fr_FR.png ...version française

Documentation for plugin developpers

This project brings together the documentations and tools for the **Computree plugin developers**.

Initialization of a Computree development environment

Initialization of a Computree development environment for Windows (64 bits, 7, 8 ou 10)

Installing the development environment

1. Install “Microsoft Visual studio 2015 Express”: https://www.visualstudio.com/fr/post-download-vs/?sku=xdesk
2. Install “Windows SDK”: https://developer.microsoft.com/fr-fr/windows/downloads/windows-10-sdk (Optional step, allowing to compile in debug mode)
3. Install “Qt”: https://www.qt.io/download-open-source/ (check at least Qt 5.9.1 msvc 2015 64 bits
4. Install a SVN client, as for example “TortoiseSVN”: https://tortoisesvn.net (During the installation, think about activating the installation of *command line client tools*)

Obtaining the source code of Computree and its plugins

1. Create a *Computree* root directory where you want

2. Dowload Computree development kit (Windows), which contains:

   - A *all.pro* file with all projects distributed as standard (core + open-source plugins)
   - A *recuperer_depots.bat* file, containing svn statements, to retrieve the source code
   - A file *LISEZ-MOI.txt*, containing these instructions

3. Unzip *kit_dev_windows_fr.zip* at the root of the *Computree* directory

4. Run the *recuperer_depots.bat* script

Installing Dependencies

Download and unzip Computree v5 dependencies folder, and paste at the root of the computree directory.

If you want to install dependencies in different locations (for example in c:/program files), you can, for each *LIBNAME_default_path.pri* file in computreev5 directory, duplicate it and rename it *LIBNAME_user_path.pri*. After that you just have to modidy this second file to use your local paths.

Computree Compilation

1. Launch Qt Creator

2. Open the project *all.pro*, select the compiler MSVC 2015 64bits, with version release and/or debug

If you have not installed PCL, delete/comment the following line in all.pro:

```
Computreev5/library/ctlibpcl/ctlibpcl.pro \
```

3. On the Project tab, disable shadow builds (check box), for release and/or debug

4. Run qmake on all.pro, then compile the project

After updating the source code, if the core of Computree has been modified significantly, it may be necessary to run qmake on each subproject and then to do Recompile on all.pro.

Execution of Computree

Once compiled, **to be run, Computree needs all the dependency dlls**, accessible from the location of the generated *CompuTreeGui.exe* file.
For that copy dlls to ComputreeInstallRelease folder (for release version) and / or ComputreeInstallDebug folder (for debug version).

Dll are available for download here: Computree v5 DLL

Then you can run from Qt-Creator (green arrow or run on all.pro).

Configure your plugin if you want to use PCL in your code

If you want to use PCL for your development some preparation steps are required:

You must configure the .pro file of your plugin (.pro) as follows (beginning of the file):

```
CT_PREFIX = ../../computreev5

Include ($$ {CT_PREFIX} /shared.pri)
Include ($$ {PLUGIN_SHARED_DIR} /include.pri)

COMPUTREE + = ctlibpcl

Include ($$ {CT_PREFIX} /include_ct_library.pri)
```

Do not forget to compile the libpcl project into the computreev5/library/ctlibpcl folder (open the ctlibpcl.pro file and compile it with QtCreator)

Just make a *qmake* on the project of your plugin (right click → qmake) and compile it.

Initialization of a Computree Development Environment on Ubuntu 16.04 LTS

Installing the development environment

1. Installing subversion

In a terminal (CTRL + ALT + T):

```
sudo apt-get update
sudo apt-get install subversion
```

2. Installing Qt (5.9.1)

- Download last Qt installer : https://www.qt.io/download-open-source/
  - You will have to create a Qt user account
  - If you are behind a proxy, you need to go in settings section to parameter it
- Install Qt

Recovering the source code of Computree and its plugins

1. Create a *Computree* root directory where you want

2. Download Computree development kit (Linux), which contains:

   - A *all.pro* file with all projects distributed as standard (core + open-source plugins)
   - A *recuperer_depots.sh* file containing svn statements to retrieve the source code
   - A file *README.txt*, containing these instructions

3. Unzip *kit_dev_linux.tar.gz* at the root of the *Computree* directory

4. *In a terminal (CTRL + ALT + T), run the _recuperer_depots.sh* script

Installing Dependencies

1. **OpenCV 3.3.0** (optional but highly recommended, allows to use images / rasters in Computree)

   - Follow instruction given in official OpenCV site: http://docs.opencv.org/3.3.0/d7/d9f/tutorial_linux_install.html

2. **PCL 1.8.0** (optional, allows to use plugins requiring PCL)

   - In a terminal (CTRL + ALT + T) :

```
sudo apt-get install git build-essential linux-libc-dev cmake cmake-gui libusb-1.0-0-dev libus
b-dev libudev-devmpi-default-dev openmpi-bin openmpi-common libflann1.8 libflann-dev libeigen3
-dev libboost-all-dev libvtk5.10-qt4 libvtk5.10 libvtk5-dev libqhull* libgtest-dev freeglut3-d
ev pkg-config libxmu-dev libxi-dev mono-complete qt-sdk openjdk-8-jdk openjdk-8-jre libproj-de
v
```

   - Download PCL 1.8.0 source code here: https://github.com/PointCloudLibrary/pcl/archive/pcl-1.8.0.tar.gz
   - Unzip the file pcl-1.8.0.tar.gz (in an explorer: right click, extract here)
   - In a terminal (CTRL + ALT + T) :

```
cd pcl-pcl-1.8.0
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr \ -DCMAKE_INSTALL_PREFIX=/usr ..
make -j7
sudo make install
```

3. **GDAL 2.2.1** (optional, gives access to GDAL/OGR vector and raster formats)

   - Download version 2.2.1 of GDAL, here: http://download.osgeo.org/gdal/2.2.1/gdal-2.2.1.tar.gz
   - Unzip the file gdal-2.2.1.tar.gz (in an explorer: right click, extract here)
   - Open a terminal in the gdal-2.2.1 folder (in an explorer: select the folder, right click, open in a terminal)
   - Launch the following commands:

```
./configure
make
sudo make install
sudo ldconfig
```

4. **GSL** (optional, gives access to a numerical calculation library used in some plugins)

   - In a terminal (CTRL + ALT + T):

```
sudo apt-get install -y gsl-bin libgsl0-dev
```

If you want to install dependencies in different locations (for example in c:/program files), you can, for each *LIBNAME_default_path.pri* file, duplicate it and rename it *LIBNAME_user_path.pri*. After that you just have to modidy this second file to use your local path.

Computree Compilation

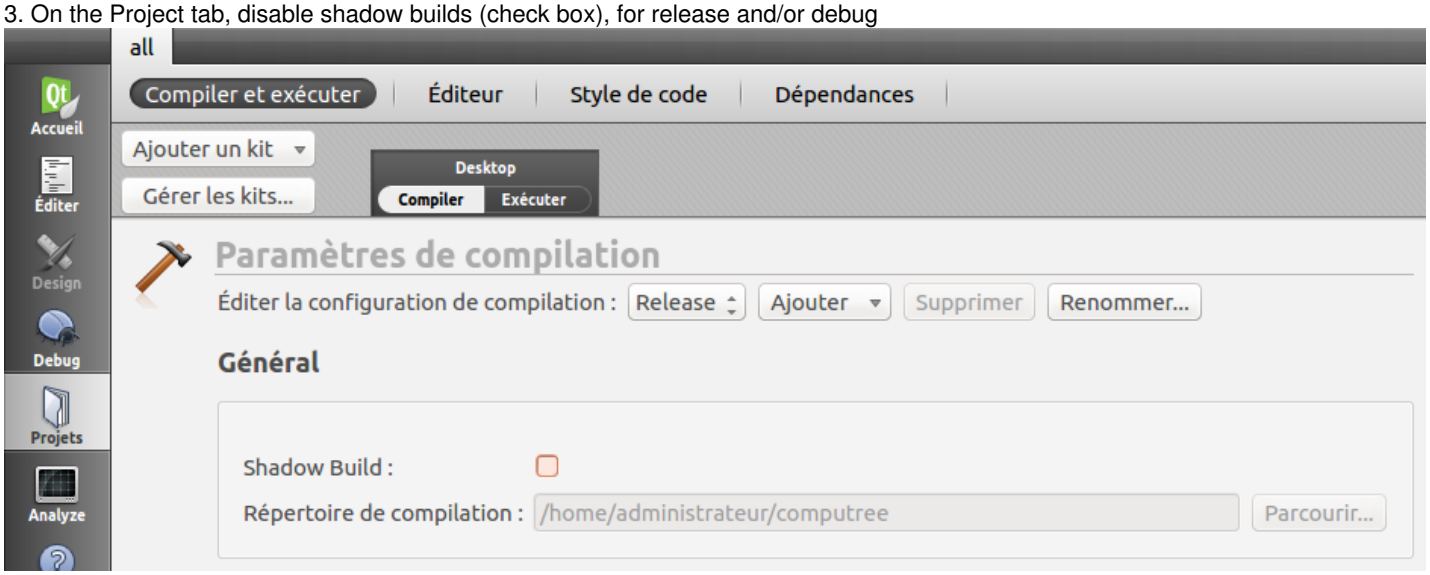
1. Launch Qt Creator

2. Open the project *all.pro*

If you have not installed PCL, delete/comment the following line in all.pro:

```
computreev5/library/ctlibpcl/ctlibpcl.pro \
```

3. On the Project tab, disable shadow builds (check box), for release and/or debug



4. Run qmake on all.pro, then compile the project

After updating the source code, if the core of Computree has been modified significantly, it may be necessary to run qmake on each subproject and then to do Recompile on all.pro.

Execution of Computree

Once compiled, you can run from Qt-Creator (green arrow or run on all.pro).

