Création d’une ville

Valentin CLIMPONT

Lorenzo MARNAT

Kirill DOROVSKIKH

Introduction

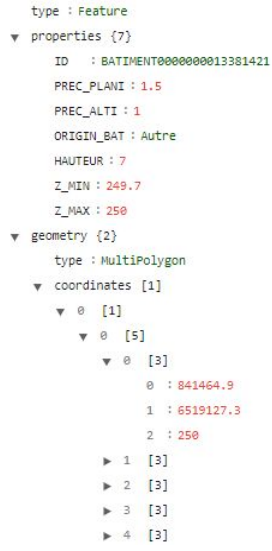
L’objectif de ce projet est de recréer une ville et ses bâtiments (ici deux quartiers de Lyon) à partir de fichiers JSON. De plus, la création d’une terrain correspondant au véritable relief de Lyon permettra de placer les bâtiments aux bonnes altitudes.

Le projet a été effectué sous Unity dans le cadre du cours de Modélisation géométrique/Mondes virtuels du Master 2 Gamagora.

Réalisation

Importation des données

Les données des bâtiments sont contenues dans des fichiers JSON. Dans un fichier, chaque ligne correspond à un bâtiment, représenté entre autres par: un identifiant, une hauteur et les points de son emprise au sol.



Exemple d’un bâtiment

Afin de transformer les données en objets sur Unity, il faut d'abord créer des classes correspondant à la sémantique des JSON.

```
[Serializable]
1 référence
public class geometry
{
    0 références
    public string type { get; set; }
    2 références
    public List<List<List<List<float>>>> coordinates { get; set; }
}
```

```
[Serializable]
1 référence
public class properties
{
    1 référence
    public string ID { get; set; }
    0 références
    public double PREC_PLANI { get; set;
    0 références
    public double PREC_ALTI { get; set; }
    0 références
    public string ORIGIN_BAT { get; set;
    2 références
    public int HAUTEUR { get; set; }
    0 références
    public double Z_MIN { get; set; }
    0 références
    public double Z_MAX { get; set; }
}
```

```
[Serializable]
3 références
public class Building
{
    0 références
    public string type { get; set; }
    3 références
    public properties properties { get; set; }
    2 références
    public geometry geometry { get; set; }
}
```

```
[Serializable]
7 références
public class RootObject
{
    0 références
    public string type { get; set; }
    0 références
    public string name { get; set; }
    3 références
    public List<Building> features { get; set; }
}
```

Il est ensuite possible de désérialiser un fichier JSON à l'aide d'un *wrapper*, en indiquant seulement le type cible, pour reconstituer les objets. Ici, on utilise la [version Unity] du wrapper open source Json.NET de NewtonSoft.

```
string json = File.ReadAllText(path);
RootObject r = JsonConvert.DeserializeObject<RootObject>(json);
```

Note: Unity possède de base son propre wrapper ([JsonUtility]), mais celui-ci n'interprète pas efficacement les listes imbriquées ou les tableaux à plusieurs dimensions.

Il arrive que les données sur la hauteur et l'altitude d'un bâtiment ne soient pas disponibles. Dans ce cas, ces valeurs sont indiquées à 0 mètre de hauteur et 9 999 mètres d'altitude. Ainsi, lors de l'import, on définit arbitrairement des valeurs par défaut afin d'éviter les incohérences. On ajoute aussi un booléen qui, si vrai, place tous les bâtiments à la même altitude (0 mètre).

```
private static float defaultHeight = 20;
private static float defaultZValue = 200;
private static bool stickToFloor = true;

0 références
public static void SetDefaultHeightValues(float height, float z)
{
    defaultHeight = height;
    defaultZValue = z;
    stickToFloor = false;
}
```

