**Documentation des projets VCity**

**Utilisation de 3DUSE**

Ouverture d'un fichier CityGML :

File → Load…

Une fois la tuile chargée, elle apparait dans l’arbre à gauche. On peut double cliquer sur un noeud dans l’arbre pour centrer la caméra dessus. On peut masquer un noeud en décochant sa case dans l’arbre. A noter qu’il est possible de cocher / décocher toutes les cases d’une hiérarchie avec “Select all” / “Select none”.

Manipulation de la caméra :

Par défaut, la caméra trackball est utilisée. Elle permet de tourner autour d’un noeud séléctionné.

Angle de vue : clic gauche + souris

Déplacement : clic molette + souris

Zoom : clic + souris ou molette

La caméra permière personne peut être activée avec Maj+2

Les touches directionnelles déplacent la caméra. Ctrl+haut / bas, permet de monter / descendre. On peut Ajuster la vitesse de déplacement avec les touches Ctrl+gauche, pour ralentir le pas de déplacement, Ctrl+droite pour l’augmenter. Cela peut être utile pour faire des mouvements plus fins. L’angle de vue est modifiable avec la souris+clic gauche.

On revient sur la caméra trackball avec Maj+1. Il y a d’autres modes de caméra qui n’ont pas été testés.

Lorsque l'on a le focus sur la vue 3D, la touche 'h' affiche une aide sur les commandes disponibles. La touche ‘s’ permet d’afficher des compteurs de performances comme les fps. On peut presser plusieurs fois ‘s’ pour afficher plus ou moins de choses.

La touche ‘t’ permet d’afficher ou non les textures.

La touche ‘l’ permet d’activer ou non l’éclairage.

La touche ‘w’ permet de passer en mode de rendu filaire, point ou triangle.

On peut vider la scène avec le menu :

Scene → Reset

Exemples d'utilisation :

Manipulation temporel :

Il faut d'abord activer l’outil temporel en cliquant sur le bouton à gauche du slider temporel qui est en bas de la fenêtre.

Ensuite avec le slider temporel, on peut choisir une date. Seuls les éléments construits à la date sélectionnée vont apparaître.

Export CityGML :

L'export citygml prend en compte les paramètres temporels et le picking.

Si l'outil temporel est activé, alors ne seront exportés que les éléments construits à la date sélectionnée.

Si un élément est sélectionnée à la souris, alors seulement celui la sera exporté.

Picking :

Par défaut, la sélection à la souris va prendre le bâtiment qui se trouve sous le curseur. Il est possible de changer de mode et de sélectionner uniquement une face, avec le menu :

Options → Picking mode → Face / Building

**Documentation pour développeur 3D-USE**

Note - documentation d’installation de Martial, peut être à mettre à jour

1) Pour démarrer sous VCity, il faut au préalable m'envoyer par email le pseudo de votre compte Github car je dois vous inscrire à la MEPP-team car VCity est pour le moment un projet privé. Une fois cette étape réalisée, vous pourrez alors forker VCity.

La même doc Git/Github que MEPP (http://liris.cnrs.fr/mepp/mepp-git-doc.html) s'applique alors pour VCity sauf qu'il faut évidemment substituer VCity à MEPP...

- Sous Linux les dépendances sont (pour Ubuntu, par exemple) :

qt4-default libopenscenegraph-dev libgdal-dev libproj-dev

- Sous Mac OS X, les dépendances (modulo un renommage de nom de paquet...) existent aussi via Homebrew: http://brew.sh/

- Sous Windows, un kit binaire est fourni afin de simplifier, d'accélérer et surtout d'uniformiser le déploiement de VCity (cf. fichier texte "kit-vs2012-64-VCity")

---

2) IMPORTANT:

-------------

Juste une information importante, surtout pour les nouveaux, ce, afin d'éviter tout problème dans l'avenir :

--> vous ne devez pas cloner le master de VCity directement, chaque personne doit d'abord obligatoirement "FORKER" le projet sur son compte Github, le clone sur votre machine ayant lieu après cela...

--> ainsi, à partir de là, vous êtes ensuite libre de travailler sur votre clone et de faire remonter sur votre FORK vos commits quand vous le souhaitez...

--> la demande de merge sur le master se fait seulement ensuite via un Pull Request (PR)

|  |  |  |
| --- | --- | --- |
|  |  |  |

Kit binaire Windows (uniquement Visual Studio 2012 - 64 bits) pour machine 64 bits avec OS 64 bits

--------------------------------------------------------------------------------------------------

Notes:

------

- avoir la dernière version de VCity (github)

Marche à suivre:

----------------

- télécharger VS2012 express edition (http://download.gforge.liris.cnrs.fr/meppbin/windows/vs2012/VS2012\_WDX\_FRA.iso), installer par exemple avec DAEMON Tools Lite (http://www.daemon-tools.cc/fra/products/dtLite)

- OBLIGATOIRE : télécharger le service pack "update 3" de VS2012 express edition (http://download.gforge.liris.cnrs.fr/meppbin/windows/vs2012/update3.zip), dézipper et installer

- télécharger le kit VCity VS2012\_64 (http://download.gforge.liris.cnrs.fr/meppbin/windows/vs2012/VCITY/VCITY\_local\_vs2012\_64.rar)

---

- extraire le kit binaire "VCITY\_local\_vs2012\_64.rar", attention il faut que la racine de ce kit soit un chemin < 50 caractères, je vous propose donc de l'installer à la racine de votre disque c:\ (éviter aussi les dossiers exotiques "Bureau", "Mes Documents", ...)

- par défaut le dossier du kit est "VCITY\_local\_vs2012\_64" mais le dossier est renommable si besoin...

- le kit binaire occupe environ 5.3 Go sur le disque (sans compter l'installation de VS2012 et de l'update 3...)

- installer cmake-2.8.12 qui gère correctement VS2012 (vous le trouverez dans C:\VCITY\_local\_vs2012\_64\\_utils\_)

- si vous n'avez pas encore doxygen et graphviz, installer les (vous les trouverez aussi dans C:\VCITY\_local\_vs2012\_64\\_utils\_)

---

- positionner une nouvelle variable d'environnement VCITY\_KIT\_ROOT à "C:\VCITY\_local\_vs2012\_64"

- ajouter dans votre variable Path le nouveau chemin suivant ";C:\VCITY\_local\_vs2012\_64\\_bin\_" ( attention au ; ), il peut être judicieux de le mettre en tête de votre Path pour éviter des conflits éventuels avec d'autres versions de .dll (comme Qt par exemple...)

- ensuite, utilisez CMake (cmake-gui.exe)

- IMPORTANT : attention Visual Studio 2012 est en fait la 11ème version de Visual Studio, donc avec CMake, il faut choisir : "Visual Studio 11 Win64"

Affichage de 3. VCity.txt en cours...

**Utilisation de QtCreator**

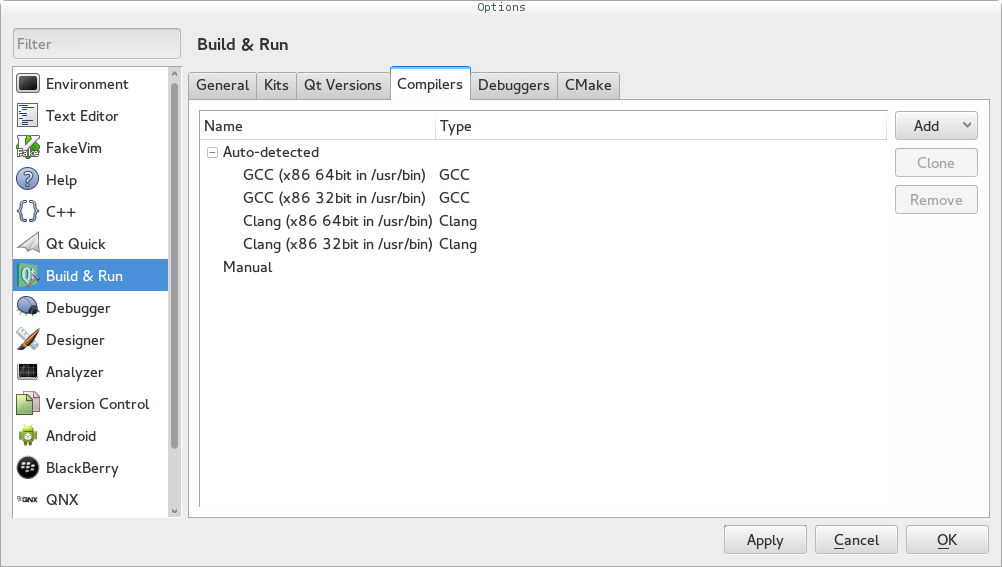
L'utilisation du cmakefile dans QtCreator n'est pas tout à fait similaire à l'utilisation d'une solution visual studio, il n'y a pas de cible debug, release. Voilà comment faire :

