IMAGE PROCESSING

Point & Neighborhood Operations

By: Khadeeja Din

CSC - Senior Design Capstone 1

(Spring 2016)

Professor George Wolberg

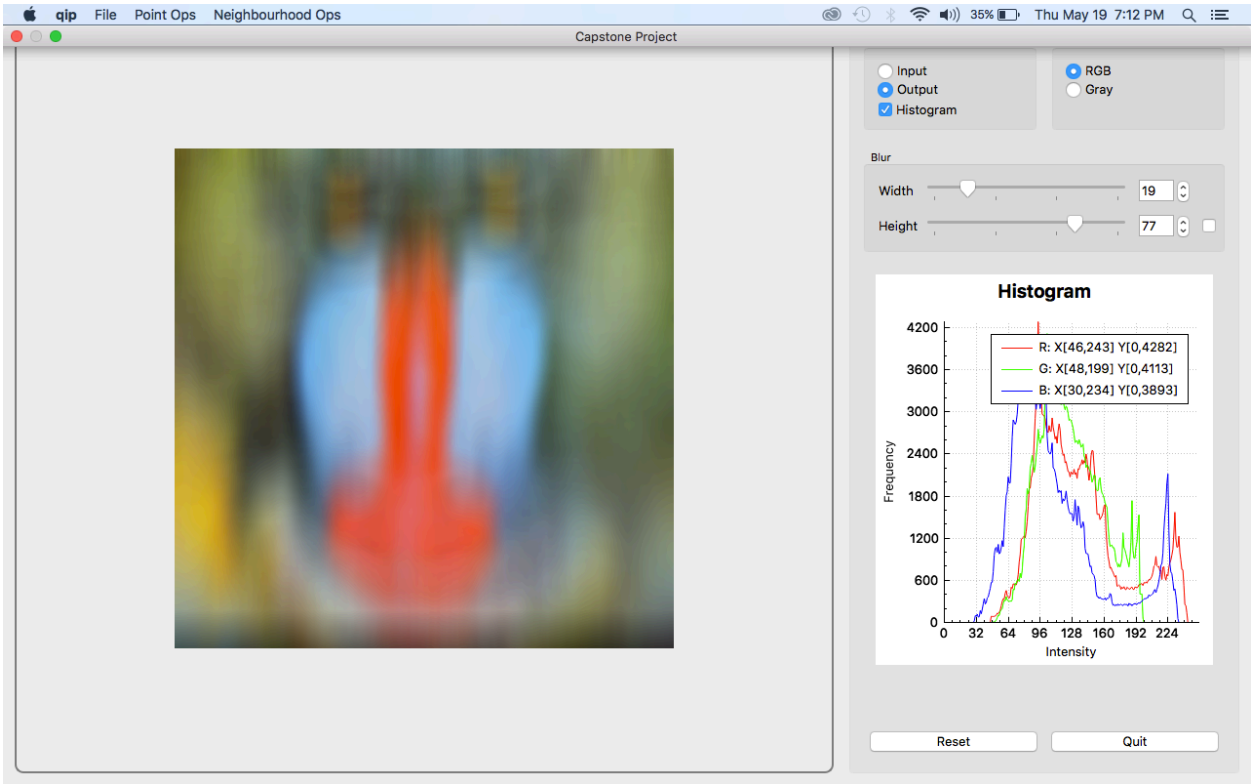
City College of New York

Table of Contents

Introduction

Project Overview

This is an image-processing project I implemented for my Senior Design I class at the City College of New York. Digital image processing is computer manipulation of pictures or images that have been converted into numeric form. The finished application lets you perform various image filters. Below is a sample window showing blur filter applied on a monkey face image.



The project is implemented in C++ and QT libraries are used for the Graphic User Interface. QT is a cross-platform application and UI framework for developers using C++ or QML, a CSS & JavaScript like language. The Qt API is used to interact with a graphics-processing unit (GPU) to achieve hardware accelerated rendering.

Program Design & Display

Main.cpp

```
10  #include "MainWindow.h"
11
12  int main(int argc, char **argv)
13  {
14    QApplication app(argc, argv);   // create application
15    MainWindow window;              // create UI window
16    window.showMaximized();         // display window
17    return app.exec();              // infinite processing loop
18  }
```

The main.cpp is the driver program that initiates and run all other programs. The main function simply creates the QApplication by calling QApplication app with the given arguments. It then creates the UI window and displays it. After setting up the window it starts running in an infinite loop until the program is quitted.

MainWindow.h

MainWindow.h is the header file for MainWindow class. All the Image Processing headers and QT libraries are called in this file to set up the skeleton for MainWindow class. The MainWindow inherits from QMainWindow class. Q_OBJECT is a macro that is required for classes that implement signals and slots. This has to be included because C++ does not know the function of signals and slots. In our program we need signals and slot because they connect our widgets to specific functions in our program. According to the authors of C++ GUI Programming with Qt 4, “In Qt and Unix terminology, a widget is a visual element in a user interface. Buttons, menus, scroll bars, and frames are all examples of widgets.” (3) Widgets allow users to interact with the interface. They give us the option to select, open and enter values. Clicking on the widget does each of these actions and generates an event. A signal is raised every time an event is generated, and when a signal occurs, the associated slot function is executed.

Below is a snippet of MainWindow.h which shows the creation of a MainWindow class that inherits from QMainWindow and contains the macro Q_OBJECT. It also shows some of the public and protected slot functions.

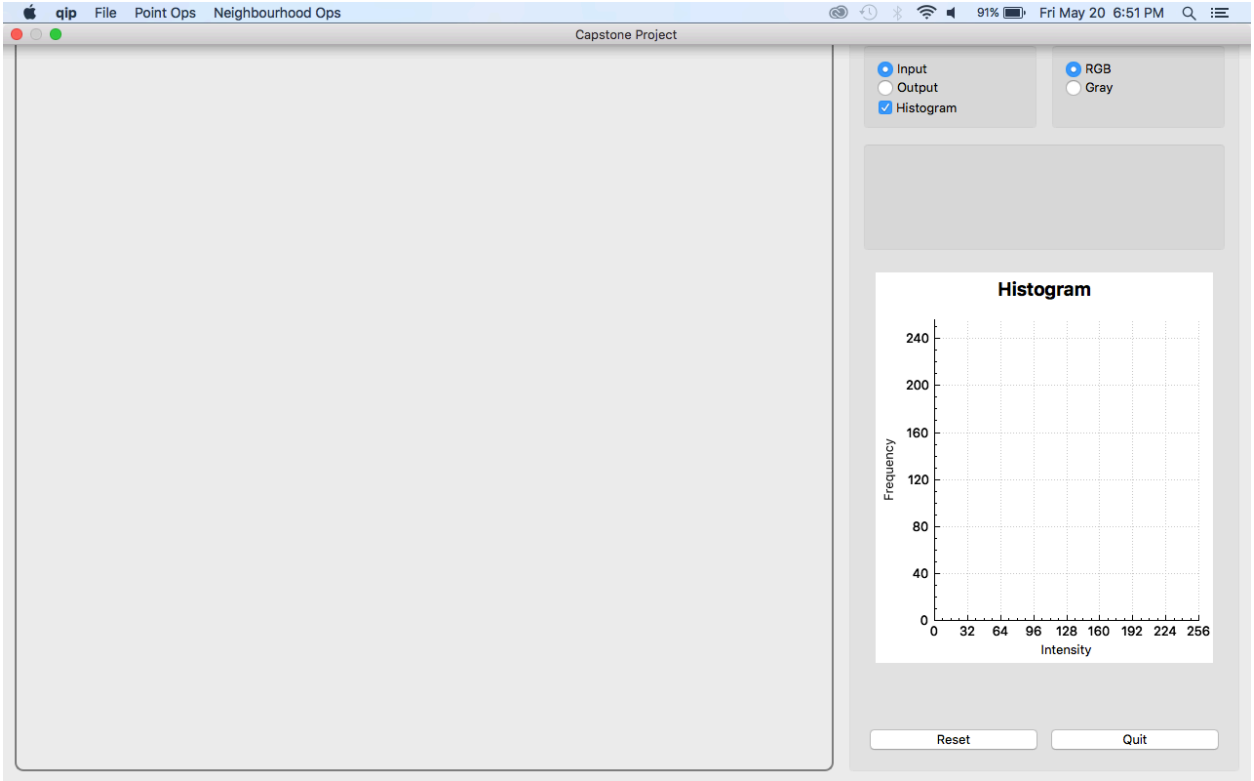
```
38  class MainWindow : public QMainWindow {
39    Q_OBJECT
40
41  public:
42    // constructor
43    MainWindow  (QWidget *parent = 0);
44    ImagePtr  imageSrc  () const;
45    ImagePtr  imageDst  () const;
46    QCustomPlot*  histogram() {return m_histogram;}
47
48  public slots:
49    void    open    ();
50    void    displayIn ();
51    void    displayOut  ();
52    void    modeRGB   ();
53    void    modeGray  ();
54    void    preview   ();
55    void    reset   ();
56    void    quit    ();
57    void    execute   (QAction*);
58
59  protected slots:
60    void    setHisto  (int);
61
```

MainWindow.cpp

MainWindow.cpp file creates all the actions, Menus, and Widgets. It defines all the functions that were declared in the MainWindow.h file. MainWindow determines which action was triggered by the user and executes the respective action, which then updates the interface. Below is the code for the MainWindow constructor.

```
38  MainWindow::MainWindow(QWidget *parent)
39    : QMainWindow(parent),
40      m_code(-1),
41      m_histoColor(0)
42  {
43    setWindowTitle("Capstone Project");
44
45    // set global variable for main window pointer
46    g_mainWindowP = this;
47
48    // INSERT YOUR ACTIONS AND MENUS
49    createActions();
50    createMenus  ();
51    createWidgets();
52  }
```

Main Window Layout

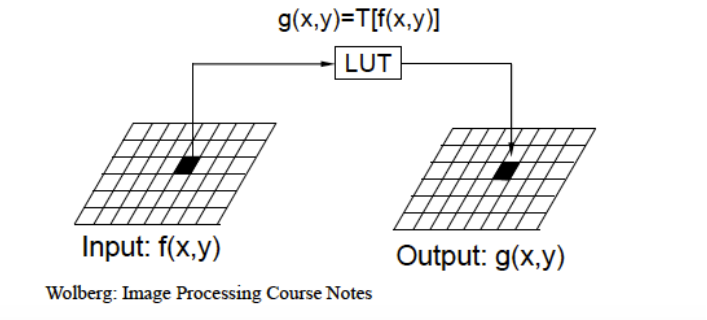


The rest of the image processing operations has their own .h and .cpp files where their classes, respective functions and layouts are defined. The right part of the above window is a control panel that changes for each different image filter. The top left groupbox contains 2 radio buttons and 1 checkbox. The 2 radio buttons let you select between displaying input or output image. The checkbox is for displaying the histogram. In the above picture the checkbox is checked and so the histogram graph is displayed. The histogram shows the frequency of 0-255 pixel values in the image. The top right groupbox contains two radio buttons, which let you choose between displaying a grey scale or color (RGB) image. The middle groupbox that is empty in the above picture is a container for the image filters. In the following pages the respective widgets for each filter will be displayed in this groupbox. The left part in the above window is a big QLabel where

the input and output image is displayed. QLabel is a Qt widdget that displays text and images. The left part remains consistent for all of the different image filters.

Point Operations

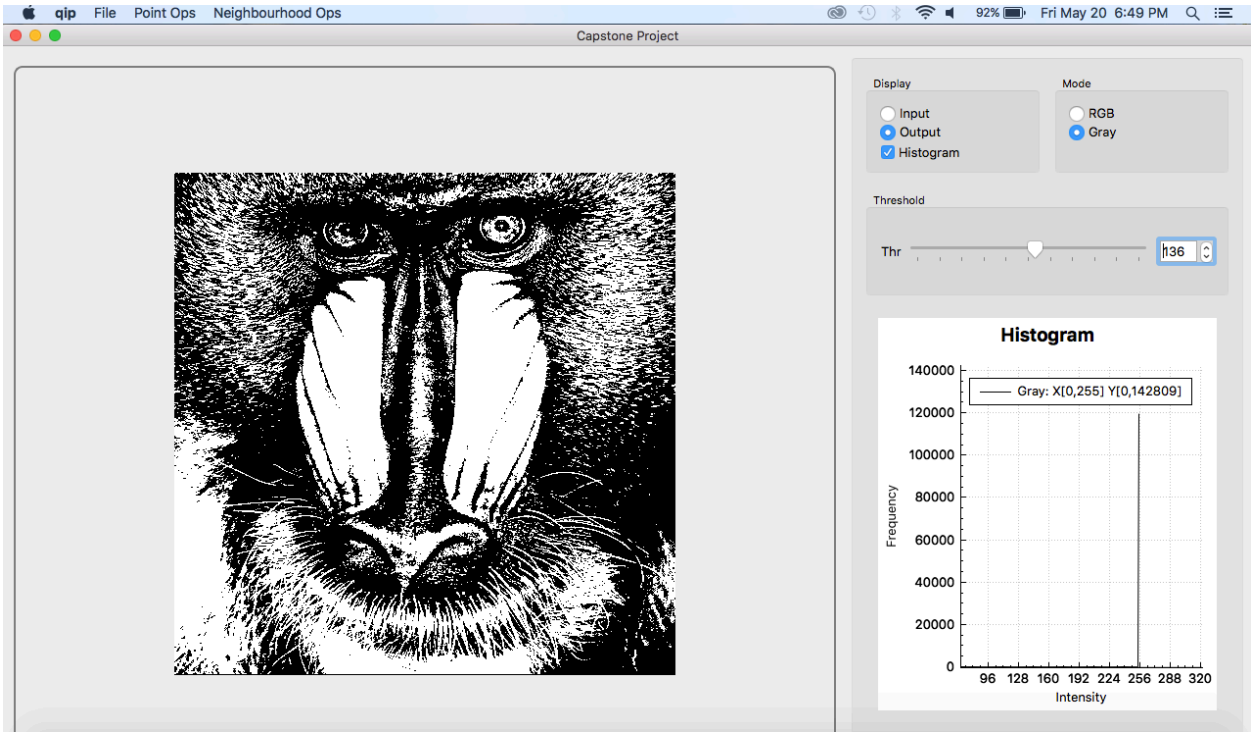
In point operations output pixels are a function of only one input pixels. Transformation T is applied to each pixel in the input image and the result is stored in the corresponding output pixel of output image. The transformations are implemented with a lookup table. The lookup table (LUT) have 256 entries for the 256 pixel values from 0 – 255. In the code 256 is defined as the variable MXGRAY and 255 is defined as the variable MaxGray. All input values index into the lookup table and the data stored there is copied to the corresponding output position. In my program I implemented some point operations that are commonly used in image processing: Threshold, Quantization, Contrast & Brightness, Histogram Stretching, and Histogram equalization and matching.



Threshold

Threshold provides a way to segment an image on the basis of different intensities or colors in the foreground and background regions of an image. This segmentation is determined by a single

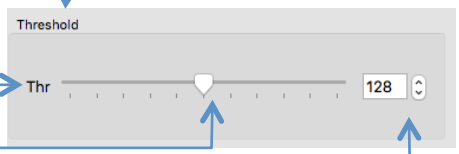
parameter known as the intensity threshold. Each pixel in the image is compared with this threshold value. The pixel values below the threshold value are replaced with 0 (black), and the pixel values above the threshold value are replaced with 255 (white). Threshold can be applied both to color and grey scale images. Below is an example of threshold applied to a grey scale image.



The threshold value can be selected by a slider or a spinbox. In the above window the threshold value is 136. Therefore, 255 replace all the pixel values above 136, which is why we also see a big spike at 255 in the histogram.

