourner un point de vue, elle prend en paramètre, le répertoire des tuiles, une liste de boîtes englobantes correspondant à la couche de données voulue, la caméra à utiliser pour l'analyse et la couche de données voulu.

Etant donné le grand nombre de tuile le but va être de minimiser le nombre de ray tracing à effectuer, si nous devons faire du ray tracing sur chacune des tuiles, cela prendrait énormément de temps. Pour ce faire on va utiliser la liste de boîte englobantes, l'objectif va être de récupérer uniquement les boîtes dont les rayons passent à travers, puis les classer de la plus proche à la plus éloignée. On va faire le ray tracing tuile par tuile suivant cet ordre et éliminer les rayons qui touchent une tuile (dû au fait que nos boîtes sont rangées de tel que si un rayon touche une tuile, on est sûr qu'il n'en touchera plus), ce qui va accélérer le ray tracing au fur et à mesure que l'on va avancer dans notre liste de boîte.

Le cheminement de l'analyse est comme suit, tout d'abord on va créer la collection de rayon, ensuite à l'aide de la fonction Setup, on va classer les boîtes englobantes suivant l'ordre dans lequel elles sont traversées par les rayons.

Une fois la liste ordonnée de boîtes récupérées on va la parcourir et faire du ray tracing sur chacune des tuiles associées. En premier il va falloir créer une collection de rayon temporaire, on ne va faire le ray tracing qu'avec les rayons qui, n'ont pas déjà touchés une tuile, et qui traverse la boîte englobante courante. On va ensuite charger la tuile correspondant à la boîte courante et effectuer le ray tracing sur cette tuile.

Une fois toutes les boites traitées on va retourner le point de vue résultat.

Fonction Setup

C'est cette fonction qui va classer une liste de boites englobantes suivant dans quel ordre elles sont traversés par une collection de rayon.

Définition de "l'ordre de traversé" d'une boite par un rayon : soit un ensemble de boite { a,b,c,d,e } et un rayon X, si ce rayon traverse les boites dans l'ordre d,b,e,c,a alors l'ordre de traversé de d = 1, b = 2, e = 3, c = 4, a = 5 pour le rayon X.

Tout d'abord on va créer un dictionnaire qui va associer à chaque boite, son ordre de traversé maximum (de n'importe quel rayon). Pour chaque rayon, on va stocker la liste de toutes les boites qu'il traverse dans sa liste de [RayBoxHit](#), on va ensuite trier cette liste par ordre croissant, et enfin on va parcourir cette liste et mettre à jours le dictionnaire d'ordre de traversé maximum.

Une fois notre dictionnaire complet, on va créer une liste de [BoxOrder](#) à partir de ce dictionnaire, puis trier cette liste pour pouvoir la retourner.

Fonction MergeViewpoint

Cette fonction va fusionner deux points de vue a et b passés en paramètre. Il n'y a pas de gestion de z-fight entre les deux couches, si besoin voir la fonction [MergeViewpointTerrainOther](#).

On va tout simplement parcourir chacun des "pixels" des points de vue et si jamais le pixel du point de vue b est devant celui du point de a alors on va remplacer le pixel de a par celui de b.

Fonction MergeViewpointTerrainOther

Même fonctionnement que la fonction [MergeViewpoint](#), cependant lors de la vérification si un pixel de b est devant celui de a, on va vérifier si ils sont à la "même" distance, cette fonction d'égalité ([AreSame](#)) est différente d'un opérateur == entre deux floats car on utilise un epsilon deux fois plus grand. Dans où les pixels sont à "même" distance on va donner priorité au pixel du point de vue b.

G. Export des données

EXPORT.HPP — EXPORT.CPP

Structure EmblematicView

C'est avec cette structure que l'utilisateur va pouvoir donner sa définition de vue emblématique, il pourra donner les différents % de différents types de données qu'il souhaite apercevoir depuis le point de vue de l'analyse.

Class ExportParameter

C'est dans cette classe que l'on va pouvoir définir des paramètres d'export, actuellement il n'y a que la vue emblématique.

Fonction LoadRemarquableBuilding

Cette fonction va charger une liste d'identifiant de bâtiment considéré comme remarquable. Le chemin vers ce fichier doit être situé dans le répertoire de tuile et avoir pour nom : Remarquable.txt. Chaque ligne correspond à un identifiant de bâtiment.