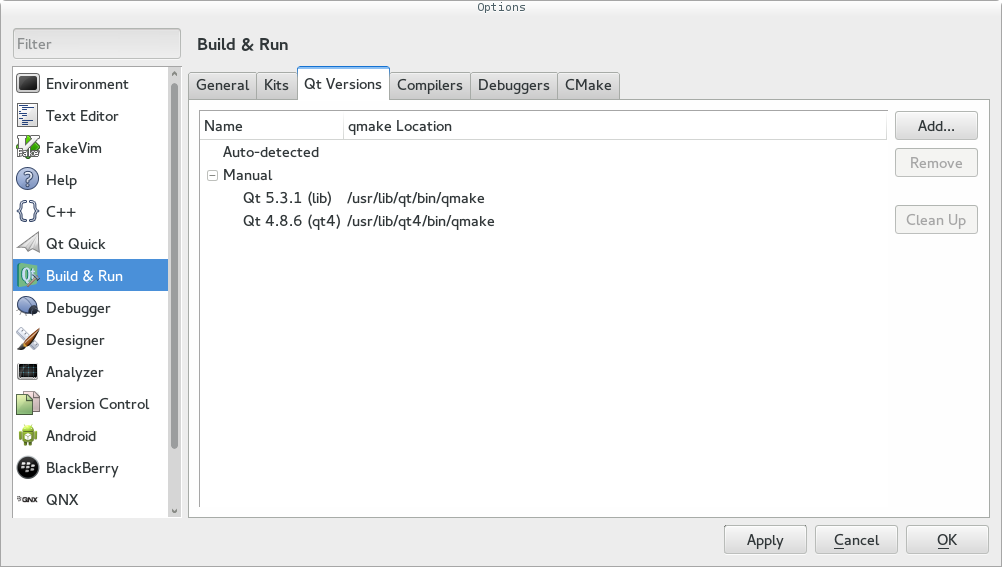
Il faut d'abord vérifier ses kits dans QtCreator : Tools -> Options -> Build & Run

On vérifie dans l'onglet "Compilers" que l'on a gcc et clang

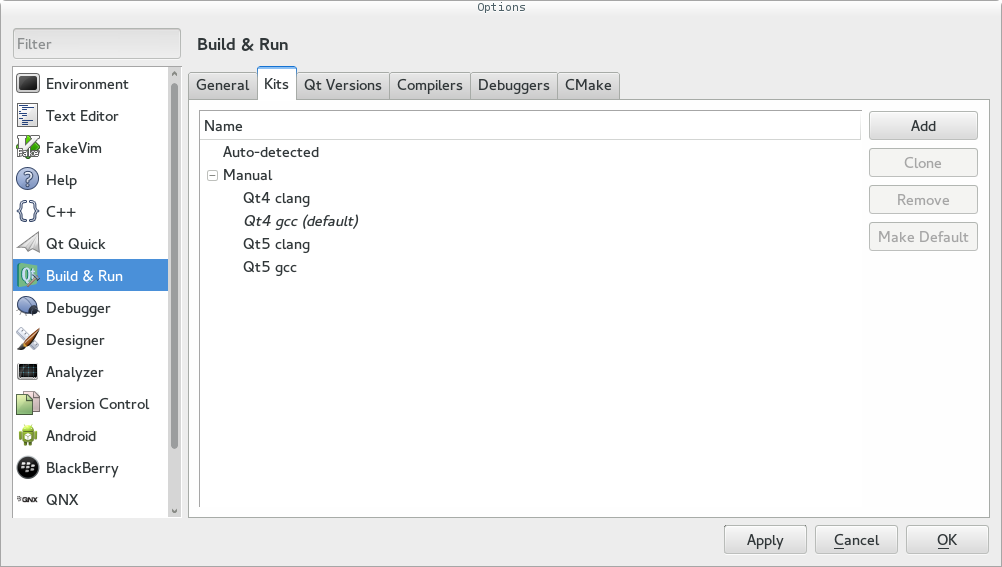
On vérifie dans l'onglet "Qt Versions" que l'on a Qt4 et Qt5. En fait Qt4 suffit, c'est cette version qui fonctionne le mieux et Qt5 n'a pas été retesté depuis longtemps.

On vérifie dans l'onglet "Kits" que l'on a un kit Qt4 gcc et un kit "Qt4 clang". Si non, les créer.

