Photo effects with Qt

This article explains how to apply a set of visual effects to pictures taken with the Camera.

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

Introduction

This article shows how to use the QML Camera API, and apply effects to the pictures taken.

Summary

First we create a simple QML UI that will allow us to access the Camera and take pictures. The first part of the QML file contains the Camera Object and a Image object that will allow us to show a preview of the picture and of the applied effect.

```
import QtQuick 1.1
import QtMultimediaKit 1.1
import com.nokia.meego 1.0

Page {
    orientationLock: PageOrientation.LockLandscape
    Rectangle {
        anchors.fill: parent
        Camera {

            anchors.left: parent.left
            id: camera
            x: 0
            y: 0
            height: parent.height
            focus: visible
            captureResolution : "1024x768"

            onImageSaved: {
                imgpreview.source = camera.capturedImagePath
                imgpreview.visible = true
            }
        }

        Image {
            id: imgpreview
            anchors.left: parent.left
            anchors.right: stillControls.left
            anchors.top: parent.top
            anchors.bottom: parent.bottom

            visible: false
            MouseArea {
                anchors.fill: parent
                onClicked: {
                    imgpreview.visible = false

                }
            }
        }
```

The following Rectangle contains the controls:

```
        Rectangle {
            id: stillControls
            width: 200
            anchors.right: parent.right

            Button{
                anchors.left: parent.left
                id: capture
                width: parent.width
                text: qsTr("Capture")
                onClicked: {
                    camera.captureImage();
                }
            }

            Button{
                id: bw
                anchors.left: parent.left
                anchors.top: capture.bottom
                width: parent.width
                text: qsTr("old look")
                onClicked: {
                    imgpreview.source = helper.oldPicture(camera.capturedImagePath)
                }
            }
            Button{
                id: pixelate
                anchors.top: bw.bottom
                anchors.left: parent.left
                width: parent.width
                text: qsTr("rasterbator")
                onClicked: {
                    imgpreview.source = helper.rasterbator(camera.capturedImagePath)
                }
            }
```


(C) Copyright Nokia 2013. All rights reserved.

```
            Button{
                id: popart
                anchors.top: pixelate.bottom
                anchors.left: parent.left
                width: parent.width
                text: qsTr("popart")
                onClicked: {
                    imgpreview.source = helper.popart(camera.capturedImagePath)
                }
            }
            Button{
                anchors.top: popart.bottom
                anchors.left: parent.left
                width: parent.width
                text: qsTr("Quit")
                onClicked: {
                    Qt.quit()
                }
            }
        }
    }
}
```

In our helper class we create the following methods, here we receive the filename of the image to process and save a new one with the desired effect.

Old picture

For the Old picture look effect, we talke the original image and convert it to an indexed image. We calculate a table of colors with the average of them. This will give it the old look effect, after that we apply the new color to the image.

```
Q_INVOKABLE QString oldPicture(const QString &filename)
   {
       if (m_photo.load(filename)) {
           qDebug() << filename;

           QImage file_in = m_photo.convertToFormat(QImage::Format_Indexed8);
           QVector<int> transform_table(file_in.numColors());
           for(int i=0;i<file_in.numColors();i++)
           {
               QRgb c1 = file_in.color(i);
               int avg = qGray(c1);
               transform_table[i] = avg;
           }
           file_in.setNumColors(256);
           for(int i=0;i<256;i++)
               file_in.setColor(i,qRgb(i,i,i));

           for(int i=0;i<file_in.numBytes();i++)
           {
               file_in.bits()[i]=transform_table[file_in.bits()[i]];
           }

           file_in.save(filename+"_gray.jpg", 0, -1);
           return filename+"_gray.jpg";
       }
       return filename;
   }
```

This is the processed file:

Image

(C) Copyright Nokia 2013. All rights reserved.

