Présentation de GLC_lib

par Laurent Ribon (http://www.glc-lib.net)

Date de publication : 04 mars 2011

Dernière mise à jour :

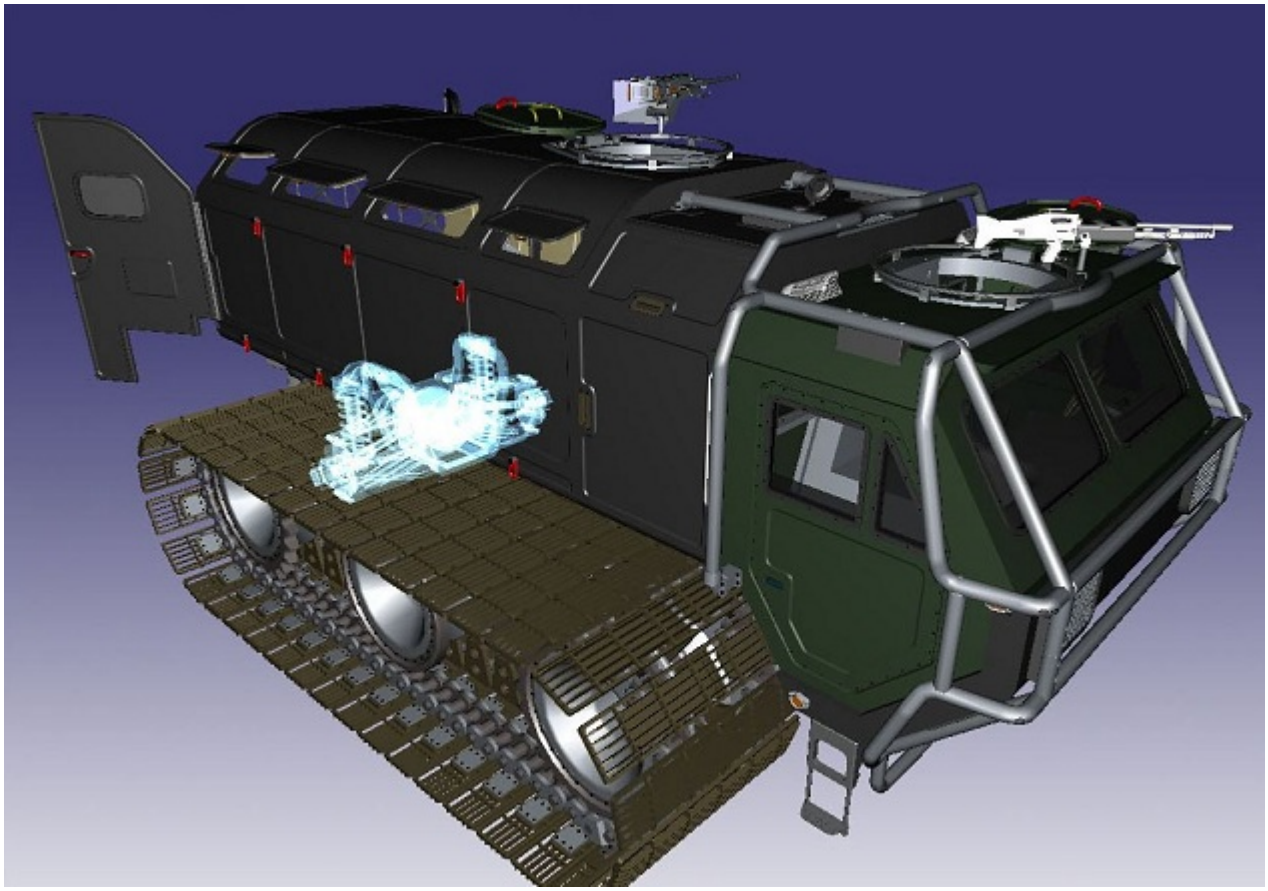
Ce tutoriel fournit une présentation de la bibliothèque GLC_lib, puis un exemple de programmation d'une application de visualisation 3D OpenGL utilisant GLC_lib.

I - Présentation de GLC_lib

I-A - Présentation générale

GLC_lib est une bibliothèque utilisant OpenGL et Qt4. Elle permet de créer et de visualiser en temps réel des scènes 3D. On peut donc dire que GLC_lib est un moteur 3D.