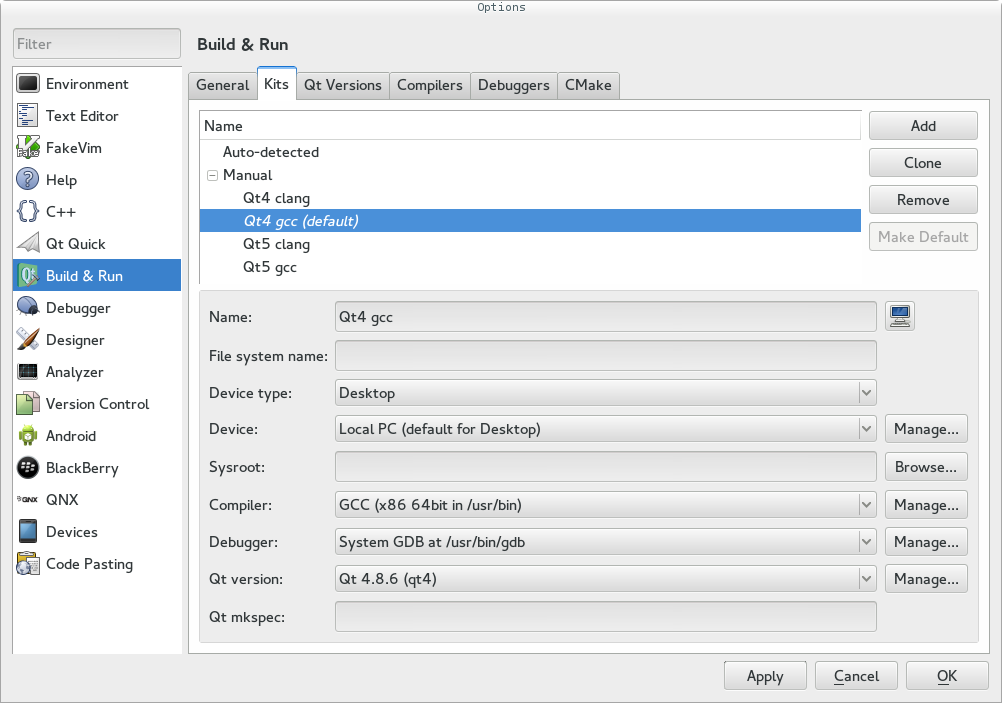
Compilateurs gcc et clang



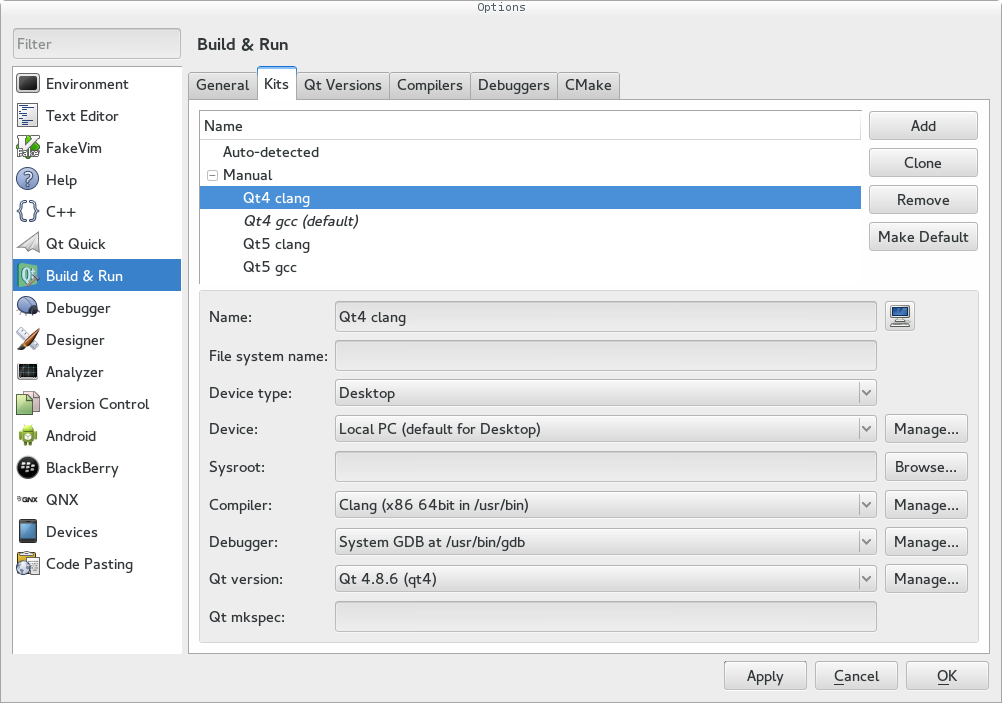
Versions de Qt 4 et 5



Kits Qt4 gcc et clang



Détails du kit Qt4 gcc



Détails du kit Qt4 clang

Ensuite, on ouvre le cmakefile du projet.

Choisir comme dossier de compilation par exemple "build-VCity-Qt4\_gcc\_debug".

Bien choisir le kit "Qt4 gcc" et ajouter comme argument à CMake : -DCMAKE\_BUILD\_TYPE=Debug

la configuration s'appelle "Defaut". On peut la renommer en "Qt4 gcc debug".

Puis on fait : Add -> Clone Selected. On l'appelle "Qt4 gcc release"

On change le dossier de compilation pour : "build-VCity-Qt4\_gcc\_release".

CMake va se relancer, on met alors comme argument : -DCMAKE\_BUILD\_TYPE=Release

Ensuite dans le panneau de gauche, dans "Projects", dans "Build Steps", faire "Details" et ajouter :

-j9 VERBOSE=1

Le nombre après -j peut être ajuster en fonction du nombre de coeur. On met générallement nb\_coeur+1. Donc 9 avec un i7. Permet de compiler en paralèlle.

Nos cibles release / debug pour gcc sont prete.

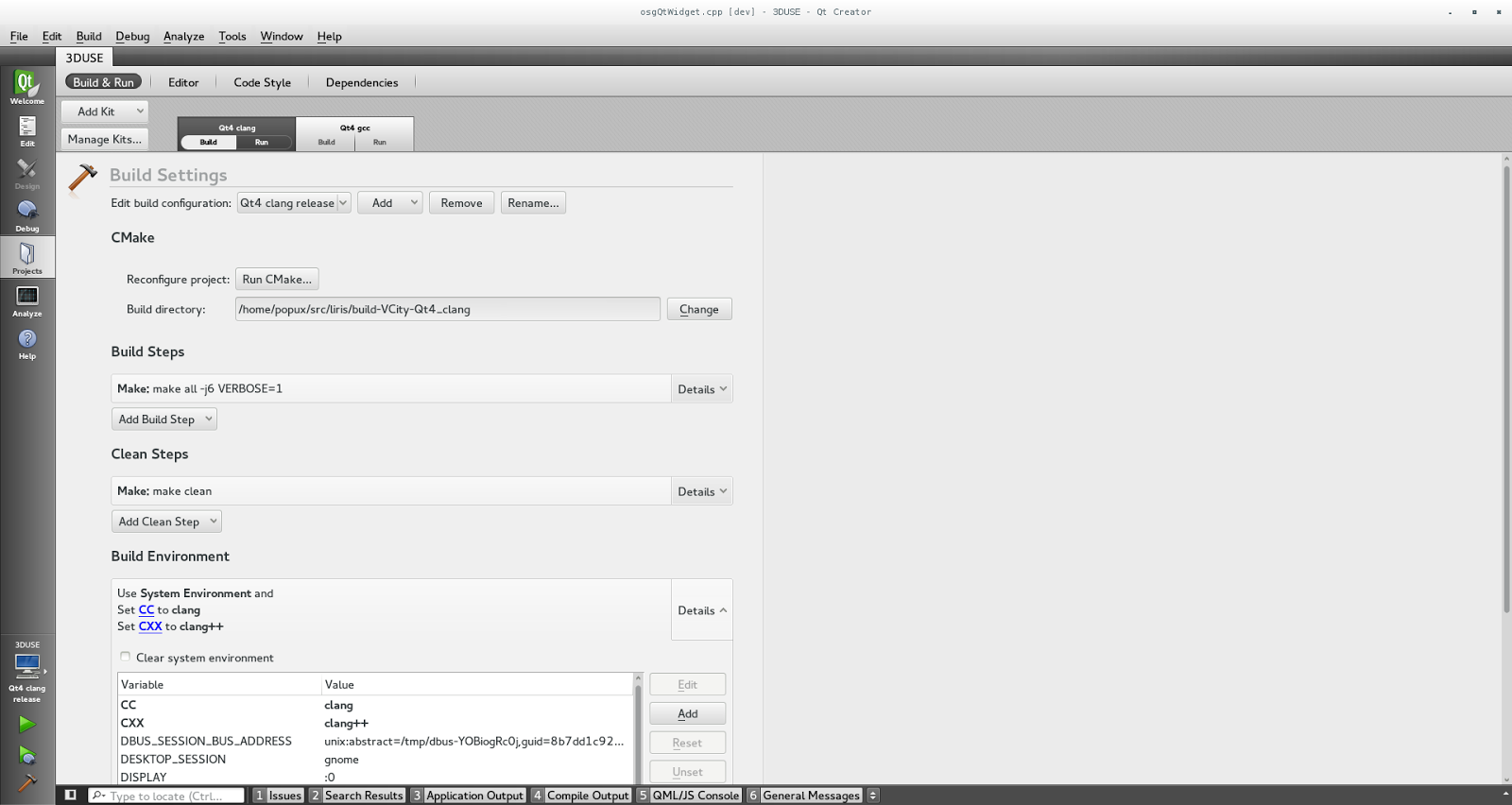
Dans l'onglet "Projects", on peut rajouter le kit clang. Ce compilateur permet de déceller des erreurs de codes ou warnings que gcc ne montre pas.

Il faut faire comme précédemment, en choisissant le kit "Qt4 clang" et en choisissant comme dossier de compilation "build-VCity-Qt4\_clang\_debug" / "build-VCity-Qt4\_clang\_release"

Par contre, pour les cibles clang, il faut ajouter dans "Build Environment" les variables :

CC = clang

CXX = clang++



Ensuite, (à tester) il faut quitter QtCreator, supprimer les dossiers "build-VCity-Qt4\_clang\_debug" / "build-VCity-Qt4\_clang\_release" et lors de l'utilisation de ces cibles, cmake se relancera (bien remettre en argument -DCMAKE\_BUILD\_TYPE=Debug ou Release) et détectera bien clang.

Il faut faire tout cela car le support de CMake dans QtCreator est un peu faible et permet peu de personnalisation.

**Documentation des modules de 3DUSE**

Le projet est développé en C++ (C++11) en utilisant principalement Qt4, OpenSceneGraph, GDAL.

**Architecture générale :**

Le projet est basé autour du pattern MVC. On retrouve alors 3 grands composants : le Modèle, la Vue et le Controlleur.

**Modèle :**

Le modèle contient tout ce qui se trouve dans le dossier core et regroupé dans le namespace vcity.

**Application :** permet de stocker et d’accéder aux principaux éléments du programme :

- la scène

- les paramètres

- le controlleur

- les algorithmes

- la liste des noeuds sélectionnées.