The code below creates the groupbox for Threshold filter. We start by naming the new groupbox, "Threshold". A *QLabel*, *QSlider*, and *QSpinBox* is created. The slider and spinbox are then connected to their corresponding slot functions. Each time the slider or spinbox is changed, the slot functions are called which applies the new filter value and displays the new output image. After the 3 widgets are created, they are added to the layout, and the *setLayout* function is called.

```
QGroupBox*
Threshold::controlPanel()
{
  // init group box
  m_ctrlGrp = new QGroupBox("Threshold");

  // init widgets
  // create label[i]
  QLabel *label = new QLabel;
  label->setText(QString("Thr"));

  // create slider
  m_slider = new QSlider(Qt::Horizontal, m_ctrlGrp);
  m_slider->setTickPosition(QSlider::TicksBelow);
  m_slider->setTickInterval(25);
  m_slider->setMinimum(1);
  m_slider->setMaximum(MXGRAY);
  m_slider->setValue  (MXGRAY>>1);

  // create spinbox
  m_spinBox = new QSpinBox(m_ctrlGrp);
  m_spinBox->setMinimum(1);
  m_spinBox->setMaximum(MXGRAY);
  m_spinBox->setValue  (MXGRAY>>1);

  // init signal/slot connections for Threshold
  connect(m_slider , SIGNAL(valueChanged(int)), this, SLOT(changeThr (int)));
  connect(m_spinBox, SIGNAL(valueChanged(int)), this, SLOT(changeThr (int)));

  // assemble dialog
  QGridLayout *layout = new QGridLayout;
  layout->addWidget(  label  , 0, 0);
  layout->addWidget(m_slider , 0, 1);
  layout->addWidget(m_spinBox, 0, 2);

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

  return(m_ctrlGrp);
}
```



Slider and SpinBox are connected to their appropriate slot functions.

Valuechanged(int) checks if the slider or spinbox value is changed. It returns 1 if value is changed, else it returns 0.

SLOT(changeThr (int)) calls the slot function, *changeThr*, with the new threshold value.

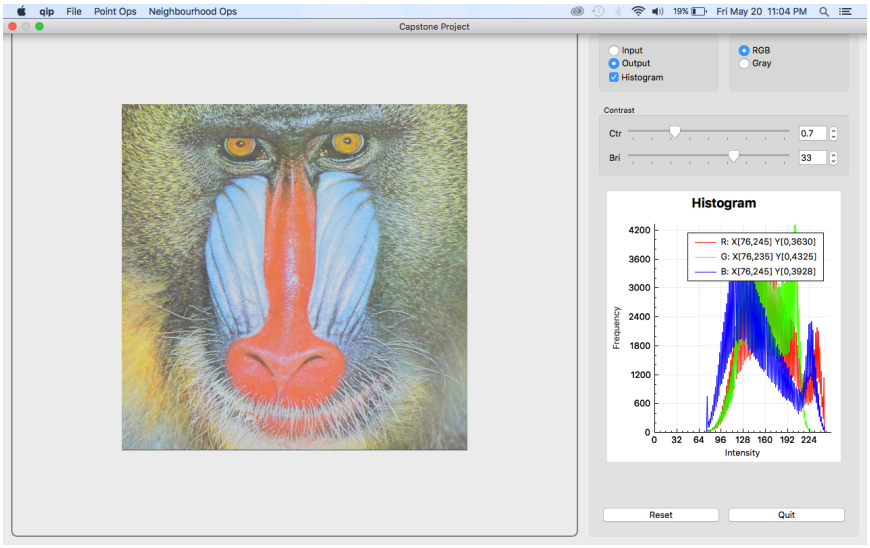
SIGNAL applied the slot function each time value changed returns true.

After the layout is created, and the threshold value is updated, the new filter is applied to all the pixels in the input image. In the code snippet below the threshold function compares each pixel with the threshold value read from the slider or spinbox, and then replaces each pixel to either 0 or 255 accordingly. The threshold function takes 3 parameters: *ImagePtr I1*, *int thr*, *ImagePtr I2*. *ImagePtr I1* is the pointer to input image, *ImagePtr I2* is the pointer to output image, and *int thr* is the integer threshold value read from slider or spinbox. The threshold function copies the image header of input image to output image, so that an output image of same width and height is created. It next initializes a lookup table (*LUT*) with 256 entries with indexes 0 – 255. In each index either 0 or 255 is saved after comparing the index value with *thr*. For each pixel in *I1*, their corresponding value is read from *LUT* and stored in *I2*.

```
void
Threshold::threshold(ImagePtr I1, int thr, ImagePtr I2) {
  IP_copyImageHeader(I1, I2);
  int w = I1->width();
  int h = I1->height();
  int total = w * h;

  // compute lut[]
  int i, lut[MXGRAY];
  for(i=0; i<thr && i<MXGRAY; ++i) lut[i] = 0;
  for(   ; i <= MaxGray;      ++i) lut[i] = MaxGray;

  int type;
  ChannelPtr<uchar> p1, p2, endd;
  for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
    IP_getChannel(I2, ch, p2, type);
    for(endd = p1 + total; p1<endd;) *p2++ = lut[*p1++];
  }
}
```

Contrast & Brightness

Brightness refers to the overall lightness or darkness of an image, whereas contrast is the difference in brightness between different objects or regions in an image.



In my implementation contrast and brightness range from -100 – 100. However, contrast is shown as a factor between 0 – 5.0. There are two separate sliders and spinboxes: one for each contrast and brightness. Since contrast spinbox is reading decimal values we use *QDoubleSpinBox*.

```
//create contrast spinbox
m_spinBoxC = new QDoubleSpinBox(m_ctrlGrp);
m_spinBoxC->setMinimum(cmin); //cmin is 0.0
m_spinBoxC->setMaximum(cmax); //cmax is 5.0
m_spinBoxC->setDecimals(1);
m_spinBoxC->setValue (2.5);
```

Since we have 2 separate sliders we need 2 separate slot functions for each brightness and contrast.

```
connect(m_sliderC , SIGNAL(valueChanged(int)), this, SLOT(changeCtr (int)));
connect(m_spinBoxC, SIGNAL(valueChanged(double)), this, SLOT(changeCtr (int)));

connect(m_sliderB , SIGNAL(valueChanged(int)), this, SLOT(changeBri (int)));
connect(m_spinBoxB, SIGNAL(valueChanged(int)), this, SLOT(changeBri (int)));
```

If the contrast value is greater than or equal to 0 we divide it by 25 and add 1. Else if the contrast value is less than 0 we divide it by 133 and add 1. Below is the code for *changeCtr* and *changeBri* slot functions.

```
// contrast:
void Contrast::changeCtr(int c)
{
  double contr;

  if (c >= 0)
    contr = c/25.0 + 1.0;
  else
    contr = c/133.0 + 1.0;

  m_sliderC ->blockSignals(true);
  m_sliderC ->setValue    (c   );
  m_sliderC ->blockSignals(false);
  m_spinBoxC->blockSignals(true);
  m_spinBoxC->setValue    (contr   );
  m_spinBoxC->blockSignals(false);

// apply filter to source image; save result in destination Image
applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

// display Output
g_mainWindowP->displayOut();

}

void
Contrast::changeBri(int b)
{
  m_sliderB ->blockSignals(true);
  m_sliderB ->setValue    (b );
  m_sliderB ->blockSignals(false);
  m_spinBoxB->blockSignals(true);
  m_spinBoxB->setValue    (b );
  m_spinBoxB->blockSignals(false);

  // apply filter to source image; save result in destination image
  applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

  // display output
  g_mainWindowP->displayOut();
}
```

To apply contrast we subtract our reference value, i.e. 128, from each pixel then multiply it by the contrast factor. This subtraction of reference point darkens all pixel levels below the reference point and brightens all pixel levels above the reference point. We then add reference and brightness to this value, and clip it between 0 – 255 before copying it to the output image. The snippet below shows the *contrast* function that takes as arguments input *ImagePtr I1*, *double brightness*, *double contrast*, and output *ImagePtr I2*. The transformed values are stored in the *LUT*, and lastly these values will be copied to *I2*.

```
void
Contrast::contrast(ImagePtr I1, double brightness, double contrast, ImagePtr I2)
{
IP_copyImageHeader(I1, I2);
int w = I1->width();
int h = I1->height();
int total = w * h;
double contr;
int reference = 128;

if (contrast >= 0)
  contr = contrast/25.0 + 1.0;
else
  contr = contrast/133.0 + 1.0;

//compute lut[]
//initialized lut of 256 entries
int i, lut[MXGRAY]; //lut an array of 256 integers
for(i=0; i<MXGRAY; ++i)
  lut[i] = (int)CLIP((i - reference)* contr + reference + brightness, 0, 255);
```

Quantization

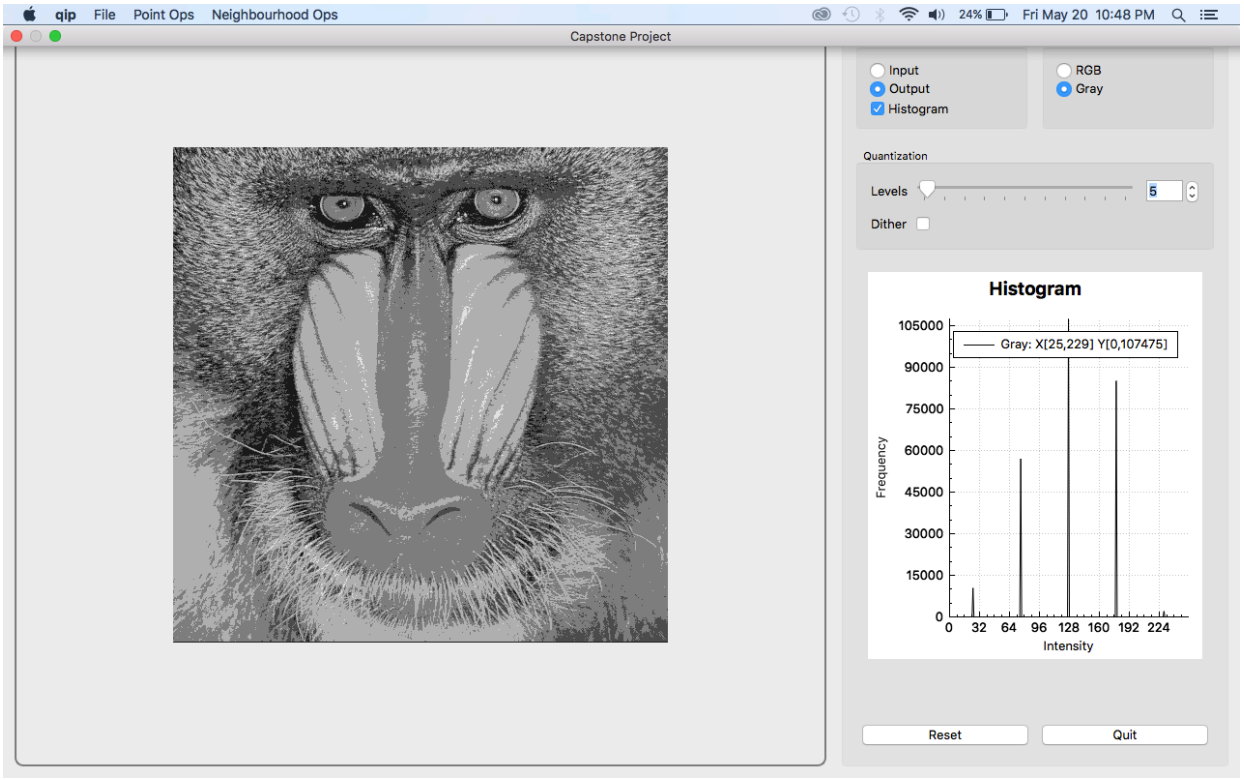
Quantization is a lossy compression technique where a range of values is compressed to a single quantum value. Color quantization reduces the number of colors used in an image. To quantize an image, we add a bias value to each pixel. The bias value is determined by the number of quantization levels. Users can select quantization levels between 0 – 255. The code below adds a

bias to quantize each pixel value and stores it in the LUT. The values from LUT are then copied to the output image.

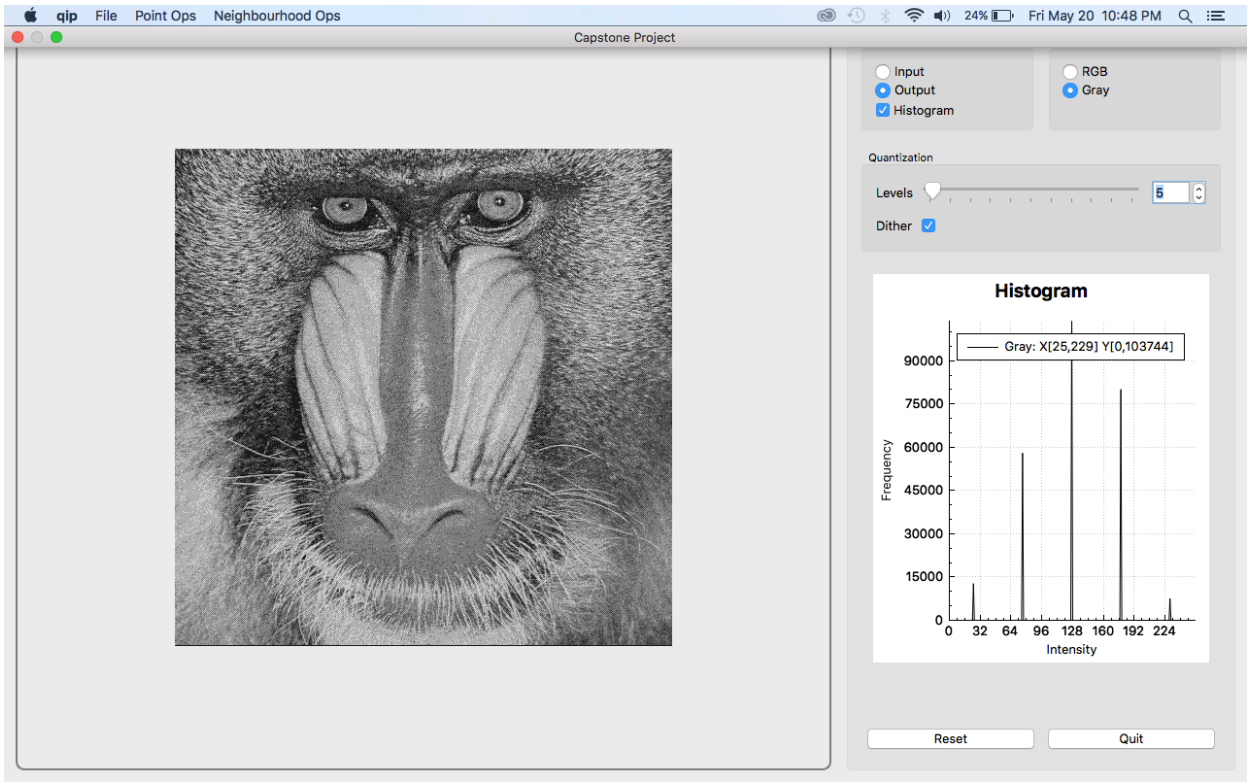
```
// compute lut[]
int i, lut[MXGRAY];
int levels = quan;                  //quantization levels
int scale = (MXGRAY + 1) / levels;
double bias = scale/2.0;            //bias brings the scale down by a factor of 2

for(i=0;  i<=MXGRAY; ++i)
    lut[i] = scale * (int) (i/scale) + bias;
```

