RSATom karma 12.0 12 votes rating 1.3

January 9 at 17:31

# QQuickRenderControl, or make friends with someone else QML OpenGL context. Part I tutorial

Qt * , C ++ * , Programming *

The recent release of Qt 5.4 , among other things, make available to developers alone, in my view, a very interesting tool. Namely, the developers of Qt QQuickRenderControl part of the public API. Amusement of this class is that it is now possible to use Qml in conjunction with any other framework, if it is able to deliver (or ask) a pointer to the OpenGL context.
On the other hand, in the process of working on one of my projects, I encountered with the need to draw on the QML scene CALayer (Mac OS X) , without the slightest possibility to access the parent window. Weekly search for possible solutions to the problem showed that the most appropriate solution would be to just use QQuickRenderControl of Qt 5.4, thanks to a good coincidence, received the status of the release simultaneously with the occurrence of the above-mentioned problems.
Initially, I assumed that the problem of Pleven, and will be resolved within a couple of nights, but how much I was wrong - the task took about a half moon on the study, and another half months for implementation (which is still far from ideal).

## Several abstracts

- QQuickRenderControl this is simply an additional interface to implement QQuickWindow to receive notifications on changes QML scene, as well as a command in the opposite direction (ie, in fact, "crutch");
- Rendering result is obtained in the form QOpenGLFramebufferObject (hereinafter FBO) , which subsequently can be used as a texture;
- Have to work directly with QuickWindow, respectively download service provided by QML QQuickView will not be available and will have to implement it yourself;
- Since no window is not actually created, it is necessary to artificially transfer mouse and keyboard events in QQuickWindow. Just need to manually control the size of the window;
- Example of use QQuickRenderControl I was able to find only one in Qt 5.4 (Examples \ Qt-5.4 \ quick \ rendercontrol) - actually it and passed all of the proceedings;

## What should be done to solve the original problem?

1) Implement setting QQuickWindow for rendering FBO and manage the process through QQuickRenderControl;
2) Implement load Qml and accession to the result QQuickWindow;
3) Implement the transfer of mouse and keyboard events;
4) Render FBO (for which all and was started); This article, I will allow myself to dwell only on paragraph 1), the other items in the future planned parts (if you find it interesting).

## Configuring QQuickWindow

### *External QOpenGLContext*

The starting point is an OpenGL context in which, ultimately, will be drawn FBO. But since, with high probability, it is necessary to work with the context was not originally unrelated to Qt, this is necessary to convert the context of the size of the operating system instance QOpenGLContext. To do this, use the method QOpenGLContext :: setNativeHandle . Example usage based NSOpenGLContext:

```
NSOpenGLContext * nativeContext = [super openGLContextForPixelFormat: pixelFormat];

QOpenGLContext * extContext = new QOpenGLContext;
extContext-> setNativeHandle (QVariant :: fromValue (QCocoaNativeContext (nativeContext)));
extContext-> create ();
```

*The list of available Native Context better to look directly into the header files Qt (include \ QtPlatformHeaders), because Documents in this part much is not complete.*

You can then use this context (but should be closely monitored to change the state of that context does not conflict with the manipulation of the owner), and can be shared context:

```
    QSurfaceFormat Format ;
     Format .setDepthBufferSize ( 16 );
     Format .setStencilBufferSize ( 8 );

    context = new QOpenGLContext;
    context-> setFormat ( Format );
    context-> setShareContext (extContext);
    context-> create ();
```

An important nuance to use OpenGL context with QML is the presence of customized Depth Buffer and Stencil Buffer, so if you have no opportunity to influence the parameters of the original context, it is necessary to use the shared context is defined «Depth Buffer Size» and «Stencil Buffer Size».

### *Creating QQuickWindow*

When creating QQuickWindow previously created QQuickRenderControl and passed to the constructor:

```
    * = QQuickRenderControl RenderControl QQuickRenderControl new () ;
    * QQuickWindow quickWindow QQuickWindow new = (RenderControl) ;
    quickWindow-> setGeometry ( 0 , 0 , 640 , 480 ) ;
```

In addition, it is important to set the window size for the further successful creation FBO.

### *Initialization QQuickRenderControl and QOpenGLFramebufferObject*

Before calling QQuickRenderControl :: initialize the current context, it is important to do because during the call signal will be generated sceneGraphInitialized, and is a good point for providing FBO (which in turn requires the exposed current context).

```
    QOpenGLFramebufferObject * FBO = nullptr;
     Connect (quickWindow, & QQuickWindow :: sceneGraphInitialized,
        [B] () {
            fbo = new QOpenGLFramebufferObject (quickWindow-> size (), QOpenGLFramebufferObject :: CombinedDepthStenc
il);
            quickWindow-> setRenderTarget (fbo);
        }
    );

    offscreenSurface = new QOffscreenSurface ();
    offscreenSurface-> setFormat (context-> format ());
    offscreenSurface-> create ();

    context-> makeCurrent (offscreenSurface);
    renderControl-> initialize (context);
    context-> doneCurrent ();
```

