MeeGo Camera VideoSurface manipulation

This article explains how to get the Qt camera example to work on MeeGo, and then how to manipulate the displayed video before it is written to the `VideoSurface`.

Note: This is an entry in the PureView Imaging Competition 2012Q2

Introduction

The QCamera example (at least in Qt 4.7) does not work on MeeGo, as discussed in the bug QTMOBILITY-1963. This article first explains why the example does not work, how you can modify it to enable support for the (required) UYVY color space, and provides a helper class you can use in your own projects.

The article then shows how you can manipulate the image prior to display. The example is relatively simple, but is sufficient to illustrate other possibilities for augmenting the video view in real-time:

1. Switch color components to help color-blind people
2. Add clothes on people without going to shop
3. We can merge visible reality with games

UYVY color space

The reason why `QCamera` example is not working on MeeGo is that MeeGo uses the UYVY color space, and this is not supported by QImage. Therefore at first you will need to do following changes to your class that subclasses QAbstractVideoSurface:

1. In function `supportedPixelFormats` you will need to add following line:

```
<< QVideoFrame::Format_UYVY
```

2. In start function you will need to add this code snippet:

```
if (format.pixelFormat() == QVideoFrame::Format_UYVY)
    imageFormat = QImage::Format_RGB32;
```

   The main problem here is that `imageFormatFromPixelFormat` will return `QImage::Format_Invalid` because QImage does not support UYVY.

3. Lastly we will need to handle UYVY frames ourselves. For that I have written helper class that you init with UYVY frame and get RGB image data:

```
if (m_frame.pixelFormat() == QVideoFrame::Format_UYVY)
    {
    uyvy2rgb.initData(m_frame);

    image = new QImage(uyvy2rgb.bits(),
                       uyvy2rgb.width(),
                       uyvy2rgb.height(),
                       uyvy2rgb.width() * 4,
                       m_imageFormat);
    }
```

I am supplying source code for this class below but let's analyze UYVY to RGB conversion algorithm. Based on Harmattan documentation and wikipedia article on YUV we can find out that Nokia is referring to Y'UV422. Therefore 4 bytes in UYVY represents 2 pixels and when converted to RGB they will occupy 8 bytes (that's because RGB pixel takes 4 bytes for performance/simplicity reasons). In wikipedia article we can find algorithm for converting from UYVY color space to RGB color space:

```
C = Y - 16
D = U - 128
E = V - 128
R = clamp((298 * C + 409 * E + 128) >> 8)
G = clamp((298 * C - 100 * D - 208 * E + 128) >> 8)
B = clamp((298 * C + 516 * E + 128) >> 8)
```

Algorithm looks quite simple and here we can do our first optimization. What we know that we have 256 different C, D and E values - therefore we can pre-calculate multiplications before using them. Multiplications are more expensive than additions and subtractions in CPU and we only need to about 4kb of memory to keep pre-calculated multiplications. If we care about performance even more we can hardcode pre-calculated arrays into the code. Here is code for helper class:

uyvy2rgb.h:

```
#ifndef UYVY2RGB_H
#define UYVY2RGB_H

#include <QVideoFrame>

class UYVY2RGB : public QObject
{
    Q_OBJECT

    public:
        UYVY2RGB();
        ~UYVY2RGB();
```


(C) Copyright Nokia 2013. All rights reserved.

```
    public:
        void initData(const QVideoFrame &source);

        unsigned char* bits();
        int width();
        int height();

    private:
        int *c298, *e409, *d100, *e208, *d516;

        unsigned char *m_bits;
        int m_width;
        int m_height;
};
#endif // UYVY2RGB_H
```

