Archived:Animated effects with the Qt Graphics View Framework

Archived: This article is archived because it is not considered relevant for third-party developers creating commercial solutions today. If you think this article is still relevant, let us know by adding the template {{ReviewForRemovalFromArchive|user=~~~~|*write your reason here*}}.

Qt Quick should be used for all UI development on mobile devices. The approach described in this article (using C++ for the Qt app UI) is deprecated.

This code snippet shows how to create animated graphics effect with the Qt's Graphics View Framework.

The media player is loading...

Implementation

The right way to apply effects to Graphics items is to use QGraphicsEffect based classes. Few pre-build effects are already part of the framework and those are not animated. Since QGraphicsEffect is a QObject derived class, creating animated effects is possible. To achieve that, you need a timer which is started by `QObject::startTimer(int interval)` and which will call the `virtual timerEvent()` function of your QObject derived class every N milliseconds as specified by the *interval*.

Using an interval of 17 ms permits the animation to run at about 58 FPS, which is a good frame-rate.

So every 17ms the timerEvent slot updates the parameter (the rotation angle in this case) which drives the animation and it calls `update()` to draw again the item using `MyQGraphicsEffect::draw` function.

Since painter transformations don't affect QGraphicsEffect::drawSource (at least in the case shown here), a QImage has been used as device buffer for an ad-hoc painter. Eventually the buffer, which contains the animation frames, is drawn on the screen by the application painter.

Code

Here is the graphics effect class which rotates the items around the Y axis.

```
#ifndef GRAPHICSEFFECT_H
#define GRAPHICSEFFECT_H

#include <QGraphicsEffect>
#include <QGraphicsItem>
#include <QPainter>

class MyQGraphicsEffect : public QGraphicsEffect
{
    Q_OBJECT;

public:
    MyQGraphicsEffect(QObject *parent = 0)
    {
        QObject::startTimer(17); // About 60 FPS
    }

    virtual ~MyQGraphicsEffect(){};

protected:
    virtual void draw(QPainter *painter){

        QSizeF s = this->sourceBoundingRect().size();

        QImage buff(s.width(), s.height(), QImage::Format_ARGB32);

        QPainter p(&buff);
        p.fillRect(0, 0, s.width(), s.height(), Qt::black);

        QTransform t;
        t.translate(+s.width()/2,+s.height()/2);
        t.rotate(mAngle, Qt::YAxis);
        t.translate(-s.width()/2,-s.height()/2);

        p.setTransform(t, true);
        drawSource(&p);

        QRectF target(0, 0, s.width(), s.height());
        QRectF source(0, 0, s.width(), s.height());

        painter->drawImage(target, buff, source);
    }

protected:
    void timerEvent(QTimerEvent *event){
        Q_UNUSED(event);

        mAngle = (mAngle >= 360) ? 0 : mAngle+= 1;

        update();
    }

private:
    qreal mAngle;
};

#endif // GRAPHICSEFFECT_H
```


(C) Copyright Nokia 2013. All rights reserved.

This effect, as any other effect, can be applied to any item as shown in the following code:

```
#include <QtGui/QApplication>
#include <QGraphicsView>
#include <QGraphicsScene>
#include <QGraphicsPixmapItem>
#include <QGraphicsEffect>
#include "graphicseffect.h"

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    QGraphicsView view;
    view.setScene(new QGraphicsScene);
    QGraphicsPixmapItem *p = view.scene()->addPixmap(QPixmap(":/meditate.png"));

    MyQGraphicsEffect e;
    p->setGraphicsEffect(&e);

    view.show();
    return a.exec();
}
```

Full code

Full source code is available here: Media:Graphicseffects-animated.zip

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.