Processed

Rasterbator

The rasterbator() function applies the Tiled printing [1] effecct to the image. It works by reading every pixel and drawing a new image where a ellipsis is drawn for each pixel. The radius of the ellipsis is calculated by getting the average color of the pixel. We can use the color of the pixel to draw the ellipsis or just paint it black.

```
Q_INVOKABLE QString rasterbator(const QString &filename)
{
    if (m_photo.load(filename)) {
        int pixelSize = 10;
        m_photo = m_photo.scaled(m_photo.width()/pixelSize, m_photo.height()/pixelSize,Qt::KeepAspectRatio);
        QImage *newImage = new QImage(m_photo.width()*pixelSize, m_photo.height()*pixelSize, QImage::Format_ARGB32);

        QPainter painter;
        QColor oldColor;
        painter.begin(newImage);
        painter.fillRect(QRect(0,0,m_photo.width()*pixelSize,m_photo.height()*pixelSize), Qt::white);
        painter.setBrush(QBrush(Qt::black));
        painter.setRenderHint(QPainter::Antialiasing, true);
        painter.setPen(Qt::NoPen);
        int r, g, b;
        double radius;
        for(int x=0; x<m_photo.width(); x++){
            for(int y=0; y<m_photo.height(); y++){
                oldColor = QColor(m_photo.pixel(x,y));
                r = oldColor.red();
                g = oldColor.green();
                b = oldColor.blue();
                painter.setBrush(QBrush(oldColor));
                radius = pixelSize/2.0 - (((r+g+b)/3)/255.0 * pixelSize/2.0);
                if (radius == 0.0)
                    continue;
                painter.drawEllipse( QPointF(x*pixelSize+pixelSize/2, y*pixelSize+pixelSize/2), radius, radius);
            }
        }
        painter.end();
        newImage->save(filename+"_rasterbator.jpg", 0, -1);
        return filename+"_rasterbator.jpg";
    }
    return filename;
}
```

This is the processed file:

Image

(C) Copyright Nokia 2013. All rights reserved.

Processed

Pop art

The popart() effect tries to mimic this effect by swaping the colors of the image and producing a new image.

```
Q_INVOKABLE QString popart(const QString &filename)
{
    if (m_photo.load(filename)) {
        QImage *newImage = new QImage(m_photo.width(), m_photo.height(), QImage::Format_ARGB32);
        m_photo = m_photo.scaled(m_photo.width()/2, m_photo.height()/2,Qt::KeepAspectRatio);
        QPainter painter;
        painter.begin(newImage);
        int r, g, b;
        int h,s,l;
        painter.drawImage(0,0, m_photo);
        QImage img2 = m_photo.copy();
        QImage img3 = m_photo.copy();
        QImage img4 = m_photo.copy();

        QColor oldColor, newColor;

        for(int x=0; x<img2.width(); x++){
            for(int y=0; y<img2.height(); y++){
                oldColor = QColor(img2.pixel(x,y));
                newColor = oldColor.toHsl();
                h = newColor.hue()+20;
                s = newColor.saturation()+80;
                l = newColor.lightness();
                s = qBound(0, s, 255);
                h = qBound(0, h, 255);
                newColor.setHsl(h, s, l);
                r = oldColor.red();
                g = oldColor.green();
                b = oldColor.blue();
                img2.setPixel(x, y, qRgb(g, b, r));
                img3.setPixel(x, y, qRgb(b, r, g));
                img4.setPixel(x, y, qRgb(newColor.red(), newColor.green(), newColor.blue()));
            }
        }
        painter.drawImage(m_photo.width(),  0, img2);
        painter.drawImage(0,  m_photo.height(), img3);
        painter.drawImage(m_photo.width(),  m_photo.height(), img4);
        painter.end();
        newImage->save(filename+"_pop.jpg", 0, -1);
        return filename+"_pop.jpg";

    }
    return filename;
}
```

This is the processed file:

Image

(C) Copyright Nokia 2013. All rights reserved.

Processed

Download

You can download source code demo from this link. File:Quickcam.tar.gz

References

Tiled printing

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.