**Settings :** stoque les paramètres de l’application et le profil de données actif.

Contient le paramètre permettant de charger ou non les textures.

**DataProfile :** contient les paramètres permettant de positionner des données dans l’espace :

- bounding box indiquant la zone couverte par les données (utilisé pour dessiner la grille)

- taille des tuiles en x et y

- offset pour décaller les données dans la vue 3D et limiter des problèmes de stabilité numérique liés à de grand nombres flottants (grande coordonnées en lambert93)

- offset pour les id de tuile (pour Paris, la première tuile est : 1286, 13714 )

Quelques méthodes static sont fournis pour créer des profils de données prédéfinis, pour Paris et Lyon.

**Scene :** stocke principalement les layers, mais fournit aussi des mécanismes de recherche de noeuds grâce aux URI.

**abstractLayer :** layer de haut niveau, abstrait. Possède un nom. On peut récupérer une URI pointant sur ce layer avec la méthode getURI.

**LayerCityGML :** layer pouvant stocker des tuiles de données CityGML. Fourni des méthdes pour accéder aux tuiles et aux noeuds CityGML avec des fonctions de recherche par URI.

**Tile :** une tuile est un conteneur pour un CityModel.

**URI :** c’est un mécanisme d’adressage de noeud. Une URI vers un noeud est la concatéation du nom des noeuds à traverser pour accéder à ce noeuds. Ressemble à un chemin vers un fichier.

Ce système d’adressage est utilisé à travers tout le programme et permet de synchroniser la vue et le modèle. Cela se fait principalement dans le ControlleurGui.

Une URI possède un curseur qui est au départ au début de l’URI (racine). On peut faire avancer ce curseur avec popFront. On peut le remettre au début avec resetCursor.

Ce mécanisme est en fait assez lent, car toute les recherche de noeud sont des comparaisons de string. Tout cela pourrait être amélioré en stoquant un hachage à la place des strings, la contrainte étant de pouvoir conserver le lien entre les 3 arbres : treeview, scene et osgscene pour permettre la synchronisation, picking.

**LayerMNT, LayerSHP, LayerAssimp :** ces layers permettent respectivement des stocker des données MNT (mnt au format asc de l’IGN), des shape, des données pouvant être lues par Assimp (ex : obj).

**Algo :** classe regroupant les codes de calcul de Frédéric. Permet de faire du calcul de LOD, de détection de changement. Ce module pourrait être découpé en plusieurs sous-modules ou réorganisé en une boite à outils.

**Vue :**

la vue contient ce qui se trouve dans le dossier gui et contient le code Qt pour l’interface, le code OSG pour l’affichage 3D et le code permettant de convertir les données source (CityGML, shp, …) en données affichable par OSG.

Les fichiers de code d’interface Qt commence par dialog et sont couplé à un fichier d’interface .ui éditable graphiquement avec QtCreator.

**ApplicationGui :** stoque les éléments liés à l’interface graphique :

- la scene OSG

- le controlleur graphique (ControlleurGui)

- le treeview (pour la gestion de l’arbre à gauche de la fenêtre)

**OsgQtWidget :** interface entre OSG et Qt pour y insérer un widget 3D. Contient la gestion des évenements / caméra pour la vue 3D.

**PickHandler :** gestion du picking dans OSG. Les noeuds pickés sont stockés dans la liste d’URI contenue dans la classe Application.

OSG permet de faire du picking avec osgUtil::LineSegmentIntersector. OSG retourne toutes les intersections trouvées sur le rayon. GetFirstIntersection permet de prendre la première, ce qui nous intéresse. Les éléments sélectionnés sont des nœuds osg. Dans notre représentation des données CityGML avec OSG, chaque élément CityGML correspond à un nœud OSG. Cela permet donc de sélectionner un mur, mais pas directement un building complet. Pour cela, il faut remonter dans l'arbre osg pour trouver le nœud correspondant au building (mode picking face / bâtiment).

Possibilité de picking par rectangle de sélection avec osgUtil::PolytopeIntersector.

**OsgScene :** la scène OSG. Elle stoque les noeuds affichable en 3D. Copie de la structure de l'arbre CityGML. Parcours de l'arbre CityGML. A chaque nœud rencontré, création d'un nœud osg et création de la géométrie associée.

**OsgCityGML :** contient des outils pour lire des données CityGML et les convertir en un noeud OSG.

**OsgGDAL, OsgAssimp, OsgMnt :** contient des outils pour lire des données shape, obj, mnt et les convertir en noeud OSG.

**TreeView :** permet la gestion de l’arbre sur la gauche de la fenêtre. Vue hiérarchique des éléments chargés. Représentation visuelle de ce qui est chargé en mémoire.

Outils : clic droit sur un nœud : menu contextuel en fonction du type de nœud.

Ex : Building → éditer → modifier la position

TAG → éditer → modifier la date

**MainWindow :** ce module relie tous les éléments de l’interface : il initialise la fenêtre, la scène OSG, le treeview. Contient les fonctionnalités de chargement de fichiers, de fichiers récents et d’accès à toutes les fonctionnalités du programme.

**Controlleur :**

**ControllerGui et Controlleur :** le controlleur graphique utilise les actions du controlleur classique. ex : reset des noeuds séléctionnés

void ControllerGui::resetSelection()

{

// reset in treeview

appGui().getTreeView()->resetSelection();

// reset in osg scene

appGui().getPickHandler()->resetPicking();

// must be done last

Controller::resetSelection();

}

3 actions :

- reset selection dans le treeview

- reset selection dans le picking OSG, pour que les noeuds ne soient plus surbrillé

- appel du controlleur classique qui vide la liste d’URI dans la classe Application

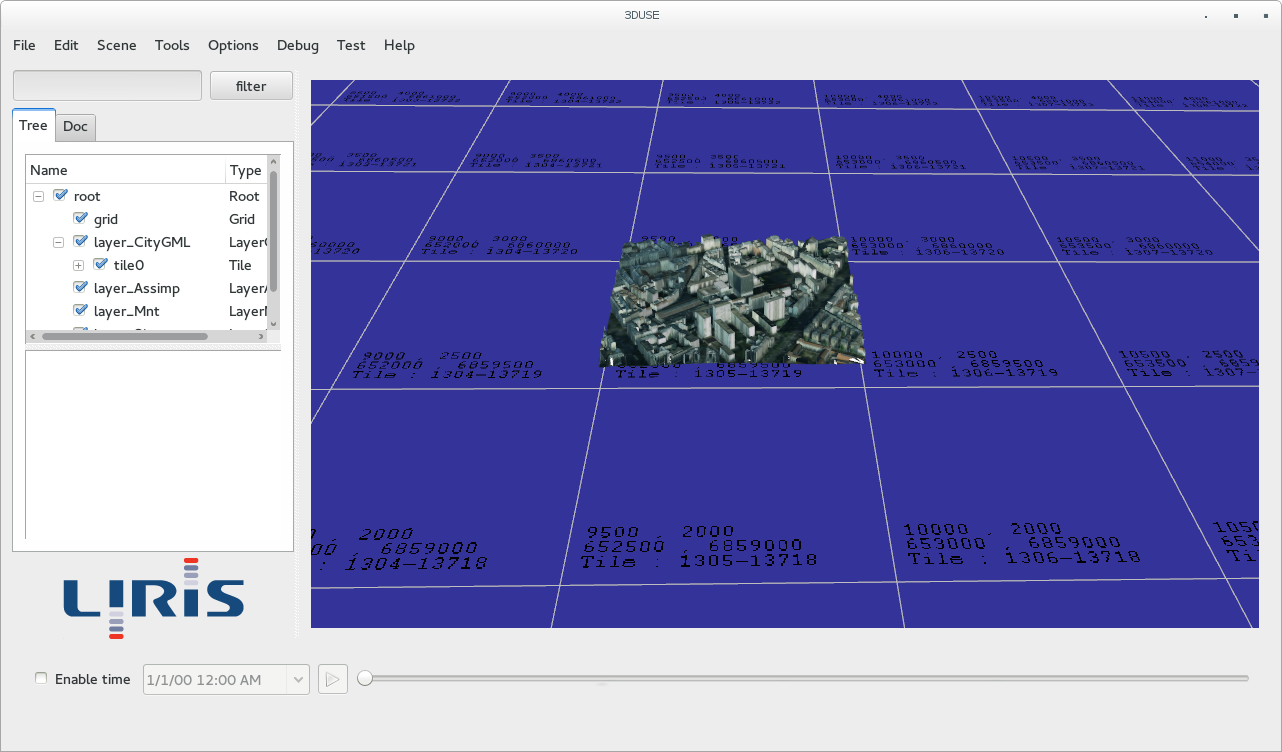
Autre exemple : Action sur les nœuds et répercussions dans les autres arbres :

deleteNode : il faut synchroniser les 3 arbres.