We can further reduce the quantization error by adding a uniformly distributed white noise (dither signal) to the input image prior to quantization. Users have an option to add dither or not. We add negative noise on odd rows and positive noise on even rows. After the noise is added the new value is clipped between the range 0 – 255, by calling the function *CLIP(pixel, 0, 255)*. The clipped value is then copied to output image. The code below gets called if dither checkbox is checked.

```
//if dither is checked, apply dither to each pixel value
{for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
  IP_getChannel(I2, ch, p2, type);
  for (int y=0; y<h; y++)
  {
     //if row is odd, intiialize s to 1
     if (y % 2)
     //sign is the + or - sign of dither noise
     sign = 1;
     else
      //if row is even, intiialize s to -1
      sign = -1;
        for (int x=0; x<w; x++)
        {
          //2^5 = 32767  // j gives a number b/w 0-1
          noise = ((rand()&0x7fff) / 32767.) * bias;
          //alternating the noise addition or subtraction
          switch(sign)
          {
            //on odd row adding negative value
            case 1:
                    //adding noise to pixel
                    pixel = *p1++ + noise;
                    sign = -1;
                    break;
            //on even row adding positive value
            case -1:
                    //subtracting noise form pixel
                    pixel = *p1++ - noise;
                    sign = 1;
                    break;
          }
          //purpose of clipping is to make sure it does not goes off range
          //clipping the pixel value after applying the dither and copying to output image
          *p2++ = lut[ CLIP(pixel, 0, MaxGray)];
          }
        }
```



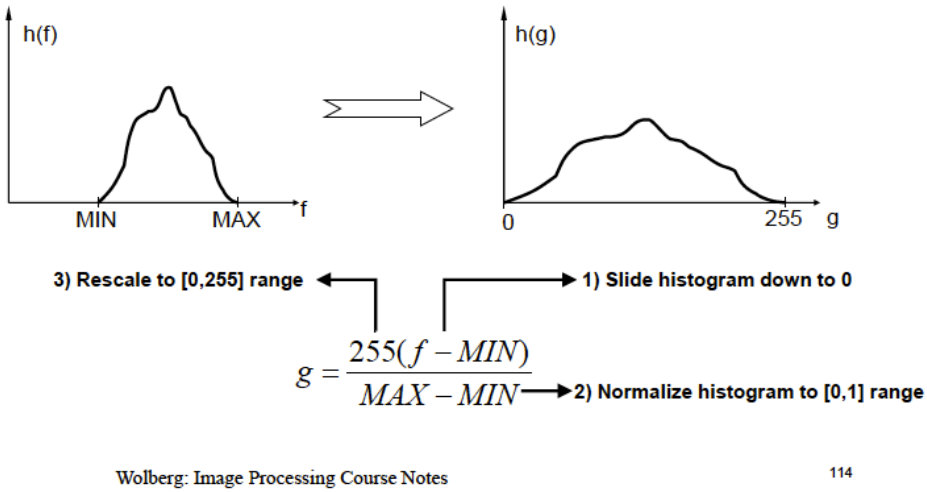
Quantization without dither



Quantization with dither

Histogram Stretching

Histogram Stretching improves the contrast in an image by stretching the range of intensity values in the image to span a range of values. In our case the range of values will be the full range of pixel values in 8 bit images, i.e. 0 – 255.



The above image shows how the range of intensity values is stretched to 0 – 255. *g* is the function that applies this stretch to an image. *f* is the current pixel value being read, and function *g* is applied to each pixel value. *MIN* is subtracted from each pixel to slide down the histogram to 0. This does not change the shape of histogram. The difference is then divided by *MAX – MIN* to normalize the histogram to [0, 1] range. Lastly the quotient is multiplied by 255 to rescale it to [0, 255] range.

In our implementation we even let users pick a min or max value of their own choice. For this reason you will see two min and max sliders in the histogram-stretching interface. We will also have 2 checkboxes one for each min and max. If the min checkbox is checked we will ignore the slider value of min, and instead read the minimum pixel intensity from the input image. Likewise if the max checkbox is checked we will ignore the slider value of max, and instead read the

maximum pixel intensity from the input image. The two checkboxes does not have to be checked or unchecked together. The min and max values are selected independent of each other. If the checkboxes are unchecked we will read the min and max values from their respective sliders. The min checkbox value is stored in *minautovalue* and max checkbox value is stored in *maxautovalue*. The min slider value is stored in *minstretch* and
max slider value is stored in *maxstretch*.

```
// apply filter
int minautovalue = m_checkBoxMin->isChecked();
int maxautovalue = m_checkBoxMax->isChecked();

int minstretch, maxstretch;
minstretch = m_sliderMin->value();
maxstretch = m_sliderMax->value();
```

We then first check if the checkboxes are checked. If the checkboxes are checked we change *minstretch* and *maxstretch* to the min and max pixel intensity values in the input image. To read the minimum value we first store the input image pixels in a Histogram.

```
int i = 0;
for (i = 0; i<MXGRAY; i++)
 {Histogram[i] = 0;}

 int type;
 ChannelPtr<uchar> p1, p2, endd; //p1 is a pointer that points to pixel. p1++ is pointing to next pixel
 for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
  for(endd = p1 + total; p1<endd; p1++)
  Histogram[*p1] ++;
 }
```

To read the minimum intensity value we start reading values from the left end of histogram and stop once we read a non-zero intensity value. That is the minimum value of the image and we store it in the variable minimum. To read the maximum intensity value we do the opposite; we start reading values from the right end of histogram. Whichever index have non-zero frequency we store that value in the variable maximum. The code for this is shown below.

```
if (maxautovalue)
{
  int maximum = MaxGray;
  int i = 0;
  for (i=MaxGray; i>= 0; i--)
  { if (!Histogram[i]) continue;
      maximum = i;  //copy max to slider and spinbox
      break; }
  maxstretch = maximum;
  m_sliderMax->setValue  (maximum);
  m_spinBoxMax->setValue (maximum);
}
```

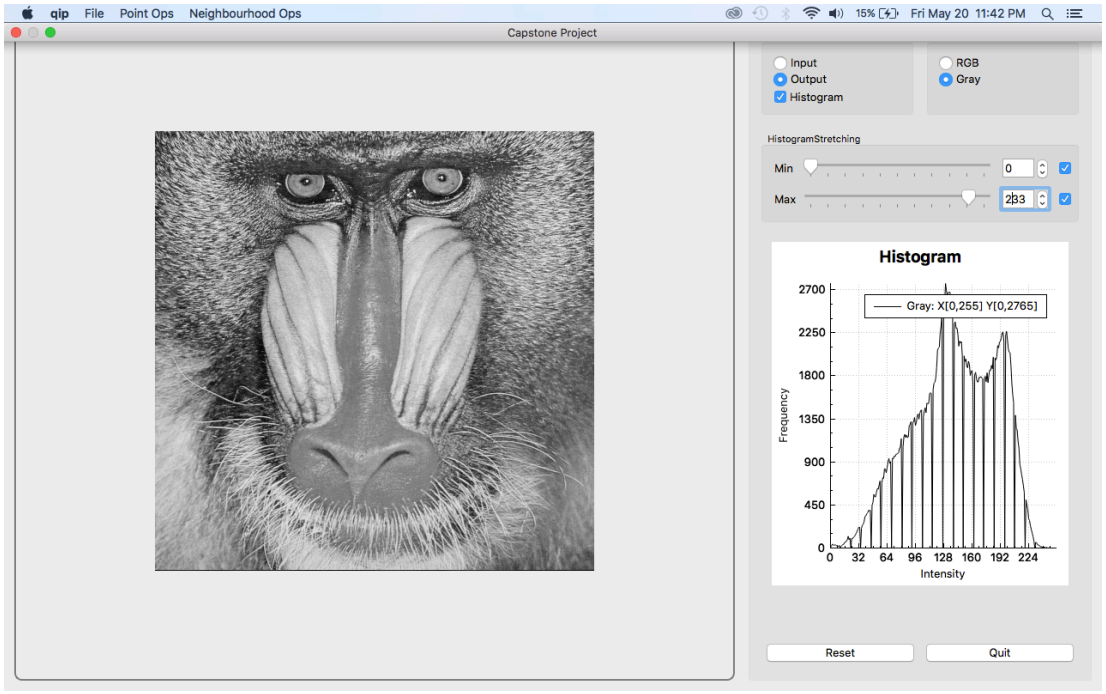
```
if (minautovalue)
{
  int minimum = min, i =0;
  for (i=0; i<MXGRAY; i++)
  { if (!Histogram[i]) continue;
      minimum = i;
      break; }
  minstretch = minimum;
  m_sliderMin->setValue (minimum);
  m_spinBoxMin->setValue (minimum);
}
```

Now *minstretch* and *maxstretch* contain the correct values, which can be either from slider or from the image. If the values were read from slider we need to make sure that *minstretch* is at least 1 less than *maxstretch*. We also need to confirm that the values are between 0 – 255. Lastly we send these values and *I1* and *I2* to *histogramstretching* function.

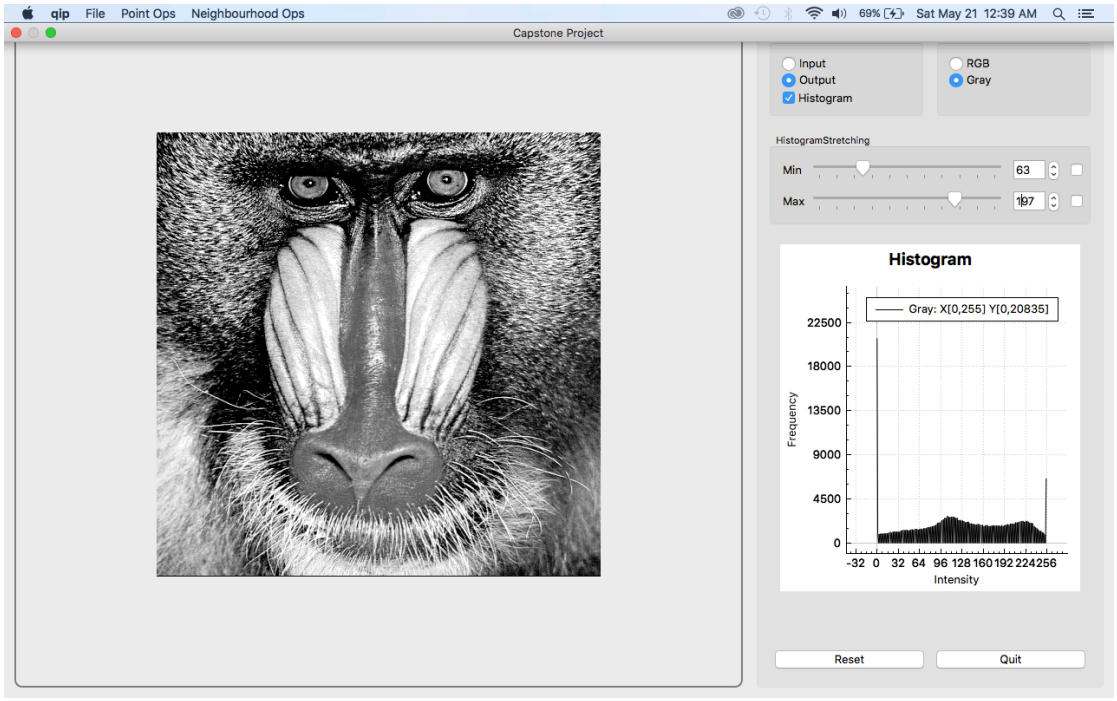
```
if ((minstretch < min || minstretch > max) || (maxstretch < min || maxstretch > max)) return 0;
if (minstretch >= maxstretch)
   {maxstretch = minstretch + 1;}

// apply filter
histogramstretching(I1, minstretch, maxstretch, I2);
return 1;
```

The *histogramstrecthing* function applies the function g to the min and max values to calculate the output pixel. The pixel value is then clipped between 0 – 255. The clipped values are then copied to the output image. This is shown in the following code snippet.

```
void
HistogramStretching::histogramstretching(ImagePtr I1, int min, int max, ImagePtr I2)
{
IP_copyImageHeader(I1, I2);
int w = I1->width();
int h = I1->height();
int total = w * h;
//compute lut[]
//initialized lut of 256 entries
int i;
for(i=0; i<MXGRAY; ++i)
{Histogram[i] = CLIP((int)(MaxGray*(i- min)) / (max - min), 0, MaxGray) ;}


//p1 points to beginning of ch array
int type;
ChannelPtr<uchar> p1, p2, endd; //p1 is a pointer that points to pixel. p1++ is pointing to next pixel
for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
  IP_getChannel(I2, ch, p2, type);
  for(endd = p1 + total; p1<endd;) *p2++ = Histogram[*p1++];
  }
}
```



Histogram Stretching example where Min and Max are read from the input image



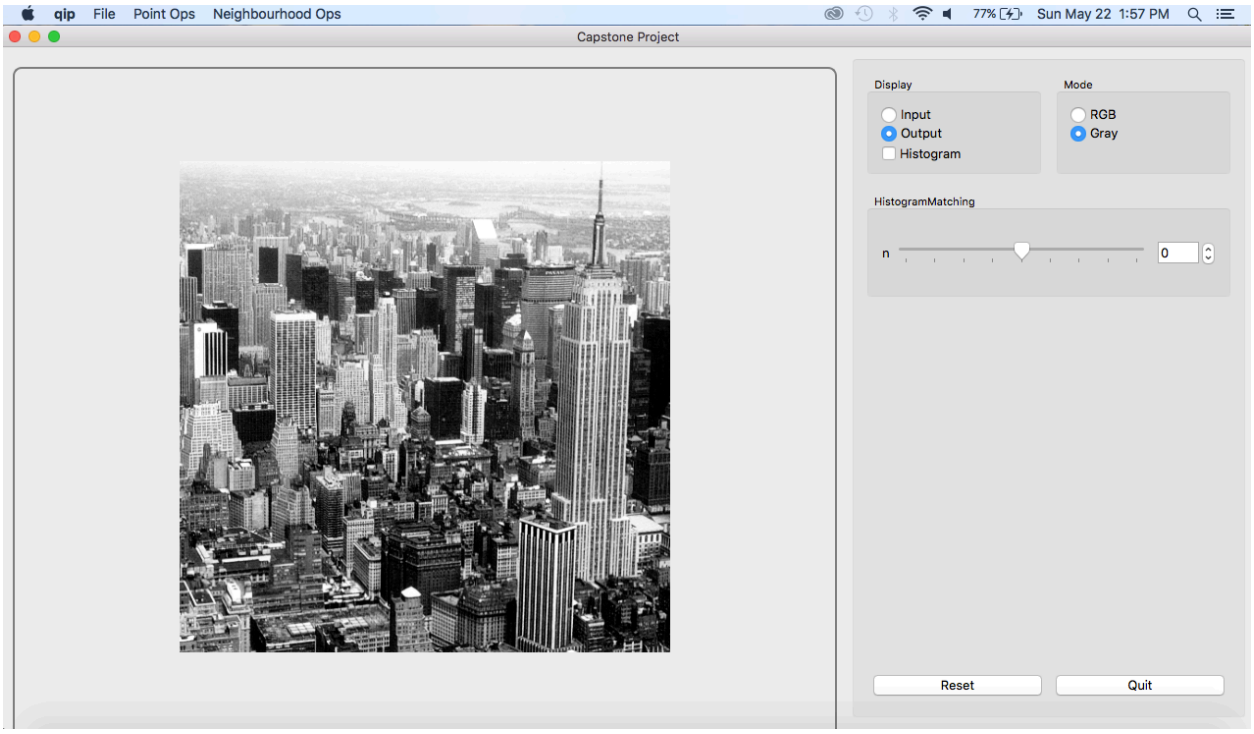
Histogram Stretching example where Min and Max are read from the slider.

Therefore, we see spikes at 0 and 255 for all the values below the selected Min: 63 are mapped to 0 and all the values above the selected Max: 197 are mapped to 255.

