THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Lecture Qt014
Graphics I

Instructor: David J. Coe

CPE 353 – Software Design and Engineering

Department of Electrical and Computer Engineering

Outline

- QPainter and QPixmap Classes
- Aliasing
- Graphics Examples
- Key Points

THE UNIVERSITY OF
ALABAMA IN HUNTSVILLE

QPainter Class

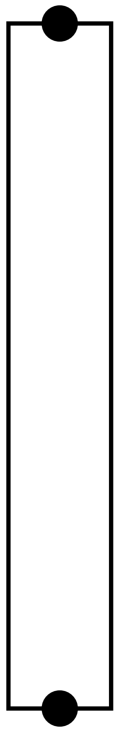
- Basis of 2-D graphics in Qt
- Enables drawing of
  - Geometric shapes
    - Points, Lines, Rectangles, Polygons, Ellipses, Arcs, Chords, Curves
  - Text, Images, Pixmaps
- Transformations
  - Translation, rotations, scaling, etc.

QPainter Class

- Methods include
  - drawPoint()
  - drawLine()
  - drawRect()
  - drawPolygon()
  - drawEllipse()
  - drawText()
  - drawPixmap()

QPainter Class

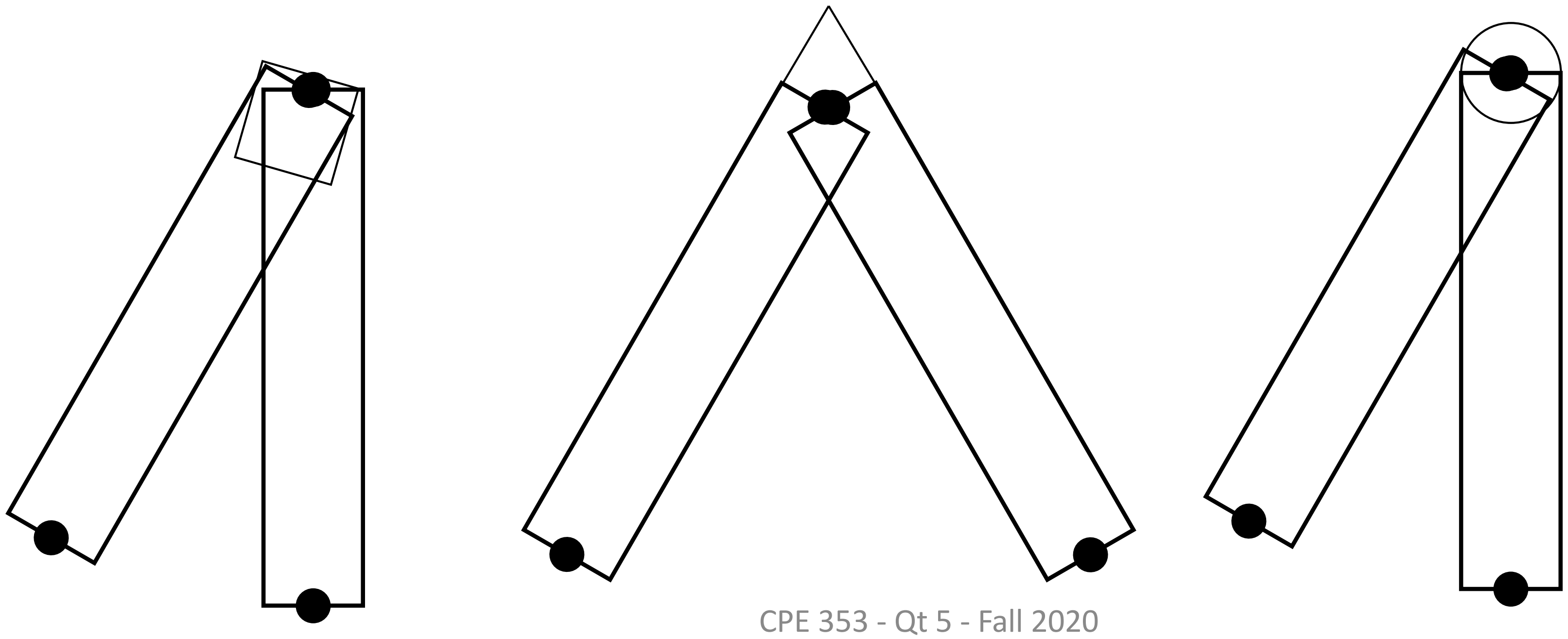
- Three key parameters to consider
  - **Pen** (QPen)
    - Color
      - Specified by setColor(someColor)
      - More on color
      -
    - Width
      - Specified in pixels by setWidth(someInt) or setWidthF(someReal)
      - Retrieved as int by width() or real by widthF()



    - Line Style
      - Specified by setStyle(someStyle) using enumerated type Qt::PenStyle which includes values Qt::SolidLine, Qt::DashLine, Qt::DotLine, Qt::DashDotLine,etc.

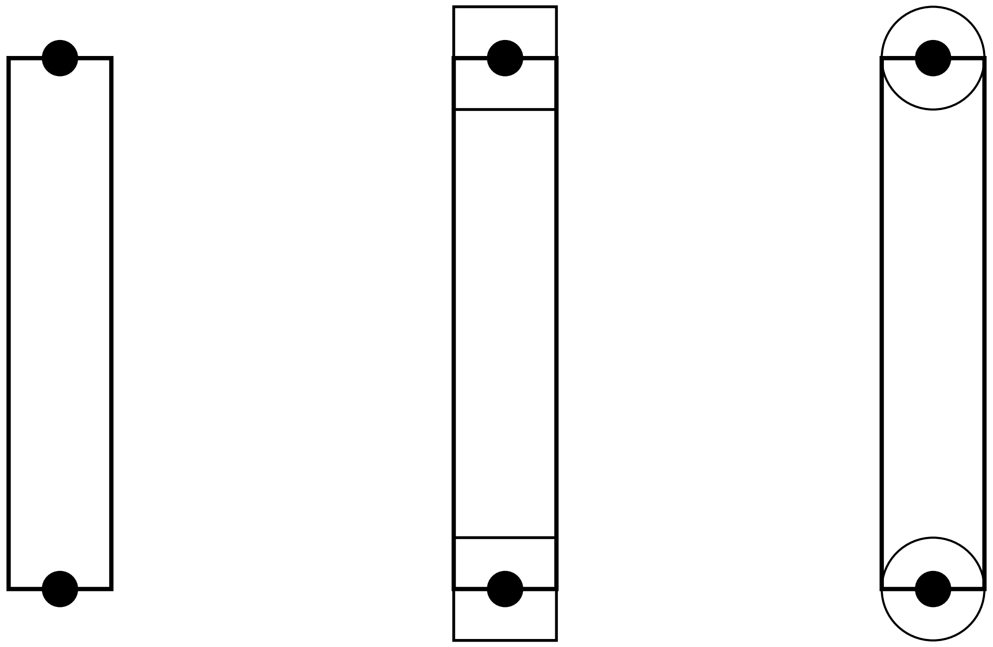
QPainter Class

- Three key parameters to consider
  - **Pen** (QPen) continued
    - Join Style
      - Specified by setJoinStyle(…)
      - Beveled, Mitered, or Rounded line joints (Qt::PenJoinStyle)



QPainter Class

- Three key parameters to consider
  - **Pen** (QPen) continued
    - Cap Style
      - Specified by setCapStyle(…)
      - Flat, Square, or Rounded endcaps (Qt::PenCapStyle)