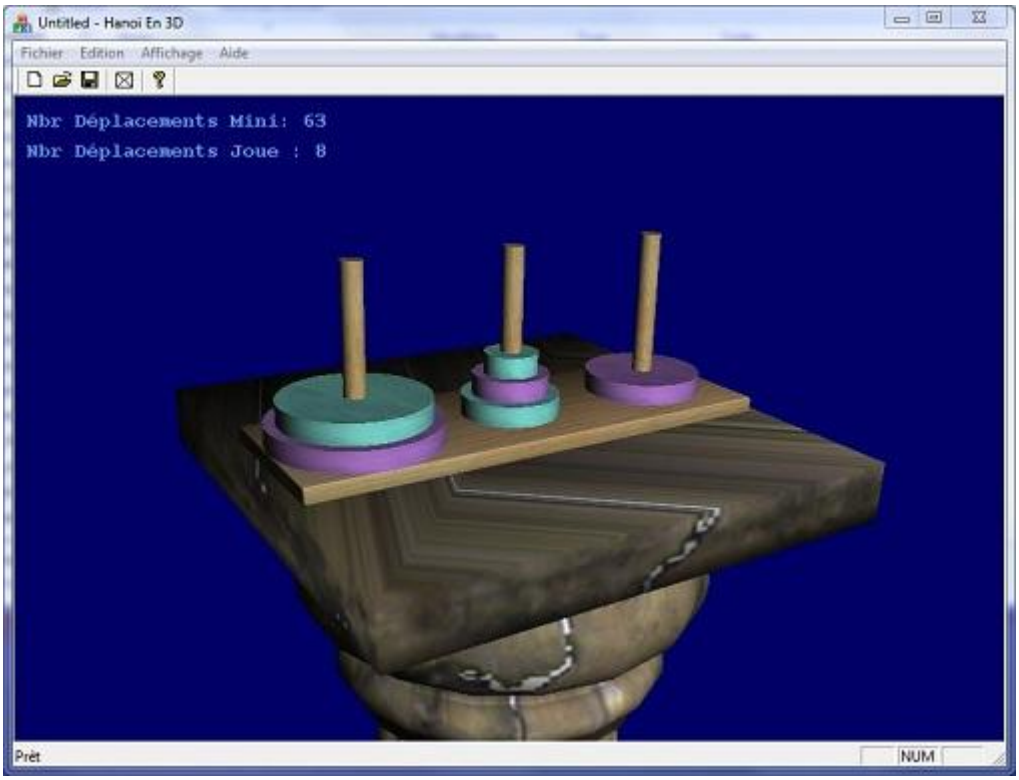
GLC_lib permet d'afficher en temps réel des scènes 3D complexes (100 000 pièces, 40 000 000 de triangles).



Exemple d'une scène contenant 241 pièces et 849 949 triangles

I-B - Bref historique

Le développement de GLC_lib a commencé en 2003 dans le but de créer des applications OpenGL. La première version de cette bibliothèque utilisait les MFC.



Les tours de Hanoï en 3D utilisant la première version de GLC_lib

En 2005, suite à la publication de Qt4 pour Windows sous licence GPL, le code de GLC_lib a été porté sous Qt4. Depuis, depuis la version 2.0, GLC_lib est sous licence LGPL.

I-C - Caractéristiques

GLC_lib permet la construction et le rendu d'une scène tridimensionnelle composée de centaines de milliers de maillages.

Pour le moment, GLC_lib n'est pas adaptée pour la représentation d'objets animés.

I-C-1 - Visualisation d'un maillage

GLC_lib permet d'afficher des géométries composées de lignes, de triangles, de triangle strips et de triangle fans. Toutes ces géométries peuvent être affichées en utilisant les *vertex array* compatibles OpenGL 1.0 ou les Vertex Buffer Object à partir d'OpenGL 1.4.

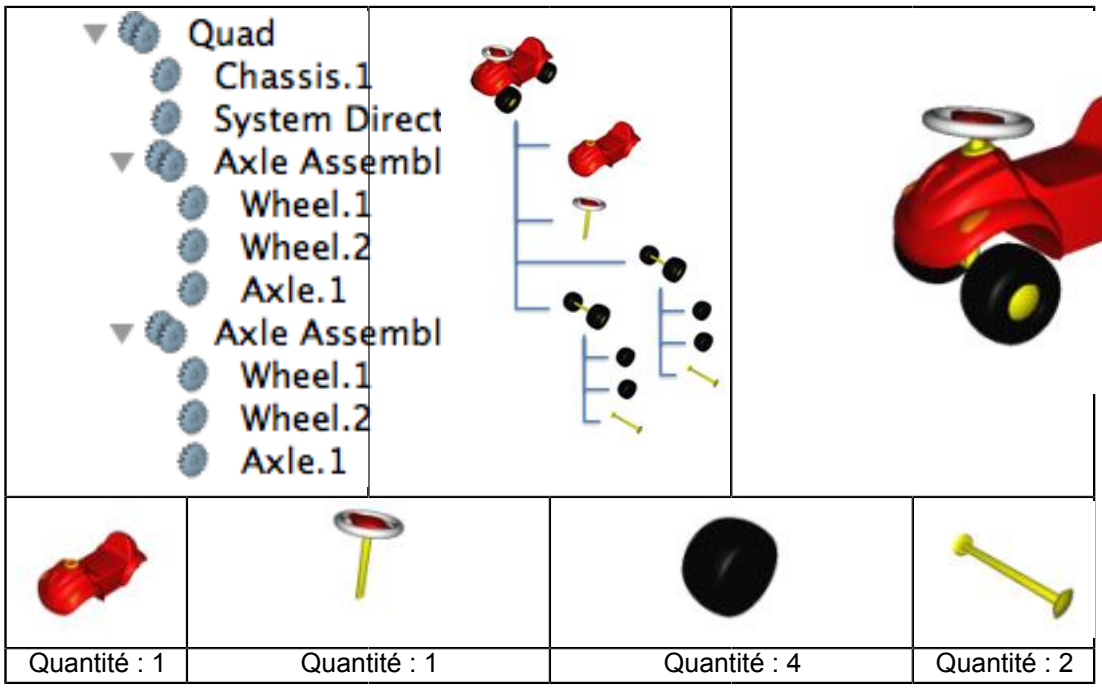
La structure de maillage de GLC_lib permet de représenter un objet solide ainsi que ses arrêtes. Voici la liste de ses possibilités :

- affichage de polylignes ;
- affichage de triangles, triangle strips et triangle fans ;
- utilisation d'un nombre quelconque de matériaux opaques ou translucides ;
- possibilité de rendu en couleurs indexées (une couleur par sommet) ;
- affichage en utilisant les *vertex array* ou les VBO ;
- gestion d'un nombre quelconque de niveaux de détails ;
- possibilité de définir des groupes afin de les sélectionner ;
- possibilité de surcharger les propriétés de rendu.