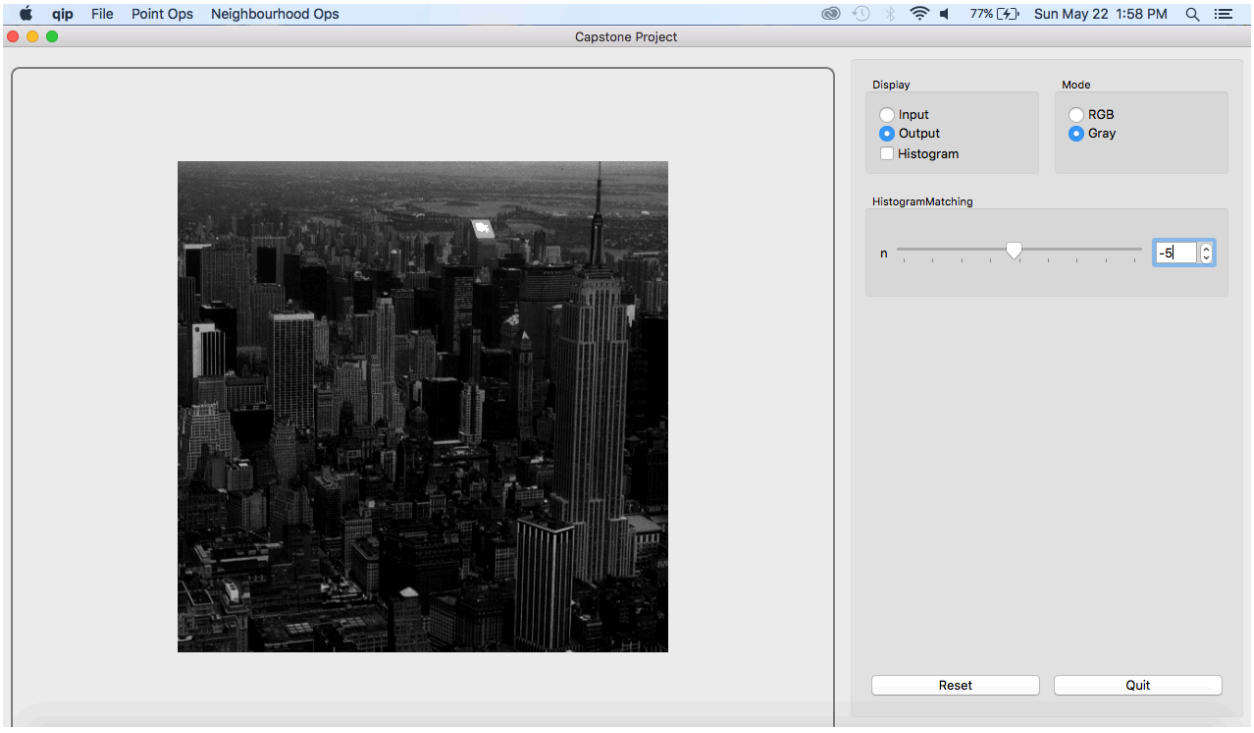
Histogram Matching

Histogram matching is a process where an image is transformed so that its histogram matches another specified histogram. Histogram equalization is the special case of histogram matching where the specified histogram is uniformly distributed. The specified histogram is specified by a number n. The interface will have a slider to read a number n. n = 0 will produce a flat histogram, which is the effect of histogram equalization. Positive n will produce an exponentially increasing histogram, and a negative n will produce an exponentially decreasing histogram. The absolute value of the number n is to be interpreted as the exponent of the exponential histogram. If the user specified value is positive, than the specified histogram is given as v^n where $0 <= v <= 1$. V is scaled from [0, 255] down to [0, 1]. On the other hand if the user specified value is negative, than the specified histogram is given as $1 - v^{|n|}$. We need to scale the histogram entries so that their sum is equal to the total number of pixels in the input image. This is shown in the code below.

```
// If n = 0 : histogram equalization
if (n == 0) {
  for (int i = 0; i < MXGRAY - 1; ++i) {
    h2[i] = (int)average;
  }
  h2[MXGRAY - 1] = total - (int) average*(MXGRAY - 1);
}

// if n > 0 exponentially increasing histogram
else if (n > 0) {
  for (int i=Havg = 0; i <MXGRAY; ++i) {
    temp = ROUND(pow((double) i/MXGRAY, (double) n) * MXGRAY);
    h2[i] = temp;
    Havg += h2[i];
  }
  scale = (double) total / Havg;
  if (scale != 1) {
    for (int i = 0; i < MXGRAY; ++i)
      h2[i] *= scale;
  }
}

// if n <0 exponentially decreasing histogram
else if (n < 0) {
  Havg = 0;
  for (int i =0; i < MXGRAY; ++i) {
    temp = ROUND(pow(1 - (double) i/MXGRAY, (double) abs(n)) * MXGRAY);
    h2[i] =  temp;
    Havg += h2[i];
  }
  scale = (double) total / Havg;
  if (scale != 1) {
    for (int i = 0; i < MXGRAY; ++i)
      h2[i] *= scale;
  }
}
```



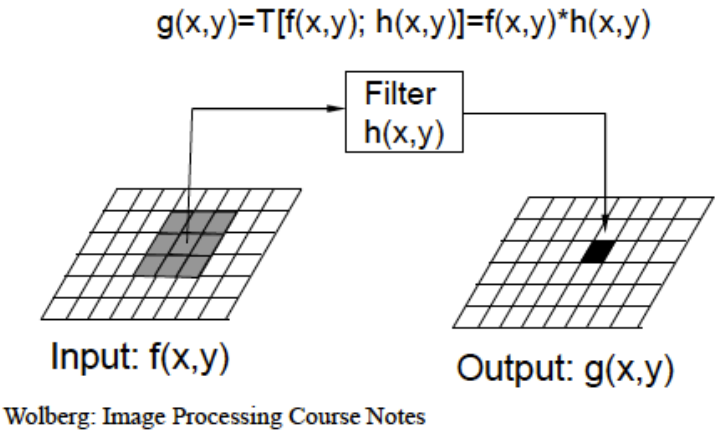
Histogram Equalization



Histogram Matching with negative n

Neighborhood Operations

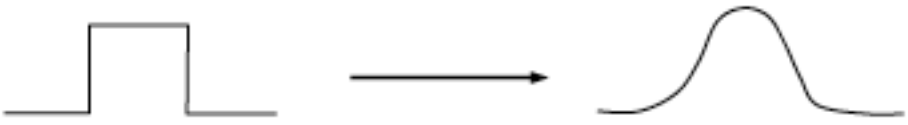
In neighborhood operations output pixels are a function of several input pixels. Transformation T is applied to few input pixels to calculate the value for each output pixel. The transformations are implemented with buffers. A collection of input pixels is stored in the buffer at a time. Transformation T is applied to the values in the buffer, and the resulting value is stored in the output pixel. In my program I implemented some neighborhood operations that are commonly used in image processing: Blur, Sharpen, and Median filter.



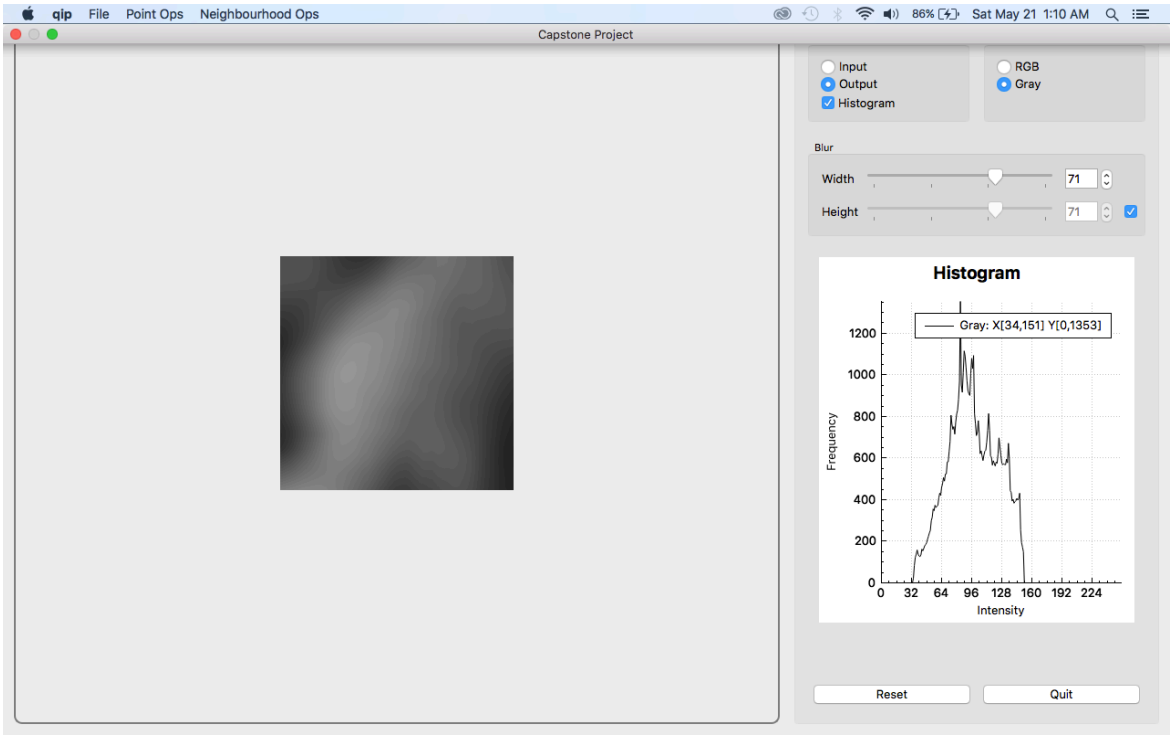
Wolberg: Image Processing Course Notes

Blur

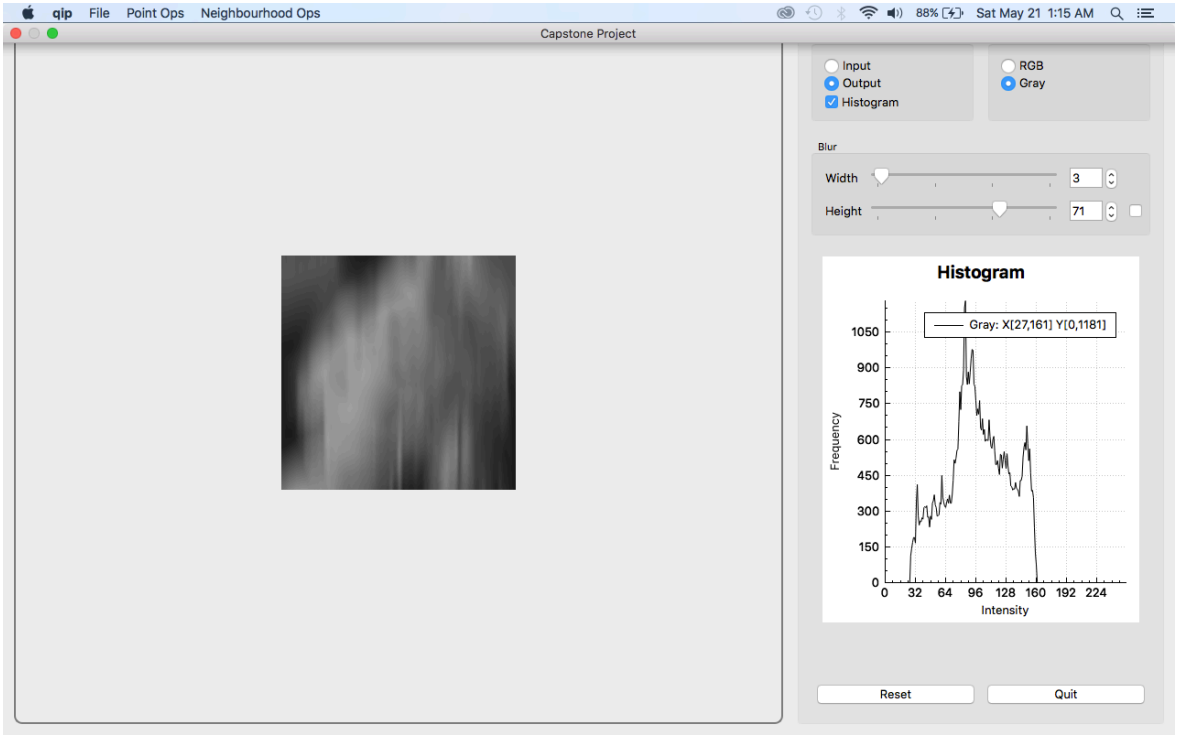
In blurring we reduce the edge content of an image, which makes the transition from one color to the other color very smooth. We apply blur with a box filter that has dimensions xsz * ysz.



We implemented the horizontal and vertical blur separately, so our program gives an option to either apply directional blurs, i.e. only vertical or horizontal blur. The images on the next page shows blur applied in both direction and blur applied in one direction only.



Blur applied in both directions



Blur applied in vertical direction

There is one checkbox in the control panel. If the checkbox is checked blur is applied in both directions with the same value. In this case we call the slot function *changeBoth*. The value from width slider is copied to the height slider in this case. Initially the checkbox is set true. Once the checkbox is unchecked we call the appropriate slot functions: *changeWidth* or *changeHeight* accordingly. Inside these slot functions we again check if the checkbox is checked now. If it is checked then we copy the value of that slider to the other slider accordingly. Blur is applied by reading pixel values from the box filter, *Width* * *Height,* into the buffer. Blur function is applied to these values and the result is stored in output pixel. To create these boxes around the boundary pixels we pad our buffer on left and right. In this implementation we only allow *Width* and *Height* to be odd numbers so that we can center the window at the current pixel. The slot functions therefore, convert even *Width* and *Height* values read from slider to odd values. Below are the three slot functions for blur.

```
void
Blur::changeWidth(int value)
{
  // if width value from slider is even add 1 to make it odd
  value += !(value %2);
  m_sliderW ->blockSignals(true);
  m_sliderW ->setValue    (value);
  m_sliderW ->blockSignals(false);
  m_spinBoxW->blockSignals(true);
  m_spinBoxW->setValue    (value);
  m_spinBoxW->blockSignals(false);

  // if checkbox is checked then set the height slider to same value as width slider
  if(m_checkBox->isChecked())
  {
    m_sliderH ->  setValue(value);
  }

  // apply filter to source image; save result in destination image
  applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

  // display output
  g_mainWindowP->displayOut();
}
```

```
void
Blur::changeHeight(int value)
{
  // if height value from slider is even add 1 to make it odd
  value += !(value %2);
  m_sliderH ->blockSignals(true);
  m_sliderH ->setValue    (value);
  m_spinBoxH->blockSignals(true);
  m_spinBoxH->setValue    (value);
  m_spinBoxH->blockSignals(false);
  m_sliderH ->blockSignals(false);

  // if checkbox is checked then set the width slider to same value as height slider
  if (m_checkBox->isChecked())
  {
    m_sliderW ->  setValue(value);
  }

  // apply filter to source image; save result in destination image
  applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

  // display output
  g_mainWindowP->displayOut();
}
```

```
void
Blur::changeBoth(int value)
{
  // checks if the checkbox is checked
  if (value)
  {
    // copy the value from width slider to height slider
    m_sliderH->setValue (m_sliderW->value());
  }

  // apply filter to source image; save result in destination image
  applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

  // display output
  g_mainWindowP->displayOut();
}
```

In our implementation we first apply blur horizontally, and then apply the blur vertically. The result is then stored in the output image. Before applying the blur we check if the blur window width and height are less than the image width and height. We also check for 1 * 1 window size, because this is a trivial case, and we do not need to apply any blur filter in this case. We will simply copy all the input pixels to output image.

```
void
Blur::blur(ImagePtr I1, int xsz, int ysz, ImagePtr I2) {
  int w = I1->width();
  int h = I1->height();
  int total = w * h;

  // error checking
  // If window width is greater than image width than we divide it by 2 and subtract 1 to make it odd
  if (xsz > w) {
    IP_copyImage(I1, I2);
    return;
  }

  // If window height is greater than image height than we divide it by 2 and subtract 1 to make it odd
  if (ysz > h) {
    IP_copyImage(I1, I2);
    return;
  }

  // trivial case:
  // if Width and Height are 1 the window size is 0 and no blurring needs to be done
  // we simple copy input image to output image
  if (xsz <= 1 && ysz<= 1){
    if (I1 != I2)
      IP_copyImage(I1, I2);
    return;
  }
```

In the above code the *blur* function does the error checking and checking for trivial case. After that it reads pixel values from the input image and send them to *IP_blur1D* function, which will create buffer and apply blur. This is shown in the code snippet on next page. The *blur* function first sends all the rows one by one to *IP_blur1D* function to apply the blur horizontally. Then it sends all the columns one by one to *IP_blur1D* to apply the blur vertically. This is done by two separate for loops as seen in the code on next page. *IP_blurID* is named so because it applies blur in 1D only.

```
IP_copyImageHeader(I1, I2);
int type;
ChannelPtr<uchar> src, dst;
ChannelPtr<float> fsrc, fdst;

for(int ch = 0; IP_getChannel(I1, ch, src, type); ch++) {
  if (type == UCHAR_TYPE) {
    if (xsz > 1.) {
      dst = I2[ch];
      for (int y =0; y<h; y++) {
        IP_blur1D(src, w, 1, xsz, dst);
        src += w;
        dst += w;
      }
      src = I2[ch];
    }

    if (ysz > 1.) {
      dst = I2[ch];
      for (int x =0; x<w; x++) {
        IP_blur1D(src, h, w, ysz, dst);
        src += 1;
        dst += 1;
      }
    }
  }
```

IP_blur1D has the arguments: input image, length of image, stride (distance from 1 element to next), width of filter, and output image. It reads one row or column at a time and applies blur to it. It first creates a buffer of size: length + width – 1. We add width to the length to pad extra spaces on both left and right of buffer. We then subtract 1 for the current pixel in the filter. Before we create the buffer we also do the error checking.