Configure your plugin if you want to use PCL in your code

If you want to use PCL for your development some preparation steps are required:

You must configure the .pro file of your plugin (.pro) as follows (beginning of the file):

```
CT_PREFIX = ../../computreev5

include($${CT_PREFIX}/shared.pri)
include($${PLUGIN_SHARED_DIR}/include.pri)

COMPUTREE += ctlibpcl

include($${CT_PREFIX}/include_ct_library.pri)
```

Do not forget to compile the libpcl project into the computreev5/library/ctlibpcl folder (open the ctlibpcl.pro file and compile it with QtCreator)

Just make a *qmake* on the project of your plugin (right click → qmake) and compile it.

List of svn repositories

If you want to add repositories, or do a manual installation without the scripts, you will find in the table below a list of repositories for all Computree plugins.

To access a repository, of course, you must have adequate rights for the project.

In general, the name of a repository is http://rdinnovation.onf.fr/svn/nom-du-projet . The name of the project being the name that appears in the address bar of the browser.

Plugin	Plugin code	Project	Svn Repository
Computree (base)	CT	computree	http://rdinnovation.onf.fr/svn/computree
ComputreeDevTools	-	computreedevtools	http://rdinnovation.onf.fr/svn/computreedevtools
Plugin Onf	ONF	plugin-onf	http://rdinnovation.onf.fr/svn/plugin-onf
Plugin Arts Free	ARFR	plugin-arts-free	http://rdinnovation.onf.fr/svn/plugin-arts-free
Plugin Onf Lsis	OL	plugin-onf-lsis	http://rdinnovation.onf.fr/svn/plugin-onf-lsis
Plugin Generate	GEN	plugin-generate	http://rdinnovation.onf.fr/svn/plugin-generate
Plugin ToolKit	TK	plugin-toolkit	http://rdinnovation.onf.fr/svn/plugin-toolkit
Plugin LVox	LVOX	plugin-lvox	http://rdinnovation.onf.fr/svn/plugin-lvox

---

Back to home

Files

shadow_build.png	62.5 KB	12/22/2016	Piboule Alexandre
kit_qt_551.png	158 KB	12/22/2016	Piboule Alexandre

fr_FR.png ...version française de cette page

Implementation of the development environment under Ubuntu 18.04 LTS

Installation of the development environment

1. Installing subversion

In a terminal (CTRL + ALT + T):

```
sudo apt-get update
sudo apt-get install subversion
```

2. Install Qt (5.9.x)

In a terminal (CTRL + ALT + T):

```
sudo apt-get install build-essential qtcreator qt5-default qt5-doc qt5-doc-html qtbase5-doc-html qtbase5-examp
les
```

Retrieving the source code of Computree and its plugins

1. Create a root directory *Computree* where you want

2. Download kit_dev_linux.zip, which contains :

- A *all.pro* file with all projects distributed as standard (core + open-sources plugins)
- A file *recover_depots.sh*, containing svn instructions, to recover the source code
- A file *Read Me.txt*, containing these instructions

3. Unzip *kit_dev_linux.tar.gz* at the root of the *Computree* directory

4. In a terminal (CTRL + ALT + T), run the script *recover_depots.sh*

Installation of dependencies

1. **OpenCV 3.2**

- In a terminal (CTRL + ALT + T):

```
sudo apt-get install libopencv-dev
```

2. **PCL 1.8** (optional, allows you to use plugins that require PCL)

- In a terminal (CTRL + ALT + T):

```
sudo apt-get install libpcl-dev
```

3. **GDAL 2.2.3** (optional, gives access to GDAL/OGR vector and raster GIS formats)

- In a terminal (CTRL + ALT + T):

```
sudo apt-get install libgdal-dev
```

4. **GSL 2.4** (optional, gives access to a numerical calculation library used in some plugins)

- In a terminal (CTRL + ALT + T):

```
sudo apt-get install libgsl-dev
```

If you want to install dependencies in different locations, you can, for each file *LIBNAME_default_path.pri*, duplicate it and rename it *LIBNAME_user_path.pri*. After that, you only have to modify this second file to use your local path.

Compilation of Computree

1. Launch Qt Creator

2. Open the project *all.pro*

If you have not installed PCL, delete/comment the following line in all.pro :

```
computreev5/library/ctlibpcl/ctlibpcl.pro \
```

3. In the project tab disable shadow builds (checkbox), for release and / or debug
shadow_build.png

4. In the project/run tab, in the “Runtime environment” section, add “**;.**” at the end of the variables **LD_LIBRARY_PATH**

5. Run qmake on all.pro, then compile the project

After a source code update, if the core of Computree has been significantly modified, it may be necessary to run qmake on each subproject, then Recompile on all.pro.

6. To get all translations, use the following script in a terminal: *computreev5/scripts/unix_updateLanguage.sh*

Running Computree

Once compiled, you can start the execution from Qt-Creator (green arrow or run/run on all.pro).

Configure your plugin if you want to use PCL in your code

If you want to use PCL for your developments some preparation steps are necessary:

You must configure the.pro file of your plugin (.pro) as follows (beginning of the file):

```
CT_PREFIX = ../.../computreev5

include($${CT_PREFIX}/shared.pri)
include($${PLUGIN_SHARED_DIR}/include.pri)

COMPUTER += ctlibpcl

include($${CT_PREFIX}/include_ct_library.pri)
```

Don’t forget to compile the libpcl project in the computreev5/library/ctlibpcl folder (open the file ctlibpcl.pro and compile it with QtCreator)

All you have to do now is to do a *qmake* on your plugin project (right click → qmake) and compile it.

Files

kit_dev_linux.zip	2.44 KB	11/29/2018	Piboule Alexandre

en_US.png ...english version of this page

Mise en place de l'environnement de développement sous Ubuntu 18.04 LTS

Installation de l'environnement de développement

1. Installer subversion

Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get update
sudo apt-get install subversion
```

2. Installer Qt (5.9.x)

Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install build-essential qtcreator qt5-default qt5-doc qt5-doc-html qtbase5-doc-html qtbase5-examp
les
```

Récupération du code source de Computree et de ses plugins

1. Créer un répertoire racine *Computree* où vous le souhaitez

2. Télécharger kit_dev_linux.zip, qui contient :

- Un fichier *all.pro* avec tous les projets diffusés en standard (cœur + plugins open-sources)
- Un fichier *recuperer_depots.sh*, contenant des instructions svn, pour récupérer le code source
- Un fichier *LISEZ-MOI.txt*, reprenant ces instructions

3. Décompresser *kit_dev_linux.tar.gz* à la racine du répertoire *Computree*

4. *Dans un terminal (CTRL + ALT + T), lancer le script _recuperer_depots.sh*

Installation des dépendances

1. **OpenCV 3.2**

- Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install libopencv-dev
```

2. **PCL 1.8** (optionnel, permet d'utiliser les plugins nécessitant PCL)

- Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install libpcl-dev
```

3. **GDAL 2.2.3** (optionnel, donne accès aux formats SIG vecteur et raster de GDAL/OGR)

- Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install libgdal-dev
```

4. **GSL 2.4** (optionnel, donne accès à une librarie de calcul numérique utilisée dans certains plugins)

- Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install libgsl-dev
```

Si vous souhaitez installer des dépendances dans des emplacements différents, vous pouvez, pour chaque fichier *LIBNAME_default_path.pri*, le dupliquer et le renommer *LIBNAME_user_path.pri*. Après cela, vous n'avez qu'à modifier ce deuxième fichier pour utiliser votre chemin local.

Compilation de Computree

1. Lancer Qt Creator

2. Ouvrir le projet *all.pro*

Si vous n'avez pas installé PCL, supprimez / commentez la ligne suivante dans all.pro :

```
computreev5/library/ctlibpcl/ctlibpcl.pro \
```

3. Dans l'onglet projet désactiver les shadow builds (case à cocher), pour la release et / ou la debug
shadow_build.png

4. Dans l'onglet projet/run, dans la section "Environnement d'exécution", rajouter " **;.** " à la fin de la variables **LD_LIBRARY_PATH**

5. Exécuter qmake sur all.pro, puis compiler le projet

Après une mise à jour du code source, si le cœur de Computree a été modifié significativement, il peut être nécessaire d'exécuter qmake sur chaque sous-projet, puis de faire Recompiler sur all.pro.

6. Pour obtenir toutes les traductions, utilisez le script suivant dans un terminal : *computreev5/scripts/unix_updateLanguage.sh*

Exécution de Computree

Une fois compilé, vous pouvez lancer l'exécution depuis Qt-Creator (flèche verte ou run/exécuter sur all.pro).

Configurer votre plugin si vous souhaitez utiliser PCL dans votre code

Si vous souhaitez utiliser PCL pour vos développements quelques étapes de préparation sont nécessaires :

Vous devez configurer le fichier .pro de votre plugin (.pro) comme suit (début du fichier) :

```
CT_PREFIX = ../../computreev5

include($${CT_PREFIX}/shared.pri)
include($${PLUGIN_SHARED_DIR}/include.pri)

COMPUTREE += ctlibpcl

include($${CT_PREFIX}/include_ct_library.pri)
```

N'oubliez pas de compiler le projet libpcl dans le dossier computreev5/library/ctlibpcl (ouvrez le fichier ctlibpcl.pro et le compiler avec QtCreator)

Il suffit maintenant de faire un *qmake* sur le projet de votre plugin (clic droit → qmake) et de le compiler.

Files

kit_dev_linux.zip	2.44 KB	07/31/2018	Piboule Alexandre

en_US.png ...english version

Documentation pour les développeurs de plugins

Ce projet regroupe les documentations et outils à destination des **développeurs de plugins** pour *Computree*.

Retour à l'accueil

Différents types de groupes: standard, abstrait et topologique

CT_StandardItemGroup: le groupe standard

Le groupe standard est le plus fréquent et est celui décrit à la section Programmer avec des objets Computree : résultats, groupes, items et attributs. À priori, on sait la nature et le nombre de niveaux hiérarchiques qu'il contient.

