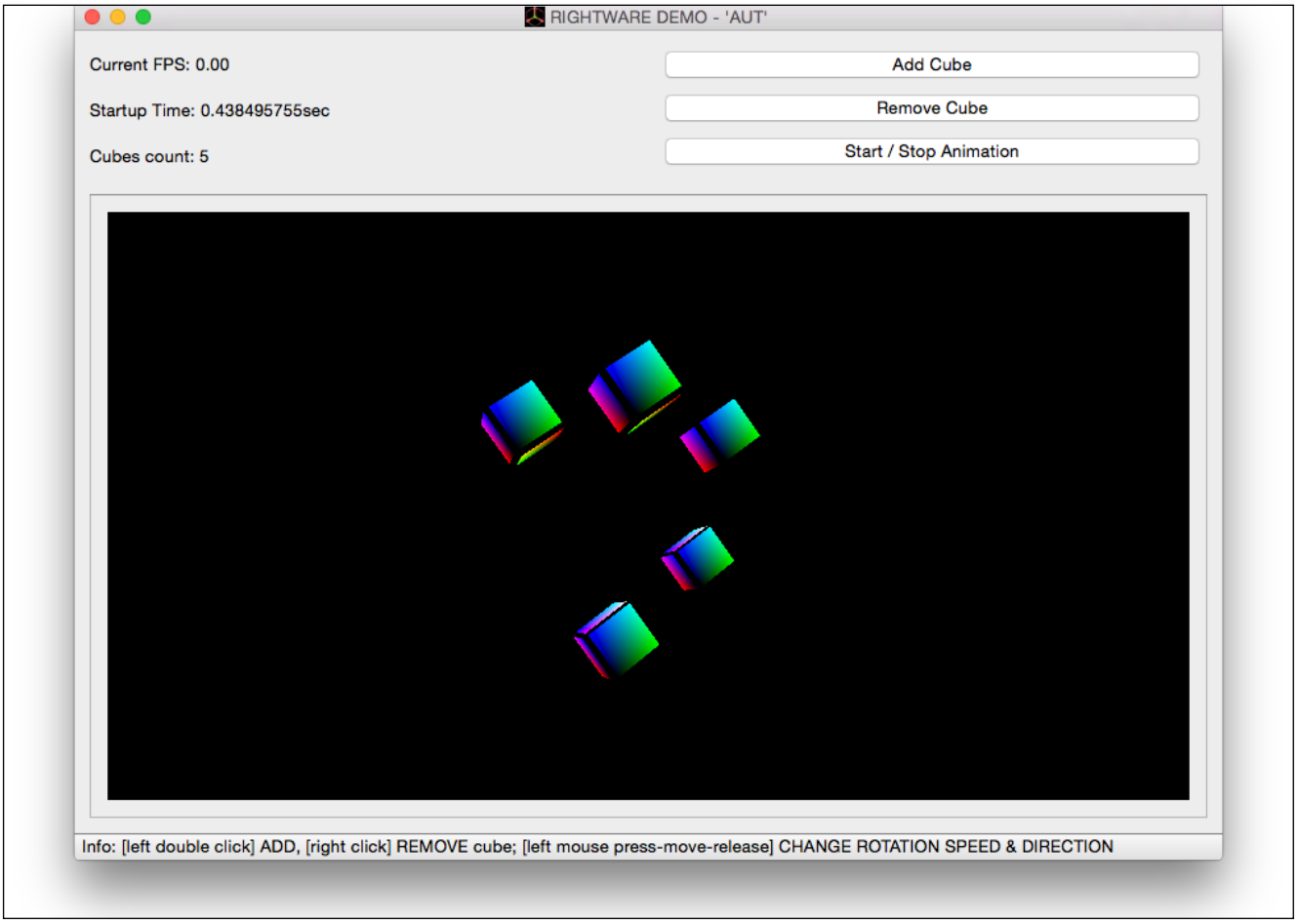
Rightware Demo

Interview test application

Goran Kostadinov - July 19, 2015



Introduction

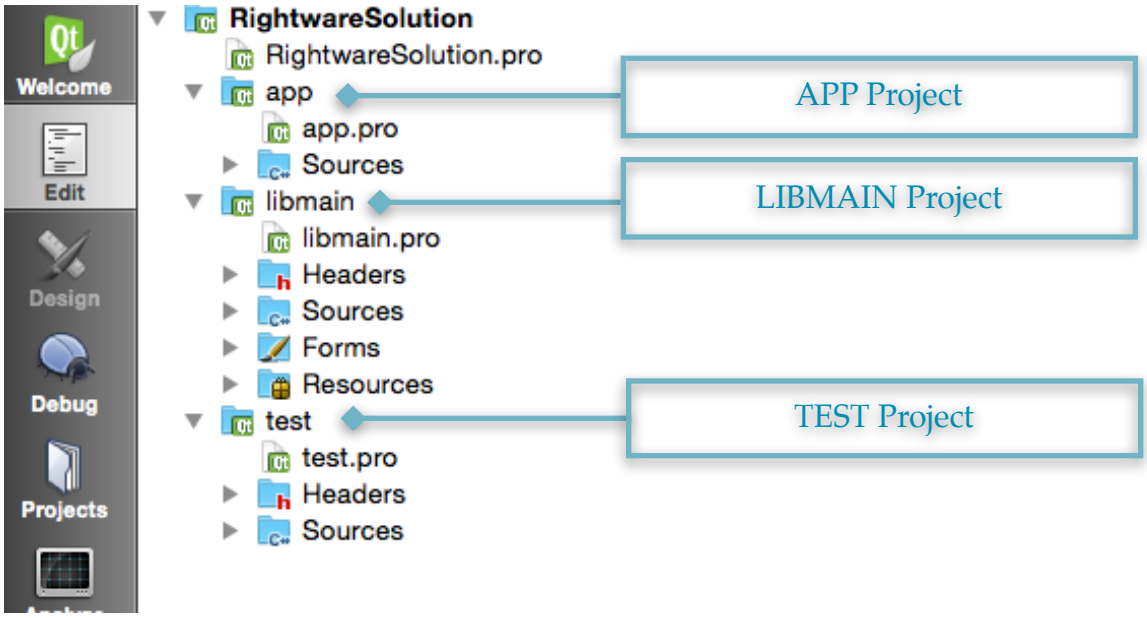
Task short description: build application which renders a scene with animated objects, measure FPS and