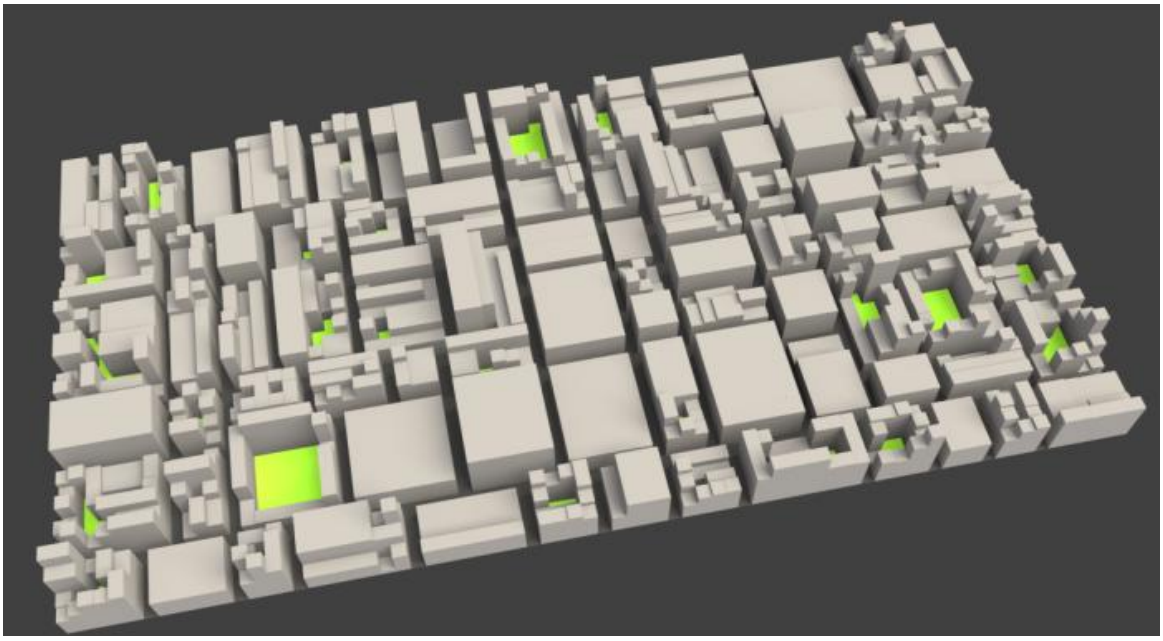
Projet 1
Générateur de villes en 3D



Le plan de la ville est représenté par une surface plane quadrillée de manière uniforme, chaque carré du quadrillage mesurant 1x1. Tous les objets qui seront positionnés dans la ville seront alignés sur cette grille et leurs dimensions dans le plan seront donc des entiers. Les dimensions du plan de la ville sont des paramètres du programme (*citySizeX*, *citySizeY*).

La première étape consiste à découper la ville en blocs avec des routes verticales et horizontales. Pour ce faire, vous devez travailler de manière récursive en positionnant à chaque niveau de récursivité une route horizontale et une route verticale. Vous découpez ainsi la zone courante en quatre blocs. Les positions des routes au sein de la zone courante sont déterminées aléatoirement (distribution uniforme). La découpe ne peut jamais générer un bloc dont une des dimensions est inférieure à *minBlockSize*. Lorsqu'un bloc a une de ses dimensions comprise entre *minBlockSize* et *maxBlockSize*, il ne peut plus être découpé dans cette dimension. La récursivité s'arrête donc quand les deux dimensions d'un bloc sont comprises entre *minBlockSize* et *maxBlockSize*, deux paramètres du programme. La largeur d'une route vaut toujours 1.

Une fois la ville découpée en blocs, vous devez découper chacun de ceux-ci en bâtiments. Pour cela, vous travaillez à nouveau par récursivité en découpant le bloc en deux dans chaque direction, la position des découpes étant choisie de manière aléatoire (distribution uniforme). Cependant, la découpe dans chaque direction doit être conditionnée par une variable aléatoire binaire (distribution uniforme).