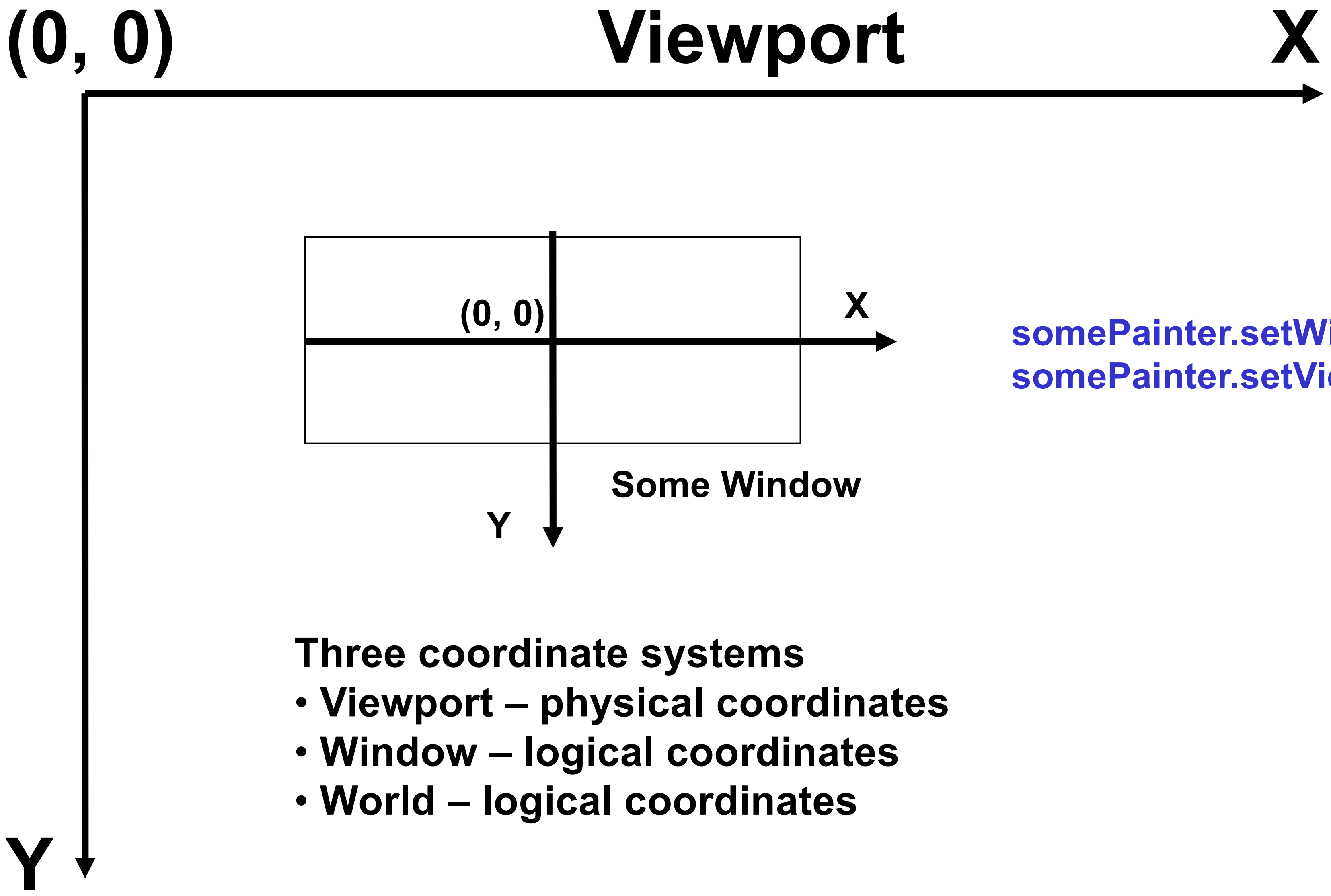
QPainter Class

- Three key parameters to consider (continued)
  - **Brush** (QBrush)
    - Specifed by setBrush(…)
    - Enumerated type Qt::BrushStyle
      - Qt::SolidPattern, Qt::LinearGradientPattern, Qt::DiagCrossPattern, etc

QPainter Class

- Three key parameters to consider (continued)
  - **Font** (QFont)
    - Specified setFont(…)
    - Can select font name, point size, bold/italics/underline, etc.

QPainter Class



somePainter.setWindow(…)
somePainter.setViewport(…)

Three coordinate systems

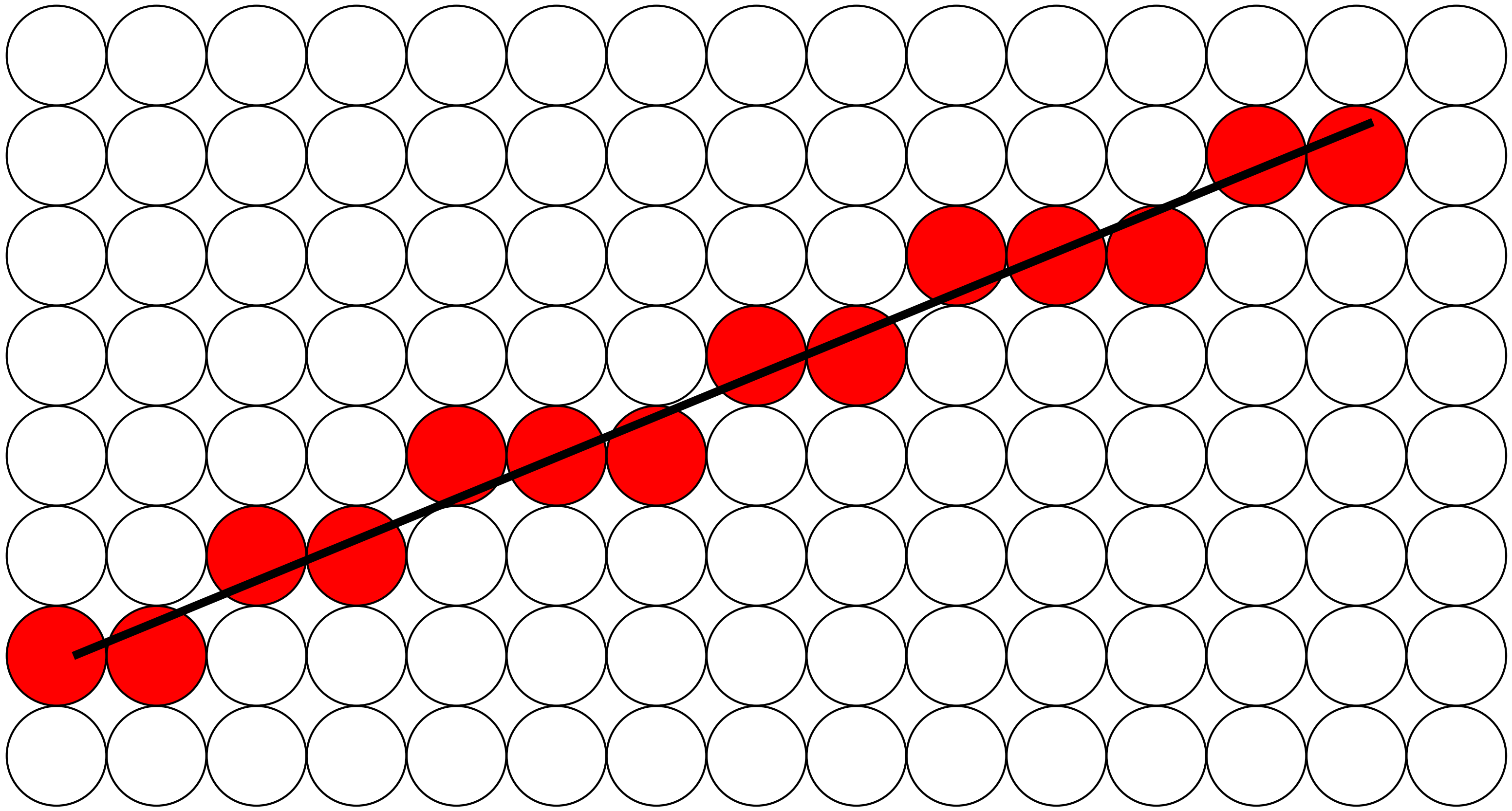
- Viewport – physical coordinates
- Window – logical coordinates
- World – logical coordinates

QPixmap Class

- One of four Qt classes used for images
- Optimized for on-screen image display

- Other classes include
  - **QImage**
    - Optimized for loading and saving of image data
  - **QPicture**
    - Recording & playback of **QPainter** commands