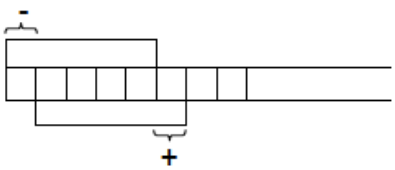
```
template <class T>
void
Blur::IP_blur1D(ChannelPtr<T> src, int len, int stride, double ww, ChannelPtr<T> dst) {

  size_t buf_size = len + ww -1;
  int padding = (ww - 1)/2;

  // error checking
  if (ww > len) {
    return;
  }

  // trivial  case
  if (ww <= 1) {
    if (src != dst) {
      for (int i =0; i<len; i++) {
        *dst = *src;
        dst += stride;
        src += stride;
      }
    }
    return;
  }

  uint16_t *buffer;
  buffer = (uint16_t*)malloc(sizeof(uint16_t*) * buf_size);
  if (buffer == NULL) {
    exit(0);
  }
```

Once the buffer is created we read values from the input image into the buffer. *Padding* is the extra spaces added on left and right of the buffer. *Padding* is always (1 – width) / 2 to ensure that it is odd. We subtract 1 from the filter width for the current pixel being read. We copy the first pixel value of the row to the left padded values, and we copy the last pixel value of the row to the right padded values. For all the remainder values in between we copy them from the row to buffer. This initialization of buffer is shown in the left image below.

```
int i = 0;
for (; i<padding; ++i) {
  buffer[i] = (*src);
}

len += i;

for (; i<len; ++i) {
  buffer[i] = (*src);
  src += stride;
}

src -= stride;

padding += i;
for (; i<padding; ++i) {
  buffer[i] = (*src);
  }
```

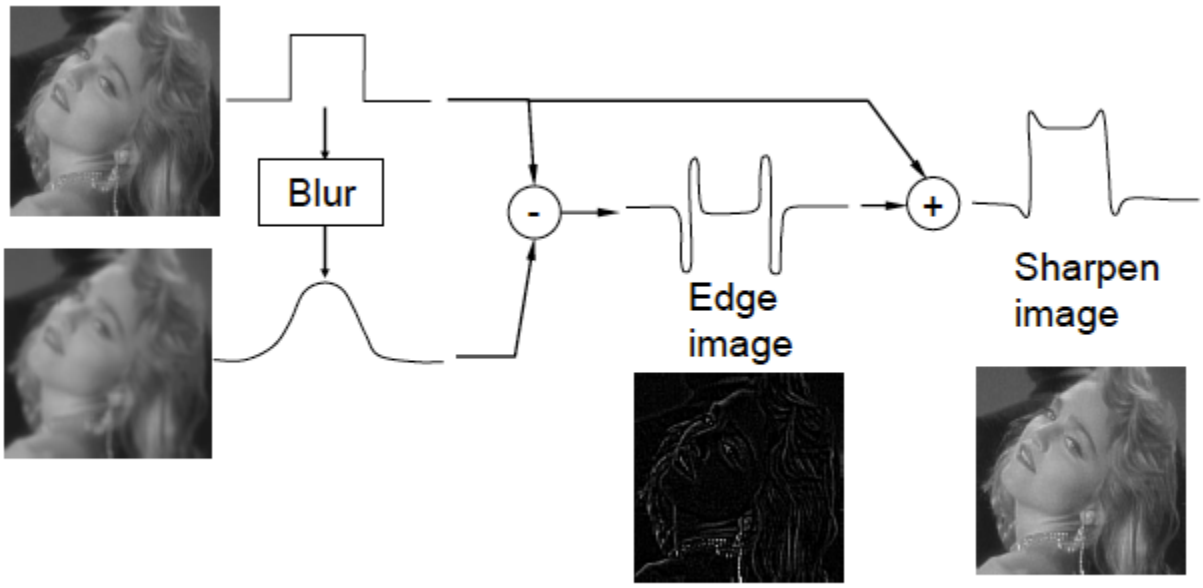
```
  ///////SUM//////////////
   double sum = 0;
   int j = 0;
   for (; j < ww; j++) {
     sum += buffer[j];
   }
   *dst = sum/ww ;
   dst += stride;

   for (; j< buf_size; j++, dst += stride) {
     int last = j - ww;
     sum += (buffer[j] - buffer[last]);
     *dst = sum/ww;
   }
   free (buffer);
}
```

Once the buffer has been initialized we apply blur to each pixel that was in the input image. This is shown in the right image above. Blurring is done by adding the pixel values for all pixels in width of filter, i.e. *ww*, and dividing it by *ww*. For example if *ww* is 3, we will apply blur to the current pixel by adding its pixel value, the pixel value of 1 next pixel on the left, and the pixel value of 1 next pixel on the right. We will then divide this sum by 3 and store the result in the corresponding output pixel. We than move filter along the buffer by subtracting the leftmost value in the filter and adding the next value to the right of filter into the filter. After the whole row is processed we clear the buffer.



Sharpen

In sharpening we enhance the edge content of an image to show sharp transitions between changes of colors. Sharpening is done by subtracting a blurred version of an image from the original image and adding the scaled difference back to the input image. The blurred version smooth out the transition regions where grayvalues vary. When we subtract this blurred version from the input image, the edges of different regions in the image are isolated. Adding these edges back onto the original image causes the edges to appear more prominent, hence, giving the image sharpening effect. The picture below visualizes this affect.

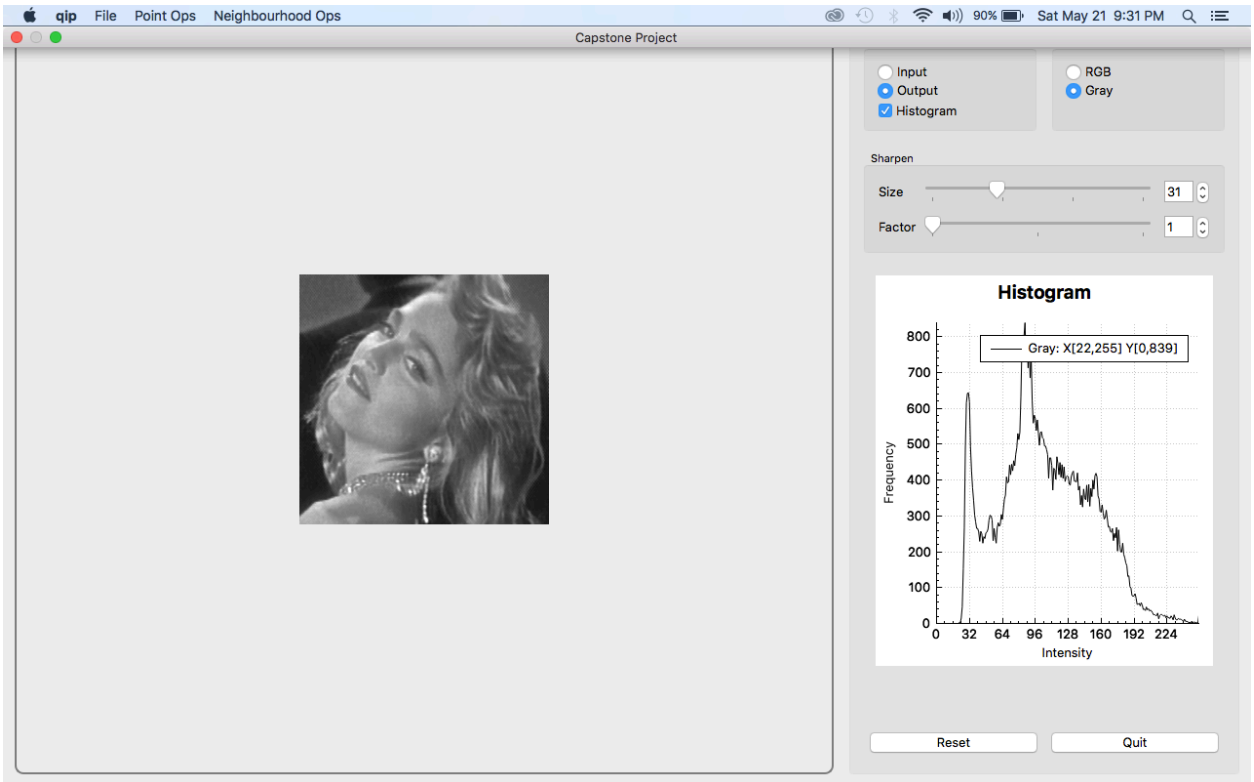


Wolberg: Image Processing Course Notes

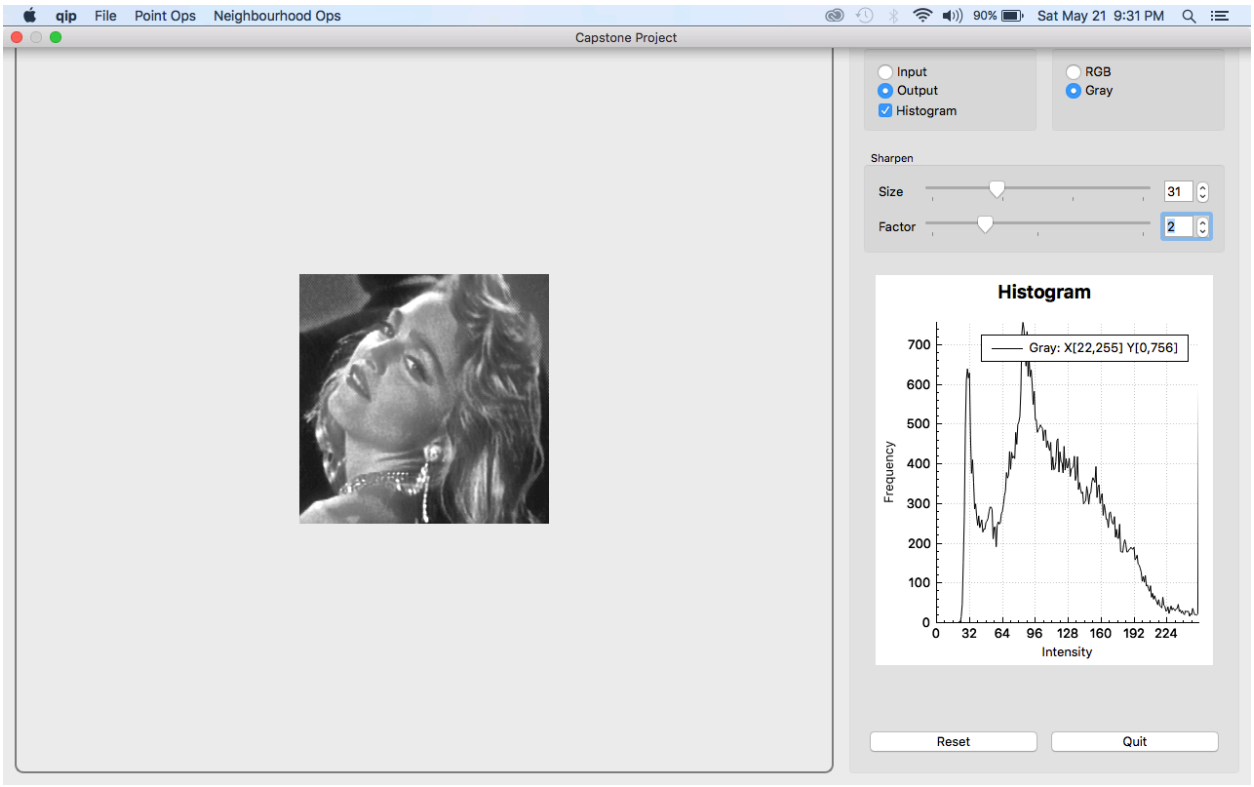
The blurred version is obtained by invoking our *blur* filter with filter dimensions xsz * xsz. The difference between input image and its blurred version is multiplied by a factor and then added back to the input image to yield the output image. We create a temporary image temp where we save the blurred image. We than subtract temp from I1 and add the difference back to I1 and output the result to I2. *Sharpblur* in the *sharpen* function below is the same blur function from

our blur filter. We CLIP the values between 0 – 255 range before we store them in the output image. The code for sharpening is shown below.

```
void
Sharpen::sharpen(ImagePtr I1, int sz, double fctr, ImagePtr I2) {
  ImagePtr temp;
  IP_copyImageHeader(I1, temp);
  IP_copyImageHeader(I1, I2);
  int w = I1->width();
  int h = I1->height();
  int total = w * h;

  // send I1 to sharpblur. The blurred image is stored in temp
  sharpblur (I1, sz, sz, temp);

  // src is pointer to I1, tempPointer is pointer to temp, dst is pointer to I2
  int type;
  ChannelPtr<uchar> src, tempPointer, dst, end;
  for(int ch = 0; IP_getChannel(I1, ch, src, type); ch++) {
    IP_getChannel(temp, ch, tempPointer, type);
    IP_getChannel(I2, ch, dst, type);
    for(end = src + total; src<end;)
      { *dst = CLIP(*src + CLIP((*src - *tempPointer), 0, 255) * fctr, 0, 255);
      ++src;
      ++tempPointer;
      ++dst;
    }
  }
}
```



Sharpen without multiplying with a Factor (Filter size: 31 * 31)



Sharpen with the Factor multiplication (Filter size: 31 * 31)

Median

In median filter pixels in a neighborhood are sorted in increasing order, and the neighborhood center is replaced with the median. It forces noisy pixels to conform to their neighbors, which is why it is excellent for noise reduction. In our implementation we included median with k-nearest neighbors. K-nearest neighbor is a variation that blurs median filtering with blurring. The median filter is applied to in over a neighborhood size of sx * sz. The input values in the neighborhood must be sorted. Then, the median is averaged with avg_nbrs pixels below and avg_nbrs pixels above in the sorted list, and the result is stored in out.