Les résultats des découpes sont stockés dans deux tableaux dynamiques à deux dimensions. Dans le premier tableau, vous allez stocker pour chaque case de la ville le type d'objet qu'elle contient : 0 pour une case de type route, -1 pour une case de type jardin (voir plus loin) et un entier supérieur à zéro pour une case de type bâtiment, cet entier représentant l'indice du bâtiment dans le bloc (1 pour le premier, 2 pour le second, etc). Le second tableau, de type

float, vous mettez 0.0 pour les cases de type route et jardin, et pour les cases de type bâtiment, une valeur H correspondant à la hauteur du bâtiment. Cette hauteur est calculée comme suit :

$$H = \max(0.5, N(\mu_H, \sigma_H))$$

où $N(\mu_H, \sigma_H)$ représente une variable aléatoire répondant à une distribution gaussienne [1] de moyenne μ_H et d'écart-type σ_H valant respectivement 3.0 et 1.0.

Lorsque les blocs sont découpés en bâtiments, il vous est demandé de supprimer tous les bâtiments qui n'ont pas un accès direct à au moins une rue et de les remplacer par des jardins. Vous ajouterez un paramètre à votre application permettant d'activer ou de désactiver cette fonctionnalité.

Afin de permettre de vérifier que votre plan de la ville est correct, vous devez générer un fichier *map.txt*[2] dans lequel vous allez sauver le contenu du premier tableau en utilisant la correspondance suivante : un espace pour représenter une case route ou un case jardin et une lettre pour représenter un case bâtiment, la lettre 'A' correspondant au bâtiment 1, 'B' au bâtiment 2, etc [3]. Voici deux exemples de fichier *map.txt* pour une ville de 30x15, sans et avec création des jardins :

```
AAD AAAAAAA ACDDD ABDGHHHH AAA
BCE AAAAAAA BCDDD CCEGHHHH AAA
FFI AAAAAAA EEEEF FFFGHHHH AAA
GHJ AAAAAAA GHHIJ
                  ABFHKLLL ABC
ACDEELP AEE ACEIL CEFIKLLL DDJ
ACFGGMQ BEE ADFJM DEGJMMMM EFK
BHHHHNR CEE AGHKM NPQRSTTU GHL
IJJKKOS DEE BNOPQ OPVVVWWW GIM

ACDH ABCIIIII ABG AABFFFFF AAB
BEFI ABCIIIII CEG AACGGIKL CEH
BEGI ADEIIIII DFH AADHHJMM CFI
JLNP FGGIIIII IJM AAENNOOO CGI
KMOP FHHIIIII KLN AAEPQRST DGJ
```

```
AAD AAAAAAA ACDDD ABDGHHHH AAA
B E AAAAAAA BCDDD CC GHHHH AAA
FFI AAAAAAA EEEEF FFFGHHHH AAA
GHJ AAAAAAA GHHIJ
                  ABFHKLLL ABC
ACDEELP AEE ACEIL C F KLLL DDJ
AC    Q BEE A   M D   MMMM E K
B     R CEE A   M NP     U G L
IJJKKOS DEE BNOPQ OPVVVWWW GIM

ACDH ABCIIIII ABG AABFFFFF AAB
B  I ABCIIIII C G AA     L C H
B  I A  IIIII D H AA    MM C I
J  P F  IIIII I M AAE  OOO CGI
KMOP FHHIIIII KLN AAEPQRST DGJ
```

Une fois le plan de votre ville généré et stocké dans les tableaux, vous devez le parcourir case par case et générer l'objet 3D correspondant, à savoir un parallélépipède à base carrée de côté 1 et de hauteur H. Les objets 3D seront générés dans un fichier de type OBJ. Pour colorer vos objets, vous utiliserez les trois matériaux prédéfinis dans le fichier *city.mtl*, à savoir *material.building*, *material.road* et *material.garden*.