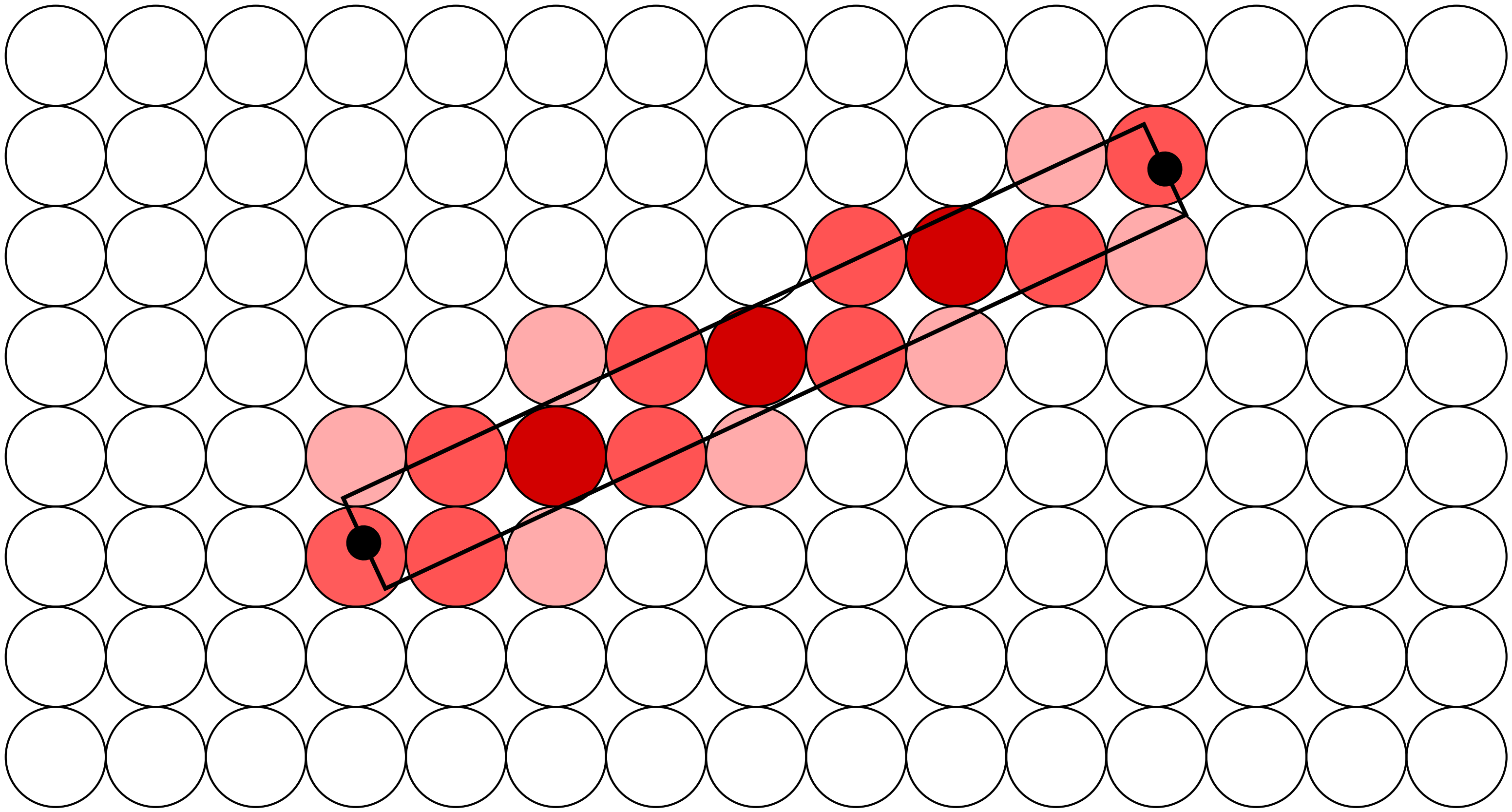
Aliasing



Aliasing occurs when trying to use discrete sampling to render a continuous shape

***Computer Graphics: Principles and Practice, Second Edition*, Foley, vanDam, Feiner, and Hughes**

Aliasing



Many anti-aliasing strategies
-- here pixel intensity is proportional to amount of line overlap area

Suggestion – use built-in Qt anti-aliasing

***Computer Graphics: Principles and Practice, Second Edition*, Foley, vanDam, Feiner, and Hughes**

Graphics Example 01

- Goals
  - Create a rectangle object
  - Use it to draw an array of rectangles
  - Explore various pen options
    - Colors
    - Line Widths
    - Cap styles
    - Join styles
    - Lines styles
    - Brush styles

Graphics Example 01

```
//
// Graphics Example 01
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                           // Need application for event loop

  QPixmap  myMap(400, 300);                                 // Establish 400 X 300 pixel pixmap

  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);            // Enable antialiasing

  // Draw frame just within perimeter
  p.setPen(QPen(Qt::black, 2, Qt::SolidLine, Qt::SquareCap));
  p.drawRect(10, 10, 380, 280);                             // At (10, 10) with width=380, height=280

  // Vary Cap Style
  p.setPen(QPen(Qt::blue, 10, Qt::SolidLine, Qt::SquareCap));
  QRect rect1(25, 25, 50, 30);                              // At (25, 25) with width=50, height=30
  p.drawRect(rect1);

  p.setPen(QPen(Qt::red, 10, Qt::SolidLine, Qt::RoundCap));
  rect1.translate(100, 0);                                  // dx = 100, dy = 0
  p.drawRect(rect1);

  p.setPen(QPen(Qt::green, 10, Qt::SolidLine, Qt::FlatCap));
  rect1.translate(100, 0);                                  // dx = 100, dy = 0
  p.drawRect(rect1);
```

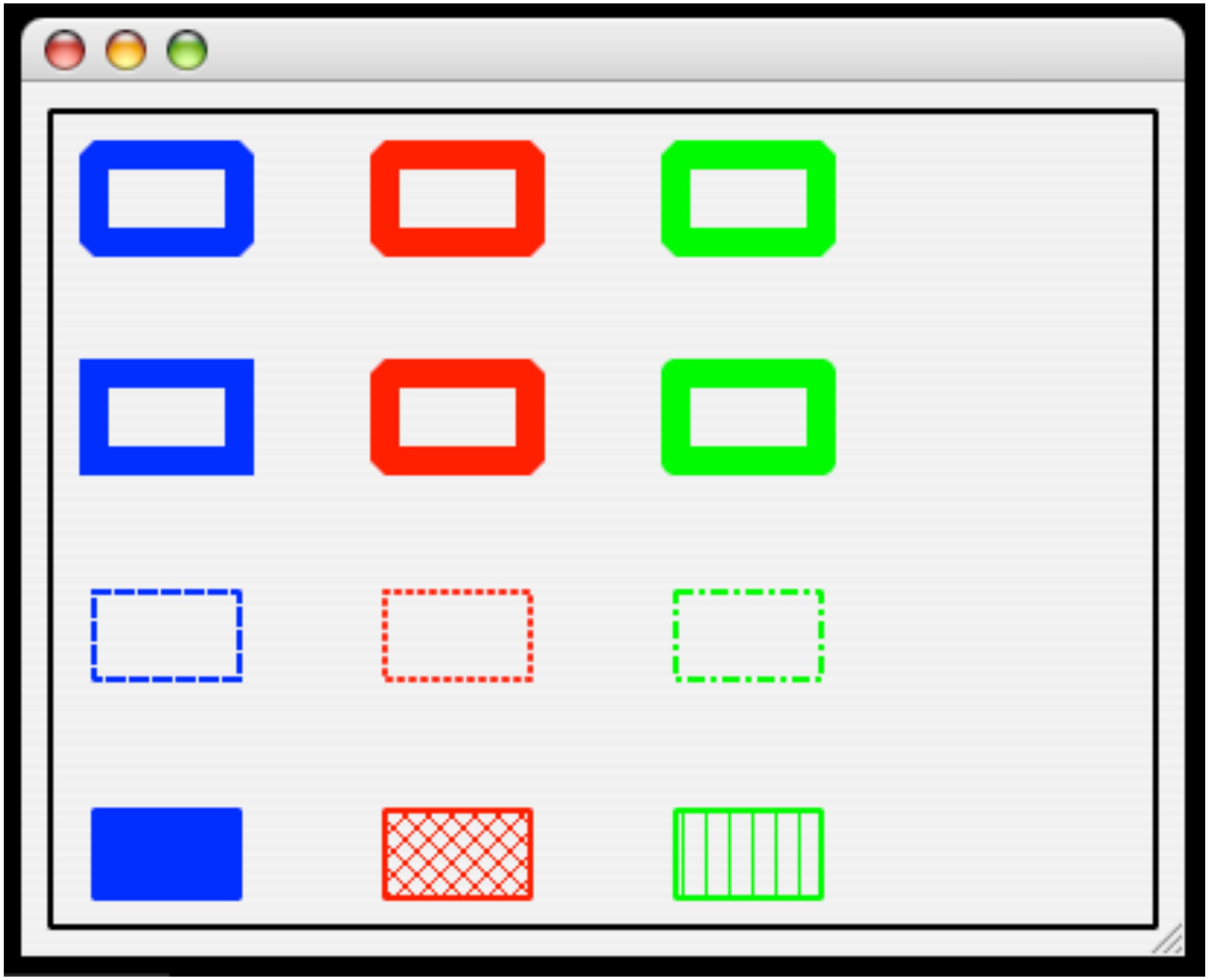
Graphics Example 01

```
// Graphics Example 01 - continued

  // Vary Join Style
  p.setPen(QPen(Qt::blue, 10, Qt::SolidLine, Qt::FlatCap, Qt::MiterJoin));
  rect1.translate(-200, 75);                                        // dx = -200, dy = 75
  p.drawRect(rect1);

  p.setPen(QPen(Qt::red, 10, Qt::SolidLine, Qt::FlatCap, Qt::BevelJoin));
  rect1.translate(100, 0);
  p.drawRect(rect1);

  p.setPen(QPen(Qt::green, 10, Qt::SolidLine, Qt::FlatCap, Qt::RoundJoin));
  rect1.translate(100, 0);
  p.drawRect(rect1);


  // Vary Line Style
  p.setPen(QPen(Qt::blue, 2, Qt::DashLine, Qt::SquareCap));
  rect1.translate(-200, 75);
  p.drawRect(rect1);

  p.setPen(QPen(Qt::red, 2, Qt::DotLine, Qt::RoundCap));
  rect1.translate(100, 0);
  p.drawRect(rect1);

  p.setPen(QPen(Qt::green, 2, Qt::DashDotLine, Qt::FlatCap));
  rect1.translate(100, 0);
  p.drawRect(rect1);
```

Graphics Example 01

```
// Graphics Example 01 - continued


  // Vary Brush Style
  p.setPen(QPen(Qt::blue, 2, Qt::SolidLine, Qt::SquareCap));
  p.setBrush(QBrush(Qt::blue, Qt::SolidPattern));
  rect1.translate(-200, 75);
  p.drawRect(rect1);

  p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::RoundCap));
  p.setBrush(QBrush(Qt::red, Qt::DiagCrossPattern));
  rect1.translate(100, 0);
  p.drawRect(rect1);

  p.setPen(QPen(Qt::green, 2, Qt::SolidLine, Qt::FlatCap));
  p.setBrush(QBrush(Qt::green, Qt::VerPattern));
  rect1.translate(100, 0);
  p.drawRect(rect1);


  QLabel myLabel;                                      // Allocate a Gui widget
  myLabel.setPixmap(myMap);                            // Associate pixmap with Gui widget
  myLabel.show();                                      // Make widget visible

  return myApp.exec();                                 // Initiate event loop
} // End main()
```

