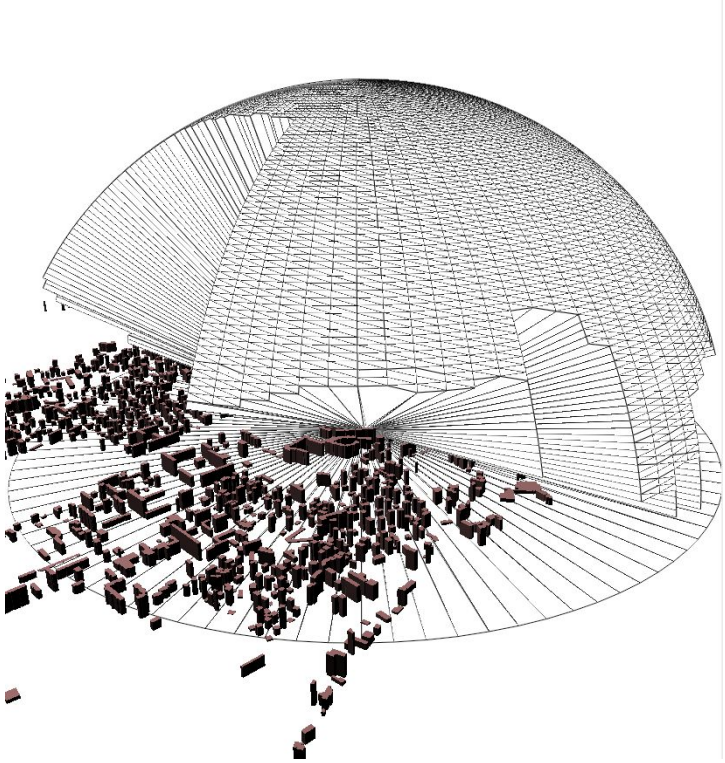
# Travaux pratiques : initiation à l'analyse spatiale 3D

Implémentation de quelques méthodes de calcul de visibilité



Auteur : Mickaël Brasebin (Équipe COGIT)

Version : V1

Cours M2TSI 2016/2017

# 1 - Objectif du TP

L'objectif de ce TP est de manipuler par le code des données géographiques 3D et de développer un outil basé sur l'analyse de visibilité. Ce TP permet de se confronter aux difficultés d'effectuer des analyses spatiales en 3D.

Il s'agira d'implémenter (en python) un outil permettant d'effectuer des calculs de lancers de rayons. Les lancers de rayons sont beaucoup utilisés lorsqu'il s'agit d'évaluer l'intervisibilité entre des objets (bâtiments, ciel, soleil, etc.) depuis un ou plusieurs sommets. Ce TP décrit brièvement les étapes nécessaires pour parvenir à un tel résultat "from scratch" et propose de tester quelques types d'analyse dérivés (vérification de la visibilité entre deux sites, calcul d'ouverture de ciel) de ce calcul de lancer de rayons.

## Préparation du TP

- Un shapefile contenant une BDTopo triangulée (couche "bati") en 3D (à regarder dans GeOxygene3D avant le TP pour voir comment est structurée la donnée)
- Un code qui permet de lire les coordonnées en 3D d'un multipolygone z en utilisant la bibliothèque OGR (beaucoup d'API de lecture de shapefile ne conservent pas la coordonnée Z et ne peuvent être utilisées)
- Un exemple de code permettant d'écrire des shapefile3D à adapter pour produire des sorties à visualiser.
- Une couche de bâtiment classique de la BD Topo utilisée dans le dernier exercice (couche "BATI_INDIFFERENCIE")

## Notes

- La visualisation des résultats intermédiaires pourra être effectuée avec GeOxygene3D ou QGIS2Threejs suivant la nature des données (QGIS2Threejs ne peut pas représenter des polygones.
- Les codes python fournies utilisent OGR et la bibliothèque python pyshp (vérifiez qu'ils sont bien installés)

# 2 - Conception d'une boîte à outil de lancer de rayons

## 2 - 1 Méthodologie générale de calcul

Le lancer de rayons est une opération qui vise à déterminer à partir d'une demi droite définie par un sommet et une direction, le point, s'il existe, le plus proche intersectant un objet de la scène.

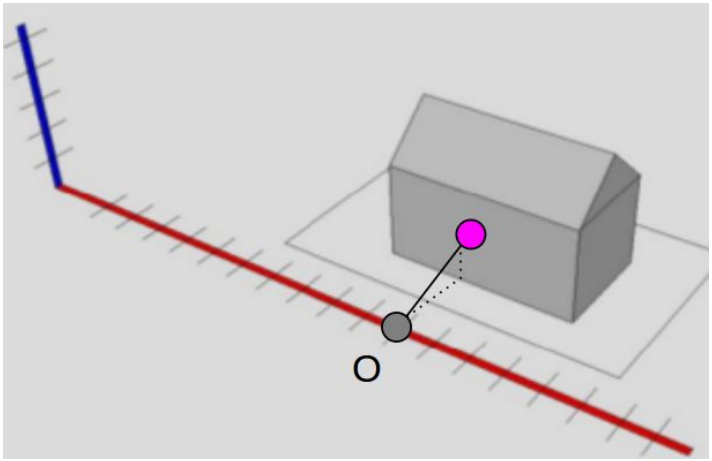



Illustration du lancer d'un rayon à partir d'un sommet (O) dans une direction donnée (ligne noire). Le point résultat est le point violet.