Exemple des possibilités de rendu de GLC_lib

I-C-2 - Structure hiérarchique d'une scène

Pour représenter la structure hiérarchique d'une scène, GLC_lib utilise « un graphe non orienté acyclique connexe » ou plus simplement un arbre.



La structure d'assemblage d'un quad (exemple fourni avec 3DXML Player)

Pour éviter de gaspiller de la mémoire, les géométries ainsi que les sous-ensembles qui apparaissent plusieurs fois sont instanciés. La définition d'un sous-ensemble ou d'une géométrie est appelée référence. La structure d'un assemblage est composée d'occurrences, d'instances et de références. Suivant cette architecture, le quad est composé des éléments suivants :

- quatre références de géométries : châssis, volant, roue et essieu ;
- deux références d'assemblages : assemblage du quad complet et assemblage essieu ;
- cinq instances de géométries : châssis.1, volant.1, roue.1, roue.2 et essieu.1 ;
- trois instances d'assemblage : quad, (essieu assemblé).1 et (essieu assemblé).2 ;
- huit occurrences de géométries ;
- trois occurrences d'assemblages.

I-C-3 - Optimisations

Afin de charger et de visualiser des scènes complexes le plus rapidement possible, GLC_lib utilise des techniques d'optimisation connues et éprouvées.

Les optimisations utilisées sont axées sur l'affichage d'une scène composée d'un grand nombre de géométries et non sur l'affichage d'une seule géométrie.