Graphics Example 01



Graphics Example 02

- Goals
  - Draw crosshairs using an ellipse, four points, and two lines
  - Add text to pixmap
  - Draw four arcs

Graphics Example 02

```
//
// Graphics Example 2
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLine>
#include <QFont>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                          // Need application for event loop

  QPixmap  myMap(400, 300);                                // Establish pixmap
  myMap.fill(Qt::black);

  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);           // Enable antialiasing
```

Graphics Example 02

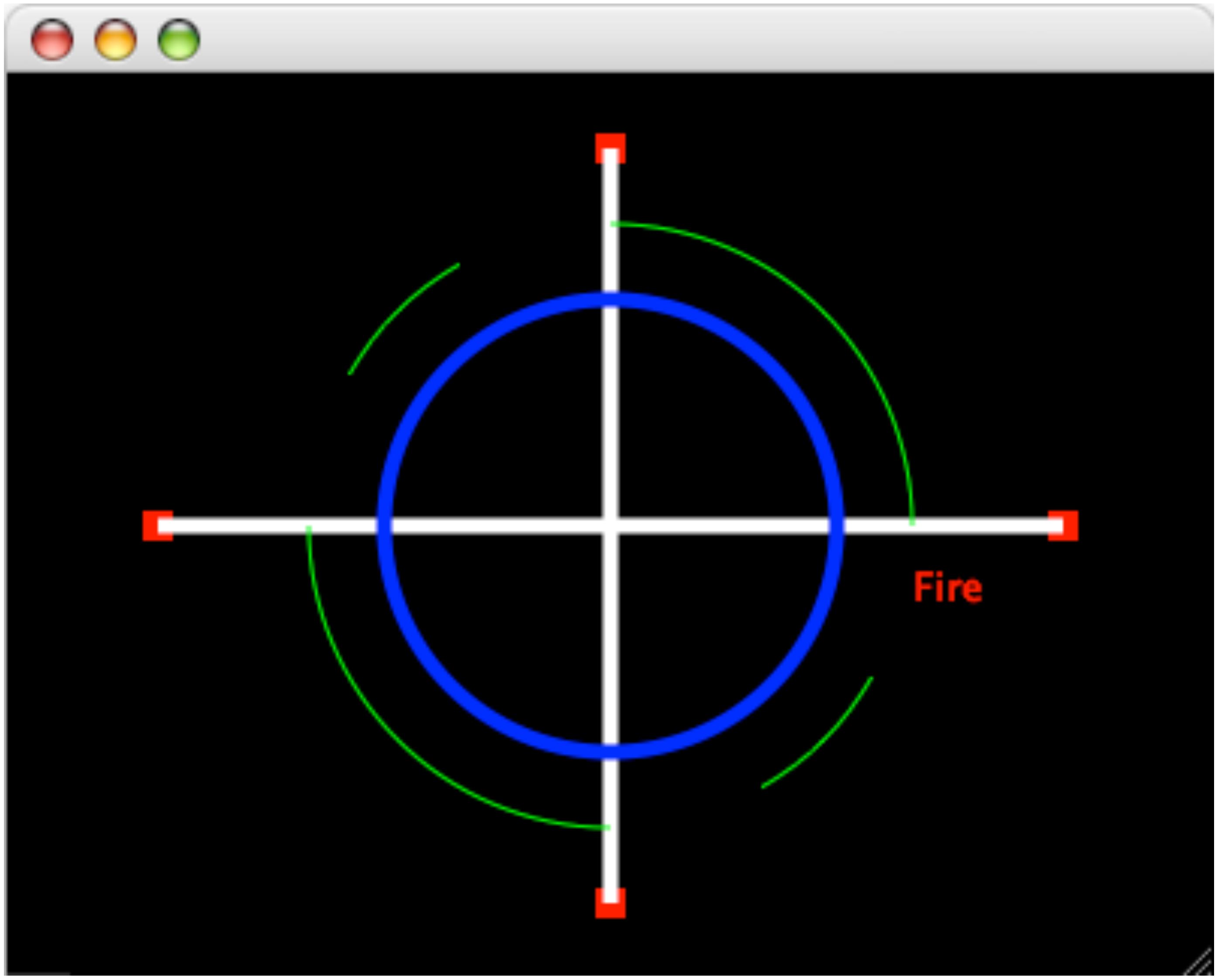
THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

```
// Graphics Example 2 --  continued

  // Draw four points and two crosshair lines
  p.setPen(QPen(Qt::red, 10, Qt::SolidLine, Qt::FlatCap));
  QPoint p1(200, 25);
  QPoint p2(200, 275);
  QPoint p3(50, 150);
  QPoint p4(350, 150);
  p.drawPoint(p1);
  p.drawPoint(p2);
  p.drawPoint(p3);
  p.drawPoint(p4);
  p.setPen(QPen(Qt::white, 5, Qt::SolidLine, Qt::FlatCap));
  QLine line1(p1, p2);
  QLine line2(p3, p4);
  p.drawLine(line1);
  p.drawLine(line2);

  // Define Bounding Rectangle, set pen, and draw ellipse
  p.setPen(QPen(Qt::blue, 5, Qt::SolidLine, Qt::SquareCap));
  QRect rect1(125, 75, 150, 150);
  p.drawEllipse(rect1);

  // Write text
  p.setPen(Qt::red);
  QPoint tp(300, 175);
  p.drawText(tp, "Fire");
```

Graphics Example 02

```
// Graphics Example 2 --  continued


  // Draw arcs --- angles are 1/16 of a degree, hence the scale factor
  p.setPen(Qt::green);
  p.drawArc(QRect(100, 50, 200, 200), 00*16, 90*16);         // Rectangle defines ellipse
  p.drawArc(QRect(100, 50, 200, 200), 180*16, 90*16);        // Arc needs start angle, span angle
  p.drawArc(QRect(100, 50, 200, 200), 120*16, 30*16);
  p.drawArc(QRect(100, 50, 200, 200), 300*16, 30*16);

  QLabel myLabel;                                            // Allocate a Gui widget
  myLabel.setPixmap(myMap);                                  // Associate pixmap with Gui widget
  myLabel.show();                                            // Make widget visible

  return myApp.exec();                                       // Initiate event loop
} // End main()
```

