OpenGL dans Qt 5.1

Guillaume Belz

1 OpenGL dans Qt5

Début est rappel d'un mon article publié le 29 mai 2012.

Le support d'OpenGL dans Qt5 a été modifié pour mieux l'intégrer avec les nouveaux modules de Qt : QtQuick2 et Qt3D. Cet article présente les modifications apportées dans Qt5.

1.1 OpenGL dans Qt4

Dans Qt4, les fonctionnalités d'OpenGL sont implémentées dans un module spécifique, QtOpenGL. Ce module était utilisé par différentes classes pour bénéficier de l'accélération matérielle. Il existe plusieurs méthodes pour activer l'accélération matérielle :

- pour activer par défaut l'utilisation de OpenGL, utiliser la ligne de commande « -graphicssystem opengl » ou la fonction QApplication::setGraphicsSystem(« opengl ») ;
- pour QGraphicsView (QtGraphics) ou QDeclarativeView (QtQuick), utiliser la fonction setViewport(new QGLWidget) ;
- on peut également utiliser QPainter directement sur un QGLWidget ;
- avec le QML Viewer, utilisez la commande « -opengl ».

1.2 L'organisation des modules

Dans Qt4, on a donc un problème dans l'organisation des modules. Beaucoup de classes de QtGui peuvent dépendre des classes appartenant à QtOpenGL alors que ce module dépend normalement de QtGui.

C'est pour cela que de nouvelles classes font leur apparition dans Qt5 et que les différents modules ont été réorganisés :

- les classes QOpenGL appartiennent au module QtGui et fournissent les fonctionnalités de base permettant l'accélération matérielle pour toute les classes de rendu de Qt ;
- la classe QWidget n'est plus le parent de toutes les classes de rendu. Cette classe et ses dérivées (QGraphicsView) sont transférées dans le module « widgets » ;
- QtQuick 2 utilise maintenant scenegraph à la place de QWidget et la classe QDeclarativeView devient QQuickView et OpenGL est utilisé par défaut ;
- QtOpenGL continue d'exister pour fournir la classe QGLWidget et ses dérivés. Le code OpenGL Qt4 est donc compatible avec Qt5 ;
- le module Qt3d fait son apparition pour fournir un moteur 3D plus avancé. Ce module est suffisamment important et fera donc l'objet d'un article spécifique.

Remarque : il faut faire attention de ne pas confondre le module QtOpenGL, contenant les classes commençant par QGL, et le module QtGui, contenant les classes commençant par QOpenGL (sans t).

1.3 Les classes de QtGui dans Qt5

Le module QtGui de Qt5 réutilise des classes existantes du module QtOpenGL de Qt4. Les noms sont explicites :

- QOpenGLBuffer ;
- QOpenGLContext ;
- QOpenGLFramebufferObject ;
- QOpenGLFramebufferObjectFormat ;
- QOpenGLShader ;
- QOpenGLShaderProgram.

Plusieurs nouvelles classes font leur apparition :

- QOpenGLContextGroup : groupe de contextes OpenGL pouvant partager des ressources. Cette classe est gérée automatiquement par les objets QOpenGLContext ;
- QOpenGLFunctions : fournit un accès indépendant de la plateforme aux fonctions d'OpenGL ;
- QOpenGLPaintDevice : permet de dessiner dans un contexte OpenGL avec QPainter ;
- QOpenGLStaticContext : manque de documentation ;
- QOpenGLContextData : manque de documentation ;

- QWindowsGLContext : manque de documentation.

1.4 Que faut-il modifier pour utiliser OpenGL dans Qt5 ?

Tout d'abord, il faut Qt5. Le plus simple est d'utiliser les dépôts PPA sur Ubuntu : https://launchpad.net/~forumnokia/+archive/fn-ppa. Qt5 sera installé dans le répertoire /opt/qt5. Pour ajouter cette version de Qt dans QtCreator, il faut aller dans le menu « Outils » puis « Options… », allez dans « Compiler et exécuter… » puis l'onglet « Versions de Qt ». Cliquer sur « Ajouter » et aller chercher le fichier « /opt/qt5/bin/qmake ». Voilà, Qt5 est prêt à être utilisé.

Normalement, vous pouvez utiliser vos projets directement, mais il est préférable de prendre l'habitude d'ajouter le module « widgets » (remarque : l'ajout des modules core et gui n'est pas obligatoire puisqu'ils sont inclus par défaut) :

```
QT += core gui opengl
greaterThan(QT_MAJOR_VERSION, 4) {
    QT += widgets
}
```

Amusez-vous bien !