Fonction SkyShadeBlue

Cette fonction prend en paramètre la direction d'un rayon, et la direction de la lumière, et va retourner une couleur du ciel en fonction de ces deux paramètres.

Fonction ExportData

C'est avec cette fonction que l'on va pouvoir exporter des données issues d'une analyse de points de vue. Cette fonction exporte deux csv ainsi que des fichiers Shp.

En premier lieu on va incrémenter différents compteurs, ils comptent le nombre de fois que l'on voit telle ou telle types de données parmi les "pixels" de l'image de la vue :

- cpt : Nombre de pixel total
- cptMiss : Pixels qui n'ont pas touchés la scène
- cptHit : Pixels qui ont touchés la scène
- cptRoof : Pixels qui ont touchés un toit
- cptWall : Pixels qui ont touchés un mur
- cptBuilding : Pixels qui ont touchés un bâtiment
- cptRemarquable : Pixels qui ont touchés un bâtiment jugé remarquable
- cptTerrain : Pixels qui ont touchés le terrain
- cptWater : Pixels qui ont touchés un corps d'eau
- cptVegetation : Pixels qui ont touchés de la végétation

Ces compteurs sont incrémentés à la suite d'un parcours de chaque "pixel" du point de vue.

On va ensuite exporter en csv. Ce csv est en plusieurs parties, la première concerne les compteurs définis plus haut, en colonne on va avoir la valeur brute des compteurs, un pourcentage global de ce qu'il représente (cptX / cpt), puis un pourcentage par rapport au pixel qui ont touchés la scène ($\text{cptY} / \text{cptHit}$) et enfin un pourcentage par rapport au pixel qui ont touchés un bâtiment (cptZ / cpt). En ligne seront les différents compteurs.

La deuxième partie concernera la vue emblématique. Pour chaque différent pourcentage de la vue emblématique on va indiquer le pourcentage dans la vue analysé, le pourcentage souhaité et la différence des deux.

Pour la dernière partie, on va pour chaque type de cityobject, indiqué sur combien de pixel il apparaît, différent pourcentage comme les compteurs ainsi que le type et sous type du cityobject.

On va ensuite exporter la skyline dans un csv. Pour chaque point de la skyline, on va écrire ses coordonnées 3D, l'identifiant, le type et le sous-type du cityobject correspondant.

Pour finir on va exporter 5 fichiers Shp.

Le premier est le viewshed (tout ce que voit l'utilisateur depuis le point de vue), c'est-à-dire chacun des "pixels" du point de vue. Ce fichier sera nommé Result.shp. Dans ce Shp on va stocker chaque pixel en tant que point, les coordonnées du point seront les coordonnées 3D du pixel, on va aussi stocker en attribut l'identifiant, le type et le sous type du cityobject.

Le deuxième fichier, CityObjectData.Shp, va contenir les informations sur chaque cityobject vue depuis le point de vue. A chaque cityobject correspondra un point dans le Shp, la coordonnée de ce point sera une moyenne des coordonnées 3D de chaque pixel correspondant à ce cityobject. On va aussi stocker en attribut, l'identifiant, le type, le sous type du cityobject ainsi que le pourcentage d'occupation de la vue par ce cityobject.

Le troisième fichier, SkylinePoints.Shp, contiendra chacun des points de la skyline, les coordonnées des points dans le Shp seront celles des points 3D de la skyline. Aussi pour chaque point on va garder en attribut, l'identifiant, le type et le sous type du cityobject de chaque point. En attribut est aussi stocké l'ordre du point dans la skyline, cela peut permettre éventuellement de recréer le polygone de la skyline sans le Shp suivant.

Le quatrième Shp, SkylineLine.Shp, est le polygone de la skyline, c'est le polygone que forment tous les points de la skyline. On va stocker en attribut de ce polygone, toutes les informations statistiques de la skyline : le rayon minimum/maximum, le rayon moyen et l'écart type (Cf. Données/Skyline).

Le dernier fichier contient unique un point qui correspond au point de vue utiliser pour l'analyse.