On retrouve un nœud dans un arbre grâce à une URI (unique dans la scène)

On utilise cette URI pour faire la même action sur les 3 arbres. (permet d'éviter de lier les éléments des différents arbres par pointeur, difficile à maintenir)

**Interface**



A gauche, le treeview.

En bas, la barre temporelle.

Au centre, la zone d’affichage 3D.

**Détail des fonctionnalités :**

**Détail chargement fichier**

-Qt

-detection extension

-chargement dans la structure adapté

-dispatch par le controlleur

- stockage dans le modele

- remplissage du treeview

- conversion geom en osg, ajout dans la scene osg

Détail pour le chargement d'une dalle CityGML :

Lorsque l'on clique sur Open dans le menu File, la méthode MainWindow::loadScene() est appelé, qui elle même appèle MainWindow::loadFile(const QString& filepath). C'est dans cette dernière que l'extension du fichier choisi est vérifiée. Si elle correspond au CityGML, une tuile est créé et se chargera de stoquer le CityModel. Pour que cette tuile soit affichable par l'interface, la méthode ControllerGui::addTile(const vcity::URI& uriLayer, vcity::Tile& tile) est utilisée. C'est le controlleur qui va s'occuper de dispatcher les actions pour synchroniser modèle et vue. La première action est d'ajouter la tuile dans la scène. Ensuite les noeuds dans l'arbre Qt (treeview) sont générés (parcours récursif des noeuds CityGML). Et les données 3D pour OpenSceneGraph sont générées. Pour Cela, on appèle OsgScene::buildTile(const vcity::Tile& tile) qui fait un parcours récursif sur tous les noeuds CityObject et génère un noeud OSG avec ReaderOsgCityGML::createCityObject(const citygml::CityObject\* object), qui sera inséré dans OsgScene. A noter que dans la fonction loadFile, il y a un test sur la date système, qui vérifie que l’on a pas dépassé le 01/01/2015. Si c’est vrai, le fichier n’est pas chargé et une boite de dialogue indique que c’est expiré.

**Mode admin**

Certaines fonctionnalitées peuvent être masqué ou affiché grâce à un mot de passe que l’on rentre dans l’onglet admin de la fenêtre des settings accessible depuis le menu Options. Avec le mot de passe “pass2”, tout est affiché et la variable m\_unlockLevel passe à 2. Si le mot, de passe est “pass1” seules les options stables et les outils temporels apparaissent et m\_unlockLevel passe à 1. Si le mot de passe est autre chose, m\_unlockLevel passe à 0 et seules les options stables apparaissent. C’est la fonction MainWindow::unlockFeatures qui gère cela. Dans MainWindow::reset(), on trouve unlockFeatures("pass2"); qui permet de tout déverouillé au lancement du programme. On peut mettre unlockFeatures(""); pour tout vérouiller dés le lancement.

**Paramètres persistants**

Il y a quelques paramètres persistants qui sont gérés par QSettings.

- dataprofile : qui permet de mémoriser le profil de données

- recentfiles : qui permet de stocker le nom des fichiers récemment chargés

- lastdir : qui permet d’ouvrir une boite de dialogue de chargement de fichier dans le même dossier que le fichier précédemment chargé

- loadtextures : qui permet de choisir si on charge ou non les textures

**Extension temporel**

State : configuration de features liée à un id de building. Contient une géométrie. Réprésente un bâtiment dans un certain état.

DynState : la même chose que pour les State, mais peut contenir des DataSources liés à un attribut. Cela permet d’associé un flux de données à un attribut.

Tag : liée à un id de building et fait référence à un State et contient une date (date de début).

L’interval du Tag se fera implicitement avec la date de début du Tag suivant.

**CityGML cut :** (à completer par Martial)

découpage par une AABB, boite aligné sur les axes.

Algorithme de coupe :

parcours de l’arbre pour faire le mapping ring / texture

parcourir chaque linear ring

pour chaque segment du linear ring (2 points consécutifs)

tester les 4 plans de coupe

si les 2 points sont à l’extérieur d'un plan, on rejette les 2

si les 2 points sont à l'intérieur d'un plan, on continue

si 1 point est à l’intérieur et 1 point est à l’extérieur

**Détails des scripts**

**Script CityGML cut :**

Ce script permet de découper des données selon une grille avec l’outil de Martial. Les offsets (1843000, 5172500) et les tailles de tuiles (500, 500) sont entrés en dur. Une double boucle for permet de parcourir les cases.

#!/bin/bash

for I in {0..9}

do

for J in {0..5}

do

FILEOUT="test\_""$I"\_"$J"".gml"

XMIN=$((1843000+$((500\*$I))))

YMIN=$((5172500+$((500\*$J))))

XMAX=$(($XMIN+500))

YMAX=$(($YMIN+500))

echo $FILEOUT "->" $XMIN $XMAX $YMIN $YMAX

./parseCityGML /mnt/docs/data/dd\_backup/GIS\_Data/Lyon3/LYON\_3.gml /mnt/docs/data/dd\_backup/GIS\_Data/Lyon3/output"$I"\_"$J"/$FILEOUT $XMIN $YMIN $XMAX $YMAX

done

done

**Script découpage shape Paris selon la grille de Paris :**

Ici, la commande ogr2ogr (outil GDAL) permet de découper les données shape selon une BBOX qui évolue grace à une double boucle.

#!/bin/bash

cd /mnt/docs/data/dd\_backup/GIS\_Data\_tmp/E\_BATI

mkdir -pv tiles

LOD=1

for I in {0..29}

do

for J in {0..20}

do

TILEX=$((1286+$I))

TILEY=$((13714+$J))

FILEOUT="tile\_""$TILEX"\_"$TILEY"".shp"

XMIN=$((643000+$(($((500\*$LOD))\*$I))))

YMIN=$((6857000+$(($((500\*$LOD))\*$J))))

XMAX=$(($XMIN+$((500\*$LOD))))

YMAX=$(($YMIN+$((500\*$LOD))))

echo lod $LOD : $I $J : $FILEOUT "->" $XMIN $XMAX $YMIN $YMAX

echo ogr2ogr -skipfailures -clipdst $XMIN $YMIN $XMAX $YMAX tiles/$FILEOUT BATI\_INDIFFERENCIE.SHP

ogr2ogr -skipfailures -clipdst $XMIN $YMIN $XMAX $YMAX tiles/$FILEOUT BATI\_INDIFFERENCIE.SHP

done

done

**Recherche de fuites mémoire avec valgrind**

Valgrind est en fait une suite d'outil mais par défaut, memcheck est utilisé.

Il faut que le programme soit compilé en debug pour avoir beaucoup d'info comme les noms de fonctions et la ligne exacte d'ou vient une fuite.

D'abord, on lance le programme avec valgrind (exécution ralenti, peut être 100x plus lent...) (pipé dans less pour pouvoir parcourir le résultat) :

valgrind --leak-check=full ./3DUSE 2>&1 | less

AU fur et à mesure de l'éxécution, des erreurs de lecture ou écriture mémoire invalides (dépassement de tableau) ou de d'utilisation de variables non initialisées sont indiquées. Il peut y en avoir des milliers (dans les lib Qt, osg et même dans la libGL nvidia...). Il faut alors faire une action dans le programme qui utilise le code que l'on veut vérifier. le rapport de fuite mémoire se fait lorsque l'on quitte le programme.

Les fuites sont indiquées dans le rapport après HEAP SUMMARY.

les fuites sont classées par ordre croissant de mémoire perdue.