uyvy2rgb.cpp:

```
#include "uyvy2rgb.h"

#include <QtCore/qmath.h>

UYVY2RGB::UYVY2RGB() : m_bits(NULL), m_width(0), m_height(0)
{
    int i;
    c298 = new int[256];
    for (i = 0; i < 256; i++)
        c298[i] = (i-16)*298;

    e409 = new int[256];
    e208 = new int[256];

    for (i = 0; i < 256; i++) {
        e409[i] = (i-128)*409;
        e208[i] = (i-128)*208;
    }

    d100 = new int[256];
    d516 = new int[256];

    for (i = 0; i < 256; i++) {
        d100[i] = (i-128)*100;
        d516[i] = (i-128)*516;
    }
}

UYVY2RGB::~UYVY2RGB()
{
    delete c298;
    delete e409;
    delete d100;
    delete e208;
    delete d516;
}

unsigned char* UYVY2RGB::bits()
{
    return m_bits;
}

int UYVY2RGB::width()
{
    return m_width;
}

int UYVY2RGB::height()
{
    return m_height;
}

void UYVY2RGB::initData(const QVideoFrame &source)
{
    if (m_bits == NULL || source.width() != m_width || source.height() != m_height)
    {
        if (m_bits != NULL)
            delete[] m_bits;

        m_width = source.width();
        m_height = source.height();
        m_bits = new unsigned char[m_width * m_height * 4];
    }

    int steps = (m_width/2)*m_height;

    unsigned char *dst = m_bits;
    const unsigned char *src = source.bits();

    for (int i = 0; i < steps; ++i)
    {
        int d   = *src    ; // u
        int y1  = *(src+1); // y
        int e   = *(src+2); // v
        int y2  = *(src+3); // y

        int rpart = e409[e] + 128;
        int gpart = d100[d] + e208[e] + 128;
        int bpart = d516[d] + 128;

        int r = (c298[y1] + rpart) >> 8;
        int g = (c298[y1] - gpart) >> 8;
        int b = (c298[y1] + bpart) >> 8;

        r = r < 0 ? 0 : (r > 255 ? 255 : r);
        g = g < 0 ? 0 : (g > 255 ? 255 : g);
        b = b < 0 ? 0 : (b > 255 ? 255 : b);

        *dst = b;
        *(dst+1) = g;
        *(dst+2) = r;
        *(dst+3) = 0;

        dst+=4;

        r = (c298[y2] + rpart) >> 8;
        g = (c298[y2] - gpart) >> 8;
        b = (c298[y2] + bpart) >> 8;
```


(C) Copyright Nokia 2013. All rights reserved.

```
        r = r < 0 ? 0 : (r > 255 ? 255 : r);
        g = g < 0 ? 0 : (g > 255 ? 255 : g);
        b = b < 0 ? 0 : (b > 255 ? 255 : b);

        *dst = b;
        *(dst+1) = g;
        *(dst+2) = r;
        *(dst+3) = 0;

        dst+=4;
        src+=4;
    }
}
```

Download sample application from here: File:Camera uyvy.zip

Here is sample image from camera:

View image manipulation

After this view image manipulation is easy part. E.g. for swapping RGB colors we can get image data using `bits()` function and swap data in place. `QImage` actually has `rgbSwapped()` function we will use in this case. Drawing image on top of view is as simple as calling `drawImage()` on `painter` object. The only thing we should be concerned about is that images should support transparency. I have drawn some images using Inkscape and exported them to PNG because PNG supports transparency (we can actually use SVG files if we need them to be resizable but that falls out of scope of this article). Here is sample code I have used to swap colors and draw image on top of view:

```
painter->drawImage(centerPic, image->rgbSwapped());

QImage pic = QImage(m_pic == 0 ? ":/icons/gentleman.png" : ":/icons/dress.png");

QPoint c((r.size().width()-pic.size().width()) / 2, (r.size().height()-pic.size().height()) / 2);
painter->drawImage(c, pic);
```

Download sample application from here: File:Camera pic.zip

Here is sample image you can get using this application (notice box color - it is the same box as above):

(C) Copyright Nokia 2013. All rights reserved.

Summary

This article show how to display camera's viewport in Meego Harmattan, and how to manipulate it before displaying it for users.

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.