Fonction ExportImages

Cette fonction va appeler en séquence toutes les fonctions d'export d'image définies dans Export.hpp

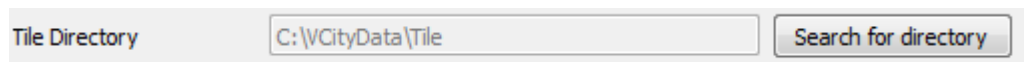
IV. Interface

A. DialogVisibilite

DIALOGVISIBILITE.HPP – DIALOGVISIBILITE.CPP

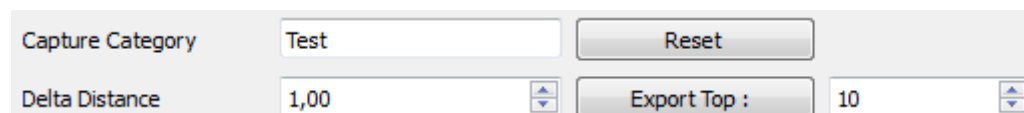
C'est la fenêtre qui va servir à faire les analyses

Répertoire des tuiles CityGML



Bouton Search for directory : Sert à définir le répertoire des tuiles CityGML. Slot : [DirButtonClicked](#)

BelvedereDB



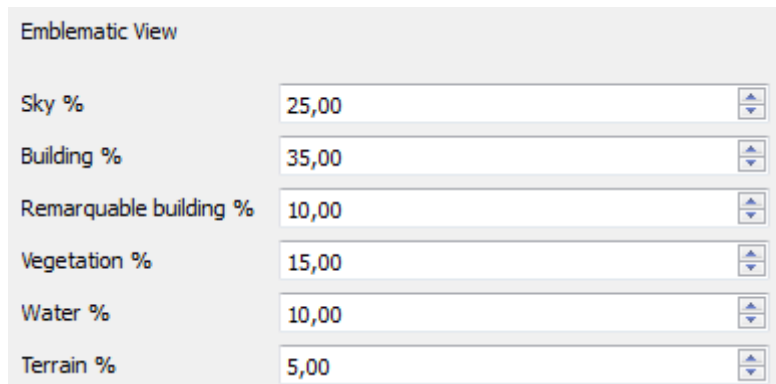
La catégorie de capture correspond au label voulu pour la base de données des belvédères.

Bouton Reset : Sert à réinitialiser la base de données des belvédères avec le label fourni. Slot : [ResetCategory](#)

Delta Distance : Distance minimum entre deux points de vue pour que les deux soient pris en compte dans la base.

Bouton Export Top : Set à exporter le polygone les plus vues de la base de données des belvédères, suivant le label fourni. Il faut entrer un nombre de polygone exporter dans la case suivante. Slot : [GetTopPolygon](#)

Vue emblématique

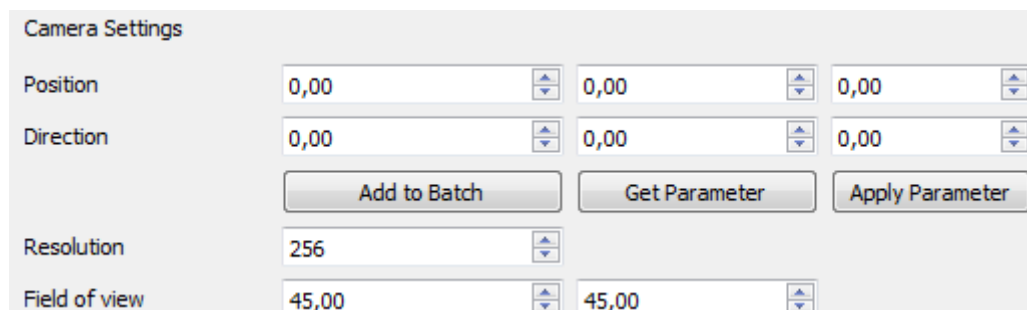


Emblematic View

Sky %	25,00
Building %	35,00
Remarquable building %	10,00
Vegetation %	15,00
Water %	10,00
Terrain %	5,00

Sert à définir la vue emblématique souhaitée, utilisé pour comparer les résultats lors de l'export.

Personnalisation de la caméra



Camera Settings

Position	0,00	0,00	0,00
Direction	0,00	0,00	0,00
	Add to Batch	Get Parameter	Apply Parameter
Resolution	256		
Field of view	45,00	45,00	

Position : Coordonnées 3D de la caméra, sans l'offset 3D-Use

Direction : Direction de la caméra, la norme du vecteur direction doit être 1.