startup time, build test application which run with incrementing workload, output result to xml file and generate code coverage report.

Application description

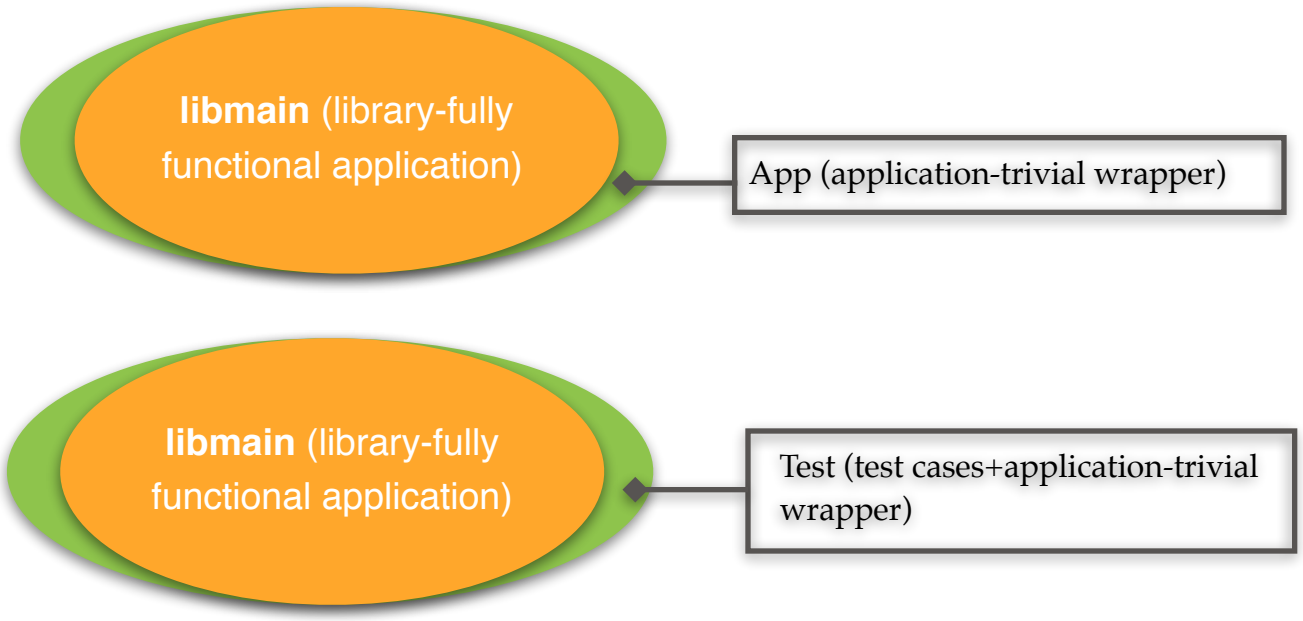
Application with all required documents and bundles can be dowloaded from:

https://github.com/gokostad/RightwareInterview

Application is build based on Qt multi-platform framework. To have testability feature, application is build as static linked library and moved to be separate project LIBMAIN, application project APP is just most trivial wrapper around the library and TEST project contains test cases and wrapper around library.



Project configuration



Application draws animated cubes which can be added or removed. Keeping left mouse button down, move and button release will accelerate moving, speed will depend of mouse moving distance and after button is released animation will deaccelerate to the normal running speed. Animation can be stopped and started at any time and, when stopped, cubes can be rotated with left mouse (down) movement. Application is measuring (class *helper_profiling*) FPS and startup time.

Main pattern used for building application (packed as library) is **MVC** pattern. Name of classes related to View have prefix *View*, classes related to controller has *Controler* prefix and model classes have the *Model* prefix. Helper classes have *Helper* prefix (used for measuring time in this project). All files and Test project files has comments at the beginning of file with description.

Test project has two test cases (*fpstest.cpp* and *startuptimetest.cpp*), file *waitforupdatedone.cpp* with corresponding class to stop test application running until main animated widget finishing drawing of scene and *xmltestreport.cpp* file with corresponding class to generate xml and html test results report.

Build instruction

1. Take source code from Github: https://github.com/gokostad/RightwareInterview

   It will contains:
   a) Source Code in folders: app, libmain, test
   b) Binaries folder:
      - under Release folder - application bundle App.app
      - under Release/test - test application which can be run by script ./testrun.sh
   c) REQUIRED FILES folder: with all required documentation files.

2. Install **Qt 5.5** on **OS X**, application can be build on Linux and Windows too with small changes related to c++11 and code coverage, for simplicity reason only OS X version is described here.

3. For code coverage **gcov**, **lcov** and **genhtml** tools are used. If they are not present on installation machine, follow next steps:

   a) Install, if already not, **Brew** (http://brew.sh/) with command (in terminal window):

   ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"

   b) Install lcov with command (in terminal window):

   brew install lcov

   Note: test application binary in Binaries folder uses lcov version 1.11, if lcov 1.10 is installed because of different command line options will not generate reports.