CT_AbstractItemGroup: le groupe abstrait

Le groupe abstrait représente l'interface implémentée par CT_StandardItemGroup et permet l'implémentation future de nouveaux types de groupes différents du groupe standard, si besoin est.

CT_TtreeGroup: le groupe topologique

Un exemple de groupe différent est le groupe topologique récursif, utilisé pour décrire la topologie d'un arbre. Contrairement au groupe standard, on ne sait pas combien de niveaux de profondeur ce groupe possède. Ce groupe permet la représentation de relations de type ramification, composition et succession, relations qui ne pourraient être correctement représentées avec un groupe standard.

Programmer avec des objets Computree : résultats, groupes, items et attributs

Le pourquoi de la (des) chose(s)

Une étape de traitement quelconque reçoit en entrée la sortie d'une autre étape et, puisqu'elle n'a pas construit le résultat en question, elle doit avoir un moyen de connaître la composition de ses intrants. Cette composition doit être standard pour toutes les étapes, même si elles font partie de différentes extensions, potentiellement développées par différentes entités. Pour cette raison, Computree présente ses données sous la forme d'items (données), groupes (hiérarchie des items) et résultats (racine unique).

Items

Les items sont la structure de données la plus facile à comprendre : ils consistent en les données elles-mêmes. Il peut s'agir d'un nuage de points, d'un cylindre, d'une grille, etc. Il s'agit toujours d'un item unique, c'est-à-dire par exemple *un* nuage de points, jamais un *groupe* de nuage de points.

Groupes

Les groupes servent à regrouper les items. Par exemple, un billot sera un groupe de plusieurs cylindres. Un groupe peut aussi composer d'autres groupes; par exemple :

- Groupe : arbre
  - Item : tronc
  - Groupe : branches
    - Item : nuage de points
    - Groupe : branchettes
      - Item : nuage de points

Les groupes ne contiennent jamais de données, seulement des items et des groupes.

Résultats

Il arrive qu'une étape traite plus d'un groupe. Par exemple, charger trois scènes mènera à la présence de trois groupes, contenant chacun les items d'une scène. Afin de simplifier l'affichage et le travail du contrôleur en général, une racine unique doit exister; c'est le rôle du résultat.
À l'occasion, et historiquement, une étape peut utiliser des résultats multiples. Par exemple, une étape fusionnant des scènes pourrait travailler avec trois résultats en entrée (chaque scène) et pourrait produire un quatrième résultat contenant la scène fusionnée.

- Résultat : scène 1
  - Groupe : scène
    - Item : …
    - Item…
- Résultat : scène 2
  - Groupe : scène
    - Item : …
    - Item…
- Résultat : scène 3
  - Groupe : scène
    - Item : …
    - Item…

→ fusion →

- Résultat : scène 1
  - Groupe : scène
    - Item : …
    - Item…
- Résultat : scène 2
  - Groupe : scène
    - Item : …
    - Item…
- Résultat : scène 3
  - Groupe : scène
    - Item : …
    - Item…
- **Résultat : scène 4**
  - **Groupe : scène**
    - **Item : …**
    - **Item…**

Attributs

Un item possède des attributs qui lui sont propres. Par exemple, un cercle possède au minimum les coordonnées de son centre et un rayon. Toutefois, il arrive qu'une étape doive modifier les attributs d'un item. Par exemple, si une étape ajoute des cercles calculés avec la méthode des moindres carrés, il serait pertinent qu'un attribut contienne une valeur indiquant la qualité de l'ajustement, r^2 ou MSE, par exemple. Comme il est impossible de prévoir tous les attributs imaginables d'un cercle dans Computree, les étapes ont la possibilité d'ajouter des attributs aux items, soit un champ scalaire. Pour ce faire, il est nécessaire de modifier le modèle des données avec addItemAttributeModel et les données elles-mêmes avec addItemAttribute (voir section Modèles vs. instances de données).

```
void LVOX_StepCombineDensityGrids::createInResultModelListProtected()
{
    CT_InResultModelGroup *resultModel = createNewInResultModel(DEF_resultIn_grids, tr("Grilles"));
    resultModel->setZeroOrMoreRootGroup();
    resultModel->addGroupModel("", DEF_groupIn_grids);
    resultModel->addItemModel(DEF_groupIn_grids, DEF_itemIn_hits, CT_Grid3D<int>::staticGetType(),
```

```
    tr("hits"), "", CT_InAbstractModel::C_ChooseOneIfMultiple, CT_InAbstractModel::F_IsOptional);
    resultModel->addItemAttributeModel(DEF_itemIn_hits, DEF_inATTisNi, QList<QString>() <<
    "LVOX_GRD_NI", CT_AbstractCategory::ANY, tr("isNi"), "",
    CT_InAbstractModel::C_ChooseOneIfMultiple, CT_InAbstractModel::F_IsOptional);
}
```

Files

groupe.PNG	16 KB	07/21/2016	Luce Myriam
fusion.PNG	35.8 KB	07/21/2016	Luce Myriam

Le système de LOG

Besoin d'enregistrer dans un fichier ou afficher sur la sortie standard (std::cout) ou dans la fenêtre de LOG de l'interface graphique un/des événements particuliers de votre plugin, étape, action ? Vous êtes dans la bonne section !

Le log d'évènements trivial

Dans votre code il faut tout d'abord inclure (si ce n'est pas déjà fait) le fichier **ct_global/ct_context.h**. Puis simplement appeler la méthode **addMessage** de l'objet **PS_LOG**.

```
#include "ct_global/ct_context.h" // Contexte

// .....

PS_LOG->addMessage(LogInterface::info, LogInterface::step, tr("Mon message"));

// ou si vous voulez que le type soit détecté automatiquement
PS_LOG->addMessage(LogInterface::info, this, tr("Mon message"));

// .....
```

Le premier paramètre permet de connaitre le type de message, afin d'offrir une première fonction de filtrage :

- trace
- debug
- info
- warning
- error
- fatal

Le deuxième paramètre permet de connaitre l'élément qui envoie l'événement, afin d'offrir une deuxième fonction de filtrage :

- core : utilisé par ComputreeCore
- gui : utilisé par ComputreeGui
- plugin
- step
- result
- itemdrawable
- action
- reader
- exporter
- unknow

La méthode **addMessage** distribue le message à tous les **CT_AbstractLogListener** qui décident de traiter ou non le message, en fonction des éléments de filtrage ou autres.

Par défaut la fenêtre "log" de l'interface graphique affiche **tous les messages** envoyé à **PS_LOG**.

CT_AbstractLogListener

Mais qu'est-ce donc ?

Un **CT_AbstractLogListener** est une interface dont vous pouvez hériter si vous souhaitez traiter différemment les messages reçu par la méthode **addMessage**.

Dans **PluginShared** il existe deux **CT_AbstractLogListener** prédéfinis :

- CT_FileLogListener : permet d'enregistrer les logs dans un fichier dont vous définissez le chemin
- CT_BasicOStreamLogListener : permet d'afficher les logs sur une sortie standard (std::cout ou std::cerr ou etc...)

Ces deux classes héritent de **CT_AbstractLogListener** et permettent ainsi de filtrer les messages reçus.

LogInterface

La classe **LogInterface** permet d'installer plusieurs **CT_AbstractLogListener** et répartit les messages reçu lors de l'appel à **PS_LOG→addMessage...** à tous les listener.

Installer un CT_AbstractLogListener prioritaire

Si vous souhaitez recevoir les messages en priorité (dès que l'objet LogInterface reçoit le message) et que **vous traiter rapidement les messages** vous devez appeler la méthode **addPrioritaryLogListener** de l'objet **PS_LOG** en lui passant le listener.

```
#include "ct_global/ct_context.h" // Contexte

// constructeur du StepPluginManager
XXXX_StepPluginManager::XXXX_StepPluginManager() : CT_AbstractStepPlugin()
{

//m_myLogListener est un attribut de ma classe XXXX_StepPluginManager. Cet attribut est du type d'une classe q
ui hérite de CT_AbstractLogListener.


// défini le filtre sur le type de message. Nous n'accepterons que les messages du type "debug" et "error"
    m_myLogListener.setSeverityAccepted(QVector<int>() << LogInterface::debug << LogInterface::error);


// défini le filtre sur l'élément qui a envoyé le message. Nous n'accepterons que les message d'une étape ou d
'une action
    m_myLogListener.setTypeAccepted(QVector<int>() << LogInterface::step << LogInterface::action);


// défini un dernier filtre afin de n'accepter que les message de mon plugin. Nous n'accepterons que les messa
ges dont la variable "filter" contiendra "xxxx"
    m_myLogListener.setFilter("xxxx");
}

// destructeur du StepPluginManager
XXXX_StepPluginManager::~XXXX_StepPluginManager()
{
    // enleve le listener de l'objet du type LogInterface

    // TRES IMPORTANT !! sous peine de planter l'application
    PS_LOG->removeLogListener(&m_myLogListener);
}

// méthode init du StepPluginManager
bool XXXX_StepPluginManager::init()
{
    // enregistrement du listener auprès de PS_LOG
    PS_LOG->addNormalLogListener(&m_myLogListener);

    // envoi d'un message pour signaler l'initialisation du plugin
    PS_LOG->addMessage(LogInterface::debug, LogInterface::plugin, QObject::tr("Plugin_XXXX initialized"), "
xxxx");


// envoi d'un message qui ne sera pas traité par notre propre listener puisque la variable "filter" ne contien
dra pas "xxxx". Par contre il sera vu dans la fenêtre de log de l'interface graphique.
    PS_LOG->addMessage(LogInterface::debug, LogInterface::plugin, QObject::tr("Plugin_XXXX test"),);

    return CT_AbstractStepPlugin::init();
}

// .....
```

Un listener enregistré en tant que listener prioritaire doit absolument traiter les messages très rapidement sous peine de voir l'application ralentir.