D'abord, il faut chercher les fuites dont on est responsable. On cherche alors le pattern "main.cpp". On trouve par exemple cela :

==16498== 202 (96 direct, 106 indirect) bytes in 1 blocks are definitely lost in loss record 6,523 of 7,124

==16498== at 0x4C28C90: operator new(unsigned long) (in /usr/lib/valgrind/vgpreload\_memcheck-amd64-linux.so)

==16498== by 0xB081E61: ??? (in /usr/lib/libQtCore.so.4.8.6)

==16498== by 0xB082506: QLibrary::setFileNameAndVersion(QString const&, int) (in /usr/lib/libQtCore.so.4.8.6)

==16498== by 0xB0825AC: QLibrary::QLibrary(QString const&, int, QObject\*) (in /usr/lib/libQtCore.so.4.8.6)

==16498== by 0xA26296A: ??? (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA26507F: ??? (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA1F6AA7: QApplicationPrivate::construct(\_XDisplay\*, unsigned long, unsigned long) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA1F6D1E: QApplication::QApplication(int&, char\*\*, int) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0x56AE11: main (main.cpp:15)

==16498== est le pid de notre programme

Ce que l'on voit est la pile d'appel. On voit bien que la dernière ligne est dans main.cpp et que celui-ci, en ligne 15, appel le constructeur de QApplication. C'est un exemple de fuite que l'on ne maitrise pas, tout se passe dans Qt ici. D'ailleurs c'est peut-être un faux positif.

En fait, il faut que dans les premières lignes du paragraphe, on soit dans notre code (souvent la première ligne est dans une librairie tierce car elle fait un new ou un malloc au final, mais provoqué par nous).

Un autre exemple :

==16498== 1,298,432 bytes in 317 blocks are definitely lost in loss record 7,124 of 7,124

==16498== at 0x4C293B0: operator new[](unsigned long) (in /usr/lib/valgrind/vgpreload\_memcheck-amd64-linux.so)

==16498== by 0x58A004: osgQtWidget::paintEvent(QPaintEvent\*) (osgQtWidget.cpp:282)

==16498== by 0xA24222F: QWidget::event(QEvent\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA1F2F0B: QApplicationPrivate::notify\_helper(QObject\*, QEvent\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA1F948C: QApplication::notify(QObject\*, QEvent\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xB09658C: QCoreApplication::notifyInternal(QObject\*, QEvent\*) (in /usr/lib/libQtCore.so.4.8.6)

==16498== by 0xA23C950: QWidgetPrivate::drawWidget(QPaintDevice\*, QRegion const&, QPoint const&, int, QPainter\*, QWidgetBackingStore\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA23D37F: QWidgetPrivate::paintSiblingsRecursive(QPaintDevice\*, QList<QObject\*> const&, int, QRegion const&, QPoint const&, int, QPainter\*, QWidgetBackingStore\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA23D1EE: QWidgetPrivate::paintSiblingsRecursive(QPaintDevice\*, QList<QObject\*> const&, int, QRegion const&, QPoint const&, int, QPainter\*, QWidgetBackingStore\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA23C50B: QWidgetPrivate::drawWidget(QPaintDevice\*, QRegion const&, QPoint const&, int, QPainter\*, QWidgetBackingStore\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA23D37F: QWidgetPrivate::paintSiblingsRecursive(QPaintDevice\*, QList<QObject\*> const&, int, QRegion const&, QPoint const&, int, QPainter\*, QWidgetBackingStore\*) (in /usr/lib/libQtGui.so.4.8.6)

==16498== by 0xA23C50B: QWidgetPrivate::drawWidget(QPaintDevice\*, QRegion const&, QPoint const&, int, QPainter\*, QWidgetBackingStore\*) (in /usr/lib/libQtGui.so.4.8.6)

Ici, la deuxième ligne est dans notre code : osgQtWidget.cpp fonction paintEvent.

(il est interessant de voir que main.cpp n'apparait pas, car c'est de la programmation évenementielle, on doit être dans un thread, dans un callback… ou alors la pile d’appel est tronquée car trop longue)

Allons voir ligne 282 :

279 void osgQtWidget::paintEvent(QPaintEvent\* /\*event\*/)

280 {

281 frame();

282 char\* p = new char[4096];

283 }

Le pointeur p est alloué mais il n'y a pas de delete. Ici, c'est un exemple simple, mais le principe est générallement de remonter la pile, trouver ou est faite l'allocation mémoire depuis notre code puis comprendre ou il faut faire la libération, pour que celle ci se passe bien dans toutes les branches de code qui suivent l'allocation.

osgQtWidget::paintEvent est une méthode d'osg qui permet de rafraichir l'écran. Elle est donc appelé plusieurs fois par seconde. C'est ce genre de fuite qu'il faut éradiqué en premier : les portions de code avec des fuites qui sont appelées plusieurs fois. A chaque passage dans ce code les fuite s'accumule. Alors qu'une zone de code qui n'est éxécutée qu'une fois n'accumule pas de mémoire perdu (à moins qu'il y ait une allocation de 1Go de ram !).

Autre utilisation :

Dans certains cas, valgrind peut permettre de débugger des crashs dont on ne peut pas rechercher la source avec un débugger car celui semble donner des informations incompréhensibles. Cela peut arriver lors de dépassements de tableaux qui peuvent corrompre la pile ou le tas. Valgrind peut alors être utilisé pour trouver l'origine de ce dépassement avec les erreurs de type "invalid read / invalid write".

Guide de valgrind en ligne :

<http://valgrind.org/docs/manual/quick-start.html>

**Quelques notes à propos du C++11 :**

Quelques notions de C++11 sont utilisées. Principalement les boucle for de ce type :

for(CityObject\* obj : \_roots)

{

obj->...;

}

à la place de :

for(std::vector<CityObject\*>::iterator it = \_roots.begin(); it < \_roots.end(); ++it)

{

(\*it)->...;

}

Pointeur nul :

Le plus souvent possible, le pointeur nul "nullptr" est utilisé à la place de NULL ou encore 0.

Pour éviter des erreurs de surcharge lors d’héritage, le mot clé override est utilisé. Cela permet de s’assurer que l’on remplace bien une méthode plutôt que d’en créer une nouvelle.

Ex : On a une classe A avec une méthode func. Une classe B héritant de A et qui remplace func.

Class A

{

public:

virtual int func(int i);

};

Class B : public A

{

public:

virtual int func(float i);

};

B b;

A\* p = &b;

p->func(5); // ici c’est A::func qui est appelé et non pas B::func comme on le pense

Avec :

Class B : public A

{

public:

virtual int func(float i) override; // erreur, pas le même prototype de fonction que dans A

};

On aura une erreur de compilation. On pourra alors corriger :

Class B : public A

{

public:

virtual int func(int i) override;

};

L’utilisation de ce mot clé permet de déceler des erreurs de polymorphisme difficiles à débugger.

**Guide mise à jour Archlinux**

Faire les mise à jours environ tous les mois, pour éviter d'être coincé par des changements majeurs d'architecture de la distribution. Vérifier pour cela le site archlinux.org. Quand Il y a des interventions manuelles à faire, tout est générallement très bien expliqué.

2 types de paquets dans archlinux, les paquets officiels et les paquets AUR (Arch User Repository). Ces derniers sont des paquets non officiels mais il y en a énormément.

Mise à jour des paquets officiels :

pacman -Syu

Mise à jour des paquets officiels + non officiels :

yaourt -Syua

Dans les paquets non officiels utilisés, il y a notamment gdal-ecw (gdal + support format ecw) et qgis.

Il y a parfois des interventions manuelles à faire pour mettre à jour les fichiers de configuration du système (générallement dans /etc). On trouve facilement ces fichiers car ils ont comme extension .pacnew et peut les trouver avec :

yaourt -C

Un assistant en mode texte permet de voir ces fichiers et des les éditer avec vimdiff. Mais on peut faire l'édition avec n'importe quel éditeur (ex : meld).

ex: /etc/toto.cfg a une nouvelle version /etc/toto.cfg.pacnew

meld /etc/toto.cfg /etc/toto.cfg.pacnew -> on repercute les modifs dans /etc/toto.cfg et on peut supprimer /etc/toto.cfg.pacnew

**Projet WebGL INSA**

Machine Rémi Baume :

Comptes sur la machine linux Gamagora :

baume / Geomod34

git / VCity

root / Geomod34

**Aperçu de l’algorithme :**

(1) init / set params / tiles / offset / camera

(2) animate

(3) update MainTile

(4) check if out of current tile

(5) if true

(6) delete invalid tiles

(7) call priority manager

(8) add loading tasks for visible tiles to scheduler

(9) render

(10) update control / camera

(11) update scheduler

(12) exec task

(13) load tile / texture

**Quelques précisions sur les étapes :**

(1) Initialisation :

Tout d’abord, chargement de la page web qui charge le premier fichier javascript et qui éxécute le point d’entrée javascript, la fonction VCC.ViewPort.