---

[1] Pour générer des nombres aléatoires suivant une distribution gaussienne, vous pouvez utiliser les fonctions suivantes (algorithme de Box-Muller, http://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform) :

```
double randUniform()
{
    return (double)rand() / RAND_MAX;
}

double randNormal(double mean = 0.0, double std = 1.0)
{
    double u1 = randUniform();
    double u2 = randUniform();
    return mean + std * sqrt(-2.0f * log(u1)) * cos(2 * PI * u2);
}
```

[2] Pour écrire dans un fichier, vous devez utiliser la bibliothèque <fstream> :

```
ofstream f("map.txt");
f << "Hello World!" << endl;
f.close();
```

[3] Si un bloc contient plus de 26 bâtiments, vous utilisez les caractères ASCII qui suivent les lettres. En réalité, il vous suffit d'utiliser l'expression `(char)(map[i][j] + 64)`.

A titre d'exemple, un cube de côté 1 sera représenté comme suit dans le fichier OBJ :

```
...
v 0 0 0
v 1 0 0
v 1 1 0
v 0 1 0
v 0 0 1
v 1 0 1
v 1 1 1
v 0 1 1
...
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
...
```

Les nombres suivant un 'v' représentent les coordonnées x, y, z des sommets. Les nombres suivant un 'f' représentent la séquence d'indices[4] des quatre sommets qui forment une face du parallélépipède. Vous trouverez plus d'informations sur le format à l'adresse suivante : http://en.wikipedia.org/wiki/Wavefront_.obj_file. Vous avez également un exemple de fichier OBJ joint à l'énoncé, dont vous vous inspirerez pour écrire la partie génération 3D de votre programme.

Remarque : pour éviter de générer des parallélépipèdes totalement plats pour les cases de types route et jardin, ce que les programmes de rendu 3D détestent, vous leur donnerez une épaisseur de 0.01 au moment de la génération du fichier OBJ.

Afin de vérifier le résultat de votre génération 3D, utilisez **Blender** (http://www.blender.org) [5] pour charger le fichier *city_lowdef.blend* [6]. Importez-y votre fichier OBJ (File | Import | Wavefront), tapez ALT-R pour aligner correctement votre ville, puis F12 pour générer un rendu 3D (peut prendre un peu de temps). Une fois le rendu généré, sauvez-le comme une image en appuyant sur F3.

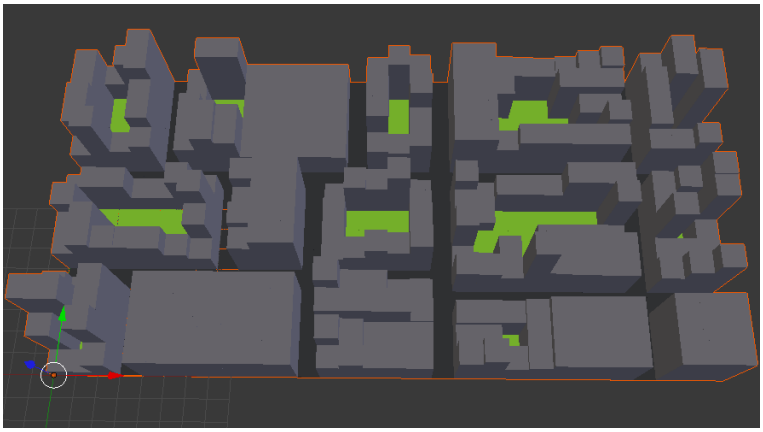


Figure 1- Exemple de ville importée dans Blender

[4] Il s'agit des indices des sommets numérotés de manière absolue pour tous les sommets du fichier. Ces indices iront donc de 1 à 8 pour le premier parallélépipède, de 9 à 16 pour le second, etc.

[5] Blender est gratuit et open source. Il est installé sur les PC du labo.

[6] Ce fichier est optimisé pour une ville générée avec les valeurs par défaut des paramètres. Il y a aussi un fichier *city_highdef.blend* qui permet de générer un rendu de meilleure qualité (avec ombrage), mais qui peut être lent sur un PC peu performant.

