Programming with Qt

Prof. George Wolberg
Dept. of Computer Science
City College of New York

Objectives

- Introduce Qt for our GUI
- Write our first “Hello World” program
- Leverage OpenGL for drawing
- Leverage Qt for user interface
- Prototype of homework submissions

Application Framework Requirements

- OpenGL applications need a place to render into
  - usually an on-screen window
- Need to communicate with native windowing system
- Each windowing system interface is different
- We use Qt
  - Advanced C++ GUI library that works everywhere
  - handles widgets and all windowing operations:
    - opening windows
    - input processing
    - widgets such as menus, sliders, spinboxes, combo boxes

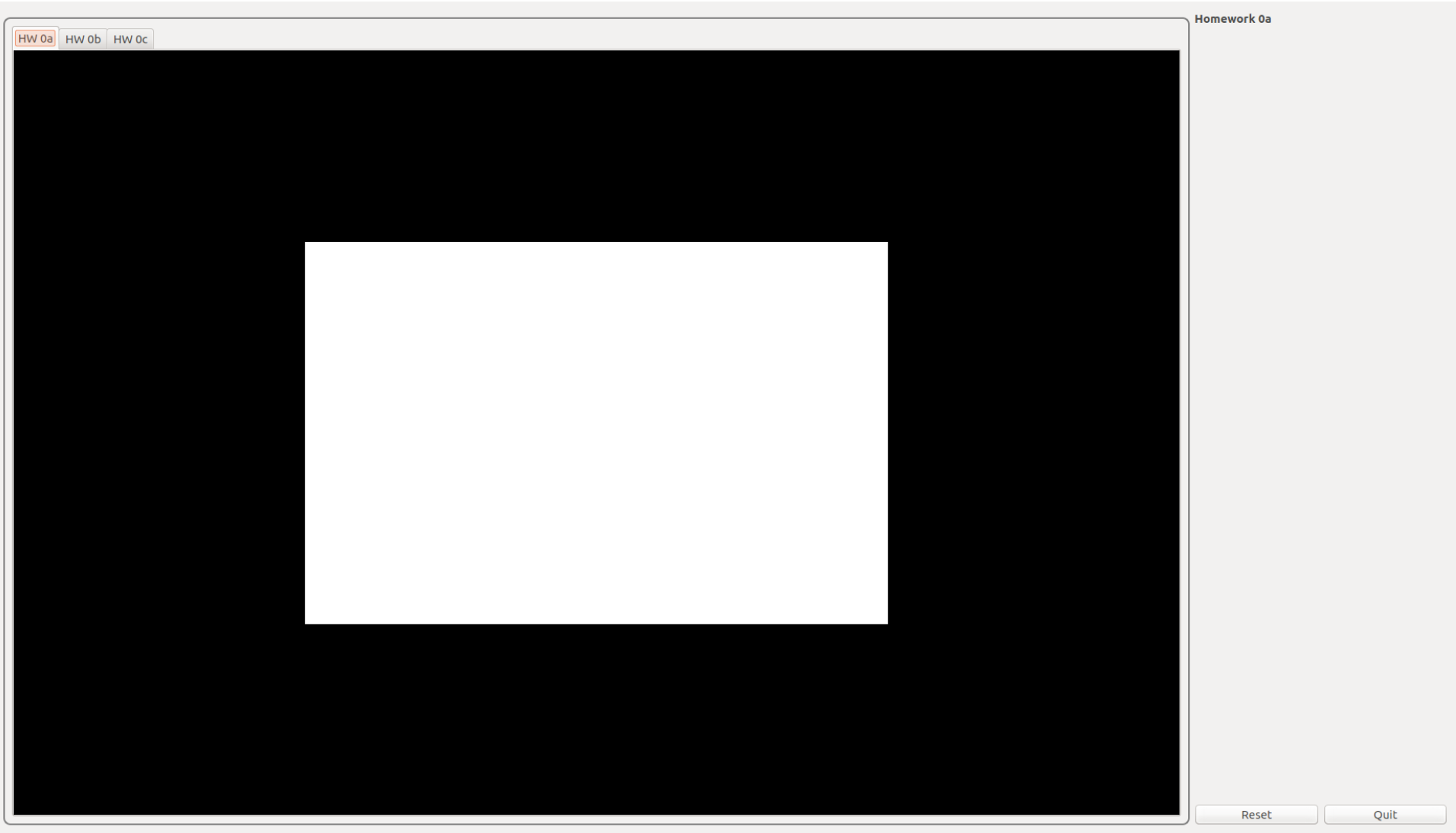
Qt

- Qt is ‘cute’
- Cross-platform GUI library (C++)
  - Write once, compile on different platforms for cross-platform use
  - Works on Windows, Mac, Linux, iOS, Android
- A complete interface environment with support for file system browser, OpenGL, WebKit API, media streamer, etc.
- Download from: http://www.qt.io/download-open-source