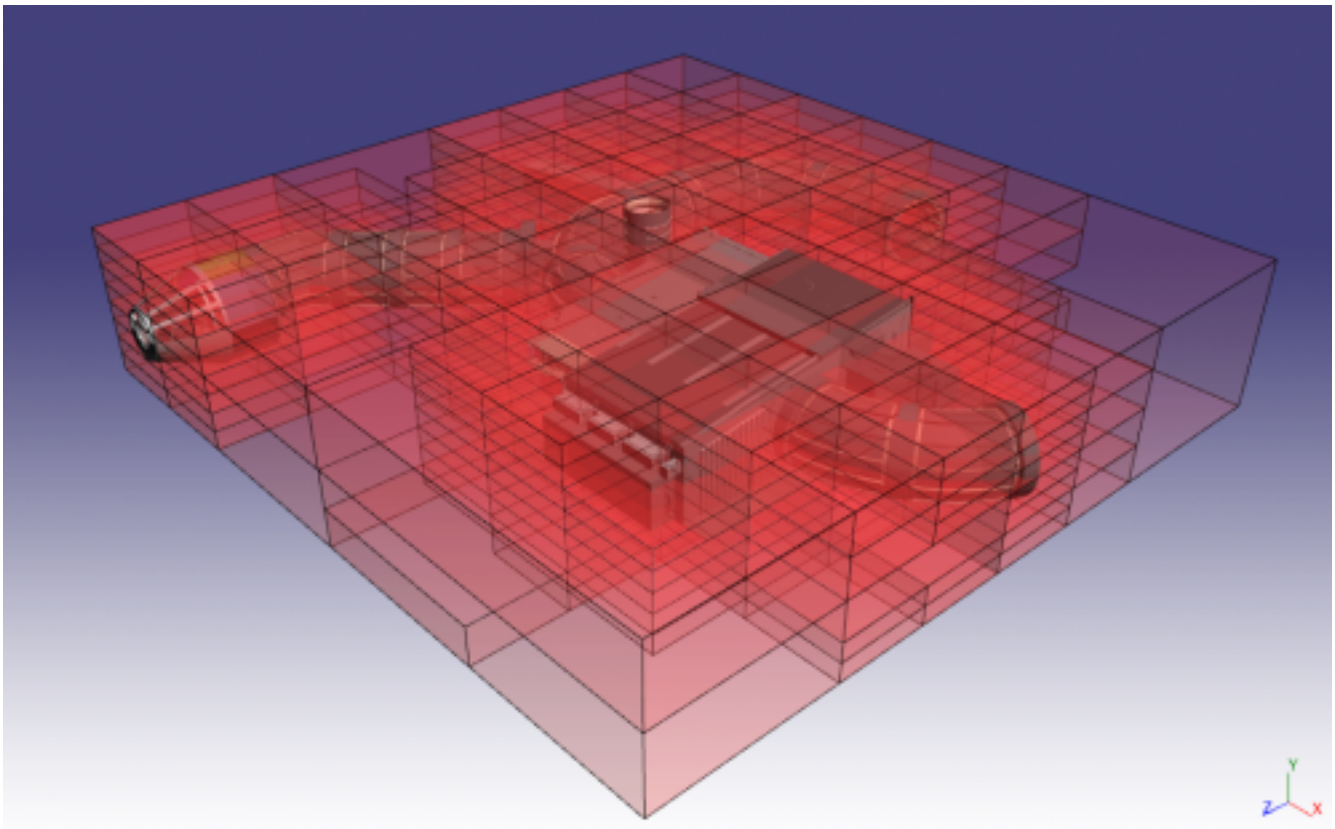
I-C-3-1 - Chargement

Uniquement pour le format 3DXML de Dassault Systèmes, GLC_lib utilise une technique de cache qui consiste à créer un cache au format binaire des géométries à afficher. Ainsi, pour les chargements ultérieurs, le cache est utilisé et permet de diviser le temps de chargement d'un facteur compris entre cinq et dix.

I-C-3-2 - Visualisation

Pour permettre l'affichage d'une scène complexe, le moteur de rendu de GLC_lib utilise les trois techniques suivantes :

- exclusion de pixel : élimination des objets dont la taille projetée est inférieure à un seuil exprimé en pixels ;
- niveaux de détails : remplacement par des objets ayant moins de triangles ;
- élimination des objet hors champ : élimination des objets situé en dehors du champ de vision de la caméra.



Partitionnement par Octree utilisé pour accélérer l'élimination des objets hors champ

I-D - Formats supportés

GLC_lib supporte les six formats suivants :

- Collada V1.4 (lecture seulement) ;
- 3DXML (XML) V3 and V4 (lecture et écriture en 3DXML V4) ;
- OBJ (lecture seulement) ;
- 3DS (lecture seulement) ;
- STL (ASCII et binaire) (lecture seulement) ;
- OFF et COFF (lecture seulement).

Le format de prédilection de GLC_lib est le 3DXML de Dassault Systèmes.

Ce format a été créé par Dassault Systèmes pour concurrencer les formats X3D et U3D.

Le 3DXML est décliné en trois variantes:

- exact : la géométrie est décrite en mode exact (B-Rep) et est encodée en binaire;
- maillage : la géométrie est décrite en mode maillage et est encodée en ASCII;
- maillage compressé : la géométrie est décrite en mode maillage et est encodée en binaire.

La variante encodée en ASCII a été rendue publique le 15 juin 2005. Cependant, il faut faire une demande auprès de Dassault Systèmes pour accéder à la spécification du 3DXML.

Le 3DXML est un conteneur compressé au format « ZIP » qui contient principalement un fichier XML décrivant la structure du produit et un fichier XML par pièce avec niveaux de détail.

Tous les logiciels de Dassault Systèmes permettent de lire ou d'exporter en 3DXML. Pour étendre l'utilisation de 3DXML, Dassault fournit gratuitement les logiciels 3DVIA et 3DXML Player ainsi qu'un site de partage de modèles 3D au format 3DXML et COLLADA.

II - Compilation et installation de GLC_lib

II-A - Présentation et prérequis

GLC_lib utilise l'API OpenGL et est très fortement couplée au framework Qt4 de Nokia. Pour compiler GLC_lib, il faut donc disposer d'une machine supportant l'OpenGL sur laquelle est installé Qt4. Comme GLC_lib n'a pas d'autre dépendance, ce sont les seuls prérequis.

À moins que vous souhaitiez modifier le code de GLC_lib, je recommande de ne pas utiliser d'EDI pour sa compilation et son installation. Pour la compilation d'une bibliothèque, il est plus complexe de paramétrer un EDI que de saisir trois lignes de commandes dans un interpréteur de commande.

Avant la compilation, il faut s'assurer que Qt4 est correctement installé en compilant un des exemples de Qt4 utilisant OpenGL (« Hello GL » me paraît être un bon candidat).

Ensuite, il faut **télécharger** et extraire les sources de GLC_lib.

Ne pas extraire le contenu de l'archive dans un dossier dont le chemin comporte des espaces ou des caractère spéciaux, beaucoup de compilateur ne le supportent pas.

II-B - Les plateformes compatibles

Théoriquement, GLC_lib peut être compilée sur toute les plateformes supportées par Qt4 prenant en charge OpenGL.

Voici la liste des plateformes testées :

- Windows Vista et Windows 7 en 32 bits avec MinGW32 ;
- Windows 7 en 64 bits avec MSVC 2008 ;
- Mac OS X 10.5, 10.6 ;
- Linux Ubuntu et Fedora en 32 et 64 bits.

Pour les plateformes Win32, il existe une version déjà compilée sur la page de **téléchargement** de GLC_lib.

Sur les plateformes Linux, il existe des paquets pour les architectures 32 bits et 64 bits sur le site : **pkgs.org**

II-C - Compilation et installation sous Windows

Si vous avez installé le SDK de Qt : lancer l'interpréteur de commandes de Windows via le raccourci du SDK de Qt. Si vous utilisez MSVC++, lancer l'interpréteur de commande de Windows via le raccourci de MSVC++.

L'exemple est donné pour la version 2.1.0 de GLC_lib.