Bouton Add to Batch : Permet d'ajouter le couple position direction dans le fichier BatchViewpoints.txt. Lors du lancement d'un batch d'analyse, les caméras sont réglés suivant les positions et directions dans ce fichier. Slot : [CopyPointToBatch](#)

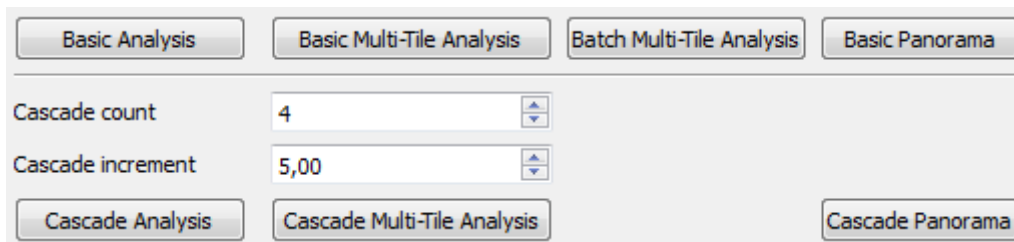
Bouton Get Parameter : Copie les coordonnées de la caméra de 3D-Use et les inscrit dans les emplacements correspondant de la fenêtre. Slot : [GetCamParam](#)

Bouton Apply Parameter : Copie les coordonnées de la caméra inscrit dans les champs de la fenêtre dans la caméra de 3D-Use. Slot : [SetCamParam](#)

Resolution : Résolution de l'analyse, correspondant aussi à la longueur de l'image résultant de l'analyse (en pixel). La largeur est calculée en fonction des champs de visions

Field of view : Champs de visions horizontal et vertical souhaités.

Analyses



Bouton Basic Analysis : Appelle la fonction d'analyse, [BasisAnalyse](#). Slot : [BasicMonoTile](#)

Bouton Basic Multi-Tile Analysis : Appelle la fonction d'analyse, [MultiTileBasicAnalyse](#). Slot : [BasicMultiTile](#)

Bouton Batch Multi-Tile Analysis : Appelle la fonction d'analyse, [MultiTileMultiViewpointAnalyse](#), avec les positions et direction lu depuis le fichier de batch. Slot : [BatchMultiTile](#)

Bouton Basic Panorama : Appelle la fonction d'analyse, [MultiTilePanoramaAnalyse](#). Slot : [BasicPanorama](#)

Cascade count : nombre de cascade à effectuer lors d'une analyse en cascade.

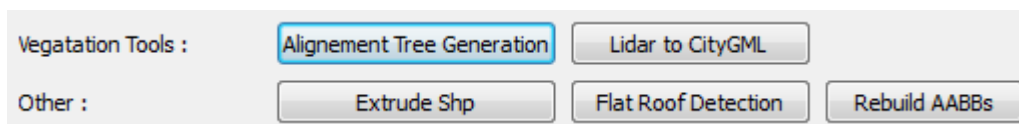
Cascade increment : Incrément vertical entre chaque analyse en cascade.

Bouton Cascade Analysis : Appelle la fonction d'analyse, Cascade Analyse. Slot : [CascadeMonoTile](#)

Bouton Cascade Multi-Tile Analysis : Appelle la fonction d'analyse, [MultiTileCascadeAnalyse](#). Slot : [CascadeMultiTile](#)

Bouton Cascade Panorama : Appelle la fonction d'analyse, [MultiTileCascadePanoramaAnalyse](#). Slot : [CascadePanorama](#)

Outils



Bouton Alignement Tree Generation : Permet la génération des modèles 3D pour les arbres d'alignement. Slot : [ToolAlignementTree](#)

Bouton Lidar to CityGML : Permet de générer les modèles 3D de la végétation à partir de données lidar et Shp. Slot : [ToolLidarToGML](#)

Bouton Extrude Shp : Permet l'extrusion d'un fichier Shp. Slot : [ToolShpExtrusion](#)

Bouton Flat Roof Detection : Permet de détection les toits plats de bâtiment CityGML. Slot : [ToolFlatRoof](#)

Bouton Rebuild AABBs : Permet de régénérer la liste des boîtes englobantes pour chaque tuile CityGML présent dans le répertoire fourni. Slot : [ToolAABBReconstruction](#)