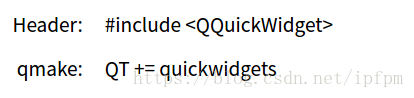
# **1ch.QT】QT的学习：使用QQuickWidget以及QQuickView**

[ipfpm](https://blog.csdn.net/ipfpm" \t "https://blog.csdn.net/ipfpm/article/details/80856640/_blank) 2018-06-29 15:36:55  11695  收藏 7

QQuickWidget以及QQuickView都可以用来加载qml文件，用于显示界面，但是到目前为止，自己并没有弄懂两者有什么区别，唯一确定的两个均能够跟widget窗口类进行混合编程。下面通过例子来记录下使用混合编程的时候两个控件的不同之处：

（1）使用QQuickwidget：



qml文件为：

import QtQuick 2.0

import QtQuick 2.2

Rectangle {

id: page

width: 500; height: 200

color: "red"

}

加载qml文件：

mainMapBoxWidget = new QQuickWidget(this);

mainMapBoxWidget->setResizeMode(QQuickWidget::SizeRootObjectToView);

mainMapBoxWidget->setSource(QUrl("qrc:/files/resource/file/mapBoxGlView.qml"));

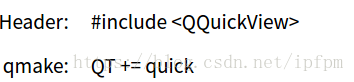
mainMapBoxWidget->resize(MAIN\_DISPALY\_WIDTH,MAIN\_DISPALY\_HEIGH);

mainMapBoxWidget->show();

得到的程序（其中有在主窗口有上下两个栏，添加的mainmapboxwidget没有把两个栏覆盖）：



（2）使用quickview：



保持qml文件不动，

mainMapBoxView = new QQuickView();

mainMapBoxWindow = QWidget::createWindowContainer(mainMapBoxView, this);

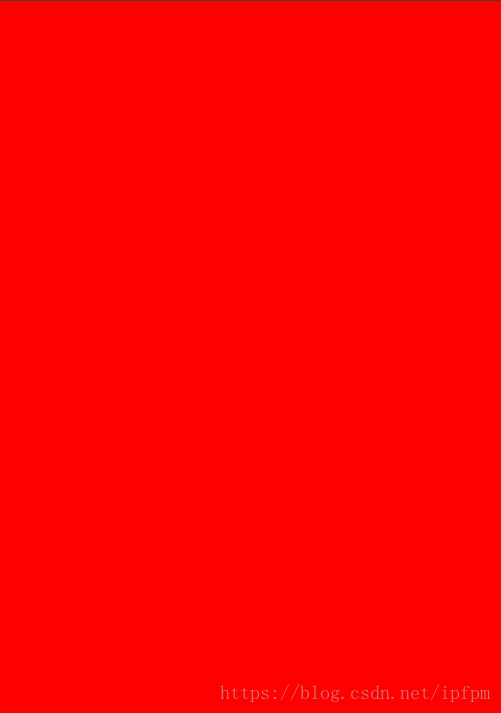
mainMapBoxWindow->resize(MAIN\_DISPALY\_WIDTH,MAIN\_DISPALY\_HEIGH);

mainMapBoxView->setResizeMode(QQuickView::SizeRootObjectToView);

mainMapBoxView->setSource(QUrl("qrc:/files/resource/file/mapBoxGlView.qml"));

mainMapBoxWindow->show();

发现上下两个栏没有了，被新增加的mainmzpboxwindow覆盖了。



（3）还是使用qquickwidget，加载mapboxgl的地图。

import QtQuick 2.0

import QtPositioning 5.5

import QtLocation 5.9

Rectangle {

id: mapWindow

//anchors.fill: parent//在地图中添加开了这一项之后地图会不显示，因此注释掉

Map {

id: map

anchors.fill: parent

plugin: Plugin { name: "mapboxgl" }

center: QtPositioning.coordinate(60.170448, 24.942046) // Helsinki

zoomLevel: 11

         MapParameter {

             type: "source"

             property var name: "routeSource"

             property var sourceType: "geojson"

             property var data: '{ "type": "FeatureCollection", "features":[{ "type": "Feature", "properties": {}, "geometry": {"type": "LineString", "coordinates": [[ 24.934938848018646, 60.16830257086771 ], [ 24.943315386772156, 60.16227776476442 ]]}}]}'