Allez dans le répertoire contenant les sources de GLC_lib :

```
C:\Qt\2010.05\qt> cd \
C:\>cd compilation\GLC_lib_src_2.1.0\glc_lib
```

Tapez la ligne de commande suivante pour générer le « *makefile* » :

```
C:\compilation\GLC_lib_src_2.1.0\glc_lib> qmake
```

II-C-1 - MinGW

Tapez la ligne de commande suivante pour compiler GLC_lib :

```
C:\compilation\GLC_lib_src_2.1.0\glc_lib>  mingw32-make
```

Tapez la ligne de commande suivante pour installer GLC_lib :

```
C:\compilation\GLC_lib_src_2.1.0\glc_lib>  mingw32-make install
```

II-C-2 - Visual C++

Tapez la ligne de commande suivante pour compiler GLC_lib :

```
C:\compilation\GLC_lib_src_2.1.0\glc_lib> nmake
```

Tapez la ligne de commande suivante pour installer GLC_lib :

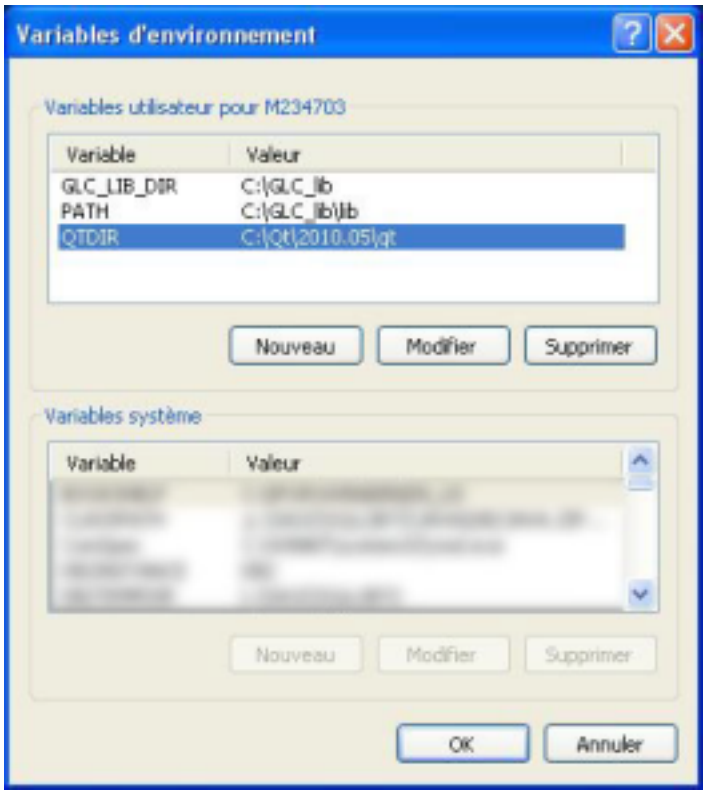
```
C:\compilation\GLC_lib_src_2.1.0\glc_lib>  nmake install
```

II-C-3 - Fin d'installation

Par défaut, GLC_lib est installée dans le dossier C:\GLC_lib qui contient :

- un dossier « lib » pour la DLL et le fichier d'importation des symboles (.lib) ;
- un dossier « include » pour les en-têtes.

Pour compléter l'installation, il reste à ajouter le dossier C:\GLC_lib\lib dans le PATH et à définir la variable « GLC_LIB_DIR » avec la valeur : C:\GLC_lib.



Vous pouvez maintenant utiliser la bibliothèque dans vos programmes.

II-D - Compilation et installation sous Unix (Linux, BSD, Mac OS X, etc.)

Pour pouvoir utiliser les outils de développement sous Linux, vous devez installer les packages nécessaires pour le développement d'applications Qt4.

Après l'installation de ces packages, lancez une fenêtre terminal et allez dans le répertoire contenant les sources de GLC_lib :

```
cd   /Users/laumaya/GLC_lib_src_2.1.0/glc_lib
```

Sous les autres systèmes que Mac OS X : tapez la ligne de commande suivante pour générer le « *makefile* » :

```
qmake
```

Sous Mac OS X, tapez la ligne de commande suivante pour générer le « *makefile*» :

```
qmake -spec macx-g++
```

Tapez la ligne de commande suivante pour compiler GLC_lib :

```
make
```

Tapez la ligne de commande suivante pour installer GLC_lib et tapez votre mot de passe

```
sudo make install
```

La bibliothèque est installée dans /usr/local/lib

Les fichiers d'en-têtes sont installés dans /usr/local/include/GLC_lib

III - Documentation de GLC_lib

Je suis conscient que le gros point faible de GLC_lib est sa documentation peut fournie.

C'est d'ailleurs l'une des raisons qui m'a poussé à écrire cet article.

Cependant, il existe plusieurs sources de documentation de GLC_lib qui sont principalement en anglais.

http://www.glc-lib.net/examples.php

III-A - Aide sur le site

http://www.glc-lib.net/help.php

Sur le site Web de GLC_lib, vous pouvez trouver :

- une aide concise sur la compilation et l'installation de GLC_lib ;
- la documentation de référence ;
- des exemples d'utilisation dans l'ordre croissant de leur complexité.

III-B - Documentation de référence

http://www.glc-lib.net/doc/index.html

Cette documentation a été générée par Doxygen.

III-C - Forum

Si vous avez besoin de plus d'aide sur GLC_lib, vous pouvez utiliser les forums dédiés à GLC_lib.