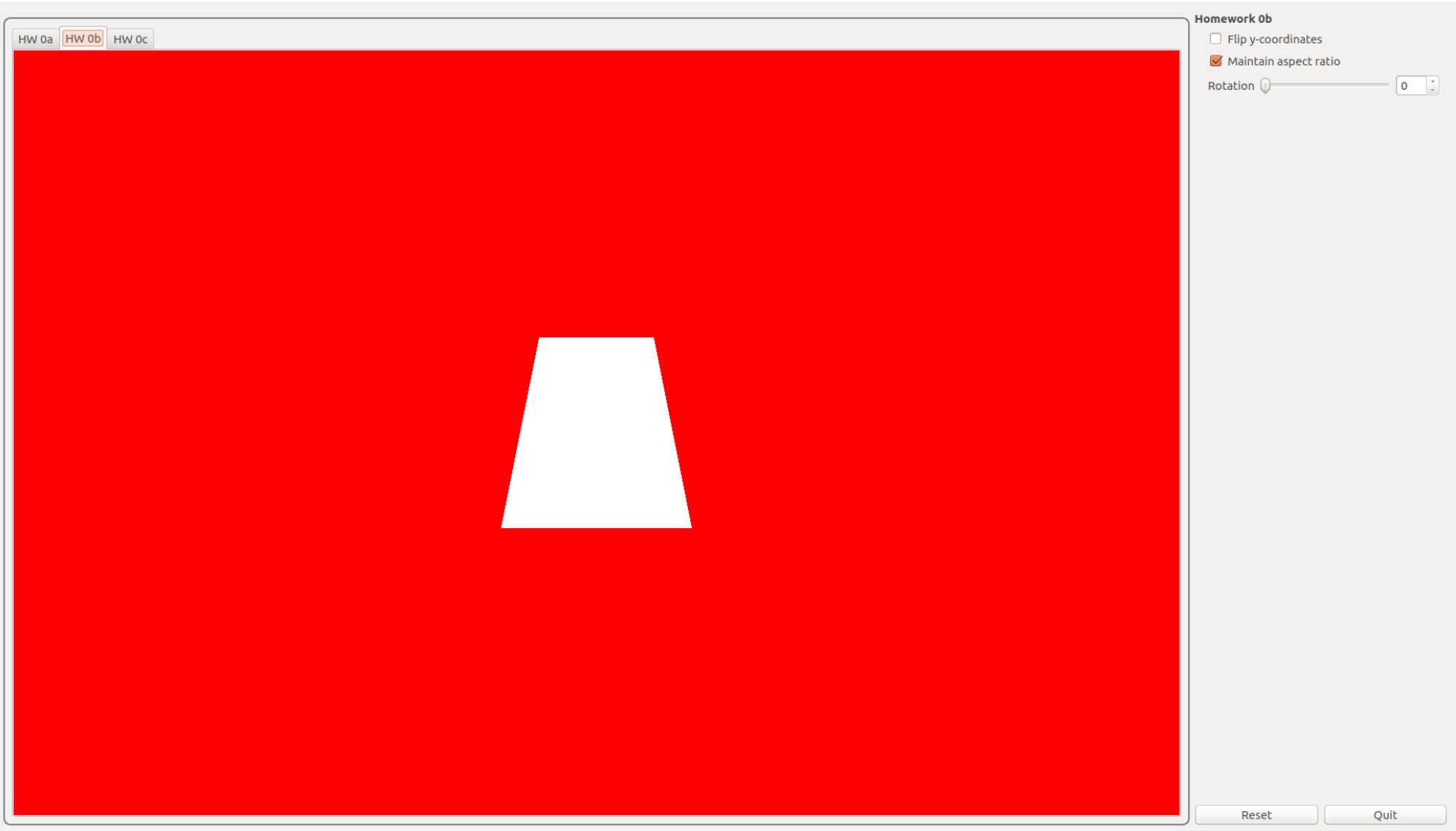
Homework Interface

- Homework submissions will use Qt
  - Renderings based on OpenGL
  - Parameter selection using advanced GUI
- The screen will be divided into two parts:
  - Tabbed widget containing OpenGL canvas
  - Control panel containing widgets to select parameters
- Each homework problem will be installed in a tabbed widget
  - Pressing a tab brings up associated homework solution and control panel

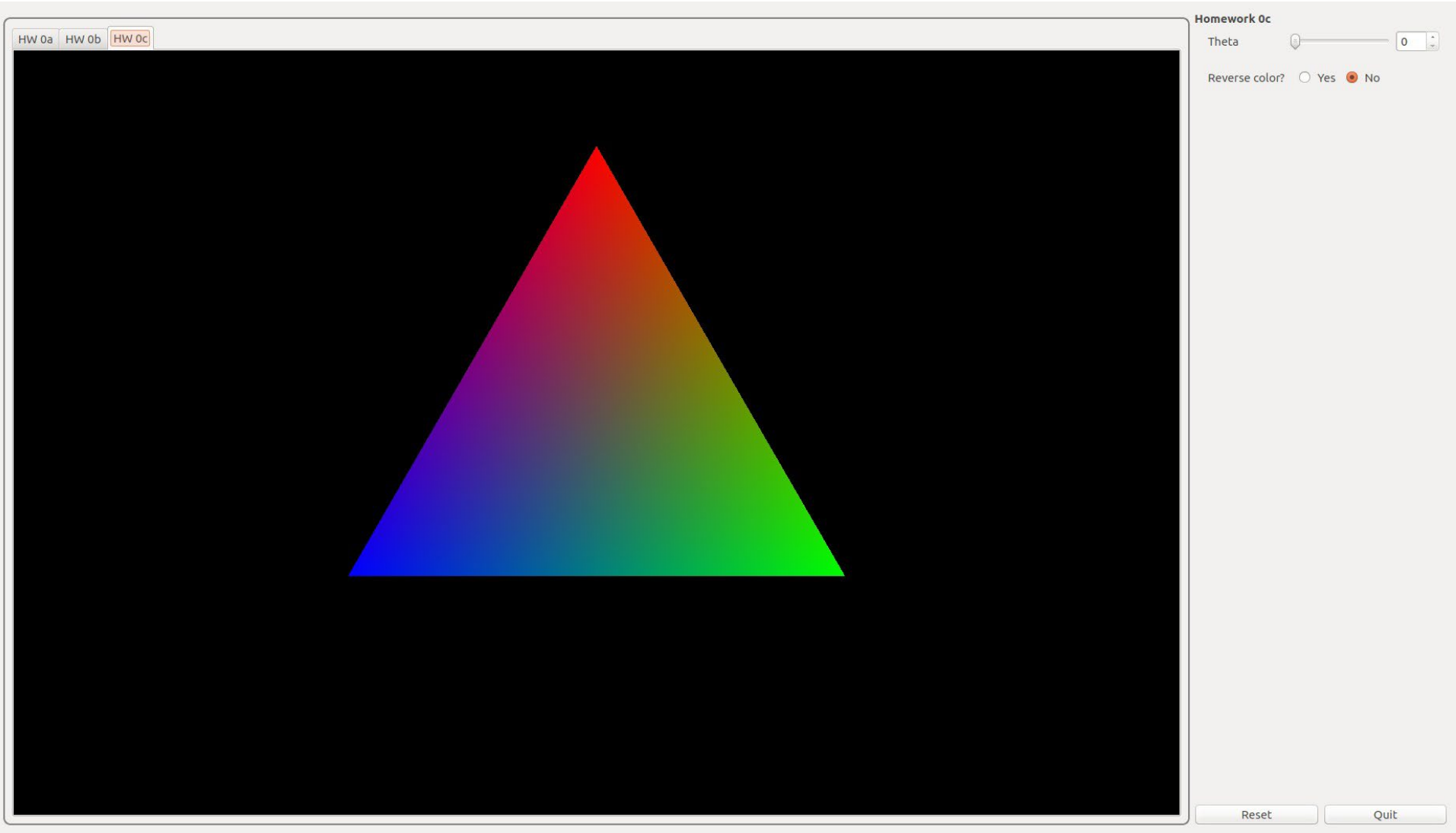
Interface (1)



