Getting available graphics memory using Qt on Symbian

This article shows how to get the amount of graphics memory (total, used, private and shared) using Qt on Symbian.

Introduction

Out of memory (OOM) errors cause applications to behave incorrectly. For example, if an application that uses the device camera for video recording runs out of memory, it throws a `QMediaRecorder::ResourceError`. When the graphics memory runs out, the application commonly fails to upload graphic assets to the GPU which will result as "freeze" of some or all parts of the UI (note that these symptoms only happen if the OpenGl rendering system is used).

These issues can be prevented by disabling resource-intensive features, such as video recording, on devices that do not have enough graphics memory. This article describes how to find out the amount of graphics memory on a Qt-enabled Symbian device in order to prevent OOM errors related to camera from occurring.

Querying graphics memory information

To query the graphics memory information, use the `eglQueryProfilingDataNOK` function found in the EGL collection of the Symbian APIs. In the following code snippet, a class called `ProfilingData` is created with a public member variable for each type of graphics memory information we're interested in. The data is obtained by first initializing the EGL display connection with the `eglInitialize` function and then calling the `eglQueryProfilingDataNOK` function with the `EGL_PROF_QUERY_MEMORY_USAGE_BIT_NOK` flag.

```
#ifndef PROFILINGDATA_H
#define PROFILINGDATA_H

#include <qglobal.h>
#include <EGL/egl.h>

// Function pointer to query the graphics memory information.
typedef EGLBoolean (*PFNEGLQUERYPROFILINGDATANOK)(EGLDisplay, EGLint, EGLint *,
                                                  EGLint, EGLint *);

class ProfilingData {
public:
    ProfilingData()
        : m_totalMemory(0),
          m_usedMemory(0),
          m_privateMemory(0),
          m_sharedMemory(0)
    {
    }

    /*!
      Resolves profiling data information about the available graphics memory.
      Returns false if the function fails. After a succesful call to this
      function the data can be read from member variables.
    */
    bool resolveData()
    {
        PFNEGLQUERYPROFILINGDATANOK query;
        EGLDisplay display = eglGetDisplay(EGL_DEFAULT_DISPLAY);

        if (display != EGL_NO_DISPLAY) {
            query = reinterpret_cast<PFNEGLQUERYPROFILINGDATANOK>(
                        eglGetProcAddress("eglQueryProfilingDataNOK"));

            if (!query) {
                return false;
            }

            // Initialize the EGL display connection
            eglInitialize(display, 0, 0);

            QByteArray buffer;
            EGLint count = 0;

            // Find out the size of the data buffer
            query(display,
                  EGL_PROF_QUERY_MEMORY_USAGE_BIT_NOK |
                  EGL_PROF_QUERY_GLOBAL_BIT_NOK,
                  NULL, 0, &count);

            buffer.resize(count * sizeof(EGLint));
            buffer.fill(0);

            EGLint *data = reinterpret_cast<EGLint *>(buffer.data());

            // Query the graphics memory information
            query(display,
                  EGL_PROF_QUERY_MEMORY_USAGE_BIT_NOK |
                  EGL_PROF_QUERY_GLOBAL_BIT_NOK,
                  data, count, &count);

            // Fill the member variables
            for (int i = 0; i < count; i++) {
                switch (data[i]) {
                case EGL_PROF_TOTAL_MEMORY_NOK: {
                    m_totalMemory = data[++i];
                    break;
                }
                case EGL_PROF_USED_MEMORY_NOK: {
                    m_usedMemory = data[++i];
```


(C) Copyright Nokia 2013. All rights reserved.

```
                    break;
                }
                case EGL_PROF_PROCESS_USED_PRIVATE_MEMORY_NOK: {
                    m_privateMemory = data[++i];
                    break;
                }
                case EGL_PROF_PROCESS_USED_SHARED_MEMORY_NOK: {
                    m_sharedMemory = data[++i];
                    break;
                }
                default:
                    // Unknown data
                    break;
                }
            }
        }
        else {
            return false;
        }

        return true;
    }

public:
    qint64 m_totalMemory;   // Amount of total graphics memory
    qint64 m_usedMemory;    // Amount of used memory
    qint64 m_privateMemory; // Amount of used private memory
    qint64 m_sharedMemory;  // Amount of used shared memory
};

#endif // PROFILINGDATA_H
```

In order to use the above code, also add the EGL library to the .pro file of the project:

```
symbian:LIBS += -llibegl
```

After the `ProfilingData` class is created, it can be used like this:

```
#include "profilingdata.h"
```

```
ProfilingData profilingData;
if (profilingData.resolveData()) {
    qDebug() << "Total gfx memory: "
             << profilingData.m_totalMemory / 1024 / 1024
             << " MB";
    qDebug() << "Used memory: "
             << profilingData.m_usedMemory / 1024 / 1024
             << " MB";
    qDebug() << "Used private memory: "
             << profilingData.m_privateMemory / 1024 / 1024
             << " MB";
    qDebug() << "Used shared memory: "
             << profilingData.m_sharedMemory / 1024 / 1024
             << " MB";
}
```

Test Code

Test code which uses the above snippets is attached. The memory information is output to debug while debugging - note that the app has no real "UI" - it simply displays a white box on the screen which you can click to close the profiler.

- Media:GfxMemoryProfiling.zip - source code
- Media:GfxMemoryProfiling.sis - installation file

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.