Appendix

Main.cpp

```
#include "MainWindow.h"

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

MainWindow.h

```
#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 "IP.h"
#include "IPtoUI.h"
#include "ImageFilter.h"
#include "qcustomplot.h"

#define MAXFILTERS 50

using namespace IP;
```

```
class MainWindow : public QMainWindow {
    Q_OBJECT

public:
    // constructor
    MainWindow      (QWidget *parent = 0);
    ImagePtr  imageSrc      () const;
    ImagePtr  imageDst      () const;
    QCustomPlot*    histogram()     {return m_histogram;}

public slots:
    void            open            ();
    void            displayIn       ();
    void            displayOut      ();
    void            modeRGB             ();
    void            modeGray        ();
    void            preview         ();
    void            reset           ();
    void            quit            ();
    void            execute         (QAction*);

protected slots:
    void            setHisto        (int);

protected:
    void            createActions   ();
    void            createMenus     ();
    void            createWidgets   ();
    QGroupBox*      createGroupPanel();
    QGroupBox*      createGroupDisplay  ();
    QGroupBox*      createDisplayButtons();
    QGroupBox*      createModeButtons();
    QHBoxLayout*    createExitButtons();
    void            displayHistogram(ImagePtr);
    void            display         (int);
    void            mode            (int);


private:
    QMenu*              m_menuFile;
    QMenu*              m_menuPtOps;
    QMenu*              m_menuNbrOps;
    QAction*        m_actionOpen;
    QAction*        m_actionQuit;
    QAction*        m_actionThreshold;
    QAction*        m_actionContrast;
    QAction*        m_actionQuantization;
    QAction*        m_actionHistogramStretching;
    QAction*        m_actionHistogramMatching;
    QAction*        m_actionBlur;
    QAction*        m_actionSharpen;
```

```
        QAction*        m_actionMedian;

        // homework objects
        ImageFilter*            m_imageFilterType[MAXFILTERS];

        // widgets
        QGroupBox*              m_groupBox;
        QStackedWidget*         m_stackWidgetImages;
        QStackedWidget*         m_stackWidgetPanels;

        // widgets for image display groupbox
        QRadioButton*           m_radioDisplay[2];      // radio buttons for input/output
        QRadioButton*           m_radioMode   [2];      // radio buttons for RGB/Gray modes
        QCheckBox*              m_checkboxHisto;        // checkbox: histogram display
        QWidget*                m_extension;            // extension widget for histogram
        QCustomPlot*            m_histogram;            // histogram plot

        int                     m_width;
        int                     m_height;
        int                     m_code;
        QString                 m_file;
        QString                 m_currentDir;
        ImagePtr        m_imageIn;
        ImagePtr        m_imageSrc;
        ImagePtr        m_imageDst;

        // histogram variables
        int                     m_histoColor;   // histogram color id: 0=RGB, 1=R, 2=G, 3=B, 4=gray
        double                  m_histoXmin[4];         // xmin for all histogram channels
        double                  m_histoXmax[4];         // xmax for all histogram channels
        double                  m_histoYmin[4];         // ymin for all histogram channels
        double                  m_histoYmax[4];         // ymax for all histogram channels
};



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::imageSrc:
//
// Source image.
//
inline ImagePtr
MainWindow::imageSrc() const
{
        return m_imageSrc;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::imageDst:
```

```
//
// Destination image.
//
inline ImagePtr
MainWindow::imageDst() const
{
	return m_imageDst;
}

#endif // MAINWINDOW_H
```

MainWindow.cpp

```
// ========================================================================
// Computer Graphics Homework Solutions
// Copyright (C) 2016 by George Wolberg
//
// MainWindow.cpp - MainWindow class
//
// Written by: George Wolberg, 2016
// ========================================================================

#include "MainWindow.h"
#include "Dummy.h"
#include "Threshold.h"
#include "Contrast.h"
#include "Quantization.h"
#include "HistogramStretching.h"
#include "HistogramMatching.h"
#include "Blur.h"
#include "Sharpen.h"
#include "Median.h"

using namespace IP;

enum {DUMMY, THRESHOLD, CONTRAST, QUANTIZATION, HISTOGRAMSTRETCHING,
HISTOGRAMMATCHING, BLUR, SHARPEN, MEDIAN};
enum {RGB, R, G, B, GRAY};

QString GroupBoxStyle = "QGroupBox {                                    \
                         border: 2px solid gray;                        \
                         border-radius: 9px;                            \
                         margin-top: 0.5em;}";

MainWindow *g_mainWindowP = NULL;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
// MainWindow::MainWindow:
//
// MainWindow constructor.
//
MainWindow::MainWindow(QWidget *parent)
        : QMainWindow(parent),
          m_code(-1),
          m_histoColor(0)
{
        setWindowTitle("Capstone Project");

        // set global variable for main window pointer
        g_mainWindowP = this;

        // INSERT YOUR ACTIONS AND MENUS
        createActions();
        createMenus  ();
        createWidgets();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createActions:
//
// Create actions to associate with menu and toolbar selection.
//
void
MainWindow::createActions()
{
        ////////////////////////////////
        // File Actions
        ////////////////////////////////

        m_actionOpen = new QAction("&Open", this);
        m_actionOpen->setShortcut(tr("Ctrl+O"));
        connect(m_actionOpen, SIGNAL(triggered()), this, SLOT(open()));

        m_actionQuit = new QAction("&Quit", this);
        m_actionQuit->setShortcut(tr("Ctrl+Q"));
        connect(m_actionQuit, SIGNAL(triggered()), this, SLOT(close()));

        ////////////////////////////////
        // Point Ops Actions
        ////////////////////////////////

        m_actionThreshold = new QAction("&Threshold", this);
        m_actionThreshold->setShortcut(tr("Ctrl+T"));
        m_actionThreshold->setData(THRESHOLD);

        m_actionContrast = new QAction("&Contrast", this);
```

```
        m_actionContrast->setShortcut(tr("Ctrl+C"));
        m_actionContrast->setData(CONTRAST);

        m_actionQuantization = new QAction("Qua&ntization", this);
        m_actionQuantization->setShortcut(tr("Ctrl+N"));
        m_actionQuantization->setData(QUANTIZATION);

        m_actionHistogramStretching = new QAction("Histogram&Stretching", this);
        m_actionHistogramStretching->setShortcut(tr("Ctrl+S"));
        m_actionHistogramStretching->setData(HISTOGRAMSTRETCHING);

        m_actionHistogramMatching = new QAction("Histogram&Matching", this);
        m_actionHistogramMatching->setShortcut(tr("Ctrl+M"));
        m_actionHistogramMatching->setData(HISTOGRAMMATCHING);

        m_actionBlur = new QAction("&Blur", this);
        m_actionBlur->setShortcut(tr("Ctrl+B"));
        m_actionBlur->setData(BLUR);

        m_actionSharpen = new QAction("Sharpen", this);
        m_actionSharpen->setShortcut(tr("Ctrl+P"));
        m_actionSharpen->setData(SHARPEN);

        m_actionMedian = new QAction("&Median", this);
        m_actionMedian->setShortcut(tr("Ctrl+M"));
        m_actionMedian->setData(MEDIAN);

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



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createMenus:
//
// Create menus and install in menubar.
//
void
MainWindow::createMenus()
{
        // File menu
        m_menuFile = menuBar()->addMenu("&File");
        m_menuFile->addAction(m_actionOpen);
        m_menuFile->addAction(m_actionQuit);

        // Point Ops menu
        m_menuPtOps = menuBar()->addMenu("&Point Ops");
        m_menuPtOps->addAction(m_actionThreshold);
        m_menuPtOps->addAction(m_actionContrast );
```

```
        m_menuPtOps->addAction(m_actionQuantization );
        m_menuPtOps->addAction(m_actionHistogramStretching );
  m_menuPtOps->addAction(m_actionHistogramMatching );

        m_menuNbrOps = menuBar()->addMenu("&Neighbourhood Ops");
        m_menuNbrOps->addAction(m_actionBlur );
        m_menuNbrOps->addAction(m_actionSharpen );
        m_menuNbrOps->addAction(m_actionMedian );

        // disable the point and neighborhood menus until image is loaded
        m_menuPtOps->setEnabled(false);
        m_menuNbrOps->setEnabled(false);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createWidgets:
//
// Create widgets for image display and filter control panels.
//
void
MainWindow::createWidgets()
{
        // assemble image display widget and control panel in horizontal layout
        QHBoxLayout *hbox = new QHBoxLayout;
        hbox->addWidget(createGroupDisplay());
        hbox->addWidget(createGroupPanel());
        hbox->setStretch(0, 1);

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

        // set central widget so that it can be displayed
        setCentralWidget(w);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createGroupPanel:
//
// Create group box for control panel.
//
QGroupBox*
MainWindow::createGroupPanel()
{
        // init group box
        m_groupBox = new QGroupBox;
        m_groupBox->setMinimumWidth(400);
```

```
        // filter's enum indexes into container of image filters
        m_imageFilterType[DUMMY         ] = new Dummy;
        m_imageFilterType[THRESHOLD] = new Threshold;
        m_imageFilterType[CONTRAST] = new Contrast;
        m_imageFilterType[QUANTIZATION ] = new Quantization;
        m_imageFilterType[HISTOGRAMSTRETCHING ] = new HistogramStretching;
        m_imageFilterType[HISTOGRAMMATCHING ] = new HistogramMatching;
        m_imageFilterType[BLUR ] = new Blur;
        m_imageFilterType[SHARPEN ] = new Sharpen;
        m_imageFilterType[MEDIAN ] = new Median;

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

        // add filter control panels to stacked widget
        m_stackWidgetPanels->addWidget(m_imageFilterType[DUMMY    ]->controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[THRESHOLD]->controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[CONTRAST ]->controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[QUANTIZATION]->controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[HISTOGRAMSTRETCHING]-
>controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[HISTOGRAMMATCHING]-
>controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[BLUR]->controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[SHARPEN]->controlPanel());
        m_stackWidgetPanels->addWidget(m_imageFilterType[MEDIAN]->controlPanel());

        // display blank dummmy panel initially
        m_stackWidgetPanels->setCurrentIndex(0);

        // assemble display and mode groups into horizontal layout
        QHBoxLayout *hbox = new QHBoxLayout;
        hbox->addWidget(createDisplayButtons());
        hbox->addWidget(createModeButtons   ());

       // create histogram plot
       m_histogram = new QCustomPlot;

        // set histogram title
       m_histogram->plotLayout()->insertRow(0);
       m_histogram->plotLayout()->addElement(0, 0, new QCPPlotTitle(m_histogram,"Histogram"));

       // assign label axes
       m_histogram->xAxis->setLabel("Intensity");
       m_histogram->yAxis->setLabel("Frequency");
       m_histogram->xAxis->setAutoTickStep(0);
       m_histogram->xAxis->setTickStep(32.);

       // set axes ranges, so we see all the data
       m_histogram->xAxis->setRange(0, MXGRAY);
```

```
        m_histogram->yAxis->setRange(0, MXGRAY);
        m_histogram->setMinimumHeight(400);
        m_histogram->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom);

        // create extension and insert histogram
        m_extension = new QWidget;
        QVBoxLayout *vBox2 = new QVBoxLayout(m_extension);
        vBox2->addWidget(m_histogram);

        // start dialog with hidden histogram
        m_extension->hide();

        // set signal-slot connection
        connect(m_checkboxHisto, SIGNAL(stateChanged(int)), this, SLOT(setHisto(int)));

        // assemble stacked widget in vertical layout
        QVBoxLayout *vbox = new QVBoxLayout;
        vbox->addLayout(hbox);
        vbox->addWidget(m_stackWidgetPanels);
        vbox->addWidget(m_extension);
        vbox->addStretch(1);
        vbox->addLayout(createExitButtons());
        m_groupBox->setLayout(vbox);

        // disable control panel until image is loaded
    m_groupBox->setEnabled(false);

        return m_groupBox;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createGroupDisplay:
//
// Create group box for displaying images.
//
QGroupBox*
MainWindow::createGroupDisplay()
{
        // init group box
        QGroupBox *groupBox = new QGroupBox;
        groupBox->setMinimumWidth (700);
        groupBox->setMinimumHeight(700);
        groupBox->setStyleSheet(GroupBoxStyle);

        // create stacked widget for input/output images
        m_stackWidgetImages = new QStackedWidget;

        // add QLabels to image stacked widget to display input/output images
        for(int i = 0; i<2; ++i)
```

```
        m_stackWidgetImages->addWidget(new QLabel);

    // add centering alignment on both labels
    QLabel *label;
    label = (QLabel *) m_stackWidgetImages->widget(0); label->setAlignment(Qt::AlignCenter);
    label = (QLabel *) m_stackWidgetImages->widget(1); label->setAlignment(Qt::AlignCenter);

    // set stacked widget to default setting: input image
    m_stackWidgetImages->setCurrentIndex(0);

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

    return groupBox;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createDisplayButtons:
//
// Create preview window groupbox.
//
QGroupBox*
MainWindow::createDisplayButtons()
{
    // init group box
    QGroupBox *groupBox = new QGroupBox("Display");

    // create radio buttons
    m_radioDisplay[0] = new QRadioButton("Input");
    m_radioDisplay[1] = new QRadioButton("Output");

    // create button group and add radio buttons to it
    QButtonGroup *bGroup = new QButtonGroup;
    for(int i = 0; i<2; ++i)
        bGroup->addButton(m_radioDisplay[i]);

    // set input radio button to be default
    m_radioDisplay[0]->setChecked(true);

    // create histogram checkbox
    m_checkboxHisto = new QCheckBox("Histogram");
    m_checkboxHisto ->setCheckState (Qt::Unchecked);

    // assemble radio buttons into vertical widget
    QVBoxLayout *vbox = new QVBoxLayout;
    vbox->addWidget(m_radioDisplay[0]);
    vbox->addWidget(m_radioDisplay[1]);
```

```
        vbox->addWidget(m_checkboxHisto);
        groupBox->setLayout(vbox);

        // init signal/slot connections
        connect(m_radioDisplay[0], SIGNAL(clicked()), this, SLOT(displayIn()));
        connect(m_radioDisplay[1], SIGNAL(clicked()), this, SLOT(displayOut()));

        return groupBox;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::createModeButtons:
//
// Create preview window groupbox.
//
QGroupBox*
MainWindow::createModeButtons()
{
        // init group box
        QGroupBox *groupBox = new QGroupBox("Mode");

        // create radio buttons
        m_radioMode[0] = new QRadioButton("RGB");
        m_radioMode[1] = new QRadioButton("Gray");

        // create button group and add radio buttons to it
        QButtonGroup *bGroup = new QButtonGroup;
        for(int i = 0; i<2; ++i)
           bGroup->addButton(m_radioMode[i]);

        // set RGB radio button to be default
        m_radioMode[0]->setChecked(true);

        // assemble radio buttons into vertical widget
        QVBoxLayout *vbox = new QVBoxLayout;
        vbox->addWidget(m_radioMode[0]);
        vbox->addWidget(m_radioMode[1]);
        vbox->addWidget(m_radioMode[1]);        // redundant radiobutton won't display; placeholder
        groupBox->setLayout(vbox);

        // init signal/slot connections
        connect(m_radioMode[0], SIGNAL(clicked()), this, SLOT(modeRGB ()));
        connect(m_radioMode[1], SIGNAL(clicked()), this, SLOT(modeGray()));

        return groupBox;
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// 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::reset:
//
// Reset application.
//
void
MainWindow::reset()
{

//      m_imageFilterType[ m_imageFilterName[index] ]->reset();
}



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

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::open:
//
// Open input image.
//
void
MainWindow::open() {
        QFileDialog dialog(this);

        // open the last known working directory
        if(!m_currentDir.isEmpty())
           dialog.setDirectory(m_currentDir);

        // display existing files and directories
        dialog.setFileMode(QFileDialog::ExistingFile);

        // invoke native file browser to select file
        m_file = dialog.getOpenFileName(this,
                                        "Open File", m_currentDir,
                                        "Images (*.jpg *.png *.ppm *.pgm *.bmp);;All files (*)");

        // verify that file selection was made
        if(m_file.isNull()) return;

        // save current directory
        QFileInfo f(m_file);
        m_currentDir = f.absolutePath();