Graphics Example 02



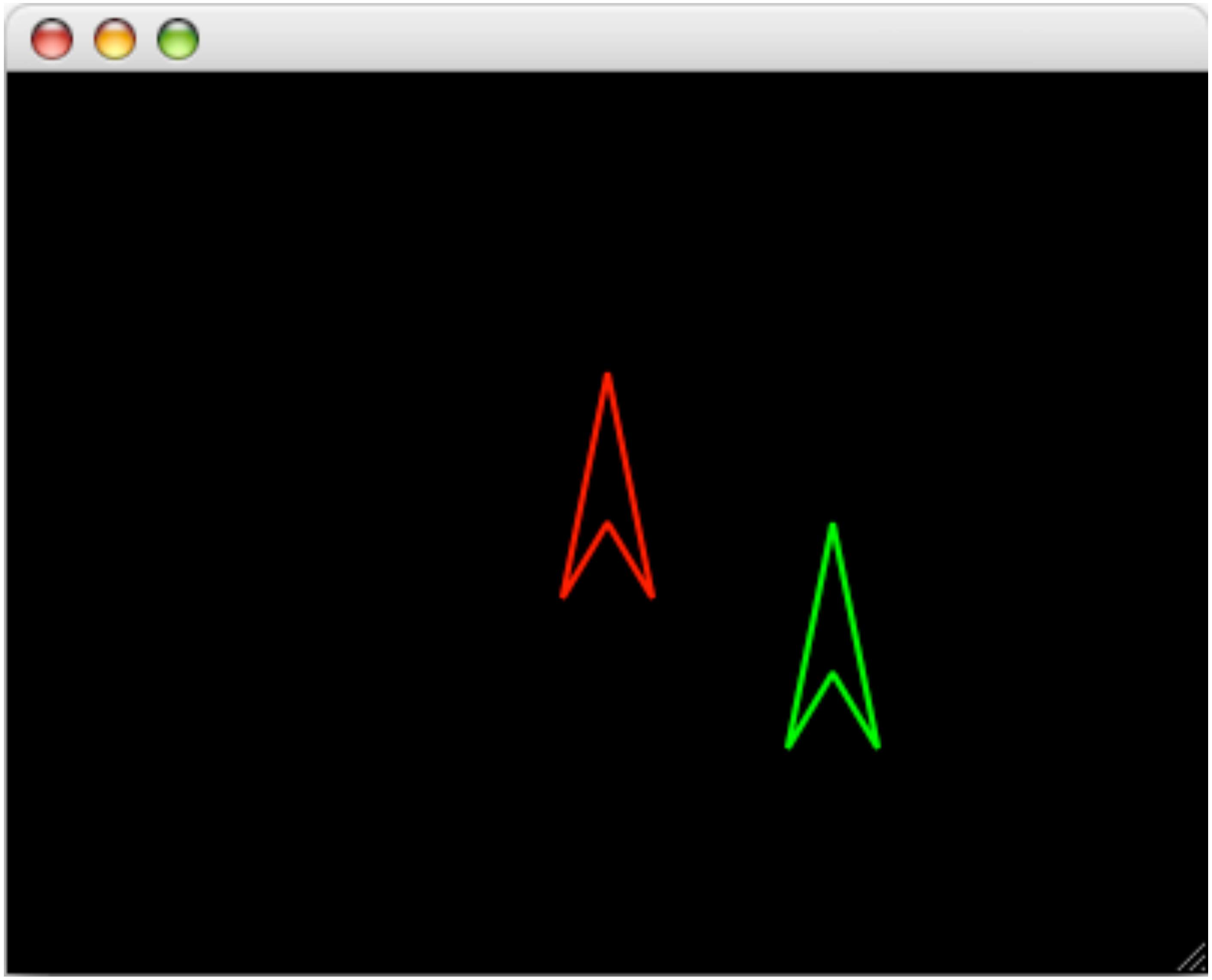
Graphics Example 03

- Goals
  - Draw an arrow using a polygon
  - Translate and redraw arrow

Graphics Example 03

```
//
// Graphics Example 3
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                          // Need application for event loop
  QPixmap  myMap(400, 300);                                // Establish pixmap
  myMap.fill(Qt::black);
  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);           // Enables antialiasing

  // Draw red arrow polygon
  p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::FlatCap));
  p.setBrush(QBrush(Qt::SolidPattern));
  QPoint points[4] = {QPoint(200, 100), QPoint(215, 175), QPoint(200, 150), QPoint(185, 175)};
  p.drawPolygon(points, 4);

  // Apply translation and redraw as green polygon
  p.translate(75, 50);                                     // Translate dx = 75, dy = 50
  p.setPen(QPen(Qt::green, 2, Qt::SolidLine, Qt::FlatCap));
  p.drawPolygon(points, 4);

  QLabel myLabel;                                          // Allocate a Gui widget
  myLabel.setPixmap(myMap);                                // Associate pixmap with Gui widget
  myLabel.show();                                          // Make widget visible
  return myApp.exec();                                     // Initiate event loop
} // End main()
```

Graphics Example 03 - 2



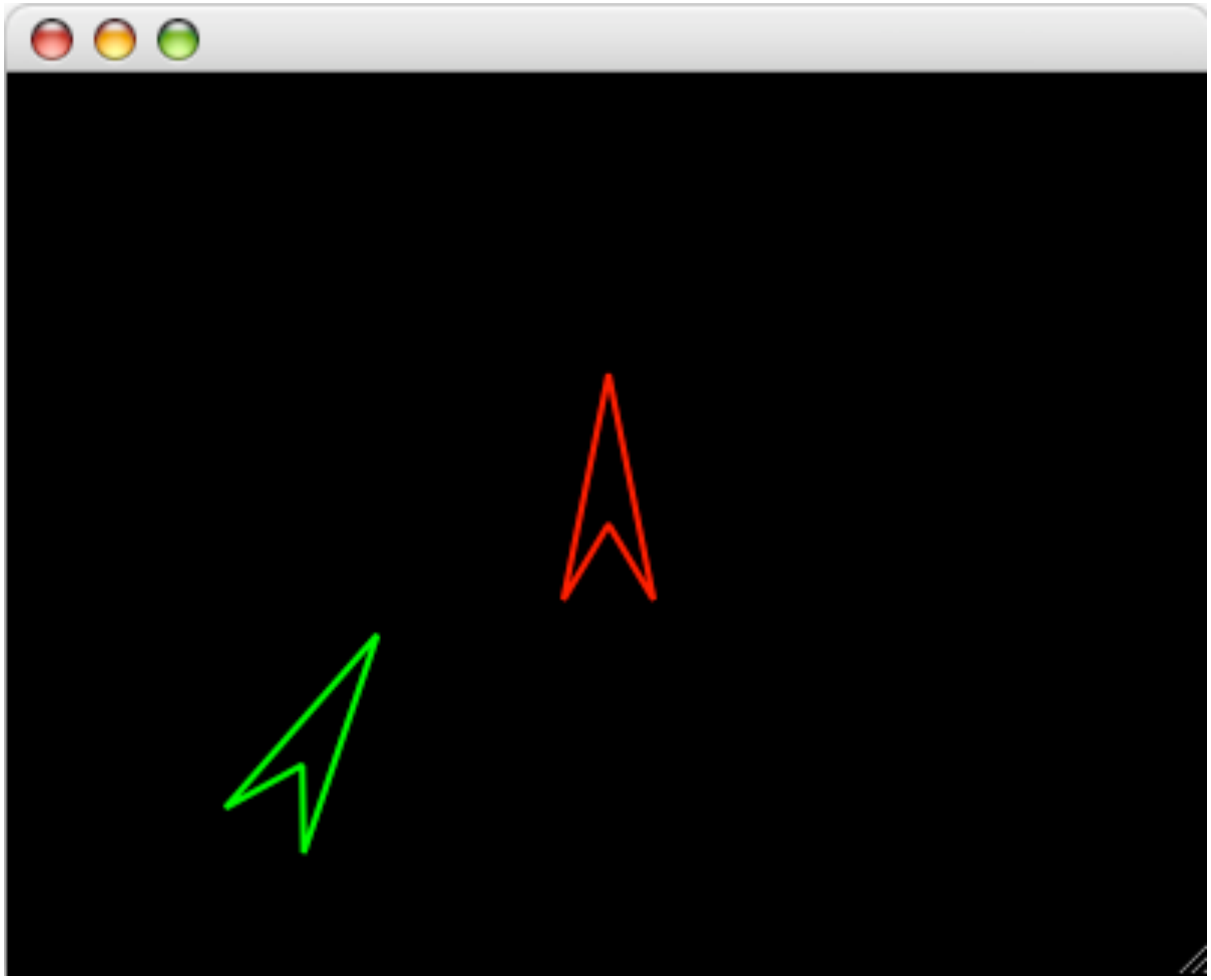
Graphics Example 04

- Goals
  - Draw an arrow using a polygon
  - Rotate by 30 degrees
  - Redraw arrow

Graphics Example 04

```
//
// Graphics Example 4
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                                 // Need application for event loop
  QPixmap  myMap(400, 300);                                       // Establish pixmap
  myMap.fill(Qt::black);
  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);                  // Enable antialiasing

  // Draw red arrow polygon
  p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::FlatCap));
  p.setBrush(QBrush(Qt::SolidPattern));
  QPoint points[4] = {QPoint(200, 100), QPoint(215, 175), QPoint(200, 150), QPoint(185, 175)};
  p.drawPolygon(points, 4);

  // Apply rotation and redraw as green polygon
  p.rotate(30);                                                   // Rotate 30 degrees
  p.setPen(QPen(Qt::green, 2, Qt::SolidLine, Qt::FlatCap));
  p.drawPolygon(points, 4);

  QLabel myLabel;                                                 // Allocate a Gui widget
  myLabel.setPixmap(myMap);                                       // Associate pixmap with Gui widget
  myLabel.show();                                                 // Make widget visible
  return myApp.exec();                                            // Initiate event loop
} // End main()
```