Si l'application plante et que vous voulez être sûr d'avoir le message avant le plantage vous devez installer votre listener de cette façon.

Installer un CT_AbstractLogListener "patient"

Si votre listener met du temps à traiter les messages vous devez l'installer de la façon suivante :

```
#include "ct_global/ct_context.h" // Contexte

// constructeur du StepPluginManager
```

```
XXXX_StepPluginManager::XXXX_StepPluginManager() : CT_AbstractStepPlugin()
{

//m_myFileLogListener est un attribut de ma classe XXXX_StepPluginManager qui est du type CT_FileLogListener

    // défini le chemin vers le fichier
    m_myFileLogListener.setFilePath("./logXXXX.txt");


// défini le filtre sur le type de message. Nous acceptons tous les types de messages. Nous aurions très bien
pu ne pas appeler cette méthode, le résultat aurait été le même.
    m_myFileLogListener.setSeverityAccepted(QVector<int>());


// défini le filtre sur l'élément qui a envoyé le message. Nous acceptons les messages de n'importe qui. Nous
aurions très bien pu ne pas appeler cette méthode, le résultat aurait été le même.
    m_myFileLogListener.setTypeAccepted(QVector<int>());


// défini un dernier filtre afin de n'accepter que les message de mon plugin. Nous n'accepterons que les messa
ges dont la variable "filter" contiendra "xxxx"
    m_myFileLogListener.setFilter("xxxx");
}

// destructeur du StepPluginManager
XXXX_StepPluginManager::~XXXX_StepPluginManager()
{
    // enleve le listener de l'objet du type LogInterface

    // TRES IMPORTANT !! sous peine de planter l'application
    PS_LOG->removeLogListener(&m_myLogListener);
}

// méthode init du StepPluginManager
bool XXXX_StepPluginManager::init()
{
    // enregistrement du listener auprès de PS_LOG
    PS_LOG->addNormalLogListener(&m_myLogListener);

    // envoi d'un message pour signaler l'initialisation du plugin
    PS_LOG->addMessage(LogInterface::debug, LogInterface::plugin, QObject::tr("Plugin_XXXX initialized"), "
xxxx");


// envoi d'un message qui ne sera pas traité par notre propre listener puisque la variable "filter" ne contien
dra pas "xxxx". Par contre il sera vu dans la fenêtre de log de l'interface graphique.
    PS_LOG->addMessage(LogInterface::debug, LogInterface::plugin, QObject::tr("Plugin_XXXX test"),);

    return CT_AbstractStepPlugin::init();
}

// .....
```

Un autre thread traite la file des messages. On peut voir le principe comme celui d'un producteur (LogInterface) et de consommateurs (CT_AbstractLogListener).

Installez toujours un CT_FileLogListener en tant que listener patient car le temps d'écriture dans le fichier est long.

Créer votre propre Listener

Il vous suffit d'hériter de la classe CT_AbstractLogListener disponible dans "ct_log/abstract" et de redéfinir la méthode *addMessage".

```
#include "ct_log/abstract/ct_abstractloglistener.h"

class XXXX_MyLogListener : public QObject, public CT_AbstractLogListener
{
public:
    XXXX_MyLogListener ();
```

```
    void addMessage(const int &severity, const int &type, const QString &s, const QString &filter);

private slots:
    void traiterMessage(const QString &message);

signals:
    void newMessageReceived(const QString &message);
};
```

```
#include "xxxx_myloglistener.h"

XXXX_MyLogListener ::XXXX_MyLogListener () : QObject(), CT_AbstractLogListener()
{
      // je m'envoie le message par l'intermédiaire d'un slot pour être dans le thread du GUI
      connect(this, SIGNAL(newMessageReceived(QString)), this
, SLOT(traiterMessage(QString)), Qt::QueuedConnection);
}

void XXXX_MyLogListener ::addMessage(const int &severity, const int &type, const QString &s, const
 QString &filter)
{
    // si j'accepte ce message
    if(acceptMessage(severity, type, filter))
    {
          // je traite mon message.

// ATTENTION cette méthode est appelée par un thread donc si j'affiche ce message dans un élément de l'interfa
ce graphique je dois m'envoyer un signal avec le message, exemple :
          emit newMessageReceived(s);
    }
}

void XXXX_MyLogListener ::traiterMessage(const QString &s)
{
    // je peux traiter sereinement mon message puisque je suis dans le thread du GUI
}
```

Travailler avec un maillage (Mesh)

Dans cette section nous allons voir comment :

- sont stockés les faces et arêtes dans le système
- créer un Mesh
- parcourir les points/faces/arêtes d'un mesh

Veuillez tout d'abord lire la section concernant l'utilisation des points Fr_points

Les nuages global des faces et arêtes

Tout comme les points lors du lancement de l'application un vecteur de faces ainsi qu'un vecteur d'arêtes sont créés afin de contenir les faces/arêtes que vous allez fabriquer dans vos steps/readers. Le mesh ne contiendra que les indices des points, faces et arêtes créés (tout comme une scène) et vous pourrez utiliser les itérateurs **CT_PointIterator**, **CT_FaceIterator** et **CT_EdgeIterator** pour parcourir les éléments du mesh.

Création d'un Mesh

L'ItemDrawable à utiliser pour créer un mesh est un **CT_MeshModel**. Celui-ci utilise un **CT_Mesh** qui est la classe contenant les indices des points, faces et arêtes (edge) créé.

Imaginons que vous vouliez créé un carré centré en position [80000, 0, 50000] et de dimension [5, 5]. Celui-ci est constitué de 4 points, 2 faces et 6 arêtes, il nous faudra alors créer un nuage de points, un nuage de faces et un nuage d'arêtes :

```
// création du mesh
CT_Mesh *mesh = new CT_Mesh();

/*****************************/
/********** POINTS ***********/
/*****************************/

// création de 4 points
CT_MutablePointIterator itP = CT_MeshAllocator::AddVertices(mesh, 4);

// passe au premier point
itP.next();

// on récupère son index global
size_t globalIndexFirstPoint = itP.currentGlobalIndex();

// modification du point 1
itP.replaceCurrentPoint(createCtPoint(79997.5, -2.5, 50000));

// modification du point 2 en une seule ligne
itP.next().replaceCurrentPoint(createCtPoint(80002,5, -2.5, 50000));

// modification du point 3 en une seule ligne
itP.next().replaceCurrentPoint(createCtPoint[1];

// modification du point 4 en une seule ligne
itP.next().replaceCurrentPoint(createCtPoint(80002,5, 2.5, 50000));

/*****************************/
/*********** FACES ***********/
/*****************************/

// création de 2 faces
CT_MutableFaceIterator itF = CT_MeshAllocator::AddFaces(mesh, 2);

// récupère la première face
CT_Face &face1 = itF.next().cT();

// récupère l'index de la face 1
size_t faceIndex = itF.cIndex();

// création de 3 arêtes
CT_MutableEdgeIterator itE = CT_MeshAllocator::AddHEdges(mesh, 3);

// création de variable contenant les indices des arêtes
size_t e1Index = itE.next().cIndex();
size_t e2Index = e1Index + 1;
```

```
size_t e3Index = e1Index + 2;

// on passe l'index de la première arête à la face. Une face doit connaitre au moins une arête.
face1.setEdge(e1Index);

/*****************************/
/*********** ARETES **********/
/******* première partie ******/
/*****************************/

// création de variables contenant les indices des points de la face
size_t p0 = globalIndexFirstPoint;
size_t p1 = p0+1;
size_t p2 = p0+2;

// on récupère l'arête 1
CT_Edge &e1 = itE.next().cT();

// et on lui affecte les indices des points qui la compose
e1.setPoint0(p0);
e1.setPoint1(p1);

// ainsi que l'indice de la face
e1.setFace(faceIndex);

// on récupère l'arête 2
CT_Edge &e2 = itE.next().cT();
e2.setPoint0(p1);
e2.setPoint1(p2);
e2.setFace(faceIndex);

// on récupère l'arête 3
CT_Edge &e3 = itE.next().cT();
e3.setPoint0(p2);
e3.setPoint1(p0);
e3.setFace(faceIndex);

// on définit qui est la précédente, qui est la suivante parmis les arêtes
e1.setNext(e2Index);
e1.setPrevious(e3Index);
e2.setNext(e3Index);
e2.setPrevious(e1Index);
e3.setNext(e1Index);
e3.setPrevious(e2Index);

/*****************************/
/*********** ARETES **********/
/******* deuxième partie ******/
/*****************************/

// création de 3 nouvelles arêtes
itE = CT_MeshAllocator::AddHEdges(mesh, 3);

// création de variable contenant les indices des arêtes
e1Index = itE.next().cIndex();
e2Index = e1Index + 1;
e3Index = e1Index + 2;

// récupère la deuxième face
CT_Face &face2 = itF.next().cT();

// récupère l'index de la face 2
faceIndex = itF.cIndex();

// on passe l'index de la première arête à la face. Une face doit connaitre au moins une arête.
face2.setEdge(e1Index);

// création de variables contenant les indices des points de la face
p0 = globalIndexFirstPoint+3;
p1 = p0+1;
p2 = p0+2;

// on récupère l'arête 1
CT_Edge &e12 = itE.next().cT();
```

```
// et on lui affecte les indices des points qui la compose
e12.setPoint0(p0);
e12.setPoint1(p1);

// ainsi que l'indice de la face
e12.setFace(faceIndex);

// on récupère l'arête 2
CT_Edge &e22 = itE.next().cT();
e22.setPoint0(p1);
e22.setPoint1(p2);
e22.setFace(faceIndex);

// on récupère l'arête 3
CT_Edge &e32 = itE.next().cT();
e32.setPoint0(p2);
e32.setPoint1(p0);
e32.setFace(faceIndex);

// on définit qui est la précédente, qui est la suivante parmis les arêtes
e12.setNext(e2Index);
e12.setPrevious(e3Index);
e22.setNext(e3Index);
e22.setPrevious(e1Index);
e32.setNext(e1Index);
e32.setPrevious(e2Index);

/*****************************/
/******** Finalisation ********/
/*****************************/

// création du CT_MeshModel
CT_MeshModel *meshModel  = new CT_MeshModel(DEF_SearchMesh, out_res, mesh);
```

