Support d'OpenGL dans Qt

Le support d'OpenGL dans Qt 5 a été modifié pour mieux l'intégrer avec les nouveaux modules de Qt : QtQuick2 et Qt3D. Cet article présente les modifications apportées dans Qt 5.

Activer OpenGL dans Qt 4

Dans Qt 4, 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, utilisez la ligne de commande "-graphicssystem opengl" ou la fonction `QApplication::setGraphicsSystem("opengl")`, dans la fonction `main` par exemple ;
- pour `QGraphicsView` (QtGraphics) ou `QDeclarativeView` (QtQuick), utilisez la fonction `setViewport(new QGLWidget)` ;

- on peut également utiliser `QPainter` directement sur un `QGLWidget`, qui est une classe dérivée de `QWidget` avec un contexte OpenGL ;
- avec le QML Viewer, utilisez la commande ”-opengl”.

L’organisation des modules dans Qt 5

Dans Qt 4, le support d'OpenGL était donc optionnel, dans un module dédié. Il fallait créer spécifiquement un contexte OpenGL et le passer en paramétré pour bénéficier de l’accélération matérielle.

Dans Qt 5, l'objectif a été de fournir un support minimal d'OpenGL dans QtGui, ce qui permet de l'utiliser dans tous les modules graphiques (widgets, Qt Quick) qui dépendent de QtGui.

- les classes `QOpenGLXxx` 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` par exemple) sont transférées dans le module “widgets” ;
- les vues QtQuick 2 ne sont plus basées sur `QWidget`, la classe `QDeclarativeView` devient `QQuickView` et OpenGL est utilisé par défaut ;
- le module QtOpenGL continue d’exister pour fournir la classe `QGLWidget` et ses dérivés (les classes qui commencent par `QGLXxx`). Ce module permet de conserver le code Qt 4 compatible avec Qt 5, mais ces classes sont dépréciées au profit des classes `QOpenGLXxx`.

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

Overview QtOpenGL

Notes de mise à jour
Ce tutoriel concerne que les classes de Qt Gui, non QtOpenGL.

Surfaces et contextes

Un contexte est une “zone” de travail d'OpenGL, contient les informations permettant à OpenGL de fonctionner. Peut correspondre à une fenêtre visible à l'écran, et dans ce cas, tous les rendus réalisés avec ce contexte seront visible dans cette fenêtre, ou hors écran.

- QOpenGLContext (Qt 5.0) : information a propos du contexte OpenGL, par exemple version d'OpenGL prise en charge, gestion des extensions OpenGL, la surface correspondante à ce contexte.
- QSurface : classe abstraite, qui représente n'importe quelle surface. Peut être une surface visible (QWindow) ou une surface non visible à l'écran (QOffscreenSurface).
  - supportsOpenGL() : teste si cette surface supporte OpenGL ou non (une surface peut également servir pour dessiner sans utiliser OpenGL, avec le moteur Raster).
  - format() : retourne un QSurfaceFormat, qui contient les informations sur le format... (à détailler). Contient en particulier version(), qui permet de tester la version d'OpenGL prise en charge (peut être différent de la version demandée lors de la création de la surface).
  - size() : dimensions de la surface.
- QWindow : surface visible à l'écran. Nombreuses informations non spécifiques à OpenGL (titre de la fenêtre, gestion des events, etc.). Sera généralement utilisé via QWidget ou QQuickView, sauf si on travaille directement avec OpenGL.

- QOffscreenSurface : surface non visible. Partager des ressources avec d'autres contexte OpenGL pour charger des textures ou dessiner dans un FBO de façon asynchrone. Remarque : à créer dans le thread principal.
- QOpenGLWindow et QOpenGLWidget (Qt 5.4) : hérite de QWindow et QWidget, en creant un contexte OpenGL en interne et proposent les fonctions `initializeGL`, `paintGL` et `resizeGL`.

```
QOpenGLContext* context = new QOpenGLContext(parent);
context->setFormat(...);
context->create();

context->makeCurrent(this);
context->swapBuffers(this);
```

- QOpenGLContextGroup (Qt 5.0) : Possibilité de partager des ressources entre plusieurs contextes (par exemple, si une texture est utilisée dans 2 contextes, on évite de la charger 2 fois, on la partage). Plus de 2 contextes ?
- QOpenGLVersionProfile (Qt 5.1) : permet de creer un contexte avec une version spécifique de GL avec versionFunctions. Préférer la version avec template.

Debug et profiling

- Debug contexte. Permet d'avoir des messages de debug générés par le driver ou le GPU. Cf Déboguer avec OpenGL 4
- QOpenGLDebugLogger (Qt 5.1)
- QOpenGLDebugMessage (Qt 5.1)

```
// gestion des erreurs
GLenum error = GL_NO_ERROR;
do {
    error = glGetError();
    if (error != GL_NO_ERROR)
```

```
        // handle the error
} while (error != GL_NO_ERROR);

// création d'un contexte debug
QSurfaceFormat format;
// asks for a OpenGL 3.2 debug context using the Core
profile
format.setMajorVersion(3);
format.setMinorVersion(2);
format.setProfile(QSurfaceFormat::CoreProfile);
format.setOption(QSurfaceFormat::DebugContext);

QOpenGLContext *context = new QOpenGLContext;
context->setFormat(format);
context->create();

QOpenGLContext *ctx = QOpenGLContext::currentContext();
ctx->hasExtension(QByteArrayLiteral("GL_KHR_debug"))
QOpenGLDebugLogger *logger = new QOpenGLDebugLogger(this);
logger->initialize(); // initializes in the current context,
i.e. ctx

// read logs
QList<QOpenGLDebugMessage> messages = logger->loggedMessages
();
foreach (const QOpenGLDebugMessage &message, messages)
    qDebug() << message;

// connexion directe
connect(logger, &QOpenGLDebugLogger::messageLogged, receiver,
&LogHandler::handleLoggedMessage);
logger->startLogging();

// send message
QOpenGLDebugMessage message =
    QOpenGLDebugMessage::createApplicationMessage(
QStringLiteral("Custom message"));
logger->logMessage(message);
```

- QOpenGLTimeMonitor (Qt 5.1)
- QOpenGLTimerQuery (Qt 5.1)

En fonction des extensions ARB_timer_query et EXT_timer_query extensions.