Version en français :

http://www.developpez.net/forums/f1553/c-cpp/bibliotheques/qt/outils/bibliotheques/glc_lib/

Version en anglais :

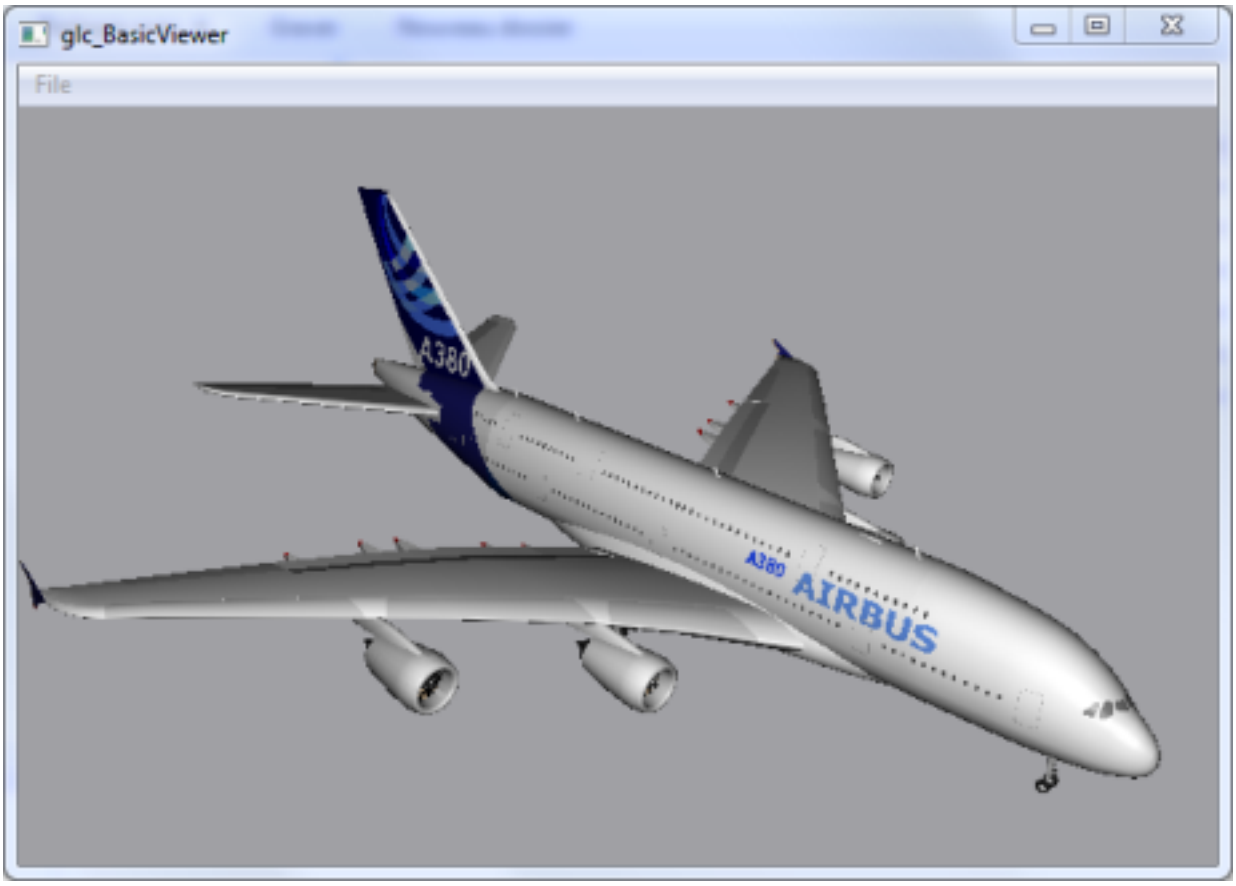
http://www.glc-lib.net/forum/

IV - Exemple d'utilisation de GLC_lib

Pour cet exemple, nous allons créer un logiciel permettant de visualiser tous les formats 3D supportés par GLC_lib.

Code source de l'exemple : **glc_BasicViewer.zip.**

Logiciel de visualisation GLC_BasicViewer (Mac)



Logiciel de visualisation GLC_BasicViewer (Windows)

IV-A - Création du projet (le fichier .pro)

Code 1 : Lignes à ajouter dans le fichier projet d'une application utilisant GLC_lib

```
win32 {
    LIBS += -L"$$(GLC_LIB_DIR)/lib" -lGLC_lib2
    INCLUDEPATH += "$$(GLC_LIB_DIR)/include"
}

unix {
     LIBS += -lGLC_lib
     INCLUDEPATH += "/usr/include/GLC_lib"
}
```

Pour utiliser GLC_lib, il faut lier le projet à la bibliothèque.

Sous Windows :

```
LIBS += -L"$$(GLC_LIB_DIR)/lib" -lGLC_lib2
```

GLC_LIB_DIR est la variable d'environnement définissant le chemin d'installation de GLC_lib.

Sous Unix (Linux et Mac) :

```
LIBS += -lGLC_lib
```

Comme GLC_lib est installée par défaut dans /usr/local/lib, il n'est pas nécessaire de spécifier son emplacement.