Parcours des points/faces/arêtes d’un mesh

Pour le parcours il suffit d’utiliser les itérateurs :

```
CT_PointIterator itP(meshModel->getPointCloudIndex());

while(itP.hasNext()) {
      itP.next();
      CT_Point p = itP.currentPoint();

      // traitement
      ....
}

// création d'un objet permettant de récupérer un point à partir de son indice global
CT_PointAccessor pAccess;

CT_FaceIterator itF(meshModel->getFaceCloudIndex());

while(itF.hasNext()) {
      itF.next();
      const CT_Face &f = itF.cT();

      // on demande à la face l'indice global du point 0 puis on demande au nuage global le point à cet indice
      CT_Point p0 = pAccess.pointAt( f.iPointAt(0) );

      // on peu demander à la face les points de 0 à 2 inclus
      CT_Point p1 = pAccess.pointAt( f.iPointAt(1) );
      CT_Point p2 = pAccess.pointAt( f.iPointAt(2) );

      // traitement
      ....
}

CT_EdgeIterator itE(meshModel->getFaceCloudIndex());
```

```
while(itE.hasNext()) {
      itE.next();
      const CT_Edge &e = itE.cT();

      // on demande à l'arête l'indice global du point 0 puis on demande au nuage global le point à cet indice
      CT_Point p0 = pAccess.pointAt( e.iPointAt(0) );

      // on peu demander à l'arête les points de 0 à 1 inclus
      CT_Point p1 = pAccess.pointAt( e.iPointAt(1) );

      // traitement
      ....
}
```

[1] 79997.5, 2.5, 50000

Modèles vs. instances de données

Un script Computree, soit une série d'étapes, n'est pas exécuté au moment où il est créé; autrement dit, les étapes sont ajoutées avant d'être exécutées. Afin de valider qu'une étape peut être ajoutée à la suite d'une autre, il faut que les sorties de la première et les entrées de la seconde soient connues et compatibles. Il est donc nécessaire de connaître leur format avant que quelques données que ce soit aient été traitées. Pour ce faire, Computree utilise une stratégie en deux temps : au moment de l'ajout de l'étape à l'arbre de traitement, l'étape spécifie le *modèle* de ses données; au moment du traitement, l'étape traite les *instances* de ses données.

Modèles

Les modèles de données doivent être spécifiés pour les entrées et les sorties de l'étape, soit dans les méthodes createInResultModelListProtected et createOutResultModelListProtected. Les méthodes utilisées indiquent d'ailleurs qu'elles traitent le modèle des données : addGroupModel et addItemModel.

```
void LVOX_StepNdNtGrids::createInResultModelListProtected()
{
    CT_InResultModelGroupToCopy* in_res = createNewInResultModelForCopy(DEF_in_res, tr("Grilles"));
    in_res->setZeroOrMoreRootGroup();
    in_res->addGroupModel("", DEF_in_grp);
    in_res->addItemModel(DEF_in_grp, DEF_in_nd, CT_Grid3D<int>::staticGetType(), tr("nb (before)"));
    in_res->addItemModel(DEF_in_grp, DEF_in_nt, CT_Grid3D<int>::staticGetType(), tr("nt (theo)"));
}

void LVOX_StepNdNtGrids::createOutResultModelListProtected()
{
    CT_OutResultModelGroupToCopyPossibilities* out_res = createNewOutResultModelToCopy(DEF_in_res);
    if (res != NULL)
    {
        out_res->addItemModel(DEF_in_grp_ndnt, _ndnt_ModelName, new CT_Grid3D<float>(), tr("nd/nt"));
    }
}
```

Instances

Les données elles-mêmes sont uniquement accessibles au cours du traitement des données, soit au sein de la méthode compute et leur structure peut être modifiée à l'aide de méthodes comme addGroup et addItemDrawable.

```
void LVOX_StepNdNtGrids::compute()
{
    // Get inputs and outputs...
    // ...

    // Iterate...
    CT_ResultGroupIterator it(out, this, DEF_in_grp);
    while (it.hasNext() && !isStopped())
    {
         // Various treatments...
     // ...

          // Add actual items and groups to result
        out_group->addItemDrawable(nbnt);
        out_res->addGroup(out_group);
        nbnt->computeMinMax();
    }
}
```

Travailler avec les points

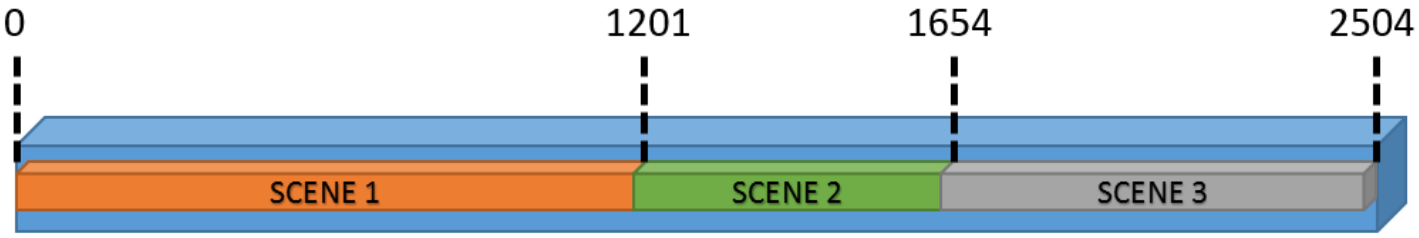
Cette section vous explique en détails comment :

- sont stocké les points
- créer un nouveau nuage de points sans perdre de précision à l'affichage
- parcourir un nuage de points à partir d'un nuage d'index
- obtenir un point à partir de son index global

Le système de nuage de points global et les indices de points

Lors du lancement de l'application le système crée un vecteur de points qui contiendra l'ensemble des points créés dans vos étapes / readers.

Voici par exemple le nuage de point global qui contient les points de 3 scènes différentes (créées à partir de 3 étapes différentes) :



La classe **CT_Scene** ne retient qu'une liste de tous les indices des points créés (classe **CT_NMPCIR**). Par exemple :

- La scène 1 a créé 1201 points et contient un **CT_NMPCIR** qui est un nuage d'index non modifiable contenant les indices des points [0;1200]
- La scène 2 a créé 453 points et contient un **CT_NMPCIR** qui est un nuage d'index non modifiable contenant les indices des points [1201;1653]
- La scène 3 a créé 850 points et contient un **CT_NMPCIR** qui est un nuage d'index non modifiable contenant les indices des points [1654;2503]

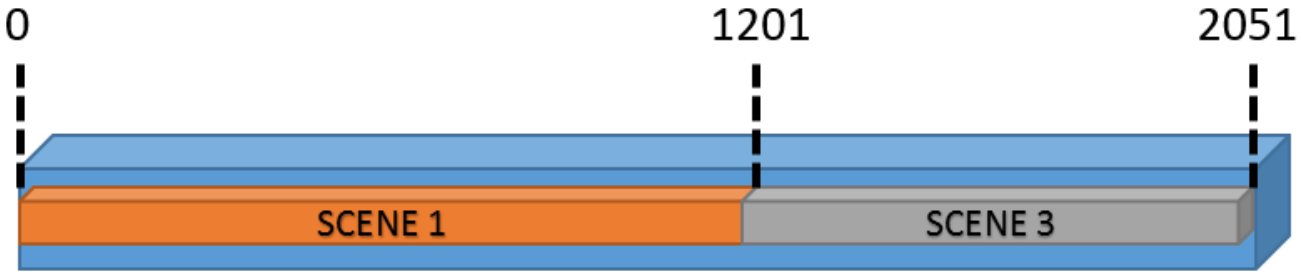
Le nuage de points global contient 2504 points au total.

CT_NMPCIR (Not Modifiable Point Cloud Index Registered) est une classe qui contient les indices des points créés et informent le système lorsqu'elle est supprimé de la mémoire. Ainsi le système peut supprimer les points créés et synchroniser tous les autres indices si cet élément n'est plus utilisé.

La synchronisation des indices de points

En utilisant ce système il a fallu mettre en place un élément qui va synchroniser les nuages d'indices si certains points sont supprimés du nuage de points global.

Par exemple si la scène 2 est supprimée de la mémoire il va falloir synchroniser les indices de la scène 3 puisqu'il ne correspondront plus avec les bon points. Pour cela le système va décaler tous les indices de la scène 3 automatiquement lorsque la scène 2 est supprimée. Vous n'avez donc pas à vous préoccuper de cet aspect puisqu'il est gérer automatiquement et est totalement transparent.



Le décalage des indices des points est transparent pour le développeur car il est géré automatiquement par le système

Les points et les systèmes de coordonnées

Le point utilisé en interne

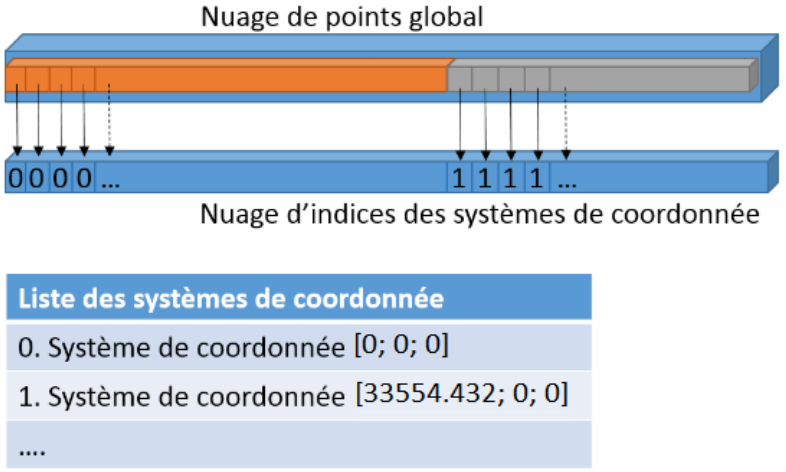
En interne le système stocke les points dans le nuage de points global en utilisant le type **CT_PointData** qui est une classe dérivant de **Eigen::Vector3f**.