Cette fonction initialise :

- threejs

- les paramètres relatifs aux données :

- offset de camera, pour avoir des coordonnnées de travail pour le GPU plus proche de 0 et ainsi avoir une meilleure précision

- offset de numéro de tuile. Dans le cas de Paris, les tuiles à notre disposition commence à 13714, 1285

- taille de tuiles, ici 500m x 500m

Ensuite la fonction animate est lancée, qui contient la boucle principale du programme.

(2) Boucle principale

Cette boucle vérifie s’il faut charger une nouvelle tuile en fonction de la position de la caméra.

Ensuite la fonction de rendu est lancée, affiche la scène threejs.

La caméra est mise à jour (mise à jour de la position) et le scheduleur est lancé ce qui permet de dépiler une tache parmis les taches précedemment empilées.

(3) Vérification de la position

Si la position de la caméra sort de la tuile courante, il faut décharger et charger d’autres tuiles.

Ici, il serait possible d’ajouter la gestion des lods en fonction de l’altitude de la caméra.

(6) Suppression des tuiles hors champ dans la fonction deleteObjetOutOfScene

(7) Appel du gestionnaire de tuiles : tilePriorityManager

Celui ci va créer des taches de chargements pour les tuiles adjacentes.

(8) Création des taches de chargements

D’abord ajout des taches haute priorité pour la tuile ou se trouve la camera si elle n’est pas chargée : terrain, batiments, textures

Ensuite, ajout de taches basse priorité pour les tuiles adjacentes.

(9) Appel du rendu de la scène threejs

(10) Gestion des controles par le module FirstPersonControls. Cela met à jour la position de la caméra.

(11) Appel du scheduleur qui va éxécuter les taches empilées précédemment. Le scheduleur exécute une seule tache par frame.

(12) Exécution des taches haute priorité, puis basse priorité

(13) En fonction du type de la tache (geometrie, texture), les ressources sont chargées et directement stocké dans le tileManager au bon endroit (on a la position de la tuile à charger dans la tache)

**Comment utiliser le projet :**

Install node and npm (included in node)

Install required packages :

npm install express

npm install throttle

To run the server, enter on linux : node index.js

on windows : node.exe index.js (or drag index.js over node.exe)

**Resource tree :**

root / ressources / paris / LOD2 / dem / json data, ex : ZoneAExporter\_1286\_13723\_terrain.json

| |

- - build / json data, ex : ZoneAExporter\_1286\_13723.json

|

- textures /

|

- 128 / dds textures data

|

- 256 / dds textures data

|

- 512 / dds textures data

|

- 1024 / dds textures data

**Documentation d'installation de Virtual City pour le développement (obsolete grâce au kit binaire de Martial)**

Windows :

Installation des lib Qt5 :

Utiliser Qt online installer for Windows :

<http://download.qt-project.org/official_releases/online_installers/1.4/qt-windows-opensource-1.4.0-2-x86-online.exe>

Cocher :

Qt -> Qt 5.1.1 -> MinG 4.8 (32 bit)

Qt Creator sera aussi automatiquement installé.

Ajouter à la variable PATH : C:\Qt\5.1.1\mingw48\_32\bin

Installation de MinGW et MSYS :

Utiliser mingw-get-setup, ici :

<http://sourceforge.net/projects/mingw/files/latest/download?source=files>

Avec le gui, installer :

Basic setup -> msys base

Basic setup -> mingw32 base

Basic setup -> mingw-developer-toolkit

lancer msys : C:\mingw\msys\1.0\msys.bat

mingw-get install g++

mingw-get install zlib

Pour la suite, on utilise le compilateur gcc fourni par MinGW.

Sous Windows, la commande "make -j" est parfois un peu capricieuse et s'arrète avec des erreurs. Il suffit de relancer la commande dans ces cas là, jusqu'à ce que tout soit compilé.

Note : ne pas mettre de valeur après make -j. Il faut laisser le système gérer cela sinon sous Windows, il y a des interbloquages...

Installation de libxml2 :

Utiliser TortoiseGit pour récupérer le code : git clone git://git.gnome.org/libxml2

Ou utiliser un snapshot : ftp://xmlsoft.org/libxml2/libxml2-git-snapshot.tar.gz

A placer dans c:/mingw-libs/

dans msys :

cd /c/mingw-libs/libxml2

./autogen.sh

make -j

make install-strip

La lib sera installée dans C:/mingw/msys/1.0/local

Installation de libjeg :

bien prendre le tar.gz, échec avec le zip

<http://www.ijg.org/files/jpegsrc.v9.tar.gz>

Extraire dans c:/mingw-libs/

cd /c/mingw-libs/jpeg-9

./configure

make -j

make install-strip

Installation de libtiff

ftp://ftp.remotesensing.org/pub/libtiff/tiff-4.0.3.zip

Extraire dans c:/mingw-libs/

cd /c/mingw-libs/tiff-4.0.3

ici, ajout de flag pour ajouter le support de jpeg dans la lib tiff

./configure CFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib

make -j

make install-strip

Installation de libpng

ftp://ftp.simplesystems.org/pub/libpng/png/src/libpng16/lpng163.zip

Extraire dans c:/mingw-libs/

cd /c/mingw-libs/lpng163

cp scripts/makefile.msys .

make -f makefile.msys -j

Pas de make install, à la main :

cp \*.h /usr/local/inculde

cp \*.a /usr/local/lib

cp \*.dll /usr/local/bin

cp \*.exe /usr/local/bin

Installation de libcurl

<http://curl.haxx.se/download/curl-7.32.0.zip>

Extraire dans c:/mingw-libs/

cd /c/mingw-libs/curl-7.32.0

./configure

make -j

make install-strip

Installation de libproj

<http://download.osgeo.org/proj/proj-4.8.0.zip>

Extraire dans c:/mingw-libs/

cd /c/mingw-libs/proj-4.8.0

./configure

make -j

touch proj.pc

make install-strip

Installation de gdal / il reste des erreus

<http://download.osgeo.org/gdal/1.10.1/gdal1101.zip>

Extraire dans c:/mingw-libs/

cd /c/mingw-libs/gdal-1.10.1

./configure --with-static-proj4=/usr/local CFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib

make -j

make install-strip

erreur : undefined getaddrinfo / freeaddrinfo

Installation de OpenSceneGraph

<http://www.openscenegraph.org/downloads/developer_releases/OpenSceneGraph-3.2.0.zip>

Utilisation de cmake-gui

Where is the source code : C:/data/src/libs/OpenSceneGraph-3.2.0

Where to build the binaries : C:/data/src/libs/OpenSceneGraph-3.2.0/build\_mingw4.8