Extensions et fonctions

- QOpenGLFunctions (Qt 5.0) : support des fonctions OpenGL. Pour utiliser une fonction d'une lib en général, besoin du prototype (nom, paramètres, etc) et l'adresse dans la lib. Fait automatiquement par le link lors de la compilation.
- QAbstractOpenGLFunctions et QOpenGLFunctions_X_x (Qt 5.1) : chargement des fonctions specifiques d'une version de GL.

OpenGL cas particulier, puisque les fonctions et fonctions dispo dépend du GPU sur lequel s'exécute le programme, peut changer pour chaque utilisateur. Besoin de gestion plus dynamique.

Versions mineurs et majeur d'OpenGL, ajout de nouvelles fonctions, enums, etc. et retrait d'autres. Donc a chaque version, correspond une liste de fonctions dispo. Possibilité d'utiliser des fonctions ne correspondant pas à la version de contexte utilisée, si supporté par le GPU, en chargeant dynamiquement la fonction par un système d'extension.

Par exemple, GPU qui supporte OpenGL 4.0, création d'un contexte GL 2.0 et chargement d'une fonction de GL 3.0.

Remarque : a partir de GL 3.2, suppression des anciennes fonctions dans Core, conservée dans Compatibility .

Qt permet de gérer tout cela (création d'un contexte avec une version spécifique, chargement d'une extension). Par défaut, OpenGL ES 2 (supporter partout, même sur mobile).

```
QOpenGLFunctions:
initializeOpenGLFunctions();
hasExtension
```

```
getProcAddress

QOpenGLFunctions_3_3_Core* funcs = 0;
funcs = context->versionFunctions<QOpenGLFunctions_3_3_Core>
();
if (!funcs) {
    qWarning() << "Could not obtain required OpenGL context
version";
    exit(1);
}
funcs->initializeOpenGLFunctions();
```

Shaders

- QOpenGLShader (Qt 5.0)
- QOpenGLShaderProgram (Qt 5.0)

```
static const char *vertexShaderSource =
    "attribute highp vec4 posAttr;\n"
    "attribute lowp vec4 colAttr;\n"
    "varying lowp vec4 col;\n"
    "uniform highp mat4 matrix;\n"
    "void main() {\n"
    "    col = colAttr;\n"
    "    gl_Position = matrix * posAttr;\n"
    "}\n";

static const char *fragmentShaderSource =
    "varying lowp vec4 col;\n"
    "void main() {\n"
    "    gl_FragColor = col;\n"
    "}\n";

// init prog
QOpenGLShaderProgram *m_program;
m_program = new QOpenGLShaderProgram(this);
m_program->addShaderFromSourceCode(QOpenGLShader::Vertex,
vertexShaderSource);
```

```
m_program->addShaderFromSourceCode(QOpenGLShader::Fragment,
fragmentShaderSource);
m_program->link();
m_posAttr = m_program->attributeLocation("posAttr");
m_colAttr = m_program->attributeLocation("colAttr");
m_matrixUniform = m_program->uniformLocation("matrix");

// load shader
GLuint shader = glCreateShader(type);
glShaderSource(shader, 1, &source, 0);
glCompileShader(shader);

// render
const qreal retinaScale = devicePixelRatio(); // iOS
glViewport(0, 0, width() * retinaScale, height() *
retinaScale);

glClear(GL_COLOR_BUFFER_BIT);

m_program->bind();

QMatrix4x4 matrix;
matrix.perspective(60.0f, 4.0f/3.0f, 0.1f, 100.0f);
matrix.translate(0, 0, -2);
matrix.rotate(100.0f * m_frame / screen()->refreshRate(), 0,
1, 0);

m_program->setUniformValue(m_matrixUniform, matrix);

GLfloat vertices[] = {
  0.0f, 0.707f,
  -0.5f, -0.5f,
  0.5f, -0.5f
};

GLfloat colors[] = {
  1.0f, 0.0f, 0.0f,
  0.0f, 1.0f, 0.0f,
  0.0f, 0.0f, 1.0f
};

glVertexAttribPointer(m_posAttr, 2, GL_FLOAT, GL_FALSE, 0,
```

```
vertices);
glVertexAttribPointer(m_colAttr, 3, GL_FLOAT, GL_FALSE, 0,
colors);

glEnableVertexAttribArray(0);
glEnableVertexAttribArray(1);

glDrawArrays(GL_TRIANGLES, 0, 3);

glDisableVertexAttribArray(1);
glDisableVertexAttribArray(0);

m_program->release();
```

Buffers et objets

- QOpenGLBuffer (Qt 5.0)
- QOpenGLFramebufferObject (Qt 5.0) + QOpenGLFramebufferObjectFormat (Qt 5.0)
- QOpenGLTexture (Qt 5.2)
- QOpenGLVertexArrayObject (Qt 5.1)

A trier

- QOpenGLPaintDevice (Qt 5.0)
- QOpenGLPixelTransferOptions (Qt ???)

Détails

Vous trouverez la liste de toutes les classes `QOpenGLXxx` dans la documentation de Qt.

OpenGL,, Qt,, Qt5