Interface (2)


HW 0a
HW 0b
HW 0c
Homework 0b
Flip y-coordinates
Maintain aspect ratio
Rotation
0
Reset
Quit

Interface (3)


HW 0a
HW 0b
HW 0c
Homework 0c
Theta
0
Reverse color?
Yes
No
Reset
Quit

Files

- main.cpp
  - Create and display UI window
  - Enter infinite loop to process events
- MainWindow.h
  - Header file for MainWindow class
- MainWindow.cpp
  - MainWindow class sets up GUI
- HW.h
  - Header file for HW class
- HW.cpp
  - Base class for homework solutions
  - Virtual functions: initializeGL(), resizeGL(), and paintGL()

main.cpp

```
// ========================================================================
// Computer Graphics Homework Solutions
// Copyright (C) 2022 by George Wolberg
//
// main.c - main() function.
//
// Written by: George Wolberg, 2022
// ========================================================================

#include "MainWindow.h"

int main(int argc, char **argv)
{
	QApplication app(argc, argv);		// create application
	MainWindow window;			// create UI window
	window.showMaximized();			// display maximized window
	return app.exec();			// infinite processing loop
}
```

MainWindow.h (1)

```
#ifndef  MAINWINDOW_H
#define MAINWINDOW_H

// ----------------------------------------------------------------
// standard include files
//
#include <QtWidgets>
#include <iostream>
#include <fstream>
#include <cstdio>
#include <cmath>
#include <cstring>
#include <cstdlib>
#include <cstdarg>
#include <cassert>
#include <vector>
#include <map>
#include <algorithm>
#include "HW.h"

enum {DUMMY, HW0A, HW0B, HW0C, HW0D, HW1A, HW1B, HW2A, HW2B, HW3A, HW3B, HW4A, HW4B};


typedef std::map<QString, HW*> hw_type;
```

MainWindow.h (2)

```
class MainWindow : public QMainWindow
{
        Q_OBJECT

public:
        // constructor
        MainWindow (QWidget *parent = 0);

public slots:
        void            createWidgets();
        void            changeHW    (QAction*);
        void            reset       ();
        void            quit        ();

protected:
        QFrame*         createGroupView         ();
        void            createActions           ();
        void            createMenus             ();
        QHBoxLayout*    createExitButtons       ();

private:
        // homework objects
        QStringList     m_hwName;
        hw_type         m_hw;

        // widgets
        QStackedWidget          *m_stackWidgetPanels;
        QStackedWidget          *m_stackWidgetViews;
```

MainWindow.h (3)

```
        // menus
        QMenu                   *m_menuHW0;
        QMenu                   *m_menuHW1;
        QMenu                   *m_menuHW2;
        QMenu                   *m_menuHW3;
        QMenu                   *m_menuHW4;

        // actions
        QAction                 *m_actionHW0a;
        QAction                 *m_actionHW0b;
        QAction                 *m_actionHW0c;
        QAction                 *m_actionHW0d;
        QAction                 *m_actionHW1a;
        QAction                 *m_actionHW1b;
        QAction                 *m_actionHW2a;
        QAction                 *m_actionHW2b;
        QAction                 *m_actionHW3a;
        QAction                 *m_actionHW3b;
        QAction                 *m_actionHW4a;
        QAction                 *m_actionHW4b;
};

extern MainWindow       *MainWindowP;

#endif // MAINWINDOW_H
```

MainWindow.cpp (1)

```
// ============================================================
// Computer Graphics Homework Solutions
// Copyright (C) 2022 by George Wolberg
//
// MainWindow.cpp - MainWindow class
//
// Written by: George Wolberg, 2022
// ============================================================

#include "MainWindow.h"
#include "dummy.h"
#include "hw0/HW0a.h"
#include "hw0/HW0b.h"
#include "hw0/HW0c.h"
#include "hw0/HW0d.h"
#include "hw1/HW1a.h"
#include "hw1/HW1b.h"
#include "hw2/HW2a.h"
#include "hw2/HW2b.h"
#include "hw3/HW3a.h"
#include "hw3/HW3b.h"
#include "hw4/HW4a.h"
#include "hw4/HW4b.h"

MainWindow *MainWindowP = NULL;
```

MainWindow.cpp (2)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::MainWindow:
//
// MainWindow constructor.
//
MainWindow::MainWindow(QWidget *parent) : QMainWindow(parent)
{
        setWindowTitle("Computer Graphics Homework");
        setWindowIcon(Qicon(“;/CS472.png”));

        // reasonable minimum size
        setMinimumSize(Qsize(800, 600));

        // set global variable
        MainWindowP = this

        // create a stacked widget to hold multiple control panels
        m_stackWidgetPanels = new QStackedWidget;
        m_stackWidgetViews  = new QStackedWidget;

        // create widgets, actions, and menus
        createWidgets();            // insert your widgets here
        createActions ();           // insert your actions  here
        createMenus  ();            // insert your menus  here