4. Start QtCreator and open solution file *RightwareSolution.pro*.

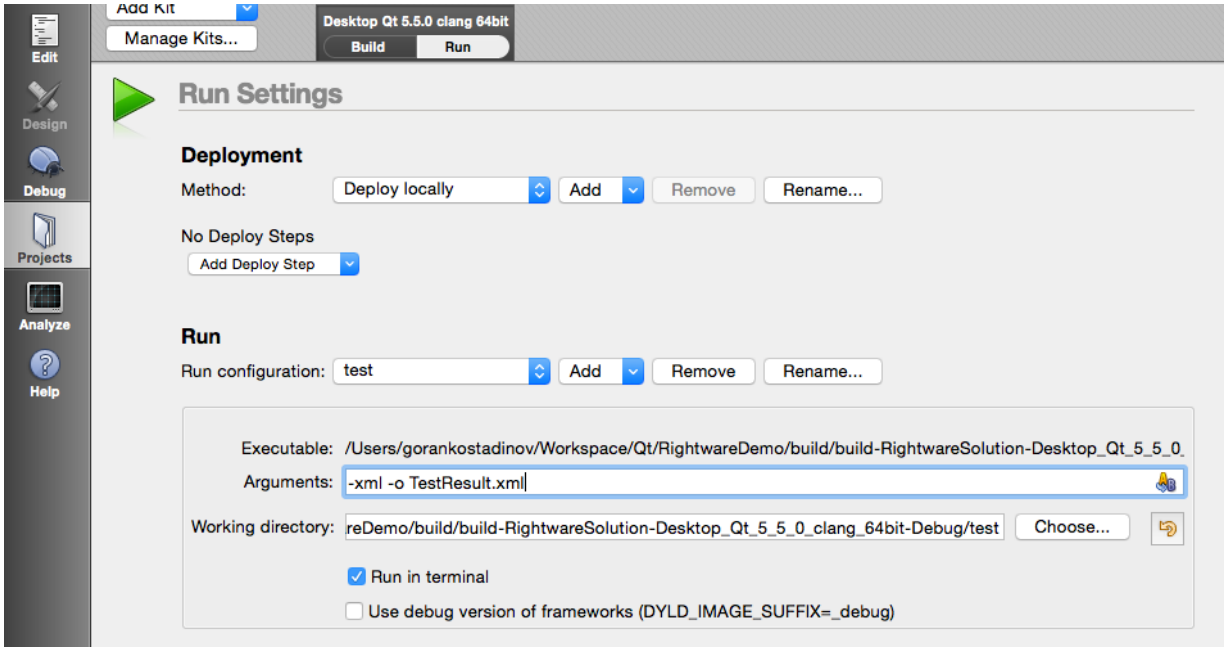
   a) Choose **clang 64bit** building kit. Compiler has to have C++11 support (and clang compiler has it).
   b) **Xcode** should be installed too on machine.
   c) app.pro, libmain.pro and test.pro project files contains everything what is needed for building application. Take a look into *contains*(MAC_CODE_COVERAGE, USE) section (at the end of project files). As project contains reference to library **clang_rt.profile_osx**, version 6.1.0 (visible in the path) be sure that the same version is installed on your machine. If not, delete that section completely and, for every project, add the same library (right click on project name, *Add Library* option and than add

External library, find clang_rt.profile_osx on your machine in similar path */Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/lib/clang/**6.1.0**/lib/darwin/* and after adding to every project, surround new section with *contains*(MAC_CODE_COVERAGE, USE) {…}.

d) Now click on RightwareSolution (top node) project, run **qmake** on it and than build. If plenty of warning message are present (-Winconsistent-missing-override), they are just known bug in some version of clang compiler and should be just ignored.

e) Now project is built in build folder.

In 'Projects' tab in the left menu, for Run configuration add next two arguments for application running (*-xml -o TestResult.xml*).



f) Copy *ApplicationTestResult.xsl* file from Test project folder to *build_folder/test*.

g) Run app project (application) and you should see application screen. After playing for a while close it.

h) Run test project . It will take a few minutes to all test cases have been executed. *build_folder/test* folder should now contains files:

CurrentDateTime_StartupTest_TestResult.xml with all StartupTimeTest cases result (PASS).
CurrentDateTime_FPSTest_TestResult.xml with FPSTest test cases result (PASS).
CurrentDateTime_ApplicationTestResult.xml output file with FPS and StartupTime measured data.

CurrentDateTime_ApplicationTestResult.html output file with HTML representation of FPS and StartupTime measured data. All of them are shown on picture below and excel presentation.

```
<?xml version="1.0" encoding="UTF-8"?>
<TestCase name="FPSTest">
<Environment>
    <QtVersion>5.5.0</QtVersion>
    <QtBuild>Qt 5.5.0 (x86_64&#x002D;little
    <QTestVersion>5.5.0</QTestVersion>
</Environment>
<TestFunction name="initTestCase">
<Incident type="pass" file="" line="0" />
    <Duration msecs="47.318086"/>
</TestFunction>
<TestFunction name="test">
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[0]]></DataTag>
</Incident>
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[1]]></DataTag>
</Incident>
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[2]]></DataTag>
</Incident>
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[5]]></DataTag>
</Incident>
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[10]]></DataTag>
</Incident>
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[20]]></DataTag>
</Incident>
<Incident type="pass" file="" line="0">
    <DataTag><![CDATA[50]]></DataTag>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<RightwareSolutionTesting>
<StartupTimeTest>
<StartupTimeMeasurement>
<Time>1.269785762</Time>
</StartupTimeMeasurement>
</StartupTimeTest>

<FPSTest>
<FPSMeasurement>
<TestNumber>1</TestNumber>
<GeometryCount>0</GeometryCount>
<FPS>59.82</FPS>
</FPSMeasurement>
<FPSMeasurement>
<TestNumber>2</TestNumber>
<GeometryCount>1</GeometryCount>
<FPS>59.85</FPS>
</FPSMeasurement>
```