        // DISABLE IMAGING FUNCTIONALITY FOR NOW....
        // read input image
        m_imageIn = IP_readImage(qPrintable(m_file));

        if(m_radioMode[1]->isChecked())
           IP_castImage(m_imageIn,  BW_IMAGE, m_imageSrc);
        else         IP_castImage(m_imageIn, RGB_IMAGE, m_imageSrc);

        // init vars
        m_width  = m_imageSrc->width ();
        m_height = m_imageSrc->height();
        preview();

        // enable the menus
        m_menuPtOps      ->setEnabled(true);
        m_menuNbrOps->setEnabled(true);

        // enable the groupBox
        m_groupBox->setEnabled(true);
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::display:
//
// Slot functions to display input and output images.
//
void MainWindow::displayIn () { display(0); }
void MainWindow::displayOut() { display(1); }

void MainWindow::display(int flag)
{
        // error checking
        if(m_imageSrc.isNull())             return;         // no input  image
        if(m_imageDst.isNull() && flag) return;             // no output image

        // raise the appropriate widget from the stack
        m_stackWidgetImages->setCurrentIndex(flag);

        // set radio button
        m_radioDisplay[flag]->setChecked(1);

        // determine image to be displayed
        ImagePtr I;
        if(flag == 0)
           I = m_imageSrc;
        else        I = m_imageDst;

        // init image dimensions
        int  w = m_imageDst->width();
        int  h = m_imageDst->height();

        // init view window dimensions
        int ww = m_stackWidgetImages->width();
        int hh = m_stackWidgetImages->height();

        // convert from ImagePtr to QImage to Pixmap
        QImage q;
        IP_IPtoQImage(I, q);

        // convert from QImage to Pixmap; rescale if image is larger than view window
        QPixmap p;
        if(MIN(w, h) > MIN(ww, hh))
           p = QPixmap::fromImage(q.scaled(QSize(ww, hh), Qt::KeepAspectRatio));
        else        p = QPixmap::fromImage(q);

        // assign pixmap to label widget for display
        QLabel *widget = (QLabel *) m_stackWidgetImages->currentWidget();
        widget->setPixmap(p);
```

```
        // compute histogram if histogram checkbox is set
        if(m_checkboxHisto->isChecked())
           displayHistogram(I);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::displayHistogram:
//
// Display image histogram in control panel
//
void MainWindow::displayHistogram(ImagePtr I)
{
        int color;
        int histo[MXGRAY];
        int yminChannel=0, ymaxChannel=0;
        int yminHisto=0,   ymaxHisto=0;
        int xmin, xmax;
        QVector<double> x, y;
        char buf[MXSTRLEN];

        // clear any previous histogram plots
        m_histogram->clearGraphs();

        // visit all selected channels in I: RGB, R, G, B, or gray
        for(int ch=0; ch<I->maxChannel(); ch++) {
           // compute histogram
           IP_histogram(I, ch, histo, MXGRAY, m_histoXmin[ch], m_histoXmax[ch]);

           // init min and max for current channel
           yminChannel = ymaxChannel = histo[0];

           // init min and max histogram value among all channels
           if(!ch) yminHisto = ymaxHisto = histo[0];

           // clear vector of x- and y-coordinates
           x.clear();
           y.clear();

           // visit all histogram entries and save into x and y vectors
           for(int i=0; i<MXGRAY; i++) {
                   x.push_back(i);
                   y.push_back(histo[i]);

                   // save channel min and max
                   yminChannel = MIN(yminChannel, histo[i]);
                   ymaxChannel = MAX(ymaxChannel, histo[i]);
           }

           // add new graph for histogram channel
```

```
m_histogram->addGraph();

   // if single channel was selected, it is in channel 0 and should
   // be drawn in corresponding color (based on m_histoColor).
   // Else, color is based on ch value: 0,1,2 corresponds to R,G,B
   // Add 1 to ch so that ch=1 corresponds to green, and ch=2 to blue
   if(!ch) color = m_histoColor;
   else       color = ch + 1;

   // convert xmin, xmax of channel into int
   xmin = m_histoXmin[ch];
   xmax = m_histoXmax[ch];

   // set histogram name and color
   switch(color) {
   case RGB:
   case R: sprintf(buf, "R: X[%d,%d] Y[%d,%d]",
               xmin, xmax, yminChannel, ymaxChannel);
               m_histogram->graph(ch)->setName(buf);
               m_histogram->graph(ch)->setPen (QPen(Qt::red  ));
               break;
   case G: sprintf(buf, "G: X[%d,%d] Y[%d,%d]",
               xmin, xmax, yminChannel, ymaxChannel);
               m_histogram->graph(ch)->setName(buf);
               m_histogram->graph(ch)->setPen(QPen(Qt::green));
               break;
   case B: sprintf(buf, "B: X[%d,%d] Y[%d,%d]",
               xmin, xmax, yminChannel, ymaxChannel);
               m_histogram->graph(ch)->setName(buf);
               m_histogram->graph(ch)->setPen(QPen(Qt::blue ));
               break;
   case GRAY:
               sprintf(buf, "Gray: X[%d,%d] Y[%d,%d]",
               xmin, xmax, yminChannel, ymaxChannel);
               m_histogram->graph(ch)->setName(buf);
               m_histogram->graph(ch)->setPen(QPen(Qt::black ));
               break;
   }

   // set data
   m_histogram->graph(ch)->setData(x,y);

   // update min and max histogram values among all channels
   yminHisto = MIN(yminChannel, yminHisto);
   ymaxHisto = MAX(ymaxChannel, ymaxHisto);
}

// turn on legend to print histogram params
m_histogram->legend->setVisible(true);

// set y-axis range and replot
```

```
	m_histogram->yAxis->setRange(yminHisto, ymaxHisto);
	m_histogram->rescaleAxes();
	m_histogram->replot();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::setHisto:
//
// Slot to show/hide histogram and set/reset histogram checkbox.
//
void
MainWindow::setHisto(int flag)
{
	m_extension->setVisible(flag);
	if(flag)
		m_checkboxHisto->setCheckState(Qt::Checked);
	else	m_checkboxHisto->setCheckState(Qt::Unchecked);
	preview();
}




// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::mode:
//
// Slot functions to display RGB and grayscale images.
//
void MainWindow::modeRGB () { mode(0); }
void MainWindow::modeGray() { mode(1); }

void MainWindow::mode(int flag)
{
	// error checking
	if(m_imageSrc.isNull()) return;			// no input image

	if(flag)
	   IP_castImage(m_imageIn,  BW_IMAGE, m_imageSrc);
	else	   IP_castImage(m_imageIn, RGB_IMAGE, m_imageSrc);

	if(m_imageSrc->imageType() == BW_IMAGE)
	   m_histoColor = GRAY;	// gray
	else	   m_histoColor = 0;	// RGB

	// re-apply filter for changed mode
	if(m_code > 0)
	   m_imageFilterType[m_code]->applyFilter(m_imageSrc, m_imageDst);

	// display image
```

```
        preview();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::preview:
//
// Display preview image.
//
void
MainWindow::preview()
{
        // display requested image
        if(m_radioDisplay[0]->isChecked())
           display(0);
        else         display(1);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// MainWindow::execute:
//
// Determine which action was triggered and execute respective action.
// Skip this if action is not image-processing related: open(), quit()
//
void
MainWindow::execute(QAction* action)
{
        // skip over menu ops that don't require image processing
        QString name = action->text();
        if(name == QString("&Open") || name == QString("&Quit")) {
           m_code = -1;
           return;
        }

        // get code from action
        m_code = action->data().toInt();

        // set output radio button to true
        m_radioDisplay[1]->setChecked(true);

        // use code to index into stack widget and array of filters
        m_stackWidgetPanels->setCurrentIndex(m_code);
        m_imageFilterType[m_code]->applyFilter(m_imageSrc, m_imageDst);
        preview();
}
```

Threshold.h

```
#ifndef THRESHOLD_H
#define THRESHOLD_H

#include "ImageFilter.h"


class Threshold : public ImageFilter {
	Q_OBJECT

public:
	Threshold(QWidget *parent = 0);			// constructor
	QGroupBox*	controlPanel	();	// create control panel
	bool		applyFilter(ImagePtr, ImagePtr);	// apply filter to input to init output
	void		reset		();		// reset parameters

protected:
	void threshold(ImagePtr I1, int thr, ImagePtr I2);

protected slots:
	void changeThr(int);

private:
	// threshold controls
	QSlider		*m_slider ;	// Threshold sliders
	QSpinBox	*m_spinBox;	// Threshold spin boxes

	// label for Otsu thresholds
	QLabel		*m_label;	// Label for printing Otsu thresholds

	// widgets and groupbox
	QGroupBox	*m_ctrlGrp;	// Groupbox for panel
};


#endif			// THRESHOLD_H
```

Threshold.cpp

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
```

```
//
// Threshold.cpp - Threshold class
//
// Written by: George Wolberg, 2016
// ==========================================================================

#include "MainWindow.h"
#include "Threshold.h"

extern MainWindow *g_mainWindowP;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Threshold::Threshold:
//
// Constructor.

Threshold::Threshold(QWidget *parent) : ImageFilter(parent)
{}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Threshold::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
Threshold::applyFilter(ImagePtr I1, ImagePtr I2)
{
        // error checking
        if(I1.isNull()) return 0;

        // get threshold value
        int thr = m_slider->value();

        // error checking
        if(thr < 0 || thr > MXGRAY) return 0;

        // apply filter
        threshold(I1, thr, I2);

        return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Threshold::controlPanel:
//
```

```
// Create group box for control panel.
//
QGroupBox*
Threshold::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("Threshold");

        // init widgets
        // create label[i]
        QLabel *label = new QLabel;
        label->setText(QString("Thr"));

        // create slider
        m_slider = new QSlider(Qt::Horizontal, m_ctrlGrp);
        m_slider->setTickPosition(QSlider::TicksBelow);
        m_slider->setTickInterval(25);
        m_slider->setMinimum(1);
        m_slider->setMaximum(MXGRAY);
        m_slider->setValue  (MXGRAY>>1);

        // create spinbox
        m_spinBox = new QSpinBox(m_ctrlGrp);
        m_spinBox->setMinimum(1);
        m_spinBox->setMaximum(MXGRAY);
        m_spinBox->setValue  (MXGRAY>>1);

        // init signal/slot connections for Threshold
        connect(m_slider , SIGNAL(valueChanged(int)), this, SLOT(changeThr (int)));
        connect(m_spinBox, SIGNAL(valueChanged(int)), this, SLOT(changeThr (int)));

        // assemble dialog
        QGridLayout *layout = new QGridLayout;
        layout->addWidget(  label  , 0, 0);
        layout->addWidget(m_slider , 0, 1);
        layout->addWidget(m_spinBox, 0, 2);

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

        return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Threshold::changeThr:
//
// Slot to process change in thr caused by moving the slider.
//
void
```

```
Threshold::changeThr(int thr)
{
        m_slider ->blockSignals(true);
        m_slider ->setValue    (thr );
        m_slider ->blockSignals(false);
        m_spinBox->blockSignals(true);
        m_spinBox->setValue    (thr );
        m_spinBox->blockSignals(false);

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Threshold::threshold:
//
// Threshold I1 using the 2-level mapping shown below.  Output is in I2.
// val<thr: 0;       val >= thr: MaxGray (255)
//! \brief            Threshold I1 using the 3-level mapping shown below.
//! \details          Output is in I2. val<t1: g1; t1<=val<t2: g2; t2<=val: g3
//! \param[in]     I1  - Input image.
//! \param[in]     thr - Threshold.
//! \param[out]   I2  - Output image.
//
void
Threshold::threshold(ImagePtr I1, int thr, ImagePtr I2) {
        IP_copyImageHeader(I1, I2);
        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // compute lut[]
        int i, lut[MXGRAY];
        for(i=0; i<thr && i<MXGRAY; ++i) lut[i] = 0;
        for(   ; i <= MaxGray;      ++i) lut[i] = MaxGray;

        int type;
        ChannelPtr<uchar> p1, p2, endd;
        for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
           IP_getChannel(I2, ch, p2, type);
           for(endd = p1 + total; p1<endd;) *p2++ = lut[*p1++];
        }
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Threshold::reset:
//
// Reset parameters.
//
void
Threshold::reset() {}
```

Contrast.h

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Contrast.h - Contrast widget
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#ifndef CONTRAST_H
#define CONTRAST_H

#include "ImageFilter.h"

class Contrast : public ImageFilter {
        Q_OBJECT

public:
        Contrast   (QWidget *parent = 0);            // constructor
        QGroupBox*      controlPanel    ();                 // create control panel
        bool                    applyFilter(ImagePtr, ImagePtr);// apply filter to input to init output
        void                    reset               ();                 // reset parameters

protected:
        void contrast(ImagePtr I1, double brightness, double contrast, ImagePtr I2);

protected slots:
        void    changeCtr (int);
        void    changeBri (int);

private:
        // brightness/contrast controls
        QSlider             *m_sliderB ;        // brightness slider
        QSlider             *m_sliderC ;        // contrast slider
        QSpinBox            *m_spinBoxB;        // brightness spin box
        QDoubleSpinBox *m_spinBoxC;  // contrast   spin box

        // labels for brightness and contrast
        QLabel              *m_labelC;          // contrast label
```

```
        QLabel          *m_labelB;      // brightness label

        // widgets and groupbox
        QGroupBox       *m_ctrlGrp;     // groupbox for panel
};