             }

         MapParameter {

             type: "layer"

             property var name: "route"

             property var layerType: "line"

             property var source: "routeSource"

             property var before: "road-label-small"

         }

         MapParameter {

             type: "paint"

             property var layer: "route"

             property var lineColor: "blue"

             property var lineWidth: 8.0

         }

         MapParameter {

             type: "layout"

             property var layer: "route"

             property var lineJoin: "round"

             property var lineCap: "round"

         }

     }

 }

得到程序：



注意：在qml中Rectangle后面不能加anchors.fill: parent否则地图会不显示。

如果还是使用QQuickView的话还是会覆盖上下两个状态栏，如下图：网上说一般只有在全部使用qml的时候才会用QQuickView，不知道是不是这样



建议：如果有知道原因的可以留言；如果还是想用QQuickView的可以使用布局：layout的形式：

QVBoxLayout \*layout = new QVBoxLayout(this);

layout->addWidget(labelBar);

layout->addWidget(mainMapBoxWidget);

layout->addWidget(buttonBar);

layout->setMargin(0);

layout->setSpacing(0);

当然使用QQuickWidget也是可以使用layout的布局的。



跟之前的显示差不多，区别就是地图原来是布满整个窗口的，状态栏是透明的话，状态栏的下面也会显示地图，但是使用布局layout的时候，地图只会占据窗口的中间部分，状态栏的下面不会显示有地图。

# **2ch.Qt5 QtQuick系列----QtQuick的Secne Graph剖析(3)-- qml与OpenGl结合**

[weixin\_34362790](https://blog.csdn.net/weixin_34362790" \t "https://blog.csdn.net/weixin_34362790/article/details/_blank) 2019-01-15 22:54:00  278  收藏 2

版权

*我读的书愈多，就愈亲近世界，愈明了生活的意义，愈觉得生活的重要。 —— 高尔基*

****需要先看：****<https://blog.csdn.net/qq_35865125/article/details/86485008> 来理解qt quick场景图的渲染过程。以下内容主要来自<http://doc.qt.io/qt-5/qtquick-visualcanvas-scenegraph.html> ， 并结合了自己的理解和实践。

Qt的场景图提供了两种方式来让用户实现与OpengGL的结合。一是，直接调用OpenGL命令函数，二是，在场景图中创建textured node。

QquickWindow类负责将qml文件中的内容渲染到屏幕上，在渲染过程中，会从渲染线程中发出信号：[QQuickWindow::beforeRendering](http://doc.qt.io/qt-5/qquickwindow.html" \l "beforeRendering)() and [QQuickWindow::afterRendering](http://doc.qt.io/qt-5/qquickwindow.html" \l "afterRendering)()，用户可以接受这些信号，并在其他线程中执行调用OpenGl的原始函数的操作，来绘制一些东东，如果用户使用的是第一个信号，用户用opengl画出的东东会在qml场景的下面，否则会在上面。 这种方式的好处是不需要二外的framebuffer或内存。缺点是，“The downside is that Qt Quick decides when to call the signals and this is the only time the OpenGL application is allowed to draw。”。这种方法的一个官方的例子： [Scene Graph - OpenGL Under QML](http://doc.qt.io/qt-5/qtquick-scenegraph-openglunderqml-example.html) 。

另一种方法：The other alternative is to create a [QQuickFramebufferObject](http://doc.qt.io/qt-5/qquickframebufferobject.html), render into it, and let it be displayed in the scene graph as a texture. The [Scene Graph - Rendering FBOs](http://doc.qt.io/qt-5/qtquick-scenegraph-textureinsgnode-example.html) example shows how this can be done. It is also possible to combine multiple rendering contexts and multiple threads to create content to be displayed in the scene graph. The [Scene Graph - Rendering FBOs in a thread](http://doc.qt.io/qt-5/qtquick-scenegraph-textureinthread-example.html) examples show how this can be done.

****注意:**** When mixing OpenGL content with scene graph rendering, it is important the application does not leave the OpenGL context in a state with buffers bound, attributes enabled, special values in the z-buffer or stencil-buffer or similar. Doing so can result in unpredictable behavior.