        // add stacked widget to vertical box layout
        QFrame *frame = new QFrame;
        frame->setObjectName(“frame”);
        frame->setStyleSheet(frameStyle);
        frame->setMinimumWidth(300);
        QVBoxLayout *vbox = new QVBoxLayout;
        vbox->addWidget(m_stackWidgetPanels);
        vbox->addStretch(1);
        vbox->addLayout(createExitButtons());
        frame->setLayout(vbox);
```

MainWindow.cpp (3)

```
    // add all widgets to grid layout
    QHBoxLayout *hbox = new QHBoxLayout;
    hbox->addWidget(createGroupView());
    hbox->setStretch(0, 1);
    hbox->addWidget(frame);

    // create container widget and set its layour
    QWidget *w = new QWidget;
    w->setLayout(hbox)
    setCentralWidget(w);

    // set stacked widget to latest solution
    m_stackWidgetViews ->setCurrentIndex(DUMMY);
    m_stackWidgetPanels->setCurrentIndex(DUMMY);
    m_hw[m_hwName[DUMMY]]->setMaximumSize(1,1); // nothing to show
    m_hw[m_hwName[DUMMY]]->setMinimumSize (1,1); // nothing to show
}
```

MainWindow.cpp (4)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createWidgets:
//
// Create user-defined widgets for display and control panel.
// INSERT YOUR LINES HERE TO CREATE A TAB PER HOMEWORK OBJECT.
//
void
MainWindow::createWidgets()
{
        // create list of hw names; m_hwName name will be used for
        // tab name and as key for class in m_hw container
        m_hwName    << “Blank page”
                    << “0a: Square          << “0b: Aspect ratio    << “0c: GLSL    << “0d: Transformations
                    << "1a: P“              << "1b: Triangle"
                    << "2a: P (GLSL)“       << "2b: Triangle (GLSL)"
                    << "3a: Triangle (Texture/Wire)“ << "3b: Wave"
                    << “4a: Bounce”         << “4b: Shadow”;

        // format for legacy OpenGL with older GLSL (supporting attribute/varying qualifiers)
        QGLFormat glfLegacy = QGLFormat::defaultFormat();       // base format
        glfLegacy.setProfile(QGLFormat::CompatibilityProfile);  // also support legacy version
        glfLegacy.setSampleBuffers(true);                       // multisample buffer support for antialiasing (AA)
        glfLegacy.setSamples(4);                                // number of samples per fragment for AA
        glfLegacy.setSwapInterval(0);
        glfLegacy.setDefaultFormat(glfLegacy);                  // use modified parameters

        // format for modern OpenGL (3.3+) with newer GLSL (supporting in/out/layout)
        QGLFormat glfModern = QGLFormat::defaultFormat();       // base format
        glfModern.setVersion(3, 3);                             // Mac requires 3.3+ for core profile
        glfModern.setProfile(QGLFormat::CoreProfile);           // don't use deprecated functions
        glfModern.setSampleBuffers(true);                       // multisample buffer support for antialiasing (AA)
        glfModern.setSamples(4);                                // number of samples per fragment (for AA)
        glfModern.setSwapInterval(0);
        glfModern.setDefaultFormat(glfModern);                  // use modified parameters
```

MainWindow.cpp (5)

```
        // instantiate homework solution classes
        m_hw[m_hwName[DUMMY]] = new Dummy(glfLegacy);
        m_hw[m_hwName[HW0A ]] = new HW0a (glfLegacy);
        m_hw[m_hwName[HW0B ]] = new HW0b (glfLegacy);
        m_hw[m_hwName[HW0C ]] = new HW0c (glfModern);
        m_hw[m_hwName[HW0D ]] = new HW0d (glfLegacy);
        m_hw[m_hwName[HW1A ]] = new HW1a (glfLegacy);
        m_hw[m_hwName[HW1B ]] = new HW1b (glfLegacy);
        m_hw[m_hwName[HW2A ]] = new HW2a (glfModern);
        m_hw[m_hwName[HW2B ]] = new HW2b (glfModern);
        m_hw[m_hwName[HW3A ]] = new HW3a (glfModern);
        m_hw[m_hwName[HW3B ]] = new HW3b (glfModern);
        m_hw[m_hwName[HW4A ]] = new HW4a (glfLegacy);
        m_hw[m_hwName[HW4B ]] = new HW4b (glfModern);

        // add control panels to stacked widget
        for(int i = 0; i < (int) m_hwName.size(); ++i)
                        m_stackWidgetPanels->addWidget(m_hw[m_hwName[i]]->controlPanel());
}
```

MainWindow.cpp (6)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createActions:
//
// Create actions to associate with menu and toolbar selection.
//
void MainWindow::createActions()
{
                ////////////////////////////
                // HW 0 actions
                ////////////////////////////

                m_actionHW0a = new QAction(m_hwName[HW0A], this);
                m_actionHW0a->setData(HW0A);

                m_actionHW0b = new QAction(m_hwName[HW0B], this);
                m_actionHW0b->setData(HW0B);

                m_actionHW0c = new QAction(m_hwName[HW0C], this);
                m_actionHW0c->setData(HW0C);

                m_actionHW0d = new QAction(m_hwName[HW0D], this);
                m_actionHW0d->setData(HW0D);

                ////////////////////////////
                // HW 1 actions
                ////////////////////////////

                m_actionHW1a = new QAction(m_hwName[HW1A], this);
                m_actionHW1a->setData(HW1A);

                m_actionHW1b = new QAction(m_hwName[HW1B], this);
                m_actionHW1b->setData(HW1B);
```

MainWindow.cpp (7)

```
////////////////////////////
// HW 2 actions
////////////////////////////
m_actionHW2a = new QAction(m_hwName[HW2A], this);
m_actionHW2a->setData(HW2A);

m_actionHW2b = new QAction(m_hwName[HW2B], this);
m_actionHW2b->setData(HW2B);