Ainsi, par convention, nous nommons O ($x_o$,$y_o$ ,$z_o$) le point à partir duquel le lancer de rayon est calculé et v le vecteur décrivant la direction dans laquelle le lancer est effectué et S ($x_s$,$y_s$,$z_s$) le sommet résultat.

Pour obtenir un tel résultat, il est nécessaire de calculer l'intersection entre une demi-droite et la surface des objets géographiques. Pour faire cela, nous allons considérer que cette surface se présente sous la forme d'une surface triangulée (ce qui simplifie les calculs).

# 2 - 2 Implémentation d'un algorithme de lancer de rayons

## 2 - 2 - 1 Création des classes de structure géométriques 3D

Afin d'instancier les informations nécessaires pour manipuler les informations géographiques en 3D, il faut créer quelques classes de base.

TODO : Créer :

- une classe Point3D (avec des attributs X, Y et Z)
- une classe Triangle3D (avec comme attributs 3 sommets (exprimés sous la forme de Point3D) et un identifiant).

En utilisant le jeu de données en ShapeFile, instanciez les classes Triangle3D et Point3D. Pour la classe Triangle3D, l'attribut id sera renseigné par le champs id du bâtiment auquel il appartient. Cela permettra de faire le lien entre la géométrie et l'objet géographique représenté pour les analyses.

## 2 - 2 - 2 Code permettant de calculer l'intersection entre une demi-droite et un triangle.

Le lancer de rayon unitaire est la base des analyses qui seront menées par la suite. Il s'agit de déterminer le point d'intersection entre une demi-droite et un triangle.
Une technique usuelle se déroule en 2 étapes (cf https://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/raycast/sld016.htm) :

- (1) Trouver le point d'intersection entre la demi-droite et le plan du triangle et
- (2) Vérifier que ce point se trouve bien dans le triangle.

1) Point d'intersection entre une demi-droite et un plan

Pour pouvoir déterminer analytiquement ce point d'intersection, il est nécessaire de définir les équations traduisants ce qu'est une ligne et ce qu'est un plan. La détermination de l'intersection entre une ligne et un plan est définie ici (http://geomalgorithms.com/a05-_intersect-1.html).

Une proposition de méthode pour implémenter cela est la suivante :

TODO : Créer une classe :

- LineEquation qui représente une équation de droite et se définit par une origine O et 3 coordonnées de direction v.x, v.y et v.z.
- PlaneEquation qui représente une équation de plan et qui se définit par les 4 coefficients d'une équation de plan (a,b,c,d). Comme cette équation de plan va être instanciée à partir d'un triangle, il sera nécessaire d'ajouter un constructeur qui prenne un triangle3D en paramètre.

1) Calculez le produit scalaire entre la direction de la droite et la normale au plan. Si ce produit scalaire est nul alors la droite et le plan sont parallèles (pas d'intersection ou droite incluse dane le plan).

TODO : Appliquer la formule suivante pour trouver la valeur du produit vectoriel.

$$m = v.x * a + v.y * b + v.z * c$$

2) Déterminer le point d'intersection entre le plan et la ligne.

En appliquant la formule fournie dans le lien ci-dessus, il est possible de calculer l'abscisse S1 le long de la ligne :

$S1 = - (a * x_o + b * y_o + c * z_o + d) / m$

Il faut vérifier que (S1 >= 0), c'est à dire que le sommet se trouve du bon côté de la demie-droite par rapport au sommet d'origine, puis à calculer les coordonnées (par exemple en ajoutant une méthode getCoordAt(s) sur la classe LineEquation) :

$x = x_o + S1 * v.x$
$y = y_o + S1 * v.y$
$z = z_o + S1 * v.z$

3) Test de la fonction

Vérifiez que pour les équations suivantes:

EquationPlan : 2x + y - 4z - 4 = 0
LineEquation :

- O = (0, 2, 0)
- v = (1 , 3, 1)

L'intersection est (x,y,z) = (2,8,2)

Et que pour :
EquationPlan : 2x + y - 4z - 4 = 0
LineEquation :

- O = (1, 2, 0)
- v = (1 , 2, 1)

Il n'existe pas d'intersection

2) Vérification que le point se trouve à l'intérieur du triangle.

Une technique très simple pour savoir si un point est dans un polygone consiste à vérifier si le point qui nous intéresse est du même côté de toutes les arêtes du triangle.

Le détail de l'algorithme est décrit ici et est assez simple à implémenter :
http://blackpawn.com/texts/pointinpoly/

Pour calculer les produits vectoriels, il est possible d'utiliser la bibliothèque numpy (http://www.numpy.org/).