****注意:**** The OpenGL rendering code must be thread aware, as the rendering might be happening outside the GUI thread.

****用上面提到的第一种方法时，需要注意：****

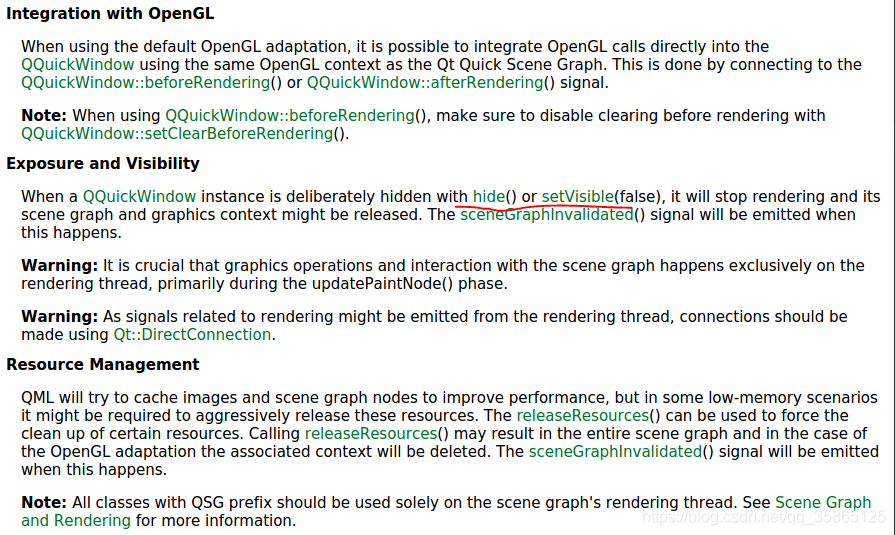
1）需要在.pro文件中添加：  
LIBS += -lglut

LIBS += -lGLU

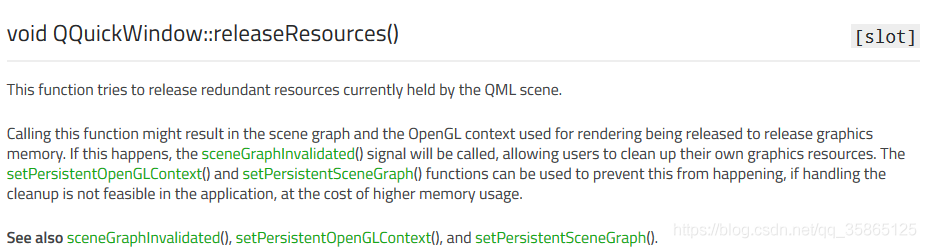
   需要在系统中安装opengl。

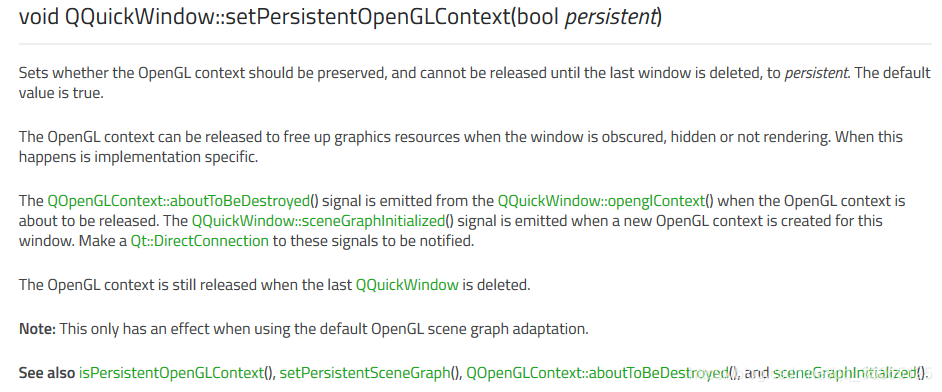
2） 需要在main中调用glutInit( &argc, argv );(#include <GL/glut.h>)。

3）资源回收问题：有必要详细了解QQuickWindow这个类。<http://doc.qt.io/qt-5/qquickwindow.html>



上图中，标出红线的部分我曾经使用过。类似后面我给出的例子，不同之处：我将写的qml文件传递给一个QquickView，在绘制出opengl图像后，调用QquckView的hide函数隐藏整个quickview。在一些性能较低的机器人上运行调用hide函数时，会遇到界面卡死问题。或许可以通过上图中提到的Resource Management部分的方法解决。





### ****My example(调用opengl画一个箭头):---基于官方例子****[Scene Graph - OpenGL Under QML](http://doc.qt.io/qt-5/qtquick-scenegraph-openglunderqml-example.html)

IMG_262

自定义类的.h：

#ifndef SELFDEFSUBITEM\_H

#define SELFDEFSUBITEM\_H

#include <QObject>

#include <QQuickItem>

#include <QtGui/QOpenGLFunctions>

class MyRenderer : public QObject, protected QOpenGLFunctions

{

Q\_OBJECT

public:

MyRenderer() ;

~MyRenderer();

void setViewportSize(const QSize &size);

void setWindow(QQuickWindow \*window);

void Init\_Before\_Paint\_arm3d();

void Paint\_Line();

void DrawLine();

public slots:

void paint();

private:

QSize m\_viewportSize;

QQuickWindow \*m\_window;

};

class SelfDefSubItem : public QQuickItem

{

Q\_OBJECT

public:

SelfDefSubItem();

~SelfDefSubItem();

static float x\_rot,y\_rot,z\_rot;

static float zoomscale;

static float x\_trans, y\_trans;

public slots:

void sync();

void cleanup();

private slots:

void handleWindowChanged(QQuickWindow \*win);

protected:

private:

MyRenderer \*m\_renderer;

};

#endif *// SELFDEFSUBITEM\_H*

自定义类的cpp：

#include "SelfDefSubItem.h"

#include <QtQuick/qquickwindow.h>

#include <QtGui/QOpenGLShaderProgram>

#include <QtGui/QOpenGLContext>

#include <GL/glut.h>

#include <GL/glu.h>

#include <GL/gl.h>

#include "stdio.h"

#include <fstream>

#include <iostream>

#include <string>

#include<vector>

using namespace std;

float SelfDefSubItem::x\_rot=0.0;float SelfDefSubItem::y\_rot=0.0;float SelfDefSubItem::z\_rot=0.0;

float SelfDefSubItem::zoomscale=1.0;

float SelfDefSubItem::x\_trans = 0;

float SelfDefSubItem::y\_trans = 0;

SelfDefSubItem::SelfDefSubItem()

{

connect(this, &QQuickItem::windowChanged, this, &SelfDefSubItem::handleWindowChanged);

}

SelfDefSubItem::~SelfDefSubItem()

{

cleanup();

}

void SelfDefSubItem::cleanup()

{

if (m\_renderer)

{

delete m\_renderer;

m\_renderer = 0;

}

}

void SelfDefSubItem::sync()

{

if (!m\_renderer)

{

m\_renderer = new MyRenderer();

connect(window(), &QQuickWindow::beforeRendering, m\_renderer, &MyRenderer::paint, Qt::DirectConnection);

*//connect(window(), &QQuickWindow::afterRendering, m\_renderer, &SquircleRenderer::paint, Qt::DirectConnection);*

}

m\_renderer->setViewportSize(window()->size() \* window()->devicePixelRatio());*//*

m\_renderer->setWindow(window());

}

void SelfDefSubItem::handleWindowChanged(QQuickWindow \*win)

{

if (win)

{

connect(win, &QQuickWindow::beforeSynchronizing, this, &SelfDefSubItem::sync, Qt::DirectConnection);

connect(win, &QQuickWindow::sceneGraphInvalidated, this, &SelfDefSubItem::cleanup, Qt::DirectConnection);

*// If we allow QML to do the clearing, they would clear what we paint*

*// and nothing would show.*

win->setClearBeforeRendering(false);

}

}

*///*

MyRenderer::MyRenderer()

{

}

void MyRenderer::paint()

{

m\_window->resetOpenGLState();

initializeOpenGLFunctions();

glViewport(0, 0, m\_viewportSize.width(), m\_viewportSize.height());

Paint\_Line();

}

void MyRenderer::DrawLine()

{

int len = 750;

GLfloat mat\_ambient\_axis1[4] = { 0.8, 0, 0, 1.0 };

glMaterialfv(GL\_FRONT, GL\_AMBIENT, mat\_ambient\_axis1);

glMaterialfv(GL\_FRONT, GL\_DIFFUSE, mat\_ambient\_axis1);

glLineWidth(18);*//设置线段宽度*

glPushMatrix();

glBegin(GL\_LINES);

glVertex3d(0.0, 0.0, 0.0);

glVertex3d(0.0, 0.0, len);

glEnd();

glTranslated(0.0, 0.0, len - 13);

glutWireCone(16, 37, 31, 32);*//void glutWireCone(GLdouble radius, GLdouble height, GLint slices, GLint stacks); 线框 圆锥体*

glPopMatrix();