#endif      // CONTRAST_H
```

Contrast.cpp

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Contrast.cpp - Brightness/Contrast class.
//
// Written by: Khadeeja Din, 2016
// ========================================================================



#include "MainWindow.h"
#include "Contrast.h"

extern MainWindow *g_mainWindowP;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

//bounds for brightness slider and spinbox
int min = -100;
int max = 100;

//bounds for contrast slider and spinbox
double cmin = 0.0;
double cmax = 5.0;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast::Contrast:
//
// Constructor.
//
Contrast::Contrast(QWidget *parent) : ImageFilter(parent)
{}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast::applyFilter:
```

```
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
Contrast::applyFilter(ImagePtr I1, ImagePtr I2)
{
        // error checking
        if (I1.isNull()) return 0;

        // brightness and contrast parameters
        double b, c;

  //get brightness value
        b = m_sliderB->value();

        // get contrast value
        c = m_sliderC->value();

        // error checking
  if ((b < min || b > max) || (c < min || c > max)) return 0;

        // apply filter
  contrast(I1, b, c, I2);

        return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast::createGroupBox:
//
// Create group box for control panel.
//
QGroupBox*
Contrast::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("Contrast");

        // init widgets
        // create label[i] for contrast
        QLabel *labelC = new QLabel;
        labelC->setText(QString("Ctr"));

        // create label[i] brightness
        QLabel *labelB = new QLabel;
        labelB->setText(QString("Bri"));
```

```
    // create contrast slider
    m_sliderC = new QSlider(Qt::Horizontal, m_ctrlGrp);
    m_sliderC->setTickPosition(QSlider::TicksBelow);
    m_sliderC->setTickInterval(25);
    m_sliderC->setMinimum(min);
    m_sliderC->setMaximum(max);
    m_sliderC->setValue (max>>min);

    // create contrast spinbox
    m_spinBoxC = new QDoubleSpinBox(m_ctrlGrp);
    m_spinBoxC->setMinimum(cmin);            //cmin is 0.0
    m_spinBoxC->setMaximum(cmax);             //cmax is 5.0
    m_spinBoxC->setDecimals(1);
    m_spinBoxC->setValue (2.5);

    // create brightness slider
    m_sliderB = new QSlider(Qt::Horizontal, m_ctrlGrp);
    m_sliderB->setTickPosition(QSlider::TicksBelow);
    m_sliderB->setTickInterval(25);
    m_sliderB->setMinimum(min);
    m_sliderB->setMaximum(max);
    m_sliderB->setValue  (max>>min);

    // create brightness spinbox
    m_spinBoxB = new QSpinBox(m_ctrlGrp);
    m_spinBoxB->setMinimum(min);
    m_spinBoxB->setMaximum(max);
    m_spinBoxB->setValue  (max>>min);

    // init signal/slot connections for Contrast
    connect(m_sliderC , SIGNAL(valueChanged(int)), this, SLOT(changeCtr (int)));
    connect(m_spinBoxC, SIGNAL(valueChanged(double)), this, SLOT(changeCtr (int)));

    // init signal/slot connections for Brightness
    connect(m_sliderB , SIGNAL(valueChanged(int)), this, SLOT(changeBri (int)));
    connect(m_spinBoxB, SIGNAL(valueChanged(int)), this, SLOT(changeBri (int)));

    // assemble dialog
    QGridLayout *layout = new QGridLayout;
    layout->addWidget(       labelC            , 0, 0);
    layout->addWidget(       labelB            , 1, 0);
    layout->addWidget(m_sliderC , 0, 1);
    layout->addWidget(m_spinBoxC, 0, 2);
    layout->addWidget(m_sliderB , 1, 1);
    layout->addWidget(m_spinBoxB, 1, 2);

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

    return(m_ctrlGrp);
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast:changeCtr:
//
// Slot to process change in c caused by moving the slider.
//
void Contrast::changeCtr(int c)
{
        // contr variable to scale our contrast factor: c
  double contr;

        // initializing contr
        if (c >= 0)
           contr = c/25.0 + 1.0;
        else
           contr = c/133.0 + 1.0;

        m_sliderC ->blockSignals(true);
        m_sliderC ->setValue                (c   );
        m_sliderC ->blockSignals(false);
        m_spinBoxC->blockSignals(true);
        m_spinBoxC->setValue              (contr);
        m_spinBoxC->blockSignals(false);

// apply filter to source image; save result in destination Image
applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

// display Output
g_mainWindowP->displayOut();

}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast:changeBri:
//
// Slot to process change in b caused by moving the slider.
//
void
Contrast::changeBri(int b)
{
        m_sliderB ->blockSignals(true);
        m_sliderB ->setValue    (b );
        m_sliderB ->blockSignals(false);
        m_spinBoxB->blockSignals(true);
        m_spinBoxB->setValue    (b );
        m_spinBoxB->blockSignals(false);
```

```
        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast::contrast:
//
// Contrast I1 using the 2-level mapping shown below.  Output is in I2.
//! \details        Output is in I2. val = (pixel - reference)* contr + reference + brightness
//! \param[in]     I1  - Input image.
//! \param[in]     brightness - Brightness.
//! \param[in]     contrast - Contrast.
//! \param[out]    I2  - Output image.
//
void
Contrast::contrast(ImagePtr I1, double brightness, double contrast, ImagePtr I2)
{
        IP_copyImageHeader(I1, I2);
        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // fixed reference pixel intensity value.
        // this is the intersection point between brightness & contrast
        int reference = 128;

        // contr variable to scale our contrast factor: contrast
        double contr;

        // initializing contr
        if (contrast >= 0)
           contr = contrast/25.0 + 1.0;
        else
           contr = contrast/133.0 + 1.0;

        // compute lut[], i.e. lookup table
        // initialized lut of 256 entries
        int i, lut[MXGRAY];

        // applying brightness & contrats algorithm to pixel intensities and storing their corresponding
values in lut
        // the algorithm darkens levels below our reference point and brightens levels above our reference
point
        // CLIP the value between 0 - 255 to make sure the value does not go off range
        for(i=0; i<MXGRAY; ++i)
           lut[i] = (int)CLIP((i - reference)* contr + reference + brightness, 0, 255);
```

```
        // for each pixel intensity in I1, read its coresponding value from lut, and output it to I2
        // p1 is a pointer that points to current pixel in I1. p1++ is pointing to next pixel in I1
        // p2 is a pointer that points to current pixel in I2. p2++ is pointing to next pixel in I2
        // initially p1 points to beginning of ch array
        int type;
        ChannelPtr<uchar> p1, p2, endd;
        for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
           IP_getChannel(I2, ch, p2, type);
           for(endd = p1 + total; p1<endd;) *p2++ = lut[*p1++];
        }
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Contrast::reset:
//
// Reset parameters.
//
void
Contrast::reset() {}
```

Quantization.h

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Quantization.h - Quantization widget
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#ifndef QUANTIZATION_H
#define QUANTIZATION_H

#include "ImageFilter.h"


class Quantization : public ImageFilter {
        Q_OBJECT

public:
        Quantization        (QWidget *parent = 0);          // constructor
        QGroupBox*          controlPanel      ();           // create control panel
```

```
        bool            applyFilter(ImagePtr, ImagePtr);// apply filter to input to init output
        void            reset           ();             // reset parameters

protected:
        void quantization(ImagePtr I1, int quan, int dither, ImagePtr I2);

protected slots:
        void changeQuan(int);

private:
        // Quantization controls
        QSlider         *m_slider ;     // Quantization sliders
        QSpinBox        *m_spinBox;     // Quantization spin boxes

        QCheckBox *m_checkBox; //Dither checkbox

        // label for Quantization
        QLabel          *m_label;       // Label for printing Quantization
        QLabel          *m_dlabel; // label for dither check box

        // widgets and groupbox
        QGroupBox       *m_ctrlGrp;     // Groupbox for panel
};


#endif  // QUANTIZATION_H
```

Quantization.cpp

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Quantization.cpp - Quantization class
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#include "MainWindow.h"
#include "Quantization.h"
#include <cstdlib>

extern MainWindow *g_mainWindowP;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Quantization::Quantization:
//
// Constructor.

Quantization::Quantization(QWidget *parent) : ImageFilter(parent)
{}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Quantization::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
Quantization::applyFilter(ImagePtr I1, ImagePtr I2)
{
	// error checking
	if(I1.isNull()) return 0;

	// get quantization value
	int quan = m_slider->value();

	// get dither value (true or false)
	int dither = m_checkBox->isChecked();

	// error checking
	if(quan < 0 || quan> MXGRAY) return 0;

	// apply filter
	quantization(I1, quan, dither, I2);

	return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Quantization::controlPanel:
//
// Create group box for control panel.
//
QGroupBox*
Quantization::controlPanel()
{
	// init group box
	m_ctrlGrp = new QGroupBox("Quantization");

	// init widgets
	// create label[i] for quantization Levels
	QLabel *label = new QLabel;
	label->setText(QString("Levels"));

	// create label[i] for dither checkbox
	QLabel *dlabel = new QLabel;
	dlabel->setText(QString("Dither"));

	// create slider
	m_slider = new QSlider(Qt::Horizontal, m_ctrlGrp);
	m_slider->setTickPosition(QSlider::TicksBelow);
```

```
        m_slider->setTickInterval(25);
        m_slider->setMinimum(1);
        m_slider->setMaximum(MXGRAY);
        m_slider->setValue  (4);

        // create spinbox
        m_spinBox = new QSpinBox(m_ctrlGrp);
        m_spinBox->setMinimum(1);
        m_spinBox->setMaximum(MXGRAY);
        m_spinBox->setValue  (4);

        //create checkbox for dither
        m_checkBox = new QCheckBox(m_ctrlGrp);


        // init signal/slot connections for Quantization
        connect(m_slider , SIGNAL(valueChanged(int)), this, SLOT(changeQuan (int)));
        connect(m_spinBox, SIGNAL(valueChanged(int)), this, SLOT(changeQuan (int)));

        // assemble dialog
        QGridLayout *layout = new QGridLayout;
        layout->addWidget(  label  , 0, 0);
        layout->addWidget(m_slider , 0, 1);
        layout->addWidget(m_spinBox, 0, 2);
        layout->addWidget(m_checkBox, 1, 1);
        layout->addWidget(dlabel, 1, 0);

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

        return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Quantization::changeQuan:
//
// Slot to process change in quan caused by moving the slider.
//
void
Quantization::changeQuan(int quan)
{
        m_slider ->blockSignals(true);
        m_slider ->setValue    (quan );
        m_slider ->blockSignals(false);
        m_spinBox->blockSignals(true);
        m_spinBox->setValue    (quan );
        m_spinBox->blockSignals(false);

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Quantization::quantization:
//
//Quantization I1 using the 2-level mapping shown below.  Output is in I2.
//! \brief To quantize we add or subtract a bias value from each pixel
//! \details            Add dither to pixel values if dither = 1 else quantize without dither
//! \param[in]      I1  - Input image.
//! \param[in]      quan - Quantization.
//! \param[in]      dither - dither.
//! \param[out]     I2  - Output image.
//
void
Quantization::quantization(ImagePtr I1, int quan, int dither, ImagePtr I2) {
        IP_copyImageHeader(I1, I2);
        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // variable for quantization levels
        int levels = quan;

        // scale is the value to add or subtract from each pixel
 int scale = (MXGRAY + 1) / levels;

        // bias brings the scale down by a factor of 2
        // will add bias to each point
        double bias = scale/2.0;

        // compute lut[]
        int i, lut[MXGRAY];
 for(i=0;  i<=MXGRAY; ++i)
                lut[i] = scale * (int) (i/scale) + bias;

        int pixel = 0;
        // int noise is the dither noise to add to each pixel
        // int sign is sign of dither noise. It tells if to add or subtract noise
        // on odd row always add negative noise, on even row always add positive noise
        int noise, sign;

 int type;
        ChannelPtr<uchar> p1, p2, endd;

        // check if dither checkbox is checked or not
        // if not checked copy values from lut to I2
 if (!dither) {
           for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
                IP_getChannel(I2, ch, p2, type);
                for(endd = p1 + total; p1<endd;) *p2++ = lut[*p1++];
           }
        }

        // else if dither is checked, apply dither to each pixel value
 else {
```

```
        for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
                IP_getChannel(I2, ch, p2, type);
                for (int y=0; y<h; y++)
                {
                        // if row is odd, intiialize sign to 1
                        if (y % 2)
                                sign = 1;

                        // else if row is even, intiialize sign to -1
                        else
                                sign = -1;

                        for (int x=0; x<w; x++)
                          {
                                // 2^5 = 32767  // gives a number b/w 0-1
                                noise = ((rand()&0x7fff) / 32767.) * bias;

                                // alternating the noise addition or subtraction
                                switch(sign)
                                {
                                // on odd row adding negative value
                                        case 1:
                                                // adding noise to pixel
                                                pixel = *p1++ + noise;
                                                sign = -1;
                                                break;
                                // on even row adding positive value
                                        case -1:
                                                // subtracting noise form pixel
                                                pixel = *p1++ - noise;
                                                sign = 1;
                                                break;
                                }

                               // purpose of clipping is to make sure output pixel value does not goes off range
                               // clipping the pixel value after applying the dither and before copying to output image
                               *p2++ = lut[ CLIP(pixel, 0, MaxGray)];
                               }
                }
        }
    }
}

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Quantization::reset:
//
// Reset parameters.
//
void
Quantization::reset() {}
```

HistogramStretching.h

```
// ============================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// HistogramStretching.h - Histogram Stretching widget
//
// Written by: Khadeeja Din, 2016
// ============================================================================



#ifndef HISTOGRAMSTRETCHING_H
#define HISTOGRAMSTRETCHING_H

#include "ImageFilter.h"

class HistogramStretching : public ImageFilter {
        Q_OBJECT

public:
        HistogramStretching        (QWidget *parent = 0);  // constructor
        QGroupBox*        controlPanel      ();                          // create control panel
        bool              applyFilter(ImagePtr, ImagePtr);    // apply filter to input to init output
        void              reset             ();                                  // reset parameters

protected:
        void histogramstretching(ImagePtr I1, int min, int max, ImagePtr I2);

protected slots:
  void    changeMin (int);
  void    changeMax (int);

private:
        // histogramstretching controls
        QSlider           *m_sliderMin ;  // Min slider
        QSlider           *m_sliderMax ; // Max slider
        QSpinBox          *m_spinBoxMin;          // Min spinbox
        QSpinBox          *m_spinBoxMax;          // Max spinbox
        QCheckBox         *m_checkBoxMin;         // Minautovalue checkbox
  QCheckBox     *m_checkBoxMax;         // Maxautovalue checkbox

        // labels for histogramstretching
        QLabel            *m_labelMin;    // Min label
        QLabel            *m_labelMax;    // Max label

        // widgets and groupbox
        QGroupBox         *m_ctrlGrp;          // Groupbox for panel
```

```
};

#endif                  // HISTOGRAMSTRETCHING_H
```

HistogramStretching.cpp

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// HistogramStretching.cpp - Histogram Stretching widget.
//
// Written by: Khadeeja Din, 2016
// ========================================================================


#include "MainWindow.h"
#include "HistogramStretching.h"

extern MainWindow *g_mainWindowP;



// Min and Max values for sliders
// initializing Histogram array of 256 entries
int Min = 0;
int Max = MaxGray;
int Histogram[MXGRAY];



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::HistogramStretching:
//
// Constructor.
//
HistogramStretching::HistogramStretching(QWidget *parent) : ImageFilter(parent)
{}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
```

```
// Return 1 for success, 0 for failure.
//
bool
HistogramStretching::applyFilter(ImagePtr I1, ImagePtr I2)
{
        // error checking
        if (I1.isNull()) return 0;

        // initializing min and max values for finding the min and max pixel intensity in the input image
        int min = 0;
        int max = MaxGray;

        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // initializing Histogram with all 0 entries
        int i = 0;
        for (i = 0; i<MXGRAY; i++)
         {Histogram[i] = 0;}

         // reading input pixels values and storing their frequencies in Histogram
         // p1 is a pointer that points to current pixel in I1. p1++ is pointing to next pixel in I1
  // p2 is a pointer that points to current pixel in I2. p2++ is pointing to next pixel in I2
         int type;
         ChannelPtr<uchar> p1, p2, endd;
         for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
           for(endd = p1 + total; p1<endd; p1++)
          Histogram[*p1] ++;
         }

  // check if auto checkboxes are checked
  int minautovalue = m_checkBoxMin->isChecked();
  int maxautovalue = m_checkBoxMax->isChecked();

        // minimum and maximum histogram stretch variables
  int minstretch, maxstretch;

        // get min and max values from siders
        minstretch = m_sliderMin->value();
        maxstretch = m_sliderMax->value();

        // if minautovalue is checked then read minimum pixel intensity from image
        // reading first non zero value from left of image and copying it to minimum
        // change values for slider and spinbox to minimum value of image
        if (minautovalue)
        {
           int minimum = min, i =0;
           for (i=0; i<MXGRAY; i++)
           { if (!Histogram[i]) continue;
                                minimum = i;
```

```
                        break; }
            minstretch = minimum;
            m_sliderMin->setValue (minimum);
            m_spinBoxMin->setValue (minimum);
        }

        // if maxautovalue is checked then read maximum pixel intensity from image
        // reading first non zero value from right of image and copying it to maximum
        // change values for slider and spinbox to maximum value of image
        if (maxautovalue)
        {
            int maximum = MaxGray;
            int i = 0;
            for (i=MaxGray; i>= 0; i--)
            { if (!Histogram[i]) continue;
                        maximum = i;
                        break; }
            maxstretch = maximum;
            m_sliderMax->setValue  (maximum);
            m_spinBoxMax->setValue (maximum);
        }

        // error checking
  if ((minstretch < min || minstretch > max) || (maxstretch < min || maxstretch > max)) return 0;

        // checking that minimum value from slider is atleast 1 less than maximum value from slider
  if (minstretch >= maxstretch)
     {maxstretch = minstretch + 1;}

        // minstretch and maxstretch are the minimum or maximum pixel values from either the slider or
image appropriately
        // apply filter
  histogramstretching(I1, minstretch, maxstretch, I2);
        return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::createGroupBox:
//
// Create group box for control panel.
//
QGroupBox*
HistogramStretching::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("HistogramStretching");

        // