Cover Flow With Qt

Description

This widget displays images with animated transition effect similar to Apple's Cover Flow. It has been built using Qt C++ and expected to work on multiple platform including Symbian, Windows, MeeGo. It doesn't need OpenGL and 3D hardware acceleration.

Prerequisite

Put some images into a folder (images) and paste it inside the project folder.

How to start

Create a new Qt C++ project using Qt Creator and name it CowerFlow. Following is the code description of each created file in the project CowerFlow.

CoverFlow.pro

```
TEMPLATE = app
TARGET = CoverFlow
QT += core \
    gui
HEADERS += inc/CoverFlowQt.h \
    CoverFlow.h
SOURCES += src/CoverFlowQt.cpp \
    CoverFlow_reg.rss \
    main.cpp \
    CoverFlow.cpp
FORMS += CoverFlow.ui
RESOURCES +=
symbian:TARGET.UID3 = 0xEA524E4F

myFiles.sources = images\*.jpg
DEPLOYMENT += myFiles
```

CoverFlowQt.h

```
#ifndef COVERFLOWQT_H
#define COVERFLOWQT_H

#include <qwidget.h>

class CoverFlowQtPrivate;



class CoverFlowQt : public QWidget
{
Q_OBJECT

  Q_PROPERTY(QColor backgroundColor READ backgroundColor WRITE setBackgroundColor)
  Q_PROPERTY(QSize slideSize READ slideSize WRITE setSlideSize)
  Q_PROPERTY(int slideCount READ slideCount)
```


(C) Copyright Nokia 2013. All rights reserved.

```
  Q_PROPERTY(int centerIndex READ centerIndex WRITE setCenterIndex)

public:

  enum ReflectionEffect
  {
    NoReflection,
    PlainReflection,
    BlurredReflection
  };

  /*!
    Creates a new CoverFlowQt widget.
  */
  CoverFlowQt(QWidget* parent = 0);

  /*!
    Destroys the widget.
  */
  ~CoverFlowQt();

  /*!
    Returns the background color.
  */
  QColor backgroundColor() const;

  /*!
    Sets the background color. By default it is black.
  */
  void setBackgroundColor(const QColor& c);

  /*!
    Returns the dimension of each slide (in pixels).
  */
  QSize slideSize() const;

  /*!
    Sets the dimension of each slide (in pixels).
  */
  void setSlideSize(QSize size);

  /*!
    Returns the total number of slides.
  */
  int slideCount() const;

  /*!
    Returns QImage of specified slide.
  */
  QImage slide(int index) const;

  /*!
    Returns the index of slide currently shown in the middle of the viewport.
  */
  int centerIndex() const;

  /*!
    Returns the effect applied to the reflection.
  */
  ReflectionEffect reflectionEffect() const;

  /*!
    Sets the effect applied to the reflection. The default is PlainReflection.
  */
  void setReflectionEffect(ReflectionEffect effect);


public slots:

  /*!
    Adds a new slide.
  */
  void addSlide(const QImage& image);

  /*!
    Adds a new slide.
  */
  void addSlide(const QPixmap& pixmap);

  /*!
    Sets an image for specified slide. If the slide already exists,
    it will be replaced.
  */
  void setSlide(int index, const QImage& image);

  /*!
    Sets a pixmap for specified slide. If the slide already exists,
    it will be replaced.
  */
  void setSlide(int index, const QPixmap& pixmap);

  /*!
    Sets slide to be shown in the middle of the viewport. No animation
    effect will be produced, unlike using showSlide.
  */
  void setCenterIndex(int index);

  /*!
    Clears all slides.
  */
  void clear();

  /*!
    Shows previous slide using animation effect.
  */
  void showPrevious();

  /*!
    Shows next slide using animation effect.
  */
  void showNext();

  /*!
    Go to specified slide using animation effect.
  */
  void showSlide(int index);

  /*!
    Rerender the widget. Normally this function will be automatically invoked
    whenever necessary, e.g. during the transition animation.
```


(C) Copyright Nokia 2013. All rights reserved.

```
  */
  void render();

  /*!
    Schedules a rendering update. Unlike render(), this function does not cause
    immediate rendering.
  */
  void triggerRender();

signals:
  void centerIndexChanged(int index);

protected:
  void paintEvent(QPaintEvent *event);
  void keyPressEvent(QKeyEvent* event);
  void mousePressEvent(QMouseEvent* event);
  void resizeEvent(QResizeEvent* event);

private slots:
  void updateAnimation();

private:
  CoverFlowQtPrivate* d;
};

#endif // COVERFLOWQT_H
```

CoverFlowQt.cpp

```
#include "inc\CoverFlowQt.h"
#include <QMessageBox>
// detect Qt version
#if QT_VERSION >= 0x040000
#define COVERFLOWQT_QT4
#elif QT_VERSION >= 0x030000
#define COVERFLOWQT_QT3
#elif QT_VERSION >= 235
#define COVERFLOWQT_QT2
#else
#error CoverFlowQt widgets need Qt 2, Qt 3 or Qt 4
#endif

#ifdef COVERFLOWQT_QT4
#include <QApplication>
#include <QCache>
#include <QHash>
#include <QImage>
#include <QKeyEvent>
#include <QPainter>
#include <QPixmap>
#include <QTimer>
#include <QVector>
#include <QWidget>
#endif

#ifdef COVERFLOWQT_QT3
#include <qapplication.h>
#include <qcache.h>
#include <qimage.h>
#include <qpainter.h>
#include <qpixmap.h>
#include <qdatetime.h>
#include <qtimer.h>
#include <qvaluevector.h>
#include <qwidget.h>

#define qMax(x,y) ((x) > (y)) ? (x) : (y)
#define qMin(x,y) ((x) < (y)) ? (x) : (y)

#define QVector QValueVector

#define toImage convertToImage
#define contains find
#define modifiers state
#define ControlModifier ControlButton
#endif

#ifdef COVERFLOWQT_QT2
#include <qapplication.h>
#include <qarray.h>
#include <qcache.h>
#include <qimage.h>
#include <qintdict.h>
#include <qpainter.h>
#include <qpixmap.h>
#include <qdatetime.h>
#include <qtimer.h>
#include <qwidget.h>

#define qMax(x,y) ((x) > (y)) ? (x) : (y)
#define qMin(x,y) ((x) < (y)) ? (x) : (y)

#define QVector QArray

#define toImage convertToImage
#define contains find
#define modifiers state
#define ControlModifier ControlButton
#define flush flushX
#endif

// for fixed-point arithmetic, we need minimum 32-bit long
// long long (64-bit) might be useful for multiplication and division
typedef long PFreal;
#define PFREAL_SHIFT 10
#define PFREAL_ONE (1 << PFREAL_SHIFT)

#define IANGLE_MAX 1024
#define IANGLE_MASK 1023

inline PFreal fmul(PFreal a, PFreal b)
{
```


(C) Copyright Nokia 2013. All rights reserved.

```
  return ((long long)(a))*((long long)(b)) >> PFREAL_SHIFT;
}

inline PFreal fdiv(PFreal num, PFreal den)
{
  long long p = (long long)(num) << (PFREAL_SHIFT*2);
  long long q = p / (long long)den;
  long long r = q >> PFREAL_SHIFT;

  return r;
}

inline PFreal fsin(int iangle)
{
  // warning: regenerate the table if IANGLE_MAX and PFREAL_SHIFT are changed!
  static const PFreal tab[] = {
     3,    103,    202,    300,    394,    485,    571,    652,
   726,    793,    853,    904,    947,    980,   1004,   1019,
  1023,   1018,   1003,    978,    944,    901,    849,    789,
   721,    647,    566,    479,    388,    294,    196,     97,
    -4,   -104,   -203,   -301,   -395,   -486,   -572,   -653,
  -727,   -794,   -854,   -905,   -948,   -981,  -1005,  -1020,
 -1024,  -1019,  -1004,   -979,   -945,   -902,   -850,   -790,
  -722,   -648,   -567,   -480,   -389,   -295,   -197,    -98,
  3
  };

  while(iangle < 0)
    iangle += IANGLE_MAX;
  iangle &= IANGLE_MASK;

  int i = (iangle >> 4);
  PFreal p = tab[i];
  PFreal q = tab[(i+1)];
  PFreal g = (q - p);
  return p + g * (iangle-i*16)/16;
}

inline PFreal fcos(int iangle)
{
  return fsin(iangle + (IANGLE_MAX >> 2));
}



struct SlideInfo
{
  int slideIndex;
  int angle;
  PFreal cx;
  PFreal cy;
  int blend;
};

class CoverFlowQtState
{
public:
  CoverFlowQtState();
  ~CoverFlowQtState();

  void reposition();
  void reset();

  QRgb backgroundColor;
  int slideWidth;
  int slideHeight;
  CoverFlowQt::ReflectionEffect reflectionEffect;
  QVector<QImage*> slideImages;

  int angle;
  int spacing;
  PFreal offsetX;
  PFreal offsetY;

  SlideInfo centerSlide;
  QVector<SlideInfo> leftSlides;
  QVector<SlideInfo> rightSlides;
  int centerIndex;
};

class CoverFlowQtAnimator
{
public:
  CoverFlowQtAnimator();
  CoverFlowQtState* state;

  void start(int slide);
  void stop(int slide);
  void update();

  int target;
  int step;
  int frame;
  QTimer animateTimer;
};

class CoverFlowQtAbstractRenderer
{
public:
  CoverFlowQtAbstractRenderer(): state(0), dirty(false), widget(0) {}
  virtual ~CoverFlowQtAbstractRenderer() {}

  CoverFlowQtState* state;
  bool dirty;
  QWidget* widget;

  virtual void init() = 0;
  virtual void paint() = 0;
};

class CoverFlowQtSoftwareRenderer: public CoverFlowQtAbstractRenderer
{
public:
  CoverFlowQtSoftwareRenderer();
  ~CoverFlowQtSoftwareRenderer();

  virtual void init();
  virtual void paint();
```


(C) Copyright Nokia 2013. All rights reserved.

```
private:
  QSize size;
  QRgb bgcolor;
  int effect;
  QImage buffer;
  QVector<PFreal> rays;
  QImage* blankSurface;
#ifdef COVERFLOWQT_QT4
  QCache<int,QImage> surfaceCache;
  QHash<int,QImage*> imageHash;
#endif
#ifdef COVERFLOWQT_QT3
  QCache<QImage> surfaceCache;
  QMap<int,QImage*> imageHash;
#endif
#ifdef COVERFLOWQT_QT2
  QCache<QImage> surfaceCache;
  QIntDict<QImage> imageHash;
#endif

  void render();
  void renderSlides();
  QRect renderSlide(const SlideInfo &slide, int col1 = -1, int col2 = -1);
  QImage* surface(int slideIndex);
};

// ------------- CoverFlowQtState -------------------------------------

CoverFlowQtState::CoverFlowQtState():
backgroundColor(0), slideWidth(150), slideHeight(200),
reflectionEffect(CoverFlowQt::BlurredReflection), centerIndex(0)
{
}

CoverFlowQtState::~CoverFlowQtState()
{
  for(int i = 0; i < (int)slideImages.count(); i++)
    delete slideImages[i];
}

// readjust the settings, call this when slide dimension is changed
void CoverFlowQtState::reposition()
{
  angle = 70 * IANGLE_MAX / 360;  // approx. 70 degrees tilted

  offsetX = slideWidth/2 * (PFREAL_ONE-fcos(angle));
  offsetY = slideWidth/2 * fsin(angle);
  offsetX += slideWidth * PFREAL_ONE;
  offsetY += slideWidth * PFREAL_ONE / 4;
  spacing = 40;
}

// adjust slides so that they are in "steady state" position
void CoverFlowQtState::reset()
{
  centerSlide.angle = 0;
  centerSlide.cx = 0;
  centerSlide.cy = 0;
  centerSlide.slideIndex = centerIndex;
  centerSlide.blend = 256;

  leftSlides.resize(6);
  for(int i = 0; i < (int)leftSlides.count(); i++)
  {
    SlideInfo& si = leftSlides[i];
    si.angle = angle;
    si.cx = -(offsetX + spacing*i*PFREAL_ONE);
    si.cy = offsetY;
    si.slideIndex = centerIndex-1-i;
    si.blend = 256;
    if(i == (int)leftSlides.count()-2)
      si.blend = 128;
    if(i == (int)leftSlides.count()-1)
      si.blend = 0;
  }

  rightSlides.resize(6);
  for(int i = 0; i < (int)rightSlides.count(); i++)
  {
    SlideInfo& si = rightSlides[i];
    si.angle = -angle;
    si.cx = offsetX + spacing*i*PFREAL_ONE;
    si.cy = offsetY;
    si.slideIndex = centerIndex+1+i;
    si.blend = 256;
    if(i == (int)rightSlides.count()-2)
      si.blend = 128;
    if(i == (int)rightSlides.count()-1)
      si.blend = 0;
  }
}

// ------------- CoverFlowQtAnimator  --------------------------------------

CoverFlowQtAnimator::CoverFlowQtAnimator():
state(0), target(0), step(0), frame(0)
{
}

void CoverFlowQtAnimator::start(int slide)
{
  target = slide;
  if(!animateTimer.isActive() && state)
  {
    step = (target < state->centerSlide.slideIndex) ? -1 : 1;
    animateTimer.start(30);
  }
}

void CoverFlowQtAnimator::stop(int slide)
{
  step = 0;
  target = slide;
  frame = slide << 16;
  animateTimer.stop();
}

void CoverFlowQtAnimator::update()
{
  if(!animateTimer.isActive())
```


(C) Copyright Nokia 2013. All rights reserved.

```
    return;
  if(step == 0)
    return;
  if(!state)
    return;

  int speed = 16384/4;

#if 1
  // deaccelerate when approaching the target
  const int max = 2 * 65536;

  int fi = frame;
  fi -= (target << 16);
  if(fi < 0)
    fi = -fi;
  fi = qMin(fi, max);

  int ia = IANGLE_MAX * (fi-max/2) / (max*2);
  speed = 512 + 16384 * (PFREAL_ONE+fsin(ia))/PFREAL_ONE;
#endif

  frame += speed*step;

  int index = frame >> 16;
  int pos = frame & 0xffff;
  int neg = 65536 - pos;
  int tick = (step < 0) ? neg : pos;
  PFreal ftick = (tick * PFREAL_ONE) >> 16;

  if(step < 0)
    index++;

  if(state->centerIndex != index)
  {
    state->centerIndex = index;
    frame = index << 16;
    state->centerSlide.slideIndex = state->centerIndex;
    for(int i = 0; i < (int)state->leftSlides.count(); i++)
      state->leftSlides[i].slideIndex = state->centerIndex-1-i;
    for(int i = 0; i < (int)state->rightSlides.count(); i++)
      state->rightSlides[i].slideIndex = state->centerIndex+1+i;
  }

  state->centerSlide.angle = (step * tick * state->angle) >> 16;
  state->centerSlide.cx = -step * fmul(state->offsetX, ftick);
  state->centerSlide.cy = fmul(state->offsetY, ftick);

  if(state->centerIndex == target)
  {
    stop(target);
    state->reset();
    return;
  }

  for(int i = 0; i < (int)state->leftSlides.count(); i++)
  {
    SlideInfo& si = state->leftSlides[i];
    si.angle = state->angle;
    si.cx = -(state->offsetX + state->spacing*i*PFREAL_ONE + step*state->spacing*ftick);
    si.cy = state->offsetY;
  }

  for(int i = 0; i < (int)state->rightSlides.count(); i++)
  {
    SlideInfo& si = state->rightSlides[i];
    si.angle = -state->angle;
    si.cx = state->offsetX + state->spacing*i*PFREAL_ONE - step*state->spacing*ftick;
    si.cy = state->offsetY;
  }

  if(step > 0)
  {
    PFreal ftick = (neg * PFREAL_ONE) >> 16;
    state->rightSlides[0].angle = -(neg * state->angle) >> 16;
    state->rightSlides[0].cx = fmul(state->offsetX, ftick);
    state->rightSlides[0].cy = fmul(state->offsetY, ftick);
  }
  else
  {
    PFreal ftick = (pos * PFREAL_ONE) >> 16;
    state->leftSlides[0].angle = (pos * state->angle) >> 16;
    state->leftSlides[0].cx = -fmul(state->offsetX, ftick);
    state->leftSlides[0].cy = fmul(state->offsetY, ftick);
  }

  // must change direction ?
  if(target < index) if(step > 0)
    step = -1;
  if(target > index) if(step < 0)
    step = 1;

  // the first and last slide must fade in/fade out
  int nleft = state->leftSlides.count();
  int nright = state->rightSlides.count();
  int fade = pos / 256;

  for(int index = 0; index < nleft; index++)
  {
    int blend = 256;
    if(index == nleft-1)
      blend = (step > 0) ? 0 : 128-fade/2;
    if(index == nleft-2)
      blend = (step > 0) ? 128-fade/2 : 256-fade/2;
    if(index == nleft-3)
      blend = (step > 0) ? 256-fade/2 : 256;
    state->leftSlides[index].blend = blend;
  }
  for(int index = 0; index < nright; index++)
  {
    int blend = (index < nright-2) ? 256 : 128;
    if(index == nright-1)
      blend = (step > 0) ? fade/2 : 0;
    if(index == nright-2)
      blend = (step > 0) ? 128+fade/2 : fade/2;
    if(index == nright-3)
      blend = (step > 0) ? 256 : 128+fade/2;
    state->rightSlides[index].blend = blend;
  }
```


(C) Copyright Nokia 2013. All rights reserved.

```
}

// ------------- CoverFlowQtSoftwareRenderer --------------------------------------

CoverFlowQtSoftwareRenderer::CoverFlowQtSoftwareRenderer():
CoverFlowQtAbstractRenderer(), size(0,0), bgcolor(0), effect(-1), blankSurface(0)
{
#ifdef COVERFLOWQT_QT3
  surfaceCache.setAutoDelete(true);
#endif
}

CoverFlowQtSoftwareRenderer::~CoverFlowQtSoftwareRenderer()
{
  surfaceCache.clear();
  buffer = QImage();
  delete blankSurface;
}

void CoverFlowQtSoftwareRenderer::paint()
{
  if(!widget)
    return;

  if(widget->size() != size)
    init();

  if(state->backgroundColor != bgcolor)
  {
    bgcolor = state->backgroundColor;
    surfaceCache.clear();
  }

  if((int)(state->reflectionEffect) != effect)
  {
    effect = (int)state->reflectionEffect;
    surfaceCache.clear();
  }

  if(dirty)
    render();

  QPainter painter(widget);
  painter.drawImage(QPoint(0,0), buffer);
}

void CoverFlowQtSoftwareRenderer::init()
{
  if(!widget)
    return;

  surfaceCache.clear();
  blankSurface = 0;

  size = widget->size();
  int ww = size.width();
  int wh = size.height();
  int w = (ww+1)/2;
  int h = (wh+1)/2;

#ifdef COVERFLOWQT_QT4
  buffer = QImage(ww, wh, QImage::Format_RGB32);
#endif
#if defined(COVERFLOWQT_QT3) || defined(COVERFLOWQT_QT2)
  buffer.create(ww, wh, 32);
#endif
  buffer.fill(bgcolor);

  rays.resize(w*2);
  for(int i = 0; i < w; i++)
  {
    PFreal gg = ((PFREAL_ONE >> 1) + i * PFREAL_ONE) / (2*h);
    rays[w-i-1] = -gg;
    rays[w+i] = gg;
  }

  dirty = true;
}

// TODO: optimize this with lookup tables
static QRgb blendColor(QRgb c1, QRgb c2, int blend)
{
  int r = qRed(c1) * blend/256 + qRed(c2)*(256-blend)/256;
  int g = qGreen(c1) * blend/256 + qGreen(c2)*(256-blend)/256;
  int b = qBlue(c1) * blend/256 + qBlue(c2)*(256-blend)/256;
  return qRgb(r, g, b);
}


static QImage* prepareSurface(const QImage* slideImage, int w, int h, QRgb bgcolor,
CoverFlowQt::ReflectionEffect reflectionEffect)
{
#ifdef COVERFLOWQT_QT4
  Qt::TransformationMode mode = Qt::SmoothTransformation;
  QImage img = slideImage->scaled(w, h, Qt::IgnoreAspectRatio, mode);
#endif
#if defined(COVERFLOWQT_QT3) || defined(COVERFLOWQT_QT2)
  QImage img = slideImage->smoothScale(w, h);
#endif

  // slightly larger, to accomodate for the reflection
  int hs = h * 2;
  int hofs = h / 3;

  // offscreen buffer: black is sweet
#ifdef COVERFLOWQT_QT4
  QImage* result = new QImage(hs, w, QImage::Format_RGB32);
#endif
#if defined(COVERFLOWQT_QT3) || defined(COVERFLOWQT_QT2)
  QImage* result = new QImage;
  result->create(hs, w, 32);
#endif
  result->fill(bgcolor);

  // transpose the image, this is to speed-up the rendering
  // because we process one column at a time
  // (and much better and faster to work row-wise, i.e in one scanline)
  for(int x = 0; x < w; x++)
    for(int y = 0; y < h; y++)
      result->setPixel(hofs + y, x, img.pixel(x, y));
```


(C) Copyright Nokia 2013. All rights reserved.

```
  if(reflectionEffect != CoverFlowQt::NoReflection)
  {
    // create the reflection
    int ht = hs - h - hofs;
    int hte = ht;
    for(int x = 0; x < w; x++)
      for(int y = 0; y < ht; y++)
      {
        QRgb color = img.pixel(x, img.height()-y-1);
        result->setPixel(h+hofs+y, x, blendColor(color,bgcolor,128*(hte-y)/hte));
      }

    if(reflectionEffect == CoverFlowQt::BlurredReflection)
    {
      // blur the reflection everything first
      // Based on exponential blur algorithm by Jani Huhtanen
      QRect rect(hs/2, 0, hs/2, w);
      rect &= result->rect();

      int r1 = rect.top();
      int r2 = rect.bottom();
      int c1 = rect.left();
      int c2 = rect.right();

      int bpl = result->bytesPerLine();
      int rgba[4];
      unsigned char* p;

      // how many times blur is applied?
      // for low-end system, limit this to only 1 loop
      for(int loop = 0; loop < 2; loop++)
      {
        for(int col = c1; col <= c2; col++)
        {
          p = result->scanLine(r1) + col*4;
          for(int i = 0; i < 3; i++)
            rgba[i] = p[i] << 4;

          p += bpl;
          for(int j = r1; j < r2; j++, p += bpl)
            for(int i = 0; i < 3; i++)
              p[i] = (rgba[i] += (((p[i]<<4)-rgba[i])) >> 1) >> 4;
        }

        for(int row = r1; row <= r2; row++)
        {
          p = result->scanLine(row) + c1*4;
          for(int i = 0; i < 3; i++)
            rgba[i] = p[i] << 4;

          p += 4;
          for(int j = c1; j < c2; j++, p+=4)
            for(int i = 0; i < 3; i++)
              p[i] = (rgba[i] += (((p[i]<<4)-rgba[i])) >> 1) >> 4;
        }

        for(int col = c1; col <= c2; col++)
        {
          p = result->scanLine(r2) + col*4;
          for(int i = 0; i < 3; i++)
            rgba[i] = p[i] << 4;

          p -= bpl;
          for(int j = r1; j < r2; j++, p -= bpl)
            for(int i = 0; i < 3; i++)
              p[i] = (rgba[i] += (((p[i]<<4)-rgba[i])) >> 1) >> 4;
        }

        for(int row = r1; row <= r2; row++)
        {
          p = result->scanLine(row) + c2*4;
          for(int i = 0; i < 3; i++)
            rgba[i] = p[i] << 4;

          p -= 4;
          for(int j = c1; j < c2; j++, p-=4)
            for(int i = 0; i < 3; i++)
              p[i] = (rgba[i] += (((p[i]<<4)-rgba[i])) >> 1) >> 4;
        }
      }

      // overdraw to leave only the reflection blurred (but not the actual image)
      for(int x = 0; x < w; x++)
        for(int y = 0; y < h; y++)
          result->setPixel(hofs + y, x, img.pixel(x, y));
    }
  }

  return result;
}

QImage* CoverFlowQtSoftwareRenderer::surface(int slideIndex)
{
  if(!state)
    return 0;
  if(slideIndex < 0)
    return 0;
  if(slideIndex >= (int)state->slideImages.count())
    return 0;

#ifdef COVERFLOWQT_QT4
  int key = slideIndex;
#endif
#if defined(COVERFLOWQT_QT3) || defined(COVERFLOWQT_QT2)
  QString key = QString::number(slideIndex);
#endif

  QImage* img = state->slideImages.at(slideIndex);
  bool empty = img ? img->isNull() : true;
  if(empty)
  {
    surfaceCache.remove(key);
    imageHash.remove(slideIndex);
    if(!blankSurface)
    {
      int sw = state->slideWidth;
      int sh = state->slideHeight;

#ifdef COVERFLOWQT_QT4
```


(C) Copyright Nokia 2013. All rights reserved.

```
      QImage img = QImage(sw, sh, QImage::Format_RGB32);

      QPainter painter(&img);
      QPoint p1(sw*4/10, 0);
      QPoint p2(sw*6/10, sh);
      QLinearGradient linearGrad(p1, p2);
      linearGrad.setColorAt(0, Qt::black);
      linearGrad.setColorAt(1, Qt::white);
      painter.setBrush(linearGrad);
      painter.fillRect(0, 0, sw, sh, QBrush(linearGrad));

      painter.setPen(QPen(QColor(64,64,64), 4));
      painter.setBrush(QBrush());
      painter.drawRect(2, 2, sw-3, sh-3);
      painter.end();
#endif
#if defined(COVERFLOWQT_QT3) || defined(COVERFLOWQT_QT2)
      QPixmap pixmap(sw, sh, 32);
      QPainter painter(&pixmap);
      painter.fillRect(pixmap.rect(), QColor(192,192,192));
      painter.fillRect(5, 5, sw-10, sh-10, QColor(64,64,64));
      painter.end();
      QImage img = pixmap.convertToImage();
#endif

      blankSurface = prepareSurface(&img, sw, sh, bgcolor, state->reflectionEffect);
    }
    return blankSurface;
  }

#ifdef COVERFLOWQT_QT4
  bool exist = imageHash.contains(slideIndex);
  if(exist)
  if(img == imageHash.find(slideIndex).value())
#endif
#ifdef COVERFLOWQT_QT3
  bool exist = imageHash.find(slideIndex) != imageHash.end();
  if(exist)
  if(img == imageHash.find(slideIndex).data())
#endif
#ifdef COVERFLOWQT_QT2
  if(img == imageHash[slideIndex])
#endif
    if(surfaceCache.contains(key))
        return surfaceCache[key];

  QImage* sr = prepareSurface(img, state->slideWidth, state->slideHeight, bgcolor, state->reflectionEffect);
  surfaceCache.insert(key, sr);
  imageHash.insert(slideIndex, img);

  return sr;
}

// Renders a slide to offscreen buffer. Returns a rect of the rendered area.
// col1 and col2 limit the column for rendering.
QRect CoverFlowQtSoftwareRenderer::renderSlide(const SlideInfo &slide, int col1, int col2)
{
  int blend = slide.blend;
  if(!blend)
    return QRect();

  QImage* src = surface(slide.slideIndex);
  if(!src)
    return QRect();

  QRect rect(0, 0, 0, 0);

  int sw = src->height();
  int sh = src->width();
  int h = buffer.height();
  int w = buffer.width();

  if(col1 > col2)
  {
    int c = col2;
    col2 = col1;
    col1 = c;
  }

  col1 = (col1 >= 0) ? col1 : 0;
  col2 = (col2 >= 0) ? col2 : w-1;
  col1 = qMin(col1, w-1);
  col2 = qMin(col2, w-1);

  int zoom = 100;
  int distance = h * 100 / zoom;
  PFreal sdx = fcos(slide.angle);
  PFreal sdy = fsin(slide.angle);
  PFreal xs = slide.cx - state->slideWidth * sdx/2;
  PFreal ys = slide.cy - state->slideWidth * sdy/2;
  PFreal dist = distance * PFREAL_ONE;

  int xi = qMax((PFreal)0, (w*PFREAL_ONE/2) + fdiv(xs*h, dist+ys) >> PFREAL_SHIFT);
  if(xi >= w)
    return rect;

  bool flag = false;
  rect.setLeft(xi);
  for(int x = qMax(xi, col1); x <= col2; x++)
  {
    PFreal hity = 0;
    PFreal fk = rays[x];
    if(sdy)
    {
      fk = fk - fdiv(sdx,sdy);
      hity = -fdiv((rays[x]*distance - slide.cx + slide.cy*sdx/sdy), fk);
    }

    dist = distance*PFREAL_ONE + hity;
    if(dist < 0)
      continue;

    PFreal hitx = fmul(dist, rays[x]);
    PFreal hitdist = fdiv(hitx - slide.cx, sdx);

    int column = sw/2 + (hitdist >> PFREAL_SHIFT);
    if(column >= sw)
      break;
    if(column < 0)
      continue;
```


(C) Copyright Nokia 2013. All rights reserved.

```
    rect.setRight(x);
    if(!flag)
      rect.setLeft(x);
    flag = true;

    int y1 = h/2;
    int y2 = y1+ 1;
    QRgb* pixel1 = (QRgb*)(buffer.scanLine(y1)) + x;
    QRgb* pixel2 = (QRgb*)(buffer.scanLine(y2)) + x;
    QRgb pixelstep = pixel2 - pixel1;

    int center = (sh/2);
    int dy = dist / h;
    int p1 = center*PFREAL_ONE - dy/2;
    int p2 = center*PFREAL_ONE + dy/2;

    const QRgb *ptr = (const QRgb*)(src->scanLine(column));
    if(blend == 256)
      while((y1 >= 0) && (y2 < h) && (p1 >= 0))
      {
        *pixel1 = ptr[p1 >> PFREAL_SHIFT];
        *pixel2 = ptr[p2 >> PFREAL_SHIFT];
        p1 -= dy;
        p2 += dy;
        y1--;
        y2++;
        pixel1 -= pixelstep;
        pixel2 += pixelstep;
      }
    else
      while((y1 >= 0) && (y2 < h) && (p1 >= 0))
      {
        QRgb c1 = ptr[p1 >> PFREAL_SHIFT];
        QRgb c2 = ptr[p2 >> PFREAL_SHIFT];
        *pixel1 = blendColor(c1, bgcolor, blend);
        *pixel2 = blendColor(c2, bgcolor, blend);
        p1 -= dy;
        p2 += dy;
        y1--;
        y2++;
        pixel1 -= pixelstep;
        pixel2 += pixelstep;
     }
   }

   rect.setTop(0);
   rect.setBottom(h-1);
   return rect;
}

void CoverFlowQtSoftwareRenderer::renderSlides()
{
  int nleft = state->leftSlides.count();
  int nright = state->rightSlides.count();

  QRect r = renderSlide(state->centerSlide);
  int c1 = r.left();
  int c2 = r.right();

  for(int index = 0; index < nleft; index++)
  {
    QRect rs = renderSlide(state->leftSlides[index], 0, c1-1);
    if(!rs.isEmpty())
      c1 = rs.left();
  }
  for(int index = 0; index < nright; index++)
  {
    QRect rs = renderSlide(state->rightSlides[index], c2+1, buffer.width());
    if(!rs.isEmpty())
      c2 = rs.right();
  }
}

// Render the slides. Updates only the offscreen buffer.
void CoverFlowQtSoftwareRenderer::render()
{
  buffer.fill(state->backgroundColor);
  renderSlides();
  dirty = false;
}

// ----------------------------------------

class CoverFlowQtPrivate
{
public:
  CoverFlowQtState* state;
  CoverFlowQtAnimator* animator;
  CoverFlowQtAbstractRenderer* renderer;
  QTimer triggerTimer;
};


CoverFlowQt::CoverFlowQt(QWidget* parent): QWidget(parent)
{
  d = new CoverFlowQtPrivate;

  d->state = new CoverFlowQtState;
  d->state->reset();
  d->state->reposition();

  d->renderer = new CoverFlowQtSoftwareRenderer;
  d->renderer->state = d->state;
  d->renderer->widget = this;
  d->renderer->init();

  d->animator = new CoverFlowQtAnimator;
  d->animator->state = d->state;
  QObject::connect(&d->animator->animateTimer, SIGNAL(timeout()), this, SLOT(updateAnimation()));

  QObject::connect(&d->triggerTimer, SIGNAL(timeout()), this, SLOT(render()));

#ifdef COVERFLOWQT_QT4
  setAttribute(Qt::WA_StaticContents, true);
  setAttribute(Qt::WA_OpaquePaintEvent, true);
  setAttribute(Qt::WA_NoSystemBackground, true);
#endif
#ifdef COVERFLOWQT_QT3
  setWFlags(getWFlags() | Qt::WStaticContents);
```


(C) Copyright Nokia 2013. All rights reserved.

```
  setWFlags(getWFlags() | Qt::WNoAutoErase);
#endif
#ifdef COVERFLOWQT_QT2
  setWFlags(getWFlags() | Qt::WPaintClever);
  setWFlags(getWFlags() | Qt::WRepaintNoErase);
  setWFlags(getWFlags() | Qt::WResizeNoErase);
#endif
}

CoverFlowQt::~CoverFlowQt()
{
  delete d->renderer;
  delete d->animator;
  delete d->state;
  delete d;
}

int CoverFlowQt::slideCount() const
{
  return d->state->slideImages.count();
}

QColor CoverFlowQt::backgroundColor() const
{
  return QColor(d->state->backgroundColor);
}

void CoverFlowQt::setBackgroundColor(const QColor& c)
{
  d->state->backgroundColor = c.rgb();
  triggerRender();
}

QSize CoverFlowQt::slideSize() const
{
  return QSize(d->state->slideWidth, d->state->slideHeight);
}

void CoverFlowQt::setSlideSize(QSize size)
{
  d->state->slideWidth = size.width();
  d->state->slideHeight = size.height();
  d->state->reposition();
  triggerRender();
}

CoverFlowQt::ReflectionEffect CoverFlowQt::reflectionEffect() const
{
  return d->state->reflectionEffect;
}

void CoverFlowQt::setReflectionEffect(ReflectionEffect effect)
{
  d->state->reflectionEffect = effect;
  triggerRender();
}

QImage CoverFlowQt::slide(int index) const
{
  QImage* i = 0;
  if((index >= 0) && (index < slideCount()))
    i = d->state->slideImages[index];
  return i ? QImage(*i) : QImage();
}

void CoverFlowQt::addSlide(const QImage& image)
{
  int c = d->state->slideImages.count();
  d->state->slideImages.resize(c+1);
  d->state->slideImages[c] = new QImage(image);
  triggerRender();
}

void CoverFlowQt::addSlide(const QPixmap& pixmap)
{
  addSlide(pixmap.toImage());
}

void CoverFlowQt::setSlide(int index, const QImage& image)
{
  if((index >= 0) && (index < slideCount()))
  {
    QImage* i = image.isNull() ? 0 : new QImage(image);
    delete d->state->slideImages[index];
    d->state->slideImages[index] = i;
    triggerRender();
  }
}

void CoverFlowQt::setSlide(int index, const QPixmap& pixmap)
{
  setSlide(index, pixmap.toImage());
}

int CoverFlowQt::centerIndex() const
{

  return d->state->centerIndex;
}

void CoverFlowQt::setCenterIndex(int index)
{
  index = qMin(index, slideCount()-1);
  index = qMax(index, 0);
  d->state->centerIndex = index;
  d->state->reset();
  d->animator->stop(index);
  triggerRender();


}

void CoverFlowQt::clear()
{
  int c = d->state->slideImages.count();
  for(int i = 0; i < c; i++)
    delete d->state->slideImages[i];
  d->state->slideImages.resize(0);

  d->state->reset();
```


(C) Copyright Nokia 2013. All rights reserved.

```
  triggerRender();
}

void CoverFlowQt::render()
{
  d->renderer->dirty = true;
  update();
}

void CoverFlowQt::triggerRender()
{
#ifdef COVERFLOWQT_QT4
  d->triggerTimer.setSingleShot(true);
  d->triggerTimer.start(0);
#endif
#if defined(COVERFLOWQT_QT3) || defined(COVERFLOWQT_QT2)
  d->triggerTimer.start(0, true);
#endif
}

void CoverFlowQt::showPrevious()
{
  int step = d->animator->step;
  int center = d->state->centerIndex;

  if(step > 0)
    d->animator->start(center);

  if(step == 0)
    if(center > 0)
      d->animator->start(center - 1);

  if(step < 0)
    d->animator->target = qMax(0, center - 2);



}

void CoverFlowQt::showNext()
{
  int step = d->animator->step;
  int center = d->state->centerIndex;

  if(step < 0)
    d->animator->start(center);

  if(step == 0)
    if(center < slideCount()-1)
      d->animator->start(center + 1);

  if(step > 0)
    d->animator->target = qMin(center + 2, slideCount()-1);



}

void CoverFlowQt::showSlide(int index)
{
  index = qMax(index, 0);
  index = qMin(slideCount()-1, index);
  if(index == d->state->centerSlide.slideIndex)
    return;

  d->animator->start(index);



}

void CoverFlowQt::keyPressEvent(QKeyEvent* event)
{
  if(event->key() == Qt::Key_Left)
  {
    if(event->modifiers() == Qt::ControlModifier)
      showSlide(centerIndex()-3);
    else
      showPrevious();
    event->accept();
    return;
  }

  if(event->key() == Qt::Key_Right)
  {
    if(event->modifiers() == Qt::ControlModifier)
      showSlide(centerIndex()+3);
    else
      showNext();
    event->accept();
    return;
  }

  event->ignore();
}

void CoverFlowQt::mousePressEvent(QMouseEvent* event)
{
  if(event->x() > width()/2)
    showNext();
  else
    showPrevious();
}

void CoverFlowQt::paintEvent(QPaintEvent* event)
{
  Q_UNUSED(event);
  d->renderer->paint();
}

void CoverFlowQt::resizeEvent(QResizeEvent* event)
{
  triggerRender();
  QWidget::resizeEvent(event);
}

void CoverFlowQt::updateAnimation()
{
  int old_center = d->state->centerIndex;
  d->animator->update();
```


(C) Copyright Nokia 2013. All rights reserved.

```
  triggerRender();
  if(d->state->centerIndex != old_center)
  {
    emit centerIndexChanged(d->state->centerIndex);
    }

}
```

main.cpp

```
#include <qapplication.h>
#include <qdir.h>
#include <qevent.h>
#include <qfileinfo.h>
#include <qimage.h>
#include "QMessageBox"

#if QT_VERSION >= 0x040000
#include <QTime>
#endif

#include "inc\coverflowqt.h"

QStringList findFiles(const QString& path = QString())
{
  QStringList files;

  QDir dir = QDir::current();
  if(!path.isEmpty())
    dir = QDir(path);

  dir.setFilter(QDir::Files | QDir::Hidden | QDir::NoSymLinks);
#if QT_VERSION >= 0x040000
  QFileInfoList list = dir.entryInfoList();
  for (int i = 0; i < list.size(); ++i)
  {
    QFileInfo fileInfo = list.at(i);
    files.append(dir.absoluteFilePath(fileInfo.fileName()));

  }
#else
  const QFileInfoList* list = dir.entryInfoList();
  if(list)
  {
    QFileInfoListIterator it( *list );
    QFileInfo * fi;
    while( (fi=it.current()) != 0 )
    {
      ++it;
      files.append(dir.absFilePath(fi->fileName()));
    }
  }
#endif

  return files;
}

#if QT_VERSION < 0x040000
#define modifiers state
#define AltModifier AltButton
#define setWindowTitle setCaption
#endif

#if QT_VERSION < 0x030000
#define flush flushX
#endif

class Browser: public CoverFlowQt
{
public:
  Browser(): CoverFlowQt()
  {
    setWindowTitle("CoverFlowQt");
  }

  void keyPressEvent(QKeyEvent* event)
  {
    if(event->key() == Qt::Key_Escape || event->key() == Qt::Key_Enter ||
      event->key() == Qt::Key_Return)
      {
        event->accept();
        close();
      }

    // checking the speed of rendering
    if(event->key() == Qt::Key_F10)
    if(event->modifiers() == Qt::AltModifier)
    {
      qDebug("benchmarking.... please wait");
      const int blit_count = 10;

      QTime stopwatch;
      stopwatch.start();
      for(int i = 0; i < blit_count; i++)
      {
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
        render(); repaint(); QApplication::flush(); QApplication::syncX();
      }
      QString msg;
      int elapsed = stopwatch.elapsed();
      if( elapsed > 0 )
        msg = QString("FPS: %1").arg( blit_count*10*1000.0/elapsed );
      else
        msg = QString("Too fast. Increase blit_count");
```


(C) Copyright Nokia 2013. All rights reserved.

```
      setWindowTitle( msg );
      event->accept();
      return;
    }

    // for debugging only: Alt+F11 cycles the reflection effect
    if(event->key() == Qt::Key_F11)
    if(event->modifiers() == Qt::AltModifier)
    {
      qDebug("changing reflection effect...");
      switch(reflectionEffect())
      {
        //case NoReflection:      setReflectionEffect(PlainReflection); break;
        case PlainReflection:   setReflectionEffect(BlurredReflection); break;
        case BlurredReflection: setReflectionEffect(PlainReflection); break;
        default:                setReflectionEffect(PlainReflection); break;
      }
      event->accept();
      return;
    }

    CoverFlowQt::keyPressEvent(event);
  }
};

int main( int argc, char ** argv )
{
  QApplication* app = new QApplication( argc, argv );
  Browser* w = new Browser;

#if defined(_WS_QWS) || defined(Q_WS_QWS)
  w->showFullScreen();
  int ww = w->width();
  int wh = w->height();
  int dim = (ww > wh) ? wh : ww;
  dim = dim * 3 / 4;
  w->setSlideSize(QSize(3*dim/5, dim));
#else
  w->setSlideSize(QSize(3*40, 5*40));
  w->resize(750, 270);
#endif

  QStringList files = (argc > 1) ? findFiles(QString(argv[1])) : findFiles();

  QImage img;
  for(int i = 0; i < (int)files.count(); i++)
    if(img.load(files[i]))
      w->addSlide(img);

  w->setCenterIndex(w->slideCount()/2);
  w->setBackgroundColor(Qt::gray);
  w->show();

  app->connect( app, SIGNAL(lastWindowClosed()), app, SLOT(quit()) );
  int result = app->exec();

  delete w;
  delete app;

  return result;
}
```

Technical Summary

CoverFlowQtState stores all necessary information to render the slides. CoverFlowQtAnimator move the sllides during transition between slids. CoverFlowQtSoftwareRenderer is the actual 3d renderer. It renders all the slides given the state.

Source Code

The full source code presented in this article is available here File:CoverFlow.zip

--somnathbanik

(C) Copyright Nokia 2013. All rights reserved.

(C) Copyright Nokia 2013. All rights reserved.