}

void MyRenderer::Init\_Before\_Paint\_arm3d()

{

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE | GLUT\_DEPTH);

*//\*\*\*\*\*Defining Position and Colors for a Light Source\*\*\*\*\*:*

GLfloat light\_position[] = { 1.0, 10.0, 0.0, 0.0 };

GLfloat light\_ambient[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat light\_diffuse[] = { 1.0, 1.0, 1.0, 1.0 };

GLfloat light\_specular[] = { 1.0, 1.0, 1.0, 1.0 };

glLightfv(GL\_LIGHT0, GL\_POSITION, light\_position); *//specify the location of lighter*

glLightfv(GL\_LIGHT0, GL\_AMBIENT, light\_ambient); *//specify light0's ambient color*

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, light\_diffuse); *//specify light0's diffuse color*

glLightfv(GL\_LIGHT0, GL\_SPECULAR, light\_specular); *//specify light0's specular color*

*//glClearColor(1, 1, 1, 1.0);//specify the backgroud color!*

glClearColor(7/255, 7/255, 18/255, 1.0);*//specify the backgroud color!*

glShadeModel(GL\_SMOOTH);

GLfloat lmodel\_ambient[] = { 0.4, 0.4, 0.3, 1.0 };

glLightModelfv(GL\_LIGHT\_MODEL\_AMBIENT, lmodel\_ambient);

glEnable(GL\_LIGHTING);

glEnable(GL\_LIGHT0);

glEnable(GL\_DEPTH\_TEST);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glOrtho(-1100.0,1200.0, -1100.0, 1100.0, -1100, 1100);

glPointSize(1);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

gluLookAt(8.3, 1.56, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

glScalef (SelfDefSubItem::zoomscale, SelfDefSubItem::zoomscale, SelfDefSubItem::zoomscale);*//scaling in 3 direction*

}

void MyRenderer::Paint\_Line()

{

initializeOpenGLFunctions();

Init\_Before\_Paint\_arm3d();

DrawLine();*//Call raw OpenGl command!*

}

MyRenderer::~MyRenderer()

{

*//delete m\_program;*

}

void MyRenderer::setViewportSize(const QSize &size)

{

m\_viewportSize = size;

}

void MyRenderer::setWindow(QQuickWindow \*window)

{

m\_window = window;

}

main.qml

import QtQuick 2.11

import QtQuick.Window 2.11

import SelfDefSubItem 1.0

Window {

visible: true

width: 640

height: 480

title: qsTr("Hello World")

SelfDefSubItem{

id:selfdef

}

}

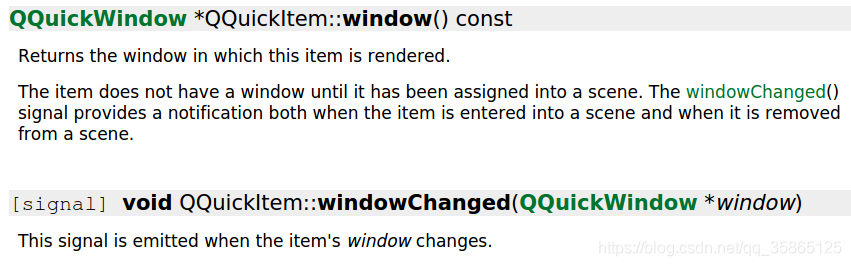
### ****几个注意的地方：****

1. connect(window(), &QQuickWindow::beforeRendering, m\_renderer, &MyRenderer::paint, Qt::DirectConnection);

*执行程序时，MyRenderer::paint函数会被执行好几次！！可能在渲染线程中会发出多个QQuickWindow::beforeRendering信号，具体过程尚未知道，需要进一步查资料，看qt公司的人怎么想的。*

        connect(this, &QQuickItem::windowChanged, this, &Arm3d::handleWindowChanged);

         QQuickWindow负责将qml场景渲染出来，用户自己写的qml文件必须依附于它才能被渲染。QquickItem::window函数返回QquickItem自己所依附的那个QuickWindow。



 转载于:https://www.cnblogs.com/butterflybay/p/10347852.html