Création des bâtiments

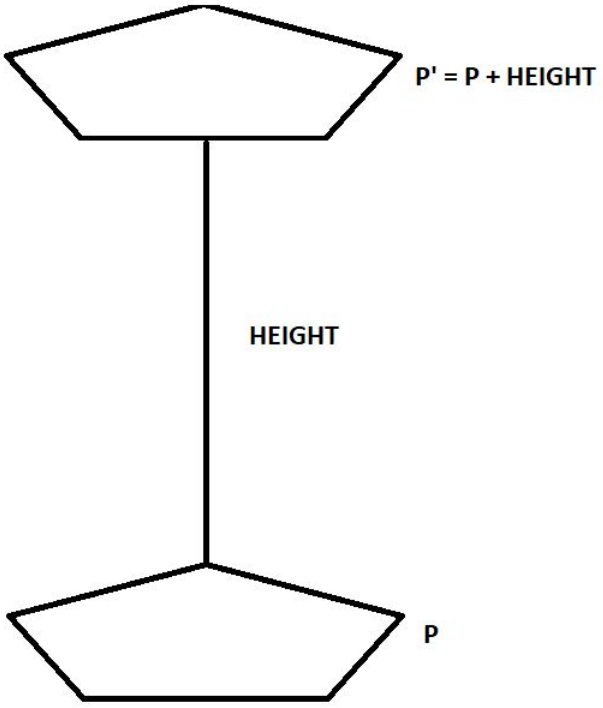
L'importation des données JSON permet de récupérer, pour chacun des bâtiments, les coordonnées des sommets qui constituent son sol, ainsi que sa hauteur. Ainsi, dans l'optique de modéliser ces bâtiments sous Unity, nous procédons en trois étapes :

1. **Création des polylignes qui constituent le sol et le toit du bâtiment.**

   L'idée est ici de récupérer chacune des coordonnées des sommets du sol d'un bâtiment définies dans le fichier JSON, et de les stocker dans un tableau de vecteurs *vertices[]*.
   Ensuite, pour chaque sommet *s* stocké, un nouveau sommet *s'* est défini et ajouté dans le tableau *vertices[]* tel que :

   s' = *s* + *height* , avec *height* la hauteur du bâtiment

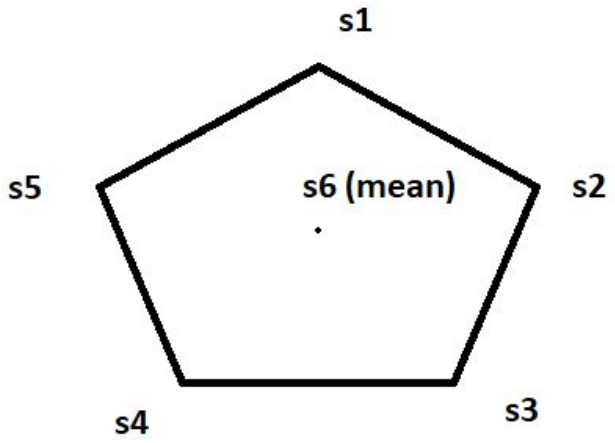
   Ainsi, le tableau de vecteurs *vertices*[] contient les coordonnées des sommets du sol et les coordonnées des sommets du toit du bâtiment.