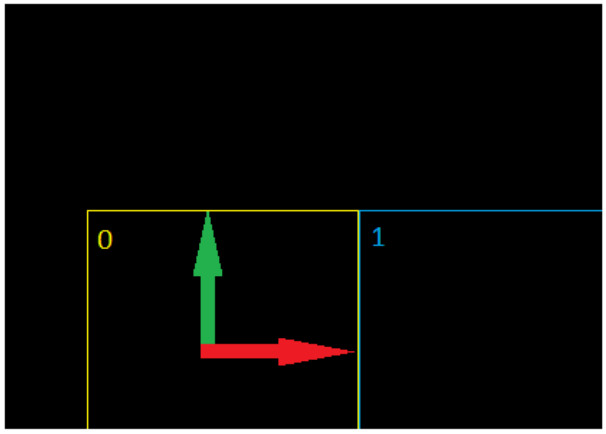
En interne les points sont stockés en utilisant des **float** pour gagner de l'espace mémoire et pour la visualisation avec OpenGL.

En effet **OpenGL n'utilise pas le type double** pour dessiner des points. De ce fait, en considérant une précision au mm, les coordonnées peuvent aller de **-16 777, 216 m à 16 777, 216 m**. Ce qui signifie qu'on ne peut avoir des scènes excédant **32 km de diamètre** à cette précision, et que des données en projection géographique dont les coordonnées sont en million de mètres perdent (fortement) en précision. Lors de l'affichage en 3D d'une scène dépassant la précision d'un float les points "sautent" d'un endroit à l'autre puisqu'OpenGL arrondi les valeurs.

Système de coordonnée

Les systèmes de coordonnées sont des éléments qui transforment les points à la lecture et à l'export pour ne pas perdre en précision lors de l'affichage. Ce chapitre n'est qu'à titre d'information puisque l'utilisation des systèmes de coordonnées est totalement transparent pour le développeur.





L'image ci-dessus présente comment les systèmes de coordonnées sont stockés/utilisés.

- L'espace 3D est découpé en cube suivant la précision que vous désiré (mm, 1/10mm, etc...). Pour une précision au mm les cubes font 33554.432 mètres de large (la distance maximum entre deux points d'un nuage pour avoir une précision au mm lors de l'affichage).
- Lors du chargement d'un point on calcul dans quel cube il se trouve et si un système de coordonnée y est associé, si non on le crée. Une liste de système de coordonnée est alors constituée. Cette liste contiendra toujours le système de coordonnée [0;0;0] à l'indice 0.
- Un nuage d'indice est synchronisé avec le nuage de points global. Ce nuage contient pour chaque point du nuage global l'indice du système de coordonnée utilisé.

A l'affichage, lors des traitements ou lors de l'exportation les points sont convertis (float → double) automatiquement en utilisant les systèmes de coordonnées pour que la précision ne soit pas perdu.

Les systèmes de coordonnées sont créés automatiquement au chargement des points. Ils sont transparents pour le développeur.

Le point utilisé par les développeurs

Afin de rendre transparent la conversion d'un point interne vers sa représentation en double nous avons créé la classe **CT_Point** qui dérive de Eigen::Vector3d (double) et qui est renvoyé par les objets permettant d'accéder aux point d'un nuage. Un **CT_PointData** est automatiquement convertit en un **CT_Point** à l'aide de son système de coordonnée assigné. Nous verrons dans un prochain chapitre comment récupérer facilement un **CT_Point**.

Tous les ItemDrawable tel que les cercles/ellipses/rectangles/etc.... utilisent des **double** pour faire leurs calculs et stocker les résultats mais n'utilisent pas de système de coordonnée.

Création d'un nuage de points de taille fixe

Dans cet exemple nous allons voir comment créer un nouveau nuage de points dont la taille est connue par avance.

```
// variable qui va contenir le nombre de points (lu dans le fichier)
size_t size = 0;

// initialisation pour la lecture du fichier
.......

// création du nuage de point. On récupère le nuage d'indice des points créés qu'il faudra donner à la scène.
CT_NMPCIR pcir = PS_REPOSITORY->createNewPointCloud(size);

// création d'un itérateur qui va permettre de parcourir les points créés et les modifier
CT_MutablePointIterator it(pcir);

// tant qu'il y a des points à parcourir et tant que l'utilisateur n'a pas stoppé le traitement
```

```
while(it.hasNext() && !isStopped()) {

      // récupération du point du fichier
      CT_Point p = nextPointInFile();

      // passe au point suivant du nuage
      it.next();


// modification du point courant afin qu'il prenne la nouvelle valeur et utilise le système de coordonnée pass
és en paramètre
      it.replaceCurrentPoint(p);
}

// création de la scène
CT_Scene *scene = new CT_Scene(....);

// définit les indices des points que la scène doit dessiner
scene->setPointCloudIndexRegistered(pcir);
```

Création d’un nuage de points de taille inconnu

Dans cet exemple nous allons voir comment créer un nouveau nuage de points dont la taille n’est pas connue par avance.

```
// initialisation pour la lecture du fichier
.......

// Demande de création d'un nuage de points dont la taille est inconnu
CT_AbstractUndefinedSizePointCloud *cloud = PS_REPOSITORY->createNewUndefinedSizePointCloud();

// tant qu'il y a des points dans le fichier à parcourir et tant que l'utilisateur n'a pas stoppé le traitemen
t
while(hasNextPointInFile() && !isStopped()) {

      // récupération du point du fichier
      CT_Point p = nextPointInFile();

      // ajout du point avec l'indice du système de coordonnée qu'il doit utiliser
      cloud->addPoint(p);
}

// récupération du nuage d'indice des points qu'on vient de créer
CT_NMPCIR pcir = PS_REPOSITORY->registerUndefinedSizePointCloud(cloud);

// création de la scène
CT_Scene *scene = new CT_Scene(....);

// définit les indices des points que la scène doit dessiner
scene->setPointCloudIndexRegistered(pcir);
```

Parcours de points à partir d’un nuage d’indice CT_PointIterator

Lorsque vous récupérer un nuage d’indices d’une scène (par exemple) et que vous souhaitez parcourir les points il vous faut utiliser un itérateur de lecture.

```
// on récupère le nuage d'indices de la scène
CT_PCIR pcir = scene->getPointCloudIndexRegistered();

// création de l'itérateur
CT_PointIterator it(pcir);

// parcours tant qu'il y a des points et tant que l'utilisateur n'a pas stoppé le traitement
while(it.hasNext() && !isStopped()) {
```

```
        it.next();

        // récupération du point
        const CT_Point &p = it.currentPoint();

        // on peut si l'on veut récupérer son index dans le nuage global
        size_t globalIndex = it.currentGlobalIndex();

        // traitement
        .......
}
```

Parcours du nuage de points global

Si vous voulez accéder directement à un point à partir de son indice global voici l'exemple qui vous montre comment faire :

```
// on crée un objet qui permet d'accéder au nuage de point global
CT_PointAccessor pAccess;

while(...) {

        // traitement
        .....

        // récupération d'un point à partir de son indice global
        CT_Point point = pAccess.pointAt(globalIndex);
}
```

Files

globalpointcloud.png	10.2 KB	02/13/2015	Krebs Michaël
globalpointcloudmodified.png	7.35 KB	02/13/2015	Krebs Michaël
coordinatesystem.png	38.6 KB	02/23/2015	Krebs Michaël

Mettre en place son environnement de développement Computree

Mise en place de l'environnement de développement sous Windows (64 bits, 7, 8 ou 10)

Installation de l'environnement de développement

1. Installer "Microsoft Visual studio 2015 Express": https://www.visualstudio.com/fr/post-download-vs/?sku=xdesk
2. Installer "Windows SDK": https://developer.microsoft.com/fr-fr/windows/downloads/windows-10-sdk (étape optionnelle, permettant de pouvoir compiler en mode debug)
3. Installer "Qt": https://www.qt.io/download-open-source/ (check at least Qt 5.9.1 msvc 2015 64 bits
4. Installer un client SVN, comme par exemple "TortoiseSVN": https://tortoisesvn.net (pendant l'installation, penser à activer l'installation à activer *command line client tools*)

Récupération du code source de Computree et de ses plugins

1. Créer un répertoire racine *Computree* où vous le souhaitez

2. Télécharger le Computree development kit (Windows), qui contient :

- Un fichier *all.pro* avec tous les projets diffusés en standard (cœur + plugins open-sources)
- Un fichier *recuperer_depots.bat*, contenant des instructions svn, pour récupérer le code source
- Un fichier *LISEZ-MOI.txt*, reprenant ces instructions

3. Décompresser *kit_dev_windows_fr.zip* à la racine du répertoire *Computree*

4. Lancer le script *recuperer_depots.bat*

Installation des dépendances

Télécharger et décompresser Computree v5 dependencies folder, et le déplacer à la racine du répertoire computree

Si vous souhaitez installer des dépendances dans des emplacements différents (par exemple dans les c:/program files), vous pouvez, pour chaque fichier *LIBNAME_default_path.pri* du répertoire computreev5, le dupliquer et le renommer *LIBNAME_user_path.pri*. Après cela, vous n'avez qu'à modifier ce deuxième fichier pour utiliser votre chemin local.