Graphics Example 04



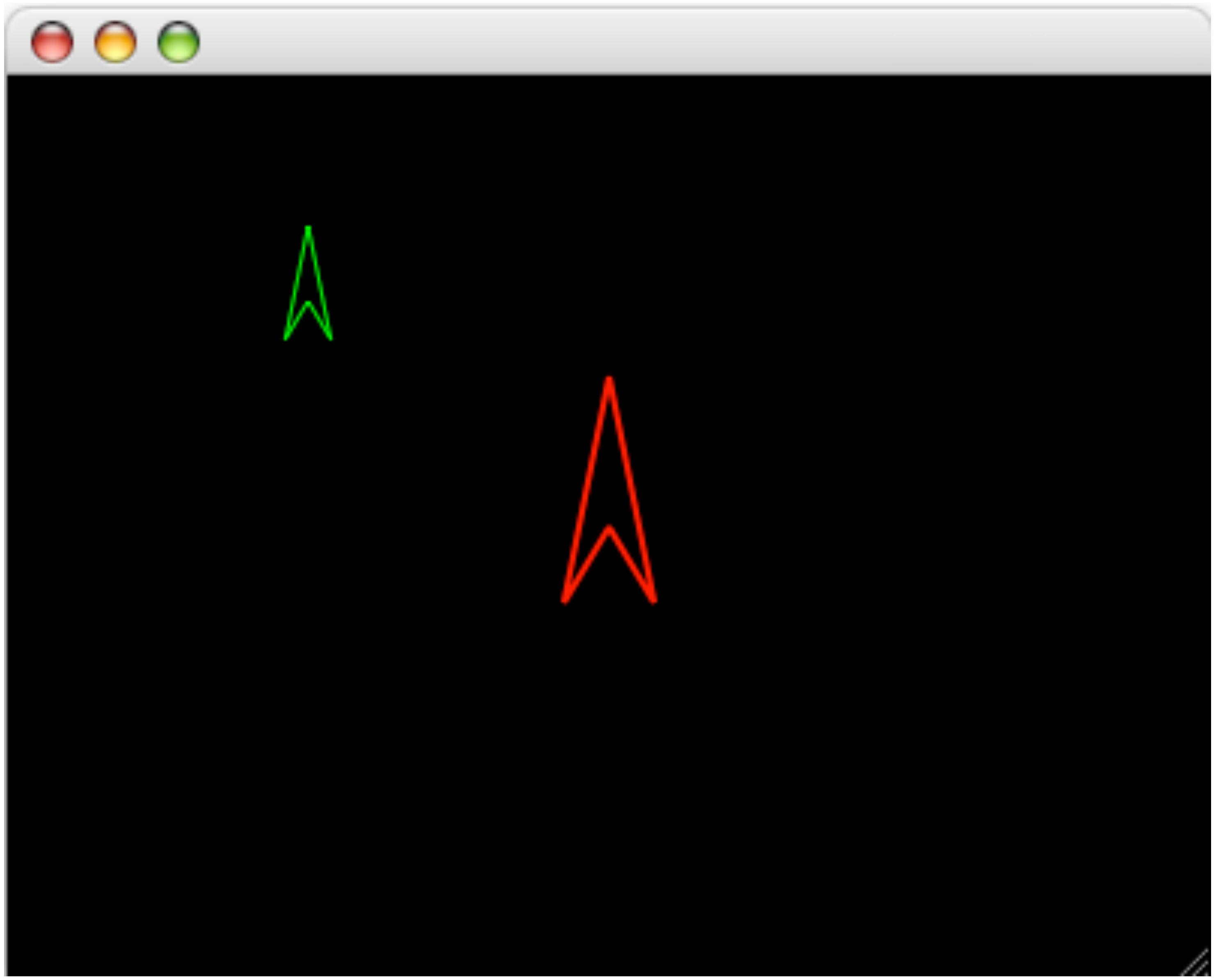
Graphics Example 05

- Goals
  - Draw an arrow using a polygon
  - Scale coordinates by factor of 50%
  - Redraw arrow

Graphics Example 05

```
//
// Graphics Example 5
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                          // Need application for event loop
  QPixmap  myMap(400, 300);                                // Establish pixmap
  myMap.fill(Qt::black);
  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);           // Enable antialiasing

  // Draw arrow
  p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::FlatCap));
  p.setBrush(QBrush(Qt::SolidPattern));
  QPoint points[4] = {QPoint(200, 100), QPoint(215, 175), QPoint(200, 150), QPoint(185, 175)};
  p.drawPolygon(points, 4);

  // Apply rotation and redraw polygon
  p.scale(0.5, 0.5);                                       // Scale X and Y by 0.5
  p.setPen(QPen(Qt::green, 2, Qt::SolidLine, Qt::FlatCap));
  p.drawPolygon(points, 4);

  QLabel myLabel;                                          // Allocate a Gui widget
  myLabel.setPixmap(myMap);                                // Associate pixmap with Gui widget
  myLabel.show();                                          // Make widget visible
  return myApp.exec();                                     // Initiate event loop
} // End main()
```

Graphics Example 05



Graphics Example 06

- Goals
  - Draw an arrow using a polygon
  - Save pixmap to a file in JPG format without displaying the pixmap
  - Load pixmap from file as a JPG image
  - Render loaded pixmap

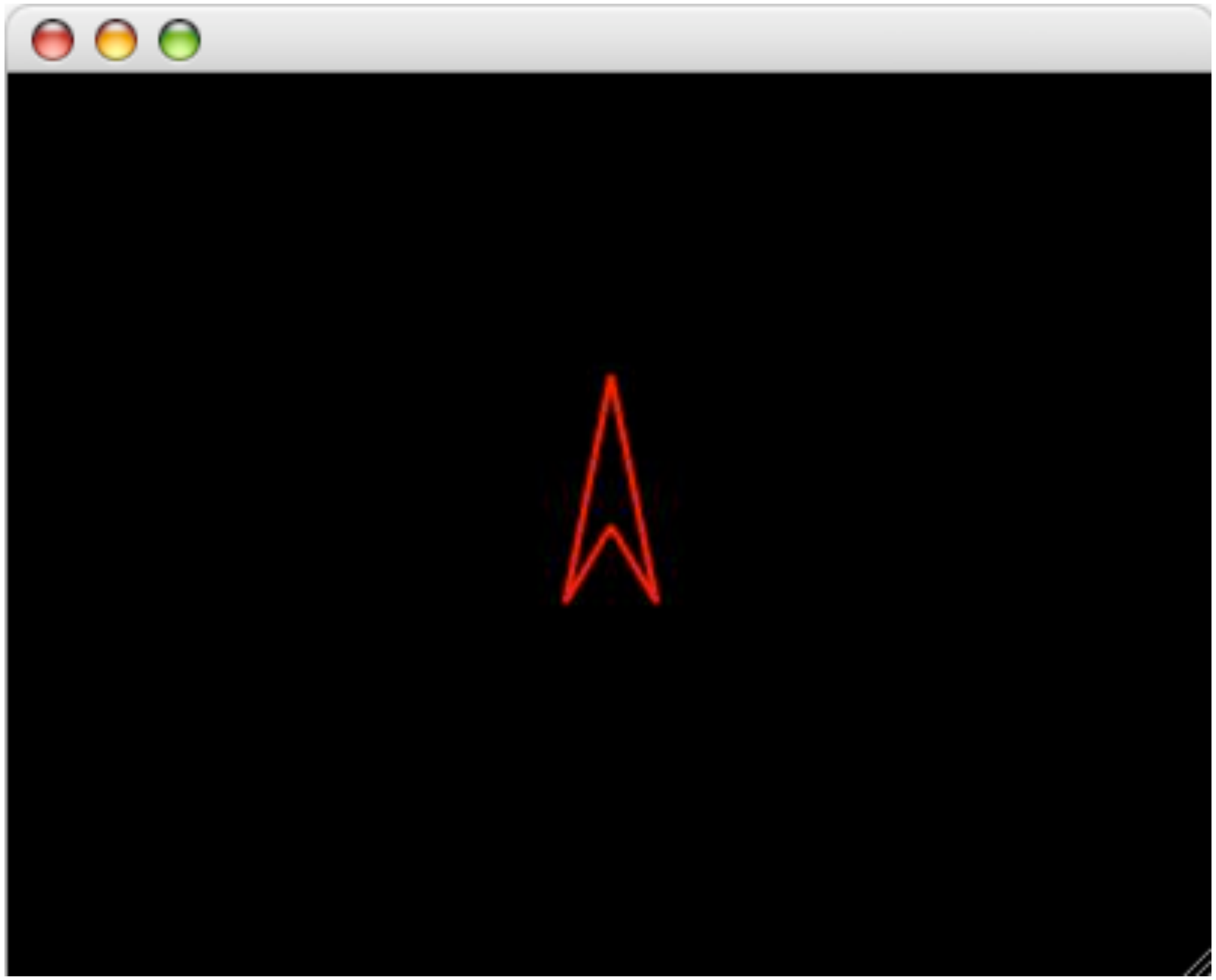
Graphics Example 06

```
//
// Graphics Example 6
//
#include <QApplication>
#include <QtDebug>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                          // Need application for event loop

  {  // Generate pixmap and save as a jpg image
    QPixmap  yourMap(400, 300);                            // Establish pixmap
    yourMap.fill(Qt::black);
    QPainter p(&yourMap);
    p.setRenderHint(QPainter::Antialiasing, true);         // Enable antialiasing

    // Draw arrow
    p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::FlatCap));
    p.setBrush(QBrush(Qt::SolidPattern));
    QPoint points[4] = {QPoint(200, 100), QPoint(215, 175), QPoint(200, 150), QPoint(185, 175)};
    p.drawPolygon(points, 4);

            // Attempt to save pixmap as jpg
            // 0 = determine image format by looking at filename; -1 = default image quality
            if (!yourMap.save("arrow.jpg", 0, -1))
              qDebug() << "Error - unable to save pixmap";
  }
```

Graphics Example 06

```
// Graphics Example 6 -- continued


  // Load jpeg image from file
  QPixmap  myMap;

  // Attempt to load pixmap as jpg
  // 0 = determine image format by looking at filename; auto conversion
  if (!myMap.load("arrow.jpg", 0, Qt::AutoColor))
    qDebug() << "Error - unable to load pixmap";

  QLabel myLabel;                              // Allocate a Gui widget
  myLabel.setPixmap(myMap);                    // Associate pixmap with Gui widget
  myLabel.show();                              // Make widget visible

  return myApp.exec();                         // Initiate event loop
} // End main()
```

Graphics Example 06



THE UNIVERSITY OF ALABAMA IN HUNTSVILLE

Graphics Example 07

- Goals
  - Establish a Window for logical coordinates
  - Draw an arrow as a red polygon
  - Apply translation
  - Redraw arrow as a green polygon