2. **Détermination des coordonnées moyennes des polylignes.**

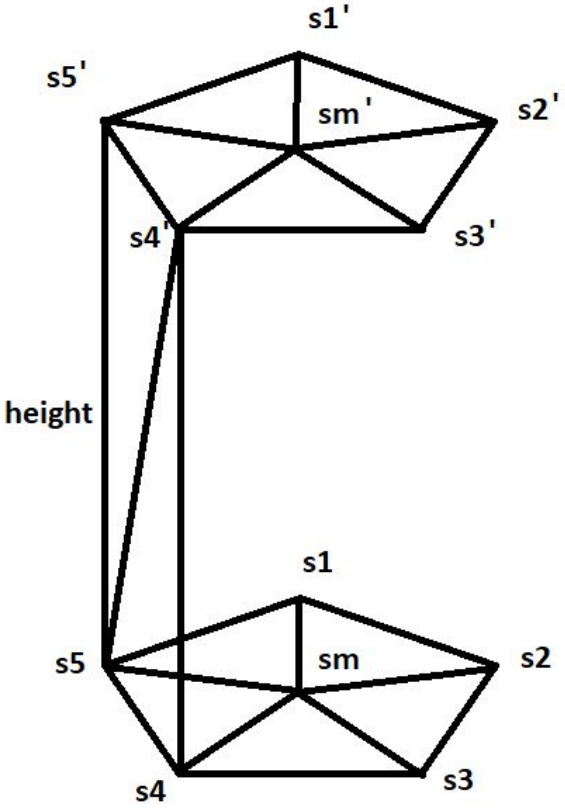
   Dans l'optique de trianguler le modèle, nous déterminons deux sommets supplémentaires définis par les coordonnées moyennes des deux polylignes créées.
   Ces coordonnées sont déterminées par la somme des coordonnées des sommets de leur polyligne, divisée par le nombre de sommets de la polyligne.

Les deux coordonnées ainsi créées sont ajoutées au tableau de vecteurs *vertices[].*



3. Triangulation du modèle bâtiment

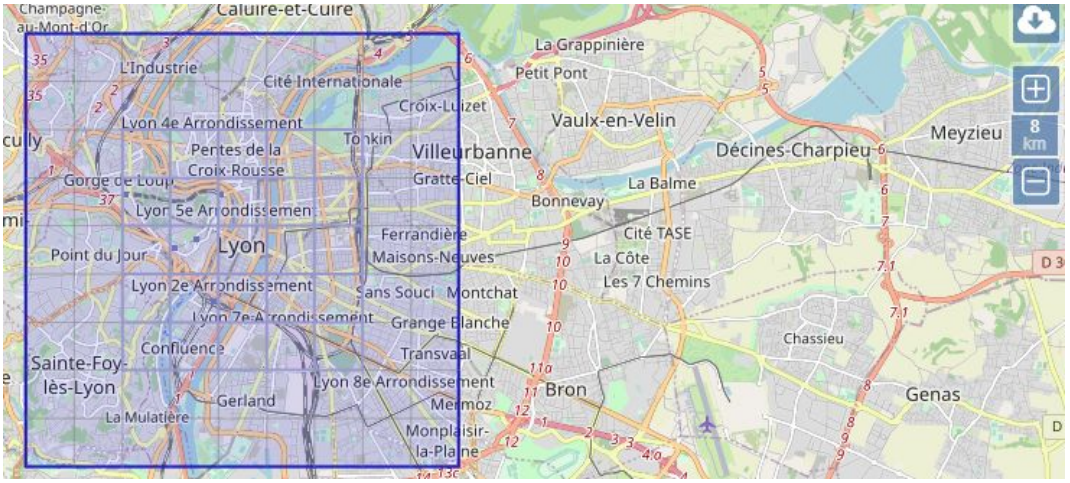
La triangulation d'un modèle bâtiment s'effectue en deux temps : triangulation des polylignes du sol et du toit, puis triangulation des faces côtés.
Dans un premier temps, la triangulation du sol (et du toit) relie chaque paire de sommets de la polyligne *s1s2, s2s3 etc.* (respectivement *s1's2', s2's3' etc.*) à son sommet moyen *sm* (respectivement *sm'*).
Dans un second temps, la triangulation des faces côtés scinde en deux triangles chaque face rectangulaire constituée de deux paires de sommets "jumeaux" (*s1s2 / s1's2', s2s3 / s2's3', etc.)*.