////////////////////////////
// HW 3 actions
////////////////////////////
m_actionHW3a = new QAction(m_hwName[HW3A], this);
m_actionHW3a->setData(HW3A);

m_actionHW3b = new QAction(m_hwName[HW3B], this);
m_actionHW3b->setData(HW3B);

////////////////////////////
// HW 4 actions
////////////////////////////
m_actionHW4a = new QAction(m_hwName[HW4A], this);
m_actionHW4a->setData(HW4A);
m_actionHW4b = new QAction(m_hwName[HW4B], this);
m_actionHW4b->setData(HW4B);

// one signal-slot connection for all actions;
// execute() will resolve which action was triggered
connect(menuBar(), SIGNAL(triggered(QAction*)), this, SLOT(changeHW(QAction*)));
}
```

MainWindow.cpp (8)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createMenus:
//
// Create menus and install in menubar.
//
void
MainWindow::createMenus()
{
        // hw0 menu
        m_menuHW0 = menuBar()->addMenu("HW0");
        m_menuHW0->addAction(m_actionHW0a);
        m_menuHW0->addAction(m_actionHW0b);
        m_menuHW0->addAction(m_actionHW0c);
        m_menuHW0->addAction(m_actionHW0d);

        // hw1 menu
        m_menuHW1 = menuBar()->addMenu("HW1");
        m_menuHW1->addAction(m_actionHW1a);
        m_menuHW1->addAction(m_actionHW1b);

        // hw2 menu
        m_menuHW2 = menuBar()->addMenu("HW2");
        m_menuHW2->addAction(m_actionHW2a);
        m_menuHW2->addAction(m_actionHW2b);

        // hw3 menu
        m_menuHW3 = menuBar()->addMenu("HW3");
        m_menuHW3->addAction(m_actionHW3a);
        m_menuHW3->addAction(m_actionHW3b);

        // hw4 menu
        m_menuHW4 = menuBar()->addMenu("HW4");
        m_menuHW4->addAction(m_actionHW4a);
        m_menuHW4->addAction(m_actionHW4b);
}
```

MainWindow.cpp (9)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createGroupView:
//
// Create preview window groupbox.
//
QFrame*
MainWindow::createGroupView()
{
        // init group box
        QFrame *frame = new QFrame();
        frame->setObjectName("frame");
        frame->setStyleSheet(frameStyle);

        // create a stack widget to handle multiple displays
        // one view per homework problem
        for(int i = 0; i < (int) m_hwName.size(); ++i)
                m_stackWidgetViews->addWidget(m_hw[m_hwName[i]]);

        // assemble stacked widget in vertical layout
        QVBoxLayout *vbox = new QVBoxLayout;
        vbox->addWidget(m_stackWidgetViews);
        frame->setLayout(vbox);

        return frame;
}
```

MainWindow.cpp (10)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::changeHW:
//
// Slot function to change OpenGL canvas and control panel in stacked widget.
//
void
MainWindow::changeHW(QAction* action)
{
        // get code from action
        int index = action->data().toInt();
        m_stackWidgetViews ->setCurrentIndex(index);        // change OpenGL widget
        m_stackWidgetPanels->setCurrentIndex(index);        // change control panel
}
```

MainWindow.cpp (11)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createExitButtons:
//
// Create save/quit buttons.
//
QHBoxLayout *
MainWindow::createExitButtons()
{
        // create pushbuttons
        QPushButton *buttonReset = new QPushButton("Reset");
        QPushButton *buttonQuit  = new QPushButton("Quit");

        // init signal/slot connections
        connect(buttonReset, SIGNAL(clicked()), this, SLOT(reset()));
        connect(buttonQuit , SIGNAL(clicked()), this, SLOT(quit ()));

        // assemble pushbuttons in horizontal layout
        QHBoxLayout *buttonLayout = new QHBoxLayout;
        buttonLayout->addWidget(buttonReset);
        buttonLayout->addWidget(buttonQuit );

        return buttonLayout;
}
```

MainWindow.cpp (12)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::reset:
//
// Reset application.
//
void
MainWindow::reset()
{
        int index = m_stackWidgetViews->currentIndex();     // index to current OpenGL widget
        m_hw[ m_hwName[index] ]->reset();                   // invoke respective reset function
}


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::quit:
//
// Quit application.
//
void
MainWindow::quit()
{
        // close the dialog window
        close();
}
```

HW.h (1)

```
// ========================================================================
// Computer Graphics Homework Solutions
// Copyright (C) 2022 by George Wolberg
//
// HW.h - Header file for HW class. Base class of homework solutions.
//
// Written by: George Wolberg, 2022
// ========================================================================

#ifndef HW_H
#define HW_H

#define MXPROGRAMS    32
#define MXUNIFORMS    32

#define PII           3.14159265358979311600E0
#define PI2           6.28318530717958623200E0
#define PI_2          1.57079632679489655800E0
#define DEGtoRAD      0.01745329277777777777E0
#define RADtoDEG      57.29577866666161658617E0

#include <QtWidgets>
#include <QGLWidget>
#include <QGLFunctions>
#include <QGLShaderProgram>
#include <QtOpenGL>

typedef QVector2D vec2;
typedef QVector3D vec3;
```

HW.h (2)

```
enum {
        ATTRIB_VERTEX,
        ATTRIB_COLOR,
        ATTRIB_TEXCOORD,
        ATTRIB_NORMAL
};

typedef std::map<QString, GLuint> UniformMap;

extern QString GroupBoxStyle;


// ------------------------------------------------------------------
// standard include files
//
class HW : public QGLWidget, protected QGLFunctions {

public:
        HW(const QGLFormat &glf, QWidget *parent = 0);
        virtual QGroupBox*      controlPanel  ();                       // create control panel
        virtual void            reset                   ();                     // reset parameters
        void                            initShader      (int, QString, QString, UniformMap);

protected:
        QGLShaderProgram  m_program[MXPROGRAMS];                // GLSL programs
        GLint                       m_uniform[MXPROGRAMS][MXUNIFORMS];      // uniform vars for each
program
};
```