2 Mise à jour Pout Qt 5.1

La nouvelle version de Qt est encore de finalisation, mais il est déjà possible de télécharger les binaires de Qt 5.1 beta pour réaliser des tests. Cette version apporte quelques nouvelles fonctionnalités, décrites dans l'annonce sur le forum Developpez.com. Je ferais prochainement des articles sur les Qt Quick Controls et les Qt Sensors, mais je vais commencer par présenter les fonctionnalités d'OpenGL prises en charge dans Qt 5.1.

3 Quelle version d'OpenGL ?

Jusque maintenant, il pouvait être compliqué de créer un contexte OpenGL de n'importe quelle version. En effet, dans les versions précédente de Qt, il fallait spécifier la version du contexte OpenGL que l'on souhaitait créer en utilisant QGLFormat. Les numéros de version étaient fixés dans une énumération, si la version souhaitait n'était pas listée, on avait un problème.

Pire, avec Qt 5.0, les développeurs de Qt ont choisit de supporter OpenGL pour être compatible sur un maximum de plate-formes, d'où le choix par défaut de OpenGL ES 2 (avec ANGLE sous Windows, donc une conversion des appels de fonction OpenGL en DirectX...) La conséquence était

qu'il était difficile de créer un contexte OpenGL 3 ou d'utiliser les geometry shaders (qui étaient pourtant présents dans Qt 4).

Avec Qt 4, la prise en charge d'OpenGL était dédié aux classes commençant par QGL, en particulier QGLWidget. Dans Qt 5, les widgets ne sont plus dans le module Qt Gui, mais dans un module séparé (voir les précédents articles de ce blog pour les détails). Il est toujours possible d'utiliser le module Qt Widget (en ajoutant QT += widget dans le fichier de projet .pro), mais ce n'est plus indispensable : toutes les fenêtres peuvent contenir un contexte OpenGL. En effet, Qt 5 est basé sur OpenGL, pour bénéficier de l'accélération matérielle et améliorer les performances de interfaces.

4 Gestion des versions de contexte

4.1 Créer une surface de rendu

Une QSurface est une zone sur laquelle on peut dessiner. Elle est dérivée en QWindow, une fenêtre dans une interface graphique, ou en QOffscreenSurface, un surface de rendu hors écran. Une QSurface est une classe abstraite et ne peut être créer directement. Pour créer une QSurface, il faut donc créer une QWindow (il faut alors spécifier que l'on souhaite une surface de type OpenGL avec la fonction setSurfaceType) ou une QOffscreenSurface, selon le besoin :

```
#include <QGuiApplication>
#include <QWindow>

int main(int argc, char *argv[])
{
    QGuiApplication a(argc, argv);

    QWindow w;
    w.setSurfaceType(QWindow::OpenGLSurface);
    w.show();

    return a.exec();
}
```

Dans la majorité des cas, on utilisera pas non plus directement une QWindow, mais une QQuickView, si l'on souhaite créer des interfaces avec Qt Quick, ou une QWidget et dérivées, si l'on souhaite travailler avec les widgets.

4.2 Connaître la version d'OpenGL par défaut

Dans un premier temps, il faut tester si une surface prend en charge OpenGL. Il suffit de tester son type :

```
bool isCompatibleOpenGL(QSurface* surface) {
    return (surface->surfaceType() == QSurface::OpenGLSurface);
}
```

Ensuite, on récupère le format (QSurfaceFormat) avec la fonction format. Le format contient différentes informations sur le contexte, comme par exemple les numéros de version d'OpenGL ou le type de profile OpenGL (Core ou Compatibility) :

```
QString glVersion(QSurface* surface) {
    QSurfaceFormat format = surface->format();
    QString result = QString("%1.%2")
            .arg(format.majorVersion())
            .arg(format.minorVersion());
    switch(format.profile()) {
    case 1: result += " Core profile"; break;
    case 2: result += " Compatibility profile"; break;
    default: break;
    }
    return result;
}
```

Il faut alors récupérer un contexte OpenGL (QOpenGLContext) valide. Pour cela, plusieurs étapes :

- il faut créer un objet QOpenGLContext avec la QWindow en paramètre (attention, pas directement la QSurface) ;
- définir le format que l'on souhaite utiliser (OpenGL ES 2.0 par défaut) ;
- créer le contexte avec la fonction create() ;