StartupTime Measurement Test Result:

Time (sec)
0.674787819

FPS Measurement Test Results:

Test number	Geometry count	FPS
1	0	59.76
2	1	58.85
3	2	59.77
4	5	678.98
5	10	574.15
6	20	645.04
7	50	610.80
8	100	463.97
9	200	302.80
10	500	301.33
11	1000	220.41
12	2000	113.16
13	5000	51.35
14	10000	22.10
15	20000	0.17

	A	B	C	D
1	Time	TestNumber	GeometryCount	FPS
2	0.962804973	1	0	59.76
3	0.962804973	2	1	59.8
4	0.962804973	3	2	59.76
5	0.962804973	4	5	59.77
6	0.962804973	5	10	59.77
7	0.962804973	6	20	59.83
8	0.962804973	7	50	59.8
9	0.962804973	8	100	59.79
10	0.962804973	9	200	59.76
11	0.962804973	10	500	59.74
12	0.962804973	11	1000	59.77
13	0.962804973	12	2000	59.76
14	0.962804973	13	5000	51.69
15	0.962804973	14	10000	27
16	0.962804973	15	20000	0.25

Running test from binaries directly

- Application App.app can be directly run as normal OS X application even without Xcode or Qt 5.5 installation.

- As there is knowing problem with making bundle for testing qt application on mac (see https://bugreports.qt.io/browse/QTBUG-46366), to be able to run test application directly from Binaries folder, some steps must be done:

  a) make folder which contain test application Read/Writable for all users, as application will write test output result in that folder.
  b) make ***testrun.sh*** script executable (**chmod +x testrun.sh**)
  c) run test script in terminal as: ./testrun.sh
  d) Xcode and Qt 5.5 have to be installed to do code coverage reports, as lcov too (1.11), described in previous section. As this is test case it is acceptable requirements for testing machine. xml and html test report should be visible in the *./testreport* folder, and code coverage report in *./profile/genhtml* folder.

Code coverage

1. For running directly from QtCreator:

   After running *Test* project, go to *build_folder/libmain* and see that files with extension ***.gcda*** are presented there. They contains human unreadable code coverage results. Run in that folder next command in terminal window:

   lcov --capture --directory . --output-file coverage.info

   LCov tool will generate coverage.info file. To generate *html* report, in the same terminal enter command:

   genhtml coverage.info --output-directory profile

   After this, in *build_folder/libmain/profile* folder should be presented index.html file as entry point in code coverage report.

2. For running from terminal in *Binaries/Release/test*

   testrun.sh script should move all generated file into the ./profile folder and ./profile/genhtml folder should contains starting index.html code coverage file.

LCOV - code coverage report

Current view:	top level - /Users/gorankostadinov/Workspace/Qt/RightwareDemo/libmain		Hit	Total	Coverage
Test:	coverage.info	**Lines:**	241	290	83.1 %
Date:	2015-07-19 23:52:44	**Functions:**	76	93	81.7 %

Filename	Line Coverage			Functions	
controler.cpp		100.0 %	23 / 23	100.0 %	11 / 11
helper_profiling.cpp		93.3 %	28 / 30	100.0 %	8 / 8
helper_profiling.h		100.0 %	1 / 1	66.7 %	2 / 3
icontroler.h		100.0 %	1 / 1	100.0 %	1 / 1
iview.h		100.0 %	1 / 1	100.0 %	1 / 1
libmain.cpp		93.5 %	29 / 31	50.0 %	1 / 2
model.cpp		95.2 %	20 / 21	100.0 %	7 / 7
model.h		100.0 %	4 / 4	100.0 %	6 / 6
view_geometryengine.cpp		100.0 %	37 / 37	100.0 %	7 / 7
view_mainwidget.cpp		61.2 %	63 / 103	73.9 %	17 / 23
view_mainwidget.h		0.0 %	0 / 2	0.0 %	0 / 2
view_mainwindow.cpp		94.1 %	32 / 34	66.7 %	10 / 15
view_mainwindow.h		100.0 %	2 / 2	71.4 %	5 / 7

Generated by: LCOV version 1.11