HW.h (3)

```
inline QMatrix3x3
setRotationMatrix(vec3 rotation)
{
        QMatrix3x3 R;
        R.setToIdentity();
        float alpha = qDegreesToRadians(rotation.x());
        float beta  = qDegreesToRadians(rotation.y());
        float gamma = qDegreesToRadians(rotation.z());

        R(0, 0) =  cos(beta)*cos(gamma);
        R(0, 1) = -cos(beta)*sin(gamma);
        R(0, 2) =  sin(beta);

        R(1, 0) =  sin(alpha)*sin(beta)*cos(gamma) + cos(alpha)*sin(gamma);
        R(1, 1) = -sin(alpha)*sin(beta)*sin(gamma) + cos(alpha)*cos(gamma);
        R(1, 2) = -sin(alpha)*cos(beta);

        R(2, 0) = -cos(alpha)*sin(beta)*cos(gamma) + sin(alpha)*sin(gamma);
        R(2, 1) =  cos(alpha)*sin(beta)*sin(gamma) + sin(alpha)*cos(gamma);
        R(2, 2) =  cos(alpha)*cos(beta);
        return R;
}
```

HW.h (4)

```
inline void
vectorRotate(QMatrix3x3 &rot, vec3 &v)
{
        vec3 t(v);
        v.setX(rot(0, 0)*t.x() + rot(0, 1)*t.y() + rot(0, 2)*t.z());
        v.setY(rot(1, 0)*t.x() + rot(1, 1)*t.y() + rot(1, 2)*t.z());
        v.setZ(rot(2, 0)*t.x() + rot(2, 1)*t.y() + rot(2, 2)*t.z());
}


inline vec3
rot_v(vec3 &v)
{
        v.normalize();
        float a = acos(vec3::dotProduct(v, vec3(1.0f, 0.0f, 0.0f)));
        float b = acos(vec3::dotProduct(v, vec3(0.0f, 1.0f, 0.0f)));
        float c = acos(vec3::dotProduct(v, vec3(0.0f, 0.0f, 1.0f)));
        vec3 r;
        r.setX(90-qRadiansToDegrees(a));
        r.setY(90-qRadiansToDegrees(c));
        r.setZ(qRadiansToDegrees(b));
        return r;

}
```

HW.h (5)

```
inline vec3
rot_v1tov2(vec3 &v2, vec3 &v1)
{
        v1.normalize();
        v2.normalize();
        float a = acos(vec3::dotProduct(v1, v2));
        vec3 axis = vec3::crossProduct(v1, v2);
        axis.normalize();
        QQuaternion q = QQuaternion::fromAxisAndAngle(axis, qRadiansToDegrees(a));
        vec3 r = q.toEulerAngles();
        vec3 rr;
        rr.setX(r.x());
        rr.setY(r.y());
        rr.setZ(-r.z());
        return rr;
}

inline vec2
cartesianToSpherical(vec3 &p)
{
        vec2 angles;
        double dist = sqrt(p.x()*p.x() + p.z()*p.z());
        angles.setX(atan2(p.z(), p.x()));
        angles.setY(atan2(dist, p.y()));
        return angles;
}
```

HW.h (6)

```
inline void
sphericalToCartesian(float theta, float phi, vec3 &eye)
{
        vec3 dir = -eye;
        float r = dir.length();

        float x = r * cos(theta) * sin(phi);
        float z = r * sin(theta) * sin(phi);
        float y = r * cos(phi);

        eye.setX(x);
        eye.setY(y);
        eye.setZ(z);
}

inline void
sphericalToCartesian(float theta, float phi, float r, vec3 &eye)
{

        float x = r * cos(theta) * sin(phi);
        float z = r * sin(theta) * sin(phi);
        float y = r * cos(phi);

        eye.setX(x);
        eye.setY(y);
        eye.setZ(z);
}
```

HW.h (7)

```
inline void
checkPhi(float &phi, float &up)
{
        // Keep phi within -2PI to +2PI for easy 'up' comparison
        if(phi > PI2)       phi -= (float) PI2;
        else if(phi < -PI2) phi += (float) PI2;

        if((phi > 0     && phi < M_PI) ||
           (phi < -M_PI && phi > -PI2))
                    up =  1.0f;
        else    up = -1.0f;
}

#endif // HW_H
```

HW.cpp (1)

```
// ============================================================
// Computer Graphics Homework Solutions
// Copyright (C) 2022 by George Wolberg
//
// HW.cpp - HW class. Base class of homework solutions.
//
// Written by: George Wolberg, 2022
// ============================================================

#ifdef SOLN
#include "../CS472.skel/HW.h"
#else
#include "HW.h"
#endif

QString GroupBoxStyle = "QGroupBox {                                \
                         border: 1px solid gray;                    \
                         border-radius: 9px;                        \
                         font-weight: bold;                         \
                         margin-top: 0.5em;}                        \
                         QGroupBox::title {                         \
                         subcontrol-origin: margin;                 \
                         left: 10px;                                \
                         padding: 0 3px 0 3px;                      \
                         }";
```

HW.cpp (2)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW::HW:
//
// HW constructor.
// This is base class for homework solutions that will replace
// the control panel, reset function, and add homework solution.
//
HW::HW(const QGLFormat &glf, QWidget *parent) : QGLWidget (glf, parent)
{}


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW::controlPanel:
//
// Create a control panel of widgets for homework solution.
//
QGroupBox*
HW::controlPanel()
{
        return NULL;
}


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW::reset:
//
// Reset parameters in control panel.
//
void
HW::reset() {}
```

HW.cpp (3)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW::initShader:
//
// Initialize vertex and fragment shaders.
//
void
HW::initShader(int shaderID, QString vshaderName, QString fshaderName, UniformMap uniforms)
{


        // due to bug in Qt, in order to use higher OpenGL version (>2.1), we need to add lines
        // up to initShader() to render properly
        uint vao;

        typedef void (APIENTRY *_glGenVertexArrays) (GLsizei, GLuint*);
        typedef void (APIENTRY *_glBindVertexArray) (GLuint);

        _glGenVertexArrays glGenVertexArrays;
        _glBindVertexArray glBindVertexArray;

        glGenVertexArrays = (_glGenVertexArrays) QGLWidget::context()->getProcAddress("glGenVertexArrays");
        glBindVertexArray = (_glBindVertexArray) QGLWidget::context()->getProcAddress("glBindVertexArray");

        glGenVertexArrays(1, &vao);
        glBindVertexArray(vao);
```