Compilation de Computree

1. Lancer Qt Creator

2. Ouvrir le projet *all.pro*, sélectionner le compilateur MSVC 2015 64bits, avec les version release et / ou debug

Si vous n'avez pas installé PCL, supprimez / commentez la ligne suivante dans all.pro :

```
computreev5/library/ctlibpcl/ctlibpcl.pro \
```

3. Dans l'onglet projet désactiver les shadow builds (case à cocher), pour la release et / ou la debug

4. Exécuter qmake sur all.pro, puis compiler le projet

Après une mise à jour du code source, si le cœur de Computree a été modifié significativement, il peut être nécessaire d'exécuter qmake sur chaque sous-projet, puis de faire Recompiler sur all.pro.

Exécution de Computree

Une fois compilé, **pour pouvoir être exécuté, Computree a besoin de toutes les dll des dépendances**, accessibles depuis l'emplacement du

fichier *CompuTreeGui.exe* généré.
Pour cela, copier les dlls dans le répertoire ComputreeInstallRelease (pour la version release) et / ou le répertoire ComputreeInstallDebug (pour la version debug).

Les Dll sont disponibles au téléchargement ici : Computree v5 DLL

Ensuite vous pouvez lancer l'exécution depuis Qt-Creator (flèche verte ou run/exécuter sur all.pro).

Configurer votre plugin si vous souhaitez utiliser PCL dans votre code

Si vous souhaitez utiliser PCL pour vos développements quelques étapes de préparation sont nécessaires :

Vous devez configurer le fichier .pro de votre plugin (.pro) comme suit (début du fichier) :

```
CT_PREFIX = ../../computreev5

include($${CT_PREFIX}/shared.pri)
include($${PLUGIN_SHARED_DIR}/include.pri)

COMPUTREE += ctlibpcl

include($${CT_PREFIX}/include_ct_library.pri)
```

N'oubliez pas de compiler le projet libpcl dans le dossier computreev5/library/ctlibpcl (ouvrez le fichier ctlibpcl.pro et le compiler avec QtCreator)

Il suffit maintenant de faire un *qmake* sur le projet de votre plugin (clic droit → qmake) et de le compiler.

Mise en place de l'environnement de développement sous Ubuntu 16.04 LTS

Installation de l'environnement de développement

1. Installer subversion

Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get update
sudo apt-get install subversion
```

2. Installer Qt (5.9.1)

- Télécharger la dernière version de l'installeur Qt installer : https://www.qt.io/download-open-source/
  - Il faudra créer un compte utilisateur Qt
  - Si vous êtes derrière un proxy, il faudra le paramétrer dans la section paramètres
- Installer Qt

Récupération du code source de Computree et de ses plugins

1. Créer un répertoire racine *Computree* où vous le souhaitez

2. Télécharger le Computree development kit (Linux), qui contient :

- Un fichier *all.pro* avec tous les projets diffusés en standard (cœur + plugins open-sources)
- Un fichier *recuperer_depots.sh*, contenant des instructions svn, pour récupérer le code source
- Un fichier *LISEZ-MOI.txt*, reprenant ces instructions

3. Décompresser *kit_dev_linux.tar.gz* à la racine du répertoire *Computree*

4. *Dans un terminal (CTRL + ALT + T), lancer le script _recuperer_depots.sh*

Installation des dépendances

1. **OpenCV 3.3.0** (optionnel mais fortement conseillé, permet d'utiliser des image / rasters dans Computree)

- Suivre les instructions données sur le site officiel d'OpenCV : http://docs.opencv.org/3.3.0/d7/d9f/tutorial_linux_install.html

2. **PCL 1.8.0** (optionnel, permet d'utiliser les plugins nécessitant PCL)

- Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install git build-essential linux-libc-dev cmake cmake-gui libusb-1.0-0-dev libusb-dev libude
v-devmpi-default-dev openmpi-bin openmpi-common libflann1.8 libflann-dev libeigen3-dev libboost-all-dev li
bvtk5.10-qt4 libvtk5.10 libvtk5-dev libqhull* libgtest-dev freeglut3-dev pkg-config libxmu-dev libxi-dev m
ono-complete qt-sdk openjdk-8-jdk openjdk-8-jre libproj-dev
```

- Télécharger le code source PCL 1.8.0 ici https://github.com/PointCloudLibrary/pcl/archive/pcl-1.8.0.tar.gz
- Décompresser le fichier pcl-1.8.0.tar.gz (dans un explorer: clic droit, extraire ici)
- Dans un terminal (CTRL + ALT + T) :

```
cd pcl-pcl-1.8.0
mkdir build
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=/usr \ -DCMAKE_INSTALL_PREFIX=/usr ..
make -j7
sudo make install
```

3. **GDAL 2.1.2** (optinnel, donne accès aux formats SIG vecteur et raster de GDAL/OGR)

- Télécharger la version 2.1.2 de GDAL, ici : http://download.osgeo.org/gdal/2.1.2/gdal-2.1.2.tar.gz
- Décompresser le fichier gdal-2.1.2.tar.gz (dans un explorateur : clic droit, extraire ici)
- Ouvrir un terminal dans le dossier gdal-2.1.2 (dans un explorateur : séléctionner le dossier, clic droit, ouvrir dans un terminal)
- Lancer les comandes suivantes :

```
./configure
make
sudo make install
sudo ldconfig
```

4. **GSL** (optinnel, donne accès à une librarie de calcul numérique utilisée dans certains plugins)

- Dans un terminal (CTRL + ALT + T) :

```
sudo apt-get install -y gsl-bin libgsl0-dev
```

Si vous souhaitez installer des dépendances dans des emplacements différents, vous pouvez, pour chaque fichier *LIBNAME_default_path.pri*, le dupliquer et le renommer *LIBNAME_user_path.pri*. Après cela, vous n'avez qu'à modifier ce deuxième fichier pour utiliser votre chemin local.

Compilation de Computree

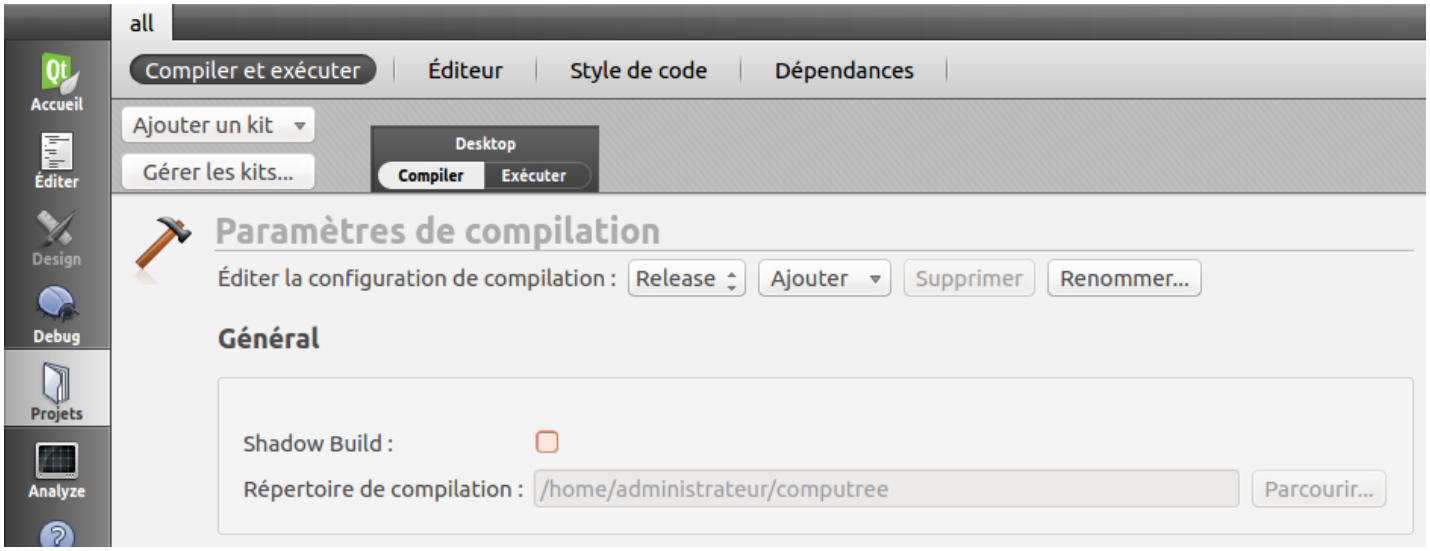
1. Lancer Qt Creator

2. Ouvrir le projet *all.pro*

Si vous n'avez pas installé PCL, supprimez / commentez la ligne suivante dans all.pro :

```
computreev5/library/ctlibpcl/ctlibpcl.pro \
```

3. Dans l'onglet projet désactiver les shadow builds (case à cocher), pour la release et / ou la debug



4. Exécuter qmake sur all.pro, puis compiler le projet

Après une mise à jour du code source, si le cœur de Computree a été modifié significativement, il peut être nécessaire d'exécuter qmake sur chaque sous-projet, puis de faire Recompiler sur all.pro.

Exécution de Computree

Une fois compilé, vous pouvez lancer l'exécution depuis Qt-Creator (flèche verte ou run/exécuter sur all.pro).

Configurer votre plugin si vous souhaitez utiliser PCL dans votre code

Si vous souhaitez utiliser PCL pour vos développements quelques étapes de préparation sont nécessaires :

Vous devez configurer le fichier .pro de votre plugin (.pro) comme suit (début du fichier) :

```
CT_PREFIX = ../../computreev5

include($${CT_PREFIX}/shared.pri)
include($${PLUGIN_SHARED_DIR}/include.pri)

COMPUTREE += ctlibpcl

include($${CT_PREFIX}/include_ct_library.pri)
```

N'oubliez pas de compiler le projet libpcl dans le dossier computreev5/library/ctlibpcl (ouvrez le fichier ctlibpcl.pro et le compiler avec QtCreator)

Il suffit maintenant de faire un *qmake* sur le projet de votre plugin (clic droit → qmake) et de le compiler.