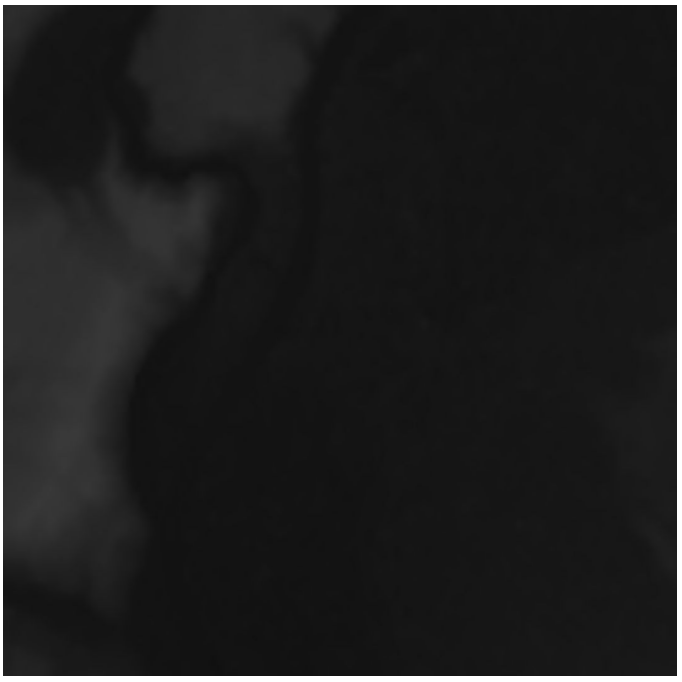
Création du terrain

Afin de placer les bâtiments de Lyon aux bonnes altitudes, il est possible de recréer le relief de la ville puis de faire correspondre les bâtiments au terrain.

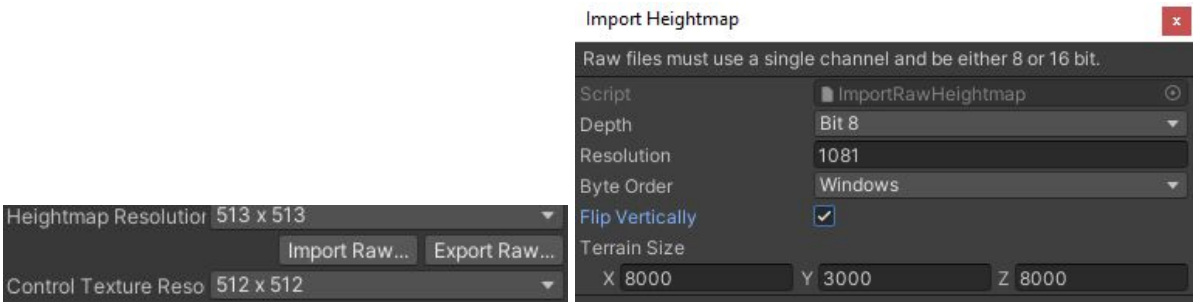
Grâce au site [terrain.party](), il est possible de récupérer une *heightmap* (ou *champ de hauteur*) de la ville de Lyon, correspondant ici à un zone de 8 km².



Une heightmap est une image en nuances de gris où chaque pixel indique une distance. Dans ce cas, la distance correspond à une hauteur par rapport à une normale. Plus le pixel est blanc, plus l'altitude est élevée.



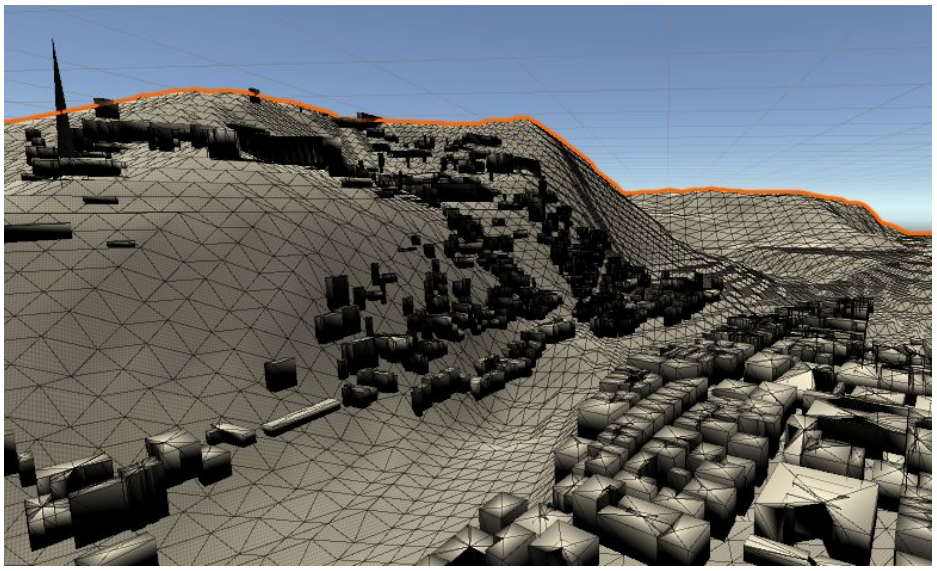
Après avoir exporté cette image en fichier .raw ou .data, il est possible d'appliquer la heightmap à un terrain sur Unity en important le fichier. On définit un terrain de 8km par 8km pour qu'il soit à l'échelle.