Ensuite, il faut définir l'emplacement des en-têtes de GLC_lib:

Sous Windows :

```
INCLUDEPATH += "$$(GLC_LIB_DIR)/include"
```

Sous Unix (Linux et Mac) :

```
INCLUDEPATH += "/usr/local/include/GLC_lib"
```

Code 2 : le fichier projet de l'exemple

```
TEMPLATE = app
QT += opengl

win32 {
    LIBS += -L"$$(GLC_LIB_DIR)/lib" -lGLC_lib2
    INCLUDEPATH += "$$(GLC_LIB_DIR)/include"
}

unix {
     LIBS += -lGLC_lib
     INCLUDEPATH += "/usr/include/GLC_lib"
}

# Input
HEADERS += glwidget.h mainwindow.h
SOURCES += glwidget.cpp main.cpp mainwindow.cpp
```

IV-B - Description de l'application

Pour cet exemple, nous allons créer une application composée d'une fenêtre principale héritant de QMainWindow ayant pour Widget central une classe héritant de QGLWidget.

IV-B-1 - Code source de la classe MainWindow

La classe MainWindow contient uniquement deux méthodes :

- son constructeur ;
- un SLOT déclenché lors de l'ouverture d'un fichier.

IV-B-1-1 - Le constructeur de la classe MainWindow

Code 3 : MainWindow::MainWindow()

```
MainWindow::MainWindow()
: QMainWindow()
, m_GLWidget(this) // La vue Opengl (QGLWidget)
, m_pOpenFileAction(new QAction(tr("Open"), this))
, m_CurrentPath(QDir::homePath())
{
 setCentralWidget(&m_GLWidget);
 // Création du menu
 QMenu* pMenu= new QMenu(tr("File"), this);
 pMenu->addAction(m_pOpenFileAction);
 this->menuBar()->addMenu(pMenu);

 // Connexion de l'action d'ouverture de fichier au SLOT void open()
 connect(m_pOpenFileAction, SIGNAL(triggered()), this, SLOT(open()));
}
```

Pour le constructeur de cette classe, il n'y a pas de code relatif à GLC_lib.

IV-B-1-2 - La méthode d'ouverture de fichier

Code 4 : void MainWindow::open()

```
void MainWindow::open()
{
 // Liste des formats de fichiers ?? filtrer
 QStringList filters;
 ....

 const QString message(tr("Select File(s) to Open and Add in album"));
 QString fileName =
          QFileDialog::getOpenFileName(this, message, m_CurrentPath, filters.join("\n"));
 if (!fileName.isEmpty())
 {
  // Affichage du sablier
  QApplication::setOverrideCursor(QCursor(Qt::WaitCursor));

  // Modification du répertoire courant
  m_CurrentPath= QFileInfo(fileName).absolutePath();

  QFile file(fileName);

  // Construction de la scène 3D à partir du fichier
  GLC_World world= GLC_Factory::instance()->createWorldFromFile(file);

  if (!world.isEmpty())
  {
   m_GLWidget.setWorld(world);
  }

  // Restauration du curseur
  QApplication::restoreOverrideCursor();
 }
}
```

Nous avons deux actions importantes dans ce code

- Lecture du fichier et création de la scène 3D :

```
// Construction de la scène 3D à partir du fichier
GLC_World world= GLC_Factory::instance()->createWorldFromFile(file);
```

On accède à l'unique instance de la classe GLC_Factory par sa méthode statique instance(), ensuite la « Factory » s'occupe de construire la classe de lecture appropriée pour le fichier passé en paramètre. Le type de fichier est déterminé par son extension. Si tout se passe bien, la scène 3D correspondant au fichier est renvoyée.

Pour simplifier cet exemple, les exceptions susceptibles d'être levées lors de la lecture du fichier ne sont pas interceptées.

- Passage de la scène à la vue :

```
m_GLWidget.setWorld(world);
```

La classe GLC_World partage ses données de façon implicite. Sa copie est donc très rapide et la destruction de sa dernière instance libérera la mémoire allouée comme pour un pointeur intelligent.

IV-B-2 - Code source de la classe GLWidget

Cette classe est le coeur de notre application.

IV-B-2-1 - Le constructeur de la classe GLWidget

Code 5 : GLWidget::GLWidget(QWidget *p_parent)

```
GLWidget::GLWidget(QWidget *p_parent)
: QGLWidget(p_parent)
, m_Light()     // Lumière (Omnidirectionnelle) : GLC_Light
, m_World()     // La scène 3D : GLC_World
, m_GlView(this)  // La vue OpenGL GLC_Viewport
, m_MoverController() // Le contrôleur des interactions liés au point de vue GLC_MoverController
{
 // Définition de la position de la lumière
 m_Light.setPosition(1.0, 1.0, 1.0);

 // Création du contrôleur des interactions par défaut
 QColor repColor;
 repColor.setRgbF(1.0, 0.11372, 0.11372, 1.0); // Couleur des éléments
 m_MoverController= GLC_Factory::instance()->createDefaultMoverController(repColor, &m_GlView);
}
```

Le constructeur se charge de créer et d'initialiser quatre objets de GLC_lib. Parmi ces objets, le plus intéressant est celui de la classe GLC_MoverController. Cet objet permet de gérer les interactions utilisateur via des évènements de la souris. La classe GLC_Factory construit un contrôleur de navigation par défaut.