### *Rendering*

Rendering should be implemented as a response to signals QQuickRenderControl :: renderRequested and QQuickRenderControl :: sceneChanged. The difference in these two cases is that in the second case it is necessary to cause further QQuickRenderControl :: polishItems and QQuickRenderControl :: sync. The second important feature is that it is not recommended strongly otsuschestvlyat rendering directly to the handlers of the above signals. Therefore, use a timer with a short interval. Well, last subtlety is that, in the case of shared OpenGL context, after rendering, you need to call glFlush - otherwise the primary context does not see changes in the FBO.

```
    bool * needSyncAndPolish = new bool;
     * needSyncAndPolish = True;

    QTimer * renderTimer = new QTimer;
    renderTimer-> setSingleShot (true);
    renderTimer-> setInterval ( 5 );
     Connect (renderTimer, & QTimer :: timeout,
        [B] () {
            if (context-> makeCurrent (offscreenSurface)) {
                if ( * needPolishAndSync ) {
                    * needPolishAndSync = false;
                    renderControl-> polishItems ();
                    renderControl-> sync ();
                }
                renderControl-> render ();
                quickWindow-> resetOpenGLState ();
                context-> functions () -> glFlush ();
                context-> doneCurrent ();
            }
    );

    Connect (RenderControl, & QQuickRenderControl :: renderRequested,
        [B] () {
            if (! renderTimer-> isActive ())
                renderTimer-> start ();
        }
    );

    Connect (RenderControl, & QQuickRenderControl :: sceneChanged,
        [B] () {
            * NeedPolishAndSync = True;
             if (! renderTimer-> isActive ())
                renderTimer-> start ();
        }
    );
```

Well, in general, if not all, the first part of the task performed. The class implements the concept of the above is available on GitHub: FboQuickWindow.h , FboQuickWindow.cpp Comments, questions, healthy criticism in the comments - welcome. Continued: Part II: Load QML , Part III: processing user input

Qt5, QML, Qt Quick 2, OpenGL

14 4802 54 RSATom 1.3

## Similar publications

Mobile application for Qt Quick: opportunities and prospects April 22, 2014 at 12:37
Model-View in QML. Part Three: Models in QML and JavaScript September 29, 2013 at 21:38
Model-View in QML. Part Two: custom presentation August 14, 2013 at 22:35
Assembling Qt 5.1 applications for Android on Mac, seriously? June 30, 2013 at 15:23
Model-View in QML. Part One: Representations of Component Based June 24, 2013 at 10:29
Overview of new features in Qt Quick 24 June 2013 at 10:07
Upcoming snacks in Qt Quick 22 June 2013 at 18:47
Model-View in QML. Part is zero, opening June 1, 2013 at 00:51
Qt Quick: the best techniques November 23, 2011 at 08:06
Creating a hybrid Qt Quick and C ++ applications June 17, 2011 at 11:49

## Comments ( 12 )

**FAQuiR** January 11, 2015 at 06:33 # 0

Please explain what caused the number 5 in renderTimer-> setInterval (5);

**RSATom** January 11, 2015 at 08:35 # ↑ 0

To be honest - I do not know the exact answer to this question. The number was taken from the example rendercontrol (window.cpp: 100).
I guess that when this interval is to focus on zhelatemy fps. But it is better, I think, does not use a timer, and attached vertical synchronization (VSync) screen.

**FAQuiR** January 11, 2015 at 08:42 # ⍈ ↑ 0

Desired fps for you - 200?
Yes, you're right, it is better not to use the timer, I would like to see and snap to the vertical sync.

**RSATom** January 11, 2015 at 08:45 # ⍈ ↑ 0

not so much for me as for the author rendercontrol :)

**semlanik** January 12, 2015 at 15:32 # ⍈ ↑ 0

Most do not even understand why the signals are not called at the time when it really needs to happen rendering, and do not shove primitives in the context, but instead need to do a separate "prerendering" texture, which later will be given on the drawing.

**RSATom** January 12, 2015 at 15:46 # ⍈ ↑ 0

I think that this may be due to the fact that the standard use QQuickWindow (ie without using QQuickRenderControl) rendering is done in a separate thread ( according to the documentation and personal experience ).
In addition, the advance is hard to imagine how and where the user will be in the future use the resulting FBO - as a consequence it is necessary to provide the maximum possible, even at the cost of complexity of use.

**semlanik** January 12, 2015 at 15:53 # ⍈ ↑ 0

There is not only a complication of use, but severe loss of performance. I've already looked at how the rendering "standard use QQuickWindow», I just do not quite understand why it was impossible to do the same mechanism to "reset" SGNode'ov, but in another context. The only adequate reason - the lack of a "synchronous" notifications about entering the drawing function. While on the other hand it is not clear as it works reset FBO. In general, we must look QQuickWindow implementation and okolostoyaschih classes when working with "foreign" context. If you have the desire and ability, it would be cool to see it in the "To Be Continued ..." :)

**RSATom** January 12, 2015 at 16:00 # ⍈ ↑ 0

I think that this is quite simply the first iteration of the implementation of the idea, and implemented "in haste" - I suppose that one of the next versions we will see completely redesigned concept - it was too many "goodies" promises this approach.

**RSATom** January 12, 2015 at 16:01 # ⍈ ↑ +1

And about the "To Be Continued ..." - I would be happy, we only think of how to increase the number of hours in a day at least to the 26;)

**FAQuiR** January 12, 2015 at 17:03 # ⍈ ↑ 0

Tell me, how do you know when rendering to occur?

**semlanik** January 12, 2015 at 17:22 # ⍈ ↑ 0

Slightly higher in the thread already written about it, and then the question arises, how Qt resets FBO in this case, because in fact the problem is the same. Although I may be wrong in some way. It is necessary to watch the code to understand.

**semlanik** January 12, 2015 at 15:35 # 0

Thanks for the article, as a whole it looks very sad. There was a desire to see how it works inside because crutch (the part of the Qt) a bit too clumsy ...

IOS Developer

all vacancies

Sticker design

all orders