HW.cpp (4)

```
// compile vertex shader
bool flag = m_program[shaderID].addShaderFromSourceFile(QGLShader::Vertex, vshaderName);
if(!flag) {
        QMessageBox::critical(0, "Error", "Vertex shader error: " + vshaderName + "\n" +
                                          m_program[shaderID].log(), QMessageBox::Ok);
        exit(-1);
}

// compile fragment shader
if(!m_program[shaderID].addShaderFromSourceFile(QGLShader::Fragment, fshaderName)) {
        QMessageBox::critical(0, "Error", "Fragment shader error: " + fshaderName + "\n" +
                                          m_program[shaderID].log(), QMessageBox::Ok);
        exit(-1);
}

// bind the attribute variable in the glsl program with a generic vertex attribute index;
// values provided via ATTRIB_VERTEX will modify the value of "a_position")
glBindAttribLocation(m_program[shaderID].programId(), ATTRIB_VERTEX,    "a_Position");
glBindAttribLocation(m_program[shaderID].programId(), ATTRIB_COLOR,     "a_Color"   );
glBindAttribLocation(m_program[shaderID].programId(), ATTRIB_TEXCOORD,  "a_TexCoord");
glBindAttribLocation(m_program[shaderID].programId(), ATTRIB_NORMAL,    "a_Normal"  );

// link shader pipeline; attribute bindings go into effect at this point
if(!m_program[shaderID].link()) {
        QMessageBox::critical(0, "Error", "Could not link shader: " + vshaderName + "\n" +
                                          m_program[shaderID].log(), QMessageBox::Ok);
        qDebug() << m_program[shaderID].log();
        exit(-1);
}
```

HW.cpp (5)

```
        // iterate over all uniform variables; map each uniform name to shader location ID
        for(std::map<QString, GLuint>::iterator iter = uniforms.begin(); iter != uniforms.end(); ++iter) {
                QString uniformName = iter->first;
                GLuint  uniformID   = iter->second;

                // get storage location
                m_uniform[shaderID][uniformID]=glGetUniformLocation(m_program[shaderID].programId(),
                        uniformName.toStdString().c_str());
                if(m_uniform[shaderID][uniformID] < 0) {
                        qDebug() << m_program[shaderID].log();
                        qDebug() << "Failed to get the storage location of " + uniformName;
                }
        }
}
```

HW0a:

White square on black background using Legacy OpenGL (pre 3.1)

Aspect ratio is not preserved

HW0a.h

```
#include "HW.h"

// ------------------------------------------------------------------
// standard include files
//

class HW0a : public HW {
	Q_OBJECT
public:
	HW0a		(const QGLFormat &glf, QWidget *parent = 0);	// constructor
	QGroupBox*	controlPanel();			// create control panel

protected:
	void		initializeGL ();		// init GL state
	void		resizeGL	(int, int);		// resize GL widget
	void		paintGL		();			// render GL scene
}
```

HW0a.cpp (1)

```
#include "HW0a.h"

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW0a::HW0a:
//
// HW0a constructor.
//
HW0a::HW0a(const QGLFormat &glf, QWidget *parent)
        : HW (glf, parent) {}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW0a::initializeGL:
//
// Initialization routine before display loop.
// Gets called once before the first time resizeGL() or paintGL() is called.
//
void
HW0a::initializeGL()
{
        // initialize GL function resolution for current context
        initializeGLFunctions();

        // init state variables
        glClearColor(0.0, 0.0, 0.0, 0.0);       // set background color
        glColor3f   (1.0, 1.0, 1.0);            // set foreground color
}
```

HW0a.cpp (2)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Resize event handler. The input parameters are the window width (w) and height (h).
//
void HW0a::resizeGL(int w, int h)
{
	// set viewport to occupy full canvas
	glViewport(0, 0, w, h);

	// init viewing coordinates for orthographic projection
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	glOrtho(0.0, 1.0, 0.0, 1.0, -1.0, 1.0);
}

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Update GL scene.
//
void HW0a::paintGL()
{
	// clear canvas with background values
	glClear(GL_COLOR_BUFFER_BIT);

	// define polygon
	glBegin(GL_POLYGON);
		glVertex2f(0.25, 0.25);
		glVertex2f(0.75, 0.25);
		glVertex2f(0.75, 0.75);
		glVertex2f(0.25, 0.75);
	glEnd();
}
```

HW0a.cpp (3)

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW0a::controlPanel:
//
// Create control panel groupbox.
//
QGroupBox*
HW0a::controlPanel()
{
        // init group box
        QGroupBox *groupBox = new QGroupBox("Homework 0a");

        return(groupBox);
}
```

HW0b:
Rotating trapezoid using Legacy OpenGL (pre 3.1)

Aspect ratio is preserved

HW0b.h

```
#include "HW.h"

// ------------------------------------------------------------------------
// standard include files
//

class HW0b : public HW {
      Q_OBJECT
public:
      HW0b          (QGLFormat &glf, QWidget *parent = 0);   // constructor
      QGroupBox*  controlPanel();             // create control panel
      void              reset          ();              // reset parameters

public slots:
      void              flipY          (int);          // flip y-coordinate
      void              aspect         (int);          // maintain aspect ratio
      void              rotate         (int);          // rotate data

protected:
      void              initializeGL   ();              // init GL state
      void              resizeGL       (int, int);     // resize GL widget
      void              paintGL        ();              // render GL scene

private:
      int               m_winW;                        // window width
      int               m_winH;                        // window height
      double            m_angle;                       // rotation angle
      QCheckBox  *m_checkBoxFlip;          // checkbox to flip y-coordinate
      QCheckBox  *m_checkBoxAR;            // checkbox to maintain aspect ratio
      QSlider        *m_slider;                        // rotation slider
      QSpinBox     *m_spinBox;                        // rotation spinbox
};
```