IV-B-2-2 - L'initialisation OpenGL

Code 6 : void GLWidget::initializeGL()

```
void GLWidget::initializeGL()
{
 m_GlView.initGl();
 m_GlView.setBackgroundColor(Qt::gray);

 GLC_State::setVboUsage(false);
}
```

Lors de l'initialisation OpenGL, la première chose à faire est d'appeler la méthode : GLC_Viewport::initGl(). Cette méthode effectue les actions suivantes :

- initialisation des états d'OpenGL requis par GLC_lib ;
- chargement des extensions OpenGL ;
- initialisation des états de GLC_lib.

Ensuite, on définit la couleur de l'arrière-plan et on désactive l'extension OpenGL « Vertex Buffer Object » qui peut poser des problèmes sur certains types de matériel.

IV-B-2-3 - Le redimensionnement de la fenêtre OpenGL

Code 7 : void GLWidget::resizeGL(int width, int height)

```
void GLWidget::resizeGL(int width, int height)
{
 m_GlView.setWinGLSize(width, height);
}
```

La méthode GLC_Viewport::setWinGLSize(int, int) permet de mettre à jour les états OpenGL liés à la taille de la fenêtre OpenGL.

IV-B-2-4 - Assignation de la scène 3D à afficher

Code 8 : void GLWidget::setWorld(const GLC_World& world)

```
void GLWidget::setWorld(const GLC_World& world)
```

Code 8 : void GLWidget::setWorld(const GLC_World& world)

```
{
 m_World= world;
 if (!m_World.boundingBox().isEmpty())
 {
  m_GlView.reframe(m_World.boundingBox()); // Recadrage de la vue sur la scène
  updateGL();                              // Mise à jour de la vue
 }
}
```

Après l'assignation de la scène 3D à afficher, on recadre la vue sur celle-ci, puis on rafraîchit la vue.

IV-B-2-5 - Dessin de la vue OpenGL

Code 9 : void GLWidget::paintGL()

```
void GLWidget::paintGL()
{
 // Effacement de la vue
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

 // Chargement de ma matrice identité
 glLoadIdentity();

 // Calcul de la profondeur de champ de la scéne
 m_GlView.setDistMinAndMax(m_World.boundingBox());

 // Eclairage de la scène
 m_Light.enable();
 m_Light.glExecute();

 // Modification de la matrice de visulisation en fonction
 // de la position de la caméra
 m_GlView.glExecuteCam();

 // Rendu de la scène
 m_World.render(0, glc::ShadingFlag);
 m_World.render(0, glc::TransparentRenderFlag);

 // Rendu du manipulateur de navigation courant
 m_MoverController.drawActiveMoverRep();
}
```

On remarquera que la scène est rendue en deux passes. La première pour les faces opaques et la deuxième pour les faces translucides.

IV-B-2-6 - Réaction à l'appui sur un bouton de la souris

Code 10 : void GLWidget::mousePressEvent(QMouseEvent *e)

```
void GLWidget::mousePressEvent(QMouseEvent *e)
{
 if (m_MoverController.hasActiveMover()) return;
 switch (e->button())
 {
 case (Qt::RightButton):
  m_MoverController.setActiveMover(GLC_MoverController::TrackBall, e);
  updateGL();
  break;
 case (Qt::LeftButton):
  m_MoverController.setActiveMover(GLC_MoverController::Pan, e);
  updateGL();
  break;
 case (Qt::MidButton):
  m_MoverController.setActiveMover(GLC_MoverController::Zoom, e);
  updateGL();
  break;
```

Code 10 : void GLWidget::mousePressEvent(QMouseEvent *e)

```
  default:
   break;
  }
 }
```

L'appui sur un des trois boutons de la souris permet de changer de mode de navigation courant :

- bouton droit : rotation ;
- bouton du milieu : déplacement panoramique ;
- bouton gauche : zoom .

IV-B-2-7 - Réaction au déplacement de la souris

Code 11 : void GLWidget::mouseMoveEvent(QMouseEvent * e)

```
void GLWidget::mouseMoveEvent(QMouseEvent * e)
{
 if (m_MoverController.hasActiveMover())
 {
  m_MoverController.move(e);
  updateGL();
 }
}
```

Grâce à la classe GLC_MoverController, la gestion des déplacements de la caméra se révèle très simple.

IV-B-2-8 - Réaction au relâchement d'un bouton de la souris

Code 12 : void GLWidget::mouseReleaseEvent(QMouseEvent*)

```
void GLWidget::mouseReleaseEvent(QMouseEvent*)
{
 if (m_MoverController.hasActiveMover())
 {
  m_MoverController.setNoMover();
  updateGL();
 }
}
```

Voici la dernière méthode de notre classe. Le relâchement d'un bouton de la souris enlève le mode de navigation courant.

V - Conclusion

J'espère que ce tutoriel vous aura donné envie d'utiliser GLC_lib.

VI - Remerciements

Je tiens à remercier **Jonathan Coutois** pour ses conseils.

Je remercie tout particulièrement **Thibaut Cuvelier** pour m'avoir proposé de rédiger cet article ainsi que pour son aide précieuse.

Sans oublier **Maxime Gault** pour la qualité de ses propositions de correction.