# 3 - Analyses de visibilité

## 3 - 1 Calcul basique d'intervisibilité basique

1) Visibilité à partir d'un rayon

Pour effectuer ce calcul, il est nécessaire de ne pas prendre en compte un seul triangle, mais d'itérer sur l'ensemble des triangles de la scène et de conserver l'intersection le plus proche du sommet O.

A partir de cette étape, il est possible de réaliser de premiers calculs. Par exemple, déterminez quel triangle est visible à partir du point (1051204.0,6840521.5) dans la direction (1,1,0).

Normalement, le temps de calcul devrait être assez long.

2) Optimisation du temps de traitement

Pour diminuer le temps de traitement, ajoutez un paramètre distMax et n'intégrez au moment du chargement que les bâtiments qui sont dans une distance inférieure à distMax du point (on pourra prendre le centroid du bâtiment comme référence).

Avec des paramètres judicieusement choisis le temps de calcul devrait diminuer.

D'autres optimisations sont envisageables (à implémenter si vous le souhaitez) :

- Ne conserver que les triangles qui font face au sommet initiale (avec un test sur la normale du triangle qui doit être opposée au vecteur reliant le sommet et le centre du triangle) ;
- Eliminez les triangles qui ne sont pas dans les directions considérées par le lancer de rayon.

## 3 - 2 Analyse d'objets visibles depuis un bâtiment

En utilisant la bibliothèque que vous avez développée et en créant une couche d'objets ponctuels adéquats, déterminez la liste des bâtiments visibles depuis le sommet du bâtiment de la Communauté Urbaine de Strasbourg et de la CUS dans un rayon de 500 m.

Effectuez un contrôle des résultats en stockants les sommets et en les visualisant dans GeOxygene.

## 3 - 3  Calcul de l'indicateur d'ouverture de ciel

L'ouverture de ciel (Sky-view factor en anglais) est un indicateur qui permet d'évaluer l'effet d'ilôt de chaleur urbain (différence de température en ville entre les zones denses et les zones périphériques (http://onlinelibrary.wiley.com/doi/10.1002/joc.3370070210/pdf). Cet indicateur consiste à mesurer le pourcentage de ciel visible dans la demi-sphère supérieur mesurée au sol. En modifiant le code déjà créé, vous proposerez une méthode pour calculer cette ouverture de ciel en discrétisant la demi-sphère supérieure.

# 4 - Application à d'autres jeux de données

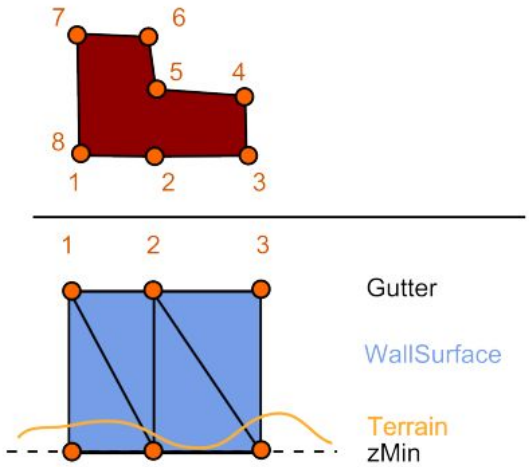
## 4 - 1 Génération 3D de la BD Topo

Les données que vous avez manipulées ont été pré-traitées, il s'agit ici d'implémenter ce traitement pour appliquer le calcul dans d'autres zones. Il s'agit de créer une méthode create3DGeometrie(layer) qui prend en entrée la couche de la BD Topo et renvoie en sortie un tableau de triangles décrivant la surface extérieur de la BD Topo après extrusion. Nous allons considérer dans cet exercice un sol plat pour simplifier les calculs.

TODO : Comme il n'existe pas de méthode permettant d'effectuer la triangulation 3D directement,il est nécessaire de devoir en concevoir une ad hoc.

Proposition de méthode en 2 étapes :

- 1) Création d'une surface 3D pour la partie supérieure
    - Création d'une couche de points à partir de polygones (**méthode extractPoints)**
    - Triangulation de cette couche **(processing.runalg('qgis:delaunaytriangulation', input, output)**
    - Suppression des triangles qui ne sont pas dans les polygones en entrée
    - Affecter comme z pour les différents sommets la hauteur du bâtiment
- 2) Création d'une surface pour les murs des bâtiments
    - En parcourant les arêtes des polygones, vous créerez pour chacune d'entre elles un couple de triangles rectangles comme décrit dans l'illustration ci-dessous.




En haut, un bâtiment vu de dessus, en bas, génération à partir de l'égout et du terrain de quelques triangles appartenant à une façade.

## 4 - 2 Application aux données CityGML

En extrayant les données CityGML en shapefile à partir de la base de données 3DCityDB, il est possible de réappliquer les méthodes proposées sur ce jeu de données.