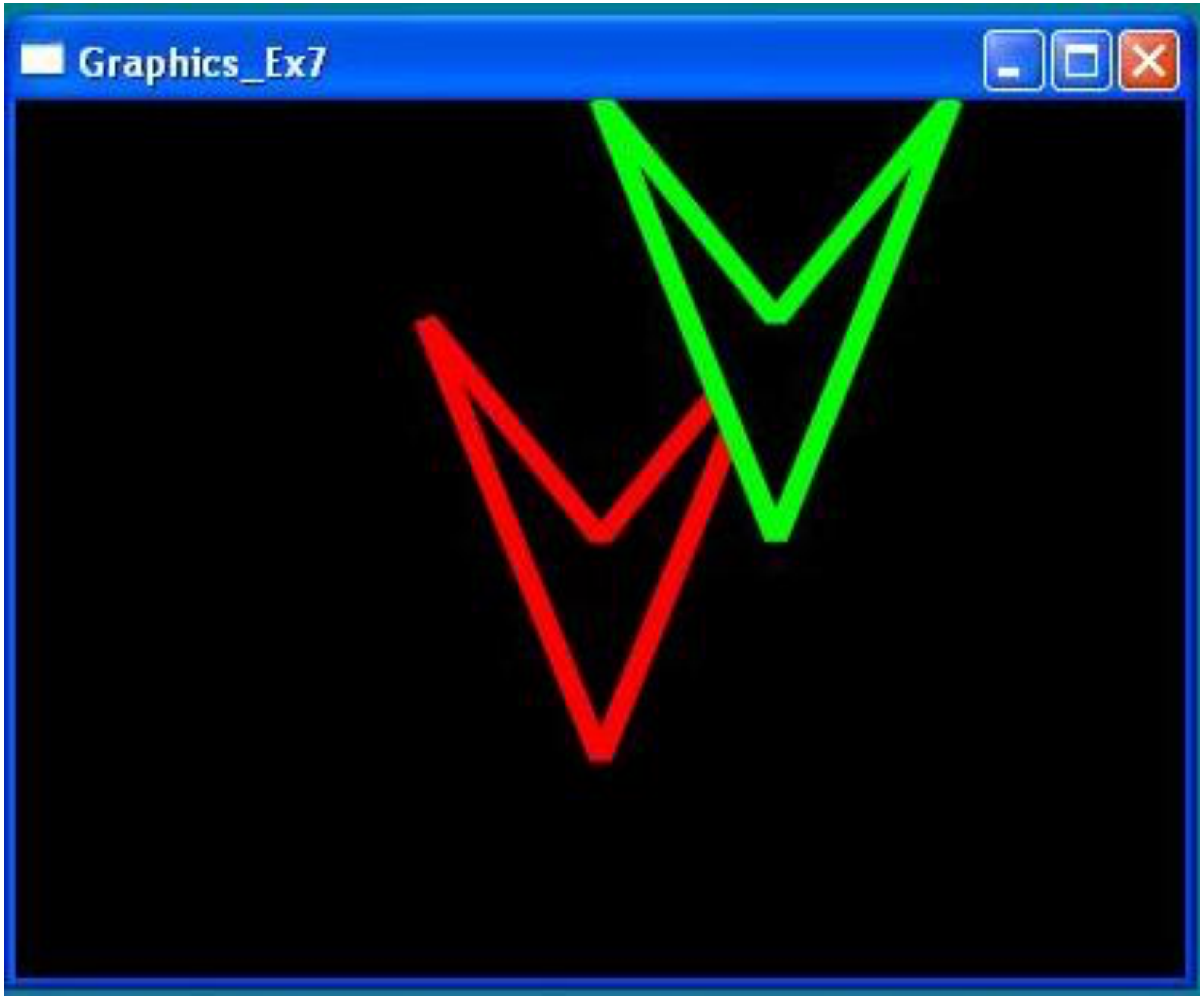
Graphics Example 07

```
//
// Graphics Example 7
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLine>
#include <QFont>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);       // Need application for event loop
  QPixmap  myMap(400, 300);             // Establish pixmap
  myMap.fill(Qt::black);
  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);  // Enables antialiasing
  p.setWindow(-50, -50, 100, 100);      // Define logical coordinate window
  // Logical (-50, -50) corresponds to Physical (0, 0)

  // Draw arrow using relative coordinates
  p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::FlatCap));
  p.setBrush(QBrush(Qt::SolidPattern));
  QPoint points[4] = {QPoint(0, 25), QPoint(15, -25), QPoint(0, 0), QPoint(-15, -25)};
  p.drawPolygon(points, 4);
```

Graphics Example 07

```
// Graphics Example 7 – continued

  // Apply translation and redraw polygon
  p.translate(15, -25);
  p.setPen(QPen(Qt::green, 2, Qt::SolidLine, Qt::FlatCap));
  p.drawPolygon(points, 4);

  QLabel myLabel;                      // Allocate a Gui widget
  myLabel.setPixmap(myMap);            // Associate pixmap with Gui widget
  myLabel.show();                      // Make widget visible

  return myApp.exec();                 // Initiate event loop
} // End main()
```

Graphics Example 07



Graphics Example 08

- Goals
  - Establish a Window for logical coordinates
  - Draw an arrow as a red polygon
  - Apply rotation
  - Redraw arrow as a green polygon

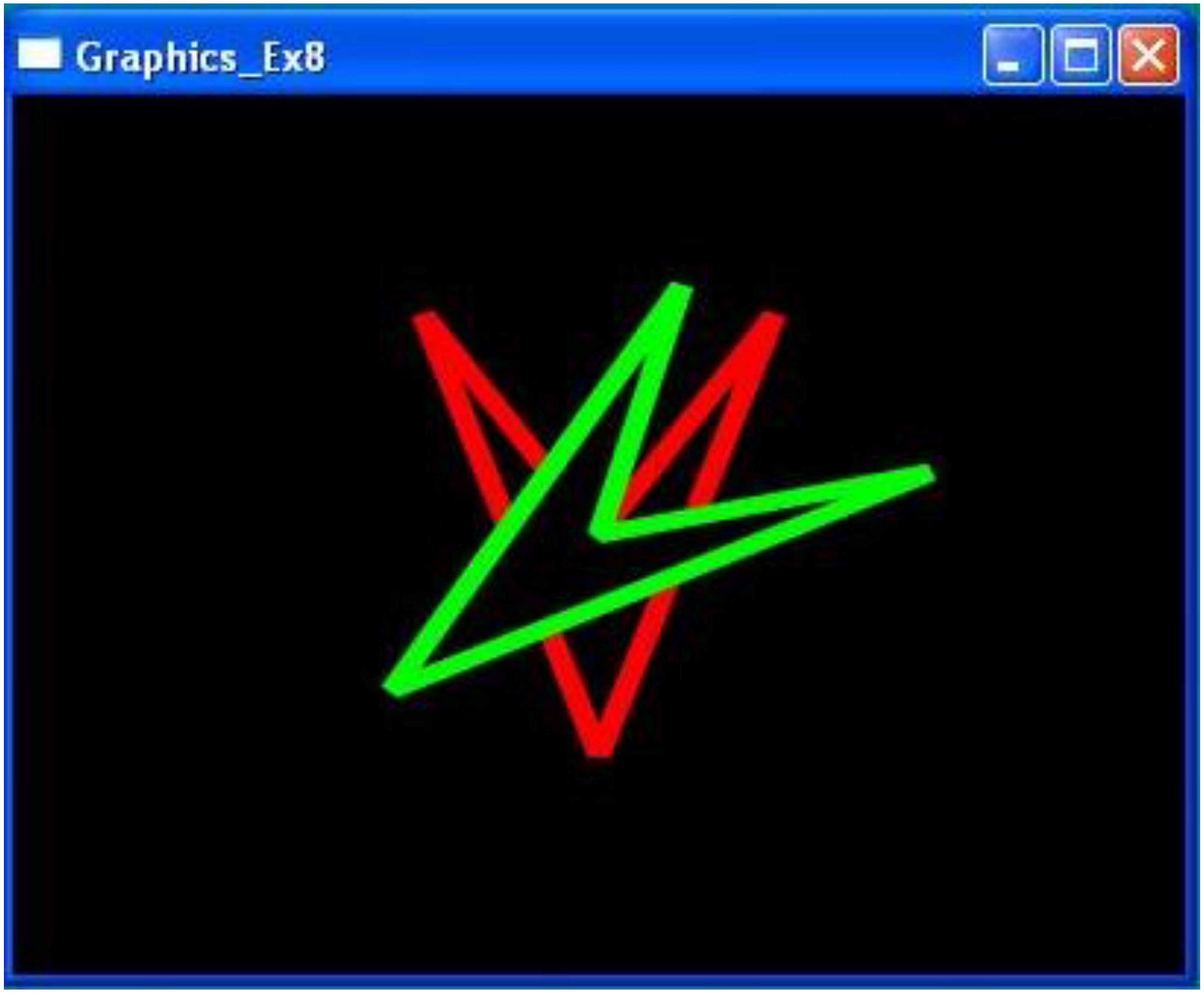
Graphics Example 08

```
//
// Graphics Example 8
//
#include <QApplication>
#include <QPainter>
#include <QPixmap>
#include <QPen>
#include <QBrush>
#include <QRect>
#include <QPoint>
#include <QLine>
#include <QFont>
#include <QLabel>

int main(int argc, char* argv[])
{
  QApplication myApp(argc, argv);                           // Need application for event loop
  QPixmap  myMap(400, 300);                                 // Establish pixmap
  myMap.fill(Qt::black);
  QPainter p(&myMap);
  p.setRenderHint(QPainter::Antialiasing, true);            // Enables antialiasing
  p.setWindow(-50, -50, 100, 100);                          // Define logical coordinate window
  // Logical (-50, -50) corresponds to Physical (0, 0)

  // Draw arrow
  p.setPen(QPen(Qt::red, 2, Qt::SolidLine, Qt::FlatCap));
  p.setBrush(QBrush(Qt::SolidPattern));
  QPoint points[4] = {QPoint(0, 25), QPoint(15, -25), QPoint(0, 0), QPoint(-15, -25)};
  p.drawPolygon(points, 4);
```