4.3 Création d'un contexte

? Gestion de version du profil : QOpenGLVersionProfile

tester les extension

passer en context debug

5 Les fonctions OpenGL

Un autre problème provient du fonctionnement de OpenGL. Pour assurer un maximum de portabilité, OpenGL ne fournit qu'une interface, la création des bibliothèques étant à la charge des vendeurs de cartes graphiques. Pour utiliser une fonction, il faut donc tester la version d'OpenGL ou une extension, puis charger la fonction avec getProcAdress. Pour faciliter la vie des développeurs, l'initialisation des fonctions peut être pris en charge par une bibliothèque. Un exemple est GLEW, très utilisée et apprécié. Qt fournit pour sa part QGLFunctions (renommé en QOpenGLFunctions dans Qt 5).

Malheureusement, Qt ne fournissait pas toutes les fonctions OpenGL. Et l'utilisation conjointe de GLEW et Qt posait des problèmes.

Dans Qt 5.1, le problème est résolu ! En plus de QOpenGLFunctions, qui fournit les fonctions de base utilisées en interne dans Qt (en gros), Qt 5.1 fournit une version de cette classe pour chaque version d'OpenGL, depuis 1.0 à 4.3. Et comble du luxe, il fournit les versions Core Profil (sans les fonctions dépréciées après OpenGL 3.2) et Compatibility Profil (avec les fonctions dépréciées, pour la compatibilité du code).

```
QOpenGLFunctions
QOpenGLFunctions_1_0
QOpenGLFunctions_1_1
QOpenGLFunctions_1_2
QOpenGLFunctions_1_3
QOpenGLFunctions_1_4
QOpenGLFunctions_1_5
QOpenGLFunctions_2_0
QOpenGLFunctions_2_1
QOpenGLFunctions_3_0
QOpenGLFunctions_3_1
QOpenGLFunctions_3_2_Compatibility
QOpenGLFunctions_3_2_Core
QOpenGLFunctions_3_3_Compatibility
QOpenGLFunctions_3_3_Core
QOpenGLFunctions_4_0_Compatibility
QOpenGLFunctions_4_0_Core
QOpenGLFunctions_4_1_Compatibility
QOpenGLFunctions_4_1_Core
QOpenGLFunctions_4_2_Compatibility
```

```
QOpenGLFunctions_4_2_Core
QOpenGLFunctions_4_3_Compatibility
QOpenGLFunctions_4_3_Core
```

Comment les utiliser ? Il suffit simplement de créer un contexte avec la version d'OpenGL souhaité, puis d'inclure le fichier QOpenGLFunctions correspondant. Rien de plus compliqué.

Autre point important : au pourrait se dire que c'est bien beau, mais comme les versions d'OpenGL est codé en dur, dès que l'on aura une mise à jour d'OpenGL, il faudra attendre que Qt soit mis à jour. C'est vrai. Mais les développeurs ont fait correctement les choses : en fait, ils n'ont pas écrit directement ces fichiers. Ils ont écrit des scripts qui utilisent les spécifications d'OpenGL fournies par le groupe Khronos pour les générer automatiquement. En cas de sortie d'une nouvelle version d'OpenGL, il suffit donc simplement de relancer les scripts et la nouvelle version sera prise en charge.

6 Plus de shaders

Prise à charge de tous les types de shaders dans QOpenGLShader et QOpenGLShaderProgram (même les computes shaders de GL 4.3) :

- QopenGLShader::Vertex
- QopenGLShader::Fragment
- QOpenGLShader::Geometry (require OpenGL >= 3.2)
- QOpenGLShader::TessellationControl (require OpenGL >= 4.0)
- QOpenGLShader::TessellationEvaluation (requires OpenGL >= 4.0)
- QOpenGLShader::Compute (requires OpenGL >= 4.3)

7 Les nouveaux contextes

Les groupes de contextes QOpenGLContextGroup : plusieurs contextes OpenGL qui partagent des ressources

Les contextes de debug (cf mon article sur les contexte debug d'openGL) :

- QopenGLDebugLogger
- QopenGLDebugMessage
- QopenGLTimeMonitor
- QOpenGLTimerQuery

8 Prise en charge de Vertex Array Object (VAO)

En natif, avec QOpenGLVertexArrayObject

Source : http://doc-snapshot.qt-project.org/qt5-stable/qtgui/openglwindow.html