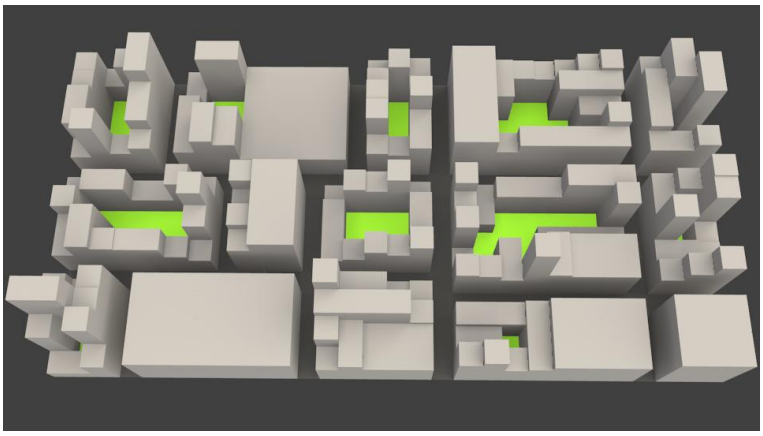


Figure 2- Exemple de rendu obtenu avec Blender (high def)

Tous les paramètres du programme doivent être entrés manuellement par l'utilisateur en début de programme, un message décrivant à chaque fois le paramètre et proposant une valeur par défaut si l'utilisateur fait directement RETURN sans rien entrer[7]. Voici un récapitulatif des paramètres, de leur valeur par défaut et des valeurs acceptables :

Paramètre	**Valeur par défaut**	**Min**	**Max**
citySizeX	75	minBlockSize	200
citySizeY	40	minBlockSize	200
minBlockSize	3	2	10
maxBlockSize	8	2	10
μ_H	3.0	1.0	10.0
σ_H	1.0	0.1	2.0
filename	« city.obj »	Ne peut pas être vide	

Consignes et modalités de remise

Nous vous demandons de respecter les règles de programmation suivantes :

1. N'utilisez aucune variable globale.
2. Définissez toutes vos constantes au moyen de `#define` ou de `const` et donnez-leur un nom en majuscules.
3. Structurez votre code en utilisant des fonctions.

[7] La commande `cin << var` ne permet pas de faire un RETURN sans avoir encodé une valeur. Vous devez donc utiliser la méthode `cin.getline(…)` : http://www.cplusplus.com/reference/iostream/istream/getline. Celle-ci vous retournera une chaîne de caractères que vous convertirez en `int` ou en `float`/`double` avec les fonctions `atoi` et `atof` (http://www.cplusplus.com/reference/clibrary/cstdlib/atoi, http://www.cplusplus.com/reference/clibrary/cstdlib/atof).

4. Donnez à vos variables, paramètres et fonctions des noms intelligibles qui favorisent la lecture et la compréhension du code.
5. Commentez votre code pour le rendre plus lisible.
6. Concevez votre programme pour qu'il soit performant.
7. Il doit également être souple pour l'utilisateur (choix des paramètres, y compris les noms de fichier).
8. Vous pouvez bien sûr séparer votre programme en plusieurs fichiers si cela est nécessaire.

Les délivrables du projet sont les suivants :

1. Le code source complet
2. Un exécutable
3. Une ville générée avec les paramètres par défaut (fichiers map.txt, city.obj et city.png pour le rendu)
4. Quelques autres exemples de villes correspondant à d'autres valeurs des paramètres
5. Un bref rapport (maximum 3 pages) décrivant votre approche du problème et la manière dont vous vous êtes réparti le travail

Le projet peut être réalisé individuellement ou par groupe de 2 étudiants maximum **du même groupe de TP (même jour),** mais la défense, qui aura lieu sur machine (de la salle info) **lors de semaine du 26/03**, comportera des questions individuelles.

Les délivrables seront remis au plus tard le **vendredi 23/03** à Benoît Penelle **sous forme numérique** (envoyez un email à benoit.penelle@ulb.ac.be avec pour sujet « INFOH200 : projet 1 – numéro série – nom_etudiant_1 nom_etudiant_2» à partir de votre compte ULB).

Aucune remise de projet au-delà de cette date ne sera prise en considération.

La défense du projet consistera en une brève démonstration de votre programme ainsi que quelques questions relatives à votre code.

Bon travail !