Graphics Example 08

```
// Graphics Example 8 -- continued

  // Rotate 45 degrees and redraw polygon
  p.rotate(45.0);
  p.setPen(QPen(Qt::green, 2, Qt::SolidLine, Qt::FlatCap));
  p.drawPolygon(points, 4);

  QLabel myLabel;                               // Allocate a Gui widget
  myLabel.setPixmap(myMap);                     // Associate pixmap with Gui widget
  myLabel.show();                               // Make widget visible

  return myApp.exec();                          // Initiate event loop
} // End main()
```

Graphics Example 08 - 3



QIcon/QPainter Example

- Goal
  - Create a push button widget that visually indicates toggle status by switching between a Black and Red icon

QIcon/QPainter Example

```
#include <QtGui/QApplication>
#include "widget.h"
int main(int argc, char *argv[])
{
  QApplication a(argc, argv);
  Widget w;
  w.show();
  return a.exec();
}
```

QIcon/QPainter Example

```
#ifndef WIDGET_H
#define WIDGET_H

#include <QPushButton>
#include <QPixmap>
#include <QPainter>
#include <QIcon>

class Widget : public QPushButton
{
  Q_OBJECT

  public:
    Widget(QWidget *parent = 0);
    ~Widget();

  protected:
    void paintEvent(QPaintEvent* pe);   // Override paint event handler

  private:
    QPixmap* redPixmap;
    QPixmap* blackPixmap;
    QPainter* painter;
    QIcon* redIcon;
    QIcon* blackIcon;
    bool illuminated;

  private slots:
    void togglePixmap();                // Toggles red/black button icon
};
#endif // WIDGET_H
```

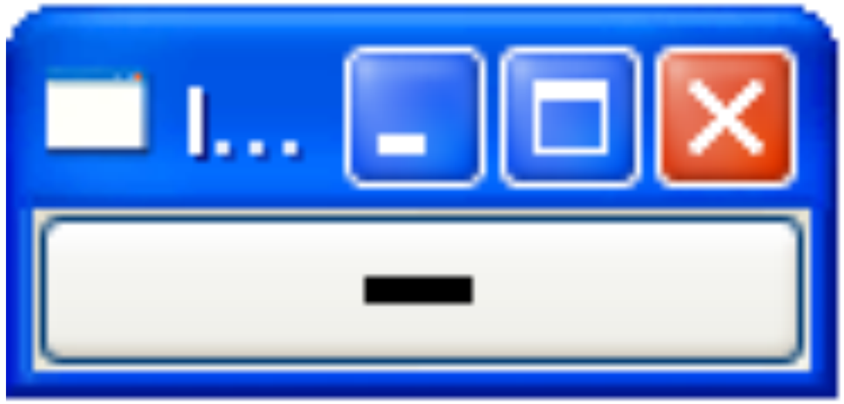
QIcon/QPainter Example

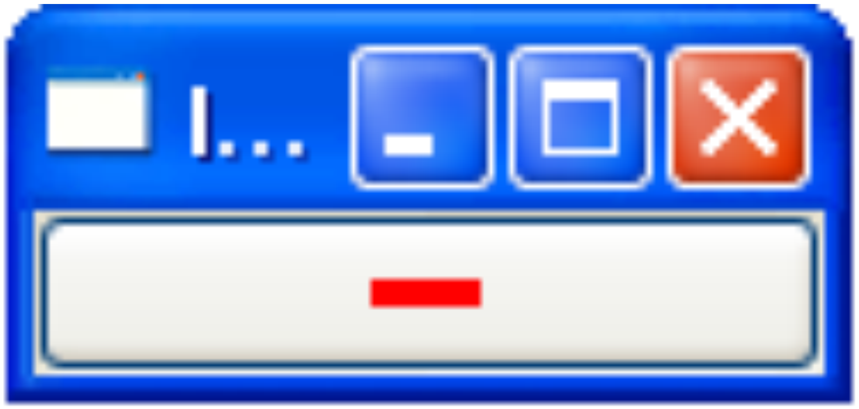
```
#include "widget.h"

Widget::Widget(QWidget *parent) : QPushButton(parent)
{
  illuminated = false;

  blackPixmap = new QPixmap(400,100);
  painter = new QPainter(blackPixmap);
  blackPixmap->fill(Qt::black);
  blackIcon = new QIcon(*blackPixmap);
  delete painter;
  delete blackPixmap;

  redPixmap = new QPixmap(400,100);
  painter = new QPainter(redPixmap);
  redPixmap->fill(Qt::red);
  redIcon = new QIcon(*redPixmap);
  delete painter;
  delete redPixmap;

  this->setIcon(*blackIcon);
  connect(this, SIGNAL(clicked()), this, SLOT(togglePixmap()));
}
```

QIcon/QPainter Example

```
#include "widget.h"

void Widget::paintEvent(QPaintEvent* pe)
{
  // Adjust icon based upon illuminated flag
  if (illuminated)
      this->setIcon(*redIcon);
  else
      this->setIcon(*blackIcon);

  // Pass on other paint events to parent class event handler
  QPushButton::paintEvent(pe);
}

void Widget::togglePixmap()
{
  // Toggle illuminated flag
  illuminated = !illuminated;

  // Trigger update of widget display
  this->update();
}

Widget::~Widget()
{
}
```

QIcon/QPainter Example





QColor Class

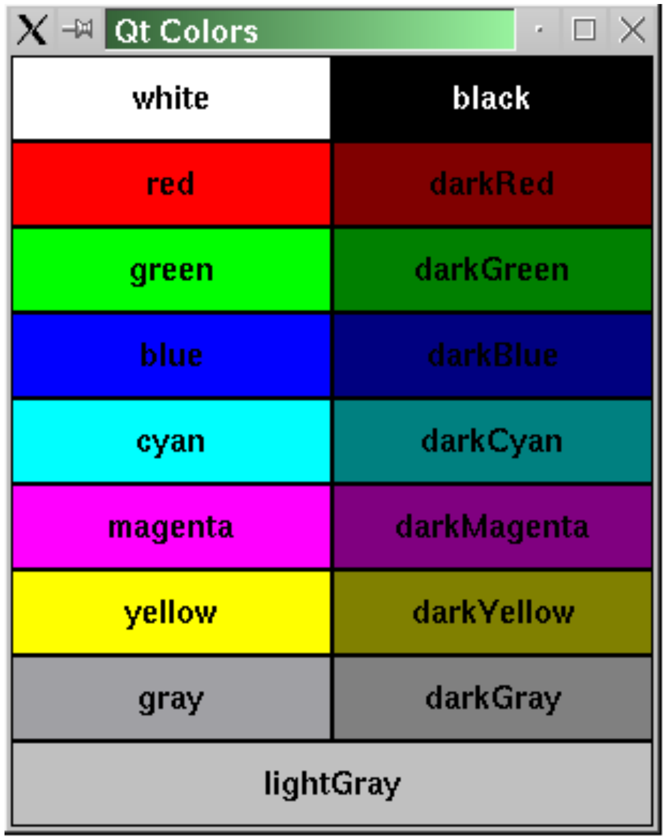
- Uses RGB representation
  - Reserves 8-bits (0-255) for each color Red, Green, Blue
  - Also reserves 8-bits for Alpha Channel
    - Describes pixel transparency
  - QColor can work with ints or floats
    - setRgb() ints
    - setRgbF() floats
  - Hex notation
    - 0xAARRGGBB

QColor Class

- Several variants of the QColor constructor
- Example
  - QColor   x(255, 127, 64, 0);
    - Red = 255
    - Green = 127
    - Blue = 64
    - Alpha = 0

QColor Class

- Another Example
  - QColor y("black");
    - Predefined SVG color names (Scalable Vector Graphics)



QColor Class

- Also supports other color models
  - HSV (Hue-Saturation-brightnessValue)
  - CMYK (Cyan-Magenta-Yellow-Black)
  - Methods included to convert to/from various color models
    - toHsv(), toCmyk(), toRgb()
- Use caution
  - Qt uses RGB internally

QPixmap vs QImage

- QPixmap
  - Available for QApplication, not QCoreApplication use
  - Operations handled by graphics card
- QImage
  - Available for either QApplication or QCoreApplication use
  - Operations performed by processor
  - Still part of Qt GUI library

QPixmap vs QImage

- QImage
  - Uses RGB representation
    - Reserves 8-bits (0-255) for each color Red, Green, Blue
    - May or may not include 8-bits for Alpha Channel
      - QImage::Format_RGB32
      - QImage::Format_ARGB32

QPixmap vs QImage

- QImage
  - scanLine() can retrieve pixel color information as unsigned char array
  - Byte order can impact interpretation
    - Little Endian
      - LSB first
    - Big Endian
      - MSB first
  - Qt resolves byte order by using QRgb type
    - Type reinterpret picks up values in platform byte order
    - See section 10.7.2

Key Points

- **QPainter** objects may be used to draw on pixmaps within a Qt program
- Standard transformations (rotation, translation, and scaling) may be applied as needed
- Be aware of the coordinate system you are using since the transformations produce different results with respect to absolute and relative coordinate systems.