Liste des dépôts svn

Si vous souhaitez ajouter des dépôts, ou faire une installation manuelle sans les scripts, vous trouverez dans le tableau ci-dessous la liste des dépôts pour tous les plugins Computree.

Pour accéder à un dépôt il faut bien sûr avoir les droits adéquats pour le projet considéré.

D'une façon générale le nom d'un dépôt est http://rdinnovation.onf.fr/svn/nom-du-projet . Le nom du projet étant celui qui s'affiche dans la barre d'adresse du navigateur.

Plugin	Code plugin	Projet	Dépôt svn
Computree (base)	CT	computree	http://rdinnovation.onf.fr/svn/computree
ComputreeDevTools	-	computreedevtools	http://rdinnovation.onf.fr/svn/computreedevtools

Plugin Onf	ONF	plugin-onf	http://rdinnovation.onf.fr/svn/plugin-onf
Plugin Arts Free	ARFR	plugin-arts-free	http://rdinnovation.onf.fr/svn/plugin-arts-free
Plugin Onf Lsis	OL	plugin-onf-lsis	http://rdinnovation.onf.fr/svn/plugin-onf-lsis
Plugin Generate	GEN	plugin-generate	http://rdinnovation.onf.fr/svn/plugin-generate
Plugin ToolKit	TK	plugin-toolkit	http://rdinnovation.onf.fr/svn/plugin-toolkit
Plugin LVox	LVOX	plugin-lvox	http://rdinnovation.onf.fr/svn/plugin-lvox

---

Retour à l'accueil

Files

kit_qt_551.png	158 KB	12/20/2016	Piboule Alexandre
shadow_build.png	62.5 KB	12/20/2016	Piboule Alexandre

Les résultats en sortie: nouveau ou copie

Les résultats d'une étape peuvent être complètement nouveaux (ex. on crée une grille à partir de nuages de points) ou alors constituer une copie des intrants à laquelle on ajoute quelque chose (ex. ajouter une grille nb/nt aux grilles de nb, nt, densité, etc.). Ces deux options se répercutent dans les méthodes createInResultModelListProtected, createOutResultModelListProtected et compute.

Créer un nouveau résultat

Dans le cas où on crée un nouveau résultat de toutes pièces, on charge le modèle intrant en tant que CT_InResultModelGroup et on y ajoute les items nécessaires au traitement.

```
// Other includes...
#include "ct_result/model/inModel/ct_inresultmodelgroup.h"
#include "ct_result/model/outModel/ct_outresultmodelgroup.h"

// Alias for indexing models
#define DEF_in_res "in_result"
#define DEF_in_grp "in_group"
#define DEF_in_nb "nb (before)"
#define DEF_in_nt "nt (theoretical)"
#define DEF_out_res "out_result"
#define DEF_out_grp "out_group"
#define DEF_out_item "nbovernt"

// Other methods...

void LVOX_StepNbNtGrids::createInResultModelListProtected()
{
    CT_InResultModelGroup *in_res_model = createNewInResultModel(DEF_in_res, tr("Grilles"));
    resultModel->setZeroOrMoreRootGroup();
    resultModel->addGroupModel("", DEF_in_grp);
    resultModel->addItemModel(DEF_in_grp, DEF_in_nb, CT_Grid3D<int>::staticGetType(),
    tr("nb (before)"));
    resultModel->addItemModel(DEF_in_grp, DEF_in_nt, CT_Grid3D<int>::staticGetType(),
    tr("nt (theoretical)"));
}
```

On crée le modèle sortant en tant que CT_OutResultModelGroup et, puisqu'il s'agit d'un nouveau résultat, on doit y ajouter un groupe avant d'ajouter les items que produira l'étape.

```
void LVOX_StepNbNtGrids::createOutResultModelListProtected()
{
    CT_OutResultModelGroup *out_res = createNewOutResultModel(DEF_out_res, tr("nb/nt"));
    if (res != NULL)
    {
        res->setRootGroup(DEF_out_grp, new CT_StandardItemGroup(), tr("grille"));
        res->addItemModel(DEF_out_grp, DEF_out_item, new CT_Grid3D<float>(), tr("nb/nt"));
    }
}
```

Lors du traitement, on doit récupérer les résultats en entrée et en sortie. Pour créer les nouvelles structures de données, on utilise la chaîne de caractère représentée par DEF_out_item et le pointeur du résultat où les données seront placées.

```
void LVOX_StepNbNtGrids::compute()
{
    // Get the input
    CT_ResultGroup* in_res = getInputResults().first();

    // Get the out res and group
    CT_ResultGroup* out_res = getOutResultList().first();
    CT_StandardItemGroup* out_group = new CT_StandardItemGroup(DEF_out_grp, out_res);

    // Iterate
    CT_ResultGroupIterator it(in_res, this, DEF_in_grp);
    while (it.hasNext() && !isStopped())
    {
```

```
        // Get necessary input items
        CT_StandardItemGroup* in_group = (CT_StandardItemGroup*) it.next();
        const CT_Grid3D<int>* nb = (CT_Grid3D<int>*) in_group->firstItemByINModelName(this, DEF_in_nb);
        const CT_Grid3D<int>* nt = (CT_Grid3D<int>*) in_group->firstItemByINModelName(this, DEF_in_nt);

        // Create the new data structure
        CT_Grid3D<float>* nbnt = new CT_Grid3D<float>(DEF_out_item, out_res,
            nb->minX(), nb->minY(), nb->minZ(),
            nb->xdim(), nb->ydim(), nb->zdim(),
            nb->resolution(), nb->NA(), nb->NA());

        // Treatment...
        // ...

        // Add item to group to result
        out_group->addItemDrawable(nbnt);
        out_res->addGroup(out_group);
        nbnt->computeMinMax();
    }
}
```

Copier le résultat intrant

En revanche, si on veut ajouter au résultant intrant, quelques modifications doivent être faites au .h. Une variable privée de type CT_AutoRenameModels doit être ajoutée, ce qui requiert l'inclusion de l'entête correspondant. CT_AutoRenameModels est une classe propre à Computree qui garantit l'unicité du nom du modèle.

```
#include "ct_tools/model/ct_autorenamemodels.h"

// ...

class LVOX_StepNbNtGrids: public CT_AbstractStep
{
    Q_OBJECT
public:
    // ...

private:
    // ...

    CT_AutoRenameModels _nbnt_ModelName;
};
```

Le résultat en entrée étant destiné à une copie, on doit le charger en tant que CT_InResultModelGroupToCopy à l'aide de la méthode createNewInResultModelForCopy.

```
// Other includes...
#include "ct_result/model/inModel/ct_inresultmodelgrouptocopy.h"
#include "ct_result/model/outModel/tools/ct_outresultmodelgrouptocopypossibilities.h"

// Alias for indexing models
#define DEF_in_res "result"
#define DEF_in_grp "group"
#define DEF_in_nb "nb (before)"
#define DEF_in_nt "nt (theoretical)"

void LVOX_StepNbNtGrids::createInResultModelListProtected()
{
    CT_InResultModelGroupToCopy* in_res = createNewInResultModelForCopy(DEF_in_res, tr("Grilles"));

    in_res->setZeroOrMoreRootGroup();
    in_res->addGroupModel("", DEF_in_grp);
    in_res->addItemModel(DEF_in_grp, DEF_in_nb, CT_Grid3D<int>::staticGetType(),    tr("nb (before)"));
    in_res->addItemModel(DEF_in_grp, DEF_in_nt, CT_Grid3D<int>::staticGetType(),    tr("nt (theo)"));
}
```

On crée le modèle sortant en tant que CT_OutResultModelGroupToCopyPossibilities à l'aide de la méthode createNewOutResultModelToCopy. On remarque que la chaîne de caractères à utiliser est celle du résultat entrant, que l'on copie. Puisque ce résultat est déjà bien formé, ajouter un groupe racine n'est pas nécessaire. L'item à ajouter est ensuite spécifié à l'aide de la variable de type CT_AutoRename.

```
void LVOX_StepNbNtGrids::createOutResultModelListProtected()
{
    CT_OutResultModelGroupToCopyPossibilities* out_res = createNewOutResultModelToCopy(DEF_in_res);
    if (out_res != NULL)
    {
        out_res->addItemModel(DEF_in_grp, _nbnt_ModelName, new CT_Grid3D<float>(),tr("ndovernt"));
    }
}
```

Au cours du traitement de données, on récupère d'abord les résultats en sortie, qui contiennent une copie des résultats en entrée. Pour créer les nouvelles données, une chaîne de caractères est nécessaire (représentée par le #define dans l'exemple "Créer un nouveau résultat"); on utilise donc la méthode CT_AutoRenameModels::completeName.

```
void LVOX_StepNbNtGrids::compute()
{
    // Get the output res, which is a copy of input result
    CT_ResultGroup* out = getOutResultList().first();

    // Iterate
    CT_ResultGroupIterator it(out, this, DEF_in_grp);
    while (it.hasNext() && !isStopped())
    {
         // Get the input group copy
        CT_StandardItemGroup* group = (CT_StandardItemGroup*) it.next();
         // Get inputs for treatment...
         // ...

         // Create the new data structure
        CT_Grid3D<float>* nbnt = new CT_Grid3D<float>(_nbnt_ModelName.completeName(), out,
            nb->minX(), nb->minY(), nb->minZ(),
            nb->xdim(), nb->ydim(), nb->zdim(),
            nb->resolution(), nb->NA(), nb->NA());

         // Treatment...
         // ...

         // Add the new data structure to the input group copy
        group->addItemDrawable(nbnt);
    }
}
```

ComputreeDevTools