HW0b.cpp (1)

```
#include "HW0b.h"

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW0b::HW0b:
//
// HW0b constructor.
//
HW0b::HW0b(const QGLFormat &glf, QWidget *parent)
        : HW (glf, parent), m_angle(0.0f)
{}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HW0b::initializeGL:
//
// Initialization routine before display loop.
// Gets called once before the first time resizeGL() or paintGL() is called.
//
void
HW0b::initializeGL()
{
        // init state variables
        glClearColor(0.0, 0.0, 0.0, 0.0);       // set background color
        glColor3f   (1.0, 1.0, 1.0);            // set foreground color
}
```

HW0b.cpp (2)

```
void
HW0b::resizeGL(int w, int h)
{
	// save window dimensions
	m_winW = w;
	m_winH = h;

	// compute aspect ratio
	float xmax, ymax;
	if(m_checkBoxAR->isChecked()) {
		float ar = (float) w / h;
		if(ar > 1.0) {		// wide screen
			xmax = ar;
			ymax = 1.;
		} else {		// tall screen
			xmax = 1.;
			ymax = 1/ar;
		}
	} else {
		xmax = 1.0;
		ymax = 1.0;
	}

	// set viewport to occupy full canvas
	glViewport(0, 0, w, h);

	// init viewing coordinates for orthographic projection
	glLoadIdentity();
	if(m_checkBoxFlip->isChecked())
		glOrtho(-xmax, xmax,  ymax, -ymax, -1.0, 1.0);
	else   glOrtho(-xmax, xmax, -ymax,  ymax, -1.0, 1.0);
}
```

HW0b.cpp (3)

```
void
HW0b::paintGL()
{
	// initial data
	QVector2D v[] = {
		QVector2D(-.25, -.25),
		QVector2D( .25, -.25),
		QVector2D( .15,  .25),
		QVector2D(-.15,  .25)
	};

	// clear canvas with background values
	glClear(GL_COLOR_BUFFER_BIT);

	// init cosine and sine variables
	double c = cos(m_angle);
	double s = sin(m_angle);

	// define polygon
	glBegin(GL_POLYGON);
		for(int i=0; i<4; i++) {
			glVertex2f(c*v[i].x() - s*v[i].y(), s*v[i].x() + c*v[i].y());
		}
	glEnd();
	glFlush();
}
```

HW0b.cpp (4)

```
QGroupBox*
HW0b::controlPanel()
{
	// init group box
	QGroupBox *groupBox = new QGroupBox("Homework 0b");
	groupBox->setMinimumWidth(300);

	// layout for assembling widgets
	QGridLayout *layout = new QGridLayout;

	// create checkboxes
	m_checkBoxFlip = new QCheckBox("Flip y-coordinates");
	m_checkBoxAR   = new QCheckBox("Maintain aspect ratio");
	m_checkBoxFlip->setChecked(false);
	m_checkBoxAR  ->setChecked(true );

	// create slider to rotate data
	m_slider = new QSlider(Qt::Horizontal);
	m_slider->setRange(0, 360);
	m_slider->setValue(0);

	// create spinBox
	m_spinBox = new QSpinBox;
	m_spinBox->setRange(0, 360);
	m_spinBox->setValue(0);

	// slider label to display name
	QLabel *label = new QLabel("Rotation");
```

HW0b.cpp (5)

```
    // assemble widgets into layout
    layout->addWidget(m_checkBoxFlip,  0, 0, 1, 3);
    layout->addWidget(m_checkBoxAR  , 1, 0, 1, 3);
    layout->addWidget(label,                  2, 0);
    layout->addWidget(m_slider,               2, 1);
    layout->addWidget(m_spinBox,              2, 2);

    // assign layout to group box
    groupBox->setLayout(layout);

    // init signal/slot connections
    connect(m_checkBoxFlip, SIGNAL(stateChanged(int)), this, SLOT(flipY(int)));
    connect(m_checkBoxAR , SIGNAL(stateChanged(int)), this, SLOT(aspect(int)));
    connect(m_slider,             SIGNAL(valueChanged(int)), this, SLOT(rotate(int)));
    connect(m_spinBox,          SIGNAL(valueChanged(int)), this, SLOT(rotate(int)));

    return(groupBox);
}
```

HW0b.cpp (6)

```
// Slot function to flip y-coordinates.
void
HW0b::flipY(int state)
{
	// update checkbox
	m_checkBoxFlip->setChecked(state);

	// call resizeGL() to reset coordinate system
	resizeGL(m_winW, m_winH);

	// redraw
	updateGL();
}

// Slot function to maintain aspect ratio.
void
HW0b::aspect(int state)
{
	// update checkbox
	m_checkBoxAR->setChecked(state);

	// call resizeGL() to reset coordinate system
	resizeGL(m_winW, m_winH);

	// redraw
	updateGL();
}
```

HW0b.cpp (7)

```
// Slot function to rotate data.
void HW0b::rotate(int angle)
{
	// update slider and spinbox
	m_slider->blockSignals(true);
	m_slider->setValue(angle);
	m_slider->blockSignals(false);

	m_spinBox->blockSignals(true);
	m_spinBox->setValue(angle);
	m_spinBox->blockSignals(false);

	// convert angle to radians
	m_angle = angle * (M_PI / 180.);

	// redraw
	updateGL();
}

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Reset parameters.
void HW0b::reset()
{
	// reset checkboxes
	m_checkBoxFlip->setChecked(false);
	m_checkBoxAR  ->setChecked(true );
	resizeGL(m_winW, m_winH);

	// reset angle and slider/spinbox settings
	m_angle = 0;
	m_slider    ->setValue(m_angle);
	m_spinBox->setValue(m_angle);
}
```