Ensuite cliquer sur Configure

ACTUAL\_3RDPARTY\_DIR : C:/MinGW/msys/1.0/local

CMAKE\_BUILD\_TYPE : RELEASE

CMAKE\_INSTALL\_PREFIX : C:\data\progs\libs\OpenSceneGraph-3.2\_mingw4.8

Ensuite cliquer sur Configure

ZLIB\_INCLUDE\_DIR : C:/MinGW/include

ZLIB\_LIBRARY : C:/MinGW/lib/libz.a

JPEG\_INCLUDE\_DIR : C:/MinGW/msys/1.0/local/include

JPEG\_LIBRARY : C:/MinGW/msys/1.0/local/lib/libjpeg.a

CURL\_INCLUDE\_DIR : C:/MinGW/msys/1.0/local/include

CURL\_LIBRARY : C:/MinGW/msys/1.0/local/lib/libcurl.dll.a

LIBXML2\_INCLUDE\_DIR : C:/MinGW/msys/1.0/local/include

LIBXML2\_LIBRARY : C:/MinGW/msys/1.0/local/lib/libxml2.a

LIBXML2\_LIBRARIES : C:/MinGW/msys/1.0/local/lib/libxml2.a

LIBXML2\_XMLLINT\_EXECUTABLE : C:/MinGW/msys/1.0/local/bin/xmllint.exe

PNG\_INCLUDE\_DIR : C:/MinGW/msys/1.0/local/include

PNG\_LIBRARY : C:/MinGW/msys/1.0/local/lib/libpng.a

PNG\_PNG\_INCLUDE\_DIR : C:/MinGW/msys/1.0/local/include

Ensuite cliquer sur Configure

Certaines de ces variables seront peut-être correctement détectées. Les librairies détectées seront les fichiers \*.dll.a qui sont les librairies dynamiques et nécessite les dll par la suite. On peut alors utiliser les librairies statiques : les fichier \*.a

ex : C:/MinGW/msys/1.0/local/lib/libxml2.dll.a -> C:/MinGW/msys/1.0/local/lib/libxml2.a

note : ne pas le faire pour libcurl, libcurl.a n'est pas généré

Ensuite cliquer sur Generate

Le Makefile est dans C:/data/src/libs/OpenSceneGraph-3.2.0/build\_mingw4.8

dans msys :

cd /c/data/src/libs/OpenSceneGraph-3.2.0/build\_mingw4.8

make -j

Si les erreurs persistent, faire : make

make install

cp -r /c/data/src/libs/OpenSceneGraph-3.2.0/build\_mingw4.8/include/osgQt /c/data/progs/libs/OpenSceneGraph-3.2\_mingw4.8/include

Erreur de link mingw\_osgdb\_tiff.dll

Editer le fichier :

C:\data\src\libs\OpenSceneGraph-3.2.0\build\_mingw4.8\src\osgPlugins\tiff\CMakeFiles\osgdb\_tiff.dir\build.make

Et ajouter à la ligne qui correspond on link (la plus longue, et qui contient des .a ) :

/C/MinGW/msys/1.0/local/lib/libjpeg.a

Configuration des variables d'environnements :

Ajouter à la variable PATH :

C:\MinGW\bin;C:\data\progs\libs\OpenSceneGraph-3.2\_mingw4.8

Récuperer le code de Virtual City sur SVN :

svn co<https://svn.liris.cnrs.fr/vcity>

Le projet Qt est virtualcity/trunk/VirtualCity.pro

Qt Creator doit détécter automatiquement le compilateur et les lib Qt5 et donc créer un kit :

dans Tools -> Options

Onglet Compilers : on a Auto detected : MinGW 4.8 32bit

Onglet Qt Versions : on a Auto detected : Qt 5.1.1 MinGW 32bit

Onglet Kits : on a Auto detected : Desktop Qt 5.1.1 MinGW 32bit -> on peut le mettre par default

On choisi ce kit, et on compile

En cas de soucis d'éxécution du binaire compilé, penser à :

- utiliser depends (dependency walker) (utiliser la bonne architecture)

- attention à l'ordre des chemins indiqués dans la variable PATH, le première dll trouvée sera utilisée

- au pire, mettre les dll dans le même dossier que le binaire, elles seront toujours prioritaires

Librairies précompilées :

<http://liris.cnrs.fr/~vcity/docs/MinGW.7z>

Extraire dans c:/

Ajouter à la variable PATH : c:\mingw\bin

<http://liris.cnrs.fr/~vcity/docs/OpenSceneGraph-3.2_mingw4.8.7z>

A suivre : gdal

**Installation des lib pour linux:**

Ubuntu : gdal-bin libgdal-dev openscenegraph libopenscenegraph-dev proj libproj-dev

Archlinux : qtcreator gdal-ecw openscenegraph proj

Draft ancienne version

Installer mingw-get

dans c:\mingw

lancer C:\mingw\msys\1.0\msys.bat

installer g++ gcc zlib xml2

mingw-get install g++

mingw-get install gcc

mingw-get install zlib

mingw-get install xml2

mingw-get install msys-wget

<http://www.gaia-gis.it/spatialite-2.4.0/mingw_how_to.html>

installer libjeg

bien prendre le tar.gz, échec avec le zip

<http://www.ijg.org/files/jpegsrc.v9.tar.gz>

./configure

make -j

make install-strip

tiff

ftp://ftp.remotesensing.org/pub/libtiff/tiff-4.0.3.zip

ici, ajout de flag pour ajouter le support de jpeg dans la lib tiff

./configure CFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib

make -j

make install-strip

png

ftp://ftp.simplesystems.org/pub/libpng/png/src/libpng16/lpng163.zip

cp scripts/makefile.msys .

make -f makefile.msys -j

pas de make install, à la main :

cp \*.h /usr/local/inculde

cp \*.a /usr/local/lib

cp \*.dll /usr/local/bin

cp \*.exe /usr/local/bin

curl

<http://curl.haxx.se/download/curl-7.32.0.zip>

./configure

make -j

make install-strip

osg

utiliser cmake.bat fourni

ajouter /C/mingw/msys/1.0/local/lib/libjpeg.a dans C:\data\src\libs\OpenSceneGraph-3.2.0\build\_mingw\src\osgPlugins\tiff\CMakeFiles\osgdb\_tiff.dir\build.make

puis :

make -j

make install

proj4

http://download.osgeo.org/proj/proj-4.8.0.zip

./configure

make -j

touch proj.pc

make install-strip

ecw

gdal\_translate -of GTiff -co "TILED=YES" tadalle.ecw tadalle.tif

geotiff

ftp://ftp.remotesensing.org/pub/geotiff/libgeotiff/libgeotiff-1.4.0.zip

./configure CFLAGS=-I/usr/local/include LDFLAGS=-L/usr/local/lib

make -j

make install-strip

gdal

./configure --with-static-proj4=/usr/local --with-xml2=/usr/bin/xml2-config --with-ecw=/usr/local CFLAGS="-I/usr/local/include -I/usr/include/libxml2" LDFLAGS=-L/usr/local/lib

make -j

make install-strip

cgal

qt5

qtcreator

<https://home.mmorel.eu:40443/zerobin/?34d8aae1bf9d5020#f5ZAggvJ5Z1WFFaYWTMkWv/XsAm9yH8rLer8U5gzE04=>

msvc 2010

Si msvc 2012 est installé, il faut installer le sp1 de msvc 2010 (coff link error)

jpeg, tiff, png, libxml2 : utilisation des binaires gnuwin32

<http://gnuwin32.sourceforge.net/packages.html>

ecw sdk v5 msvc2010

osg : cmake

ajouter les libs jpeg, png, tiff.

**Tableau récapitulatif de test de librairies sous windows :**

|  |  |  |  |
| --- | --- | --- | --- |
|  | mingw4.4 | mingw4.8 | vs |
| qt4 | ok | Compile mais crash à l’exécution | Compile mais crash à l’exécution |
| qt5 |  | Lib et binaire fourni  ok, fonctionne avec osg | Binaire vs2010  nécessite vs2010 sp1  avec cette version, on peut utiliser toutes les autres libs avec vs |
| osg |  | Support avec cmake  Ok avec jpeg, tiff, png... | vs2012 |
| ecw |  | Source disponible, mais ne compile pas (#idef WIN32 avec du code spécifique vs) | vs2012 |
| gdal |  | Support avec cmake    compilation en cours  nécessite des modifications du makefile |  |
| cgal |  | Support avec cmake |  |

compilation windows de osg :

avec mingw : cmake ok mais echec de la compil

avec cygwin : cmake ok mais echec de la compil -> pb link graphicswidgetx11.o

avec visual C++ 11 : ok

-> utilisation dans qt creator :

kit 4.8.4 msvc2010 -> necessite le compilo msvc2010

crash dés l'utilisation de std::string

kit 5.1 opengl 64bit msvc 2012 (pas d'autre choix) -> compilo msvc2012 64bit

echec, il faut linker avec osg 64 bit (pas dispo)

sous linux -> parfait

mais tests dans vm, pas de support opengl suffisant