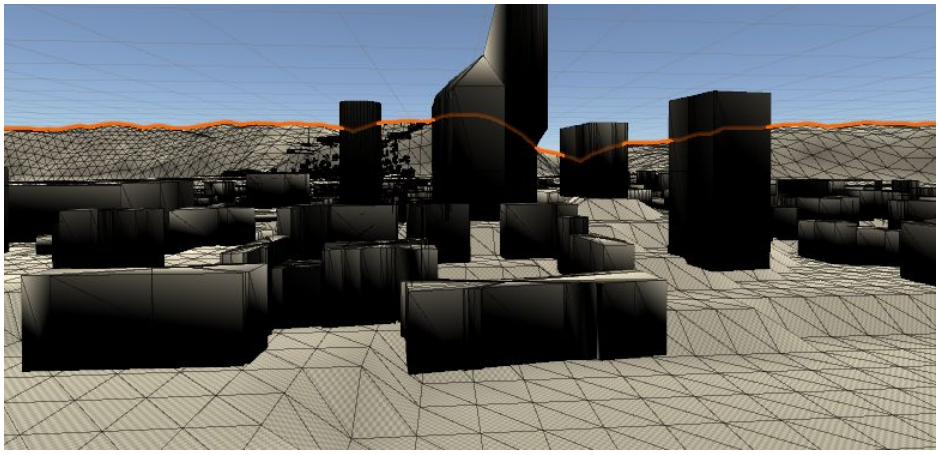


Une fois le terrain généré, on le positionne correctement sur les axes horizontaux (sur Unity, x et z). Par soucis de temps, le positionnement du terrain a été fait manuellement en utilisant certains bâtiments comme points de repère. Maintenant, il n'y a plus qu'à gérer l'altitude des bâtiments.

Pour cela, on tire un rayon vers le bas depuis le centre de chacun des bâtiments. Si le terrain est touché, on déplace le bâtiment de la distance parcourue par le rayon avant d'entrer en collision avec le terrain.

Ainsi, les bâtiments sont placés à la bonne altitude sur le terrain:





Conclusion

Le programme final permet de générer facilement et rapidement une ville ou un quartier à partir d'un fichier JSON. Le résultat obtenu est totalement satisfaisant si on ne cherche pas à avoir des bâtiments aux modèles très précis.

L'algorithme utilisé pour créer les bâtiments ne permet pas de représenter les toits. De plus, il cause parfois des triangles incohérents lorsque la forme du bâtiment n'est pas un simple polygone. Afin d'améliorer les bâtiments, une amélioration future serait d'utiliser un algorithme de triangulation minimisant le nombre de triangles, par exemple en ne créant pas de triangle à l'intérieur d'un bâtiment.

Pour ce qui est du terrain, se baser sur une image possède deux désavantages:

- Le bruit de l'image génère beaucoup de petites bosses et de petits trous sur le terrain
- La résolution de l'image ne permet pas de représenter le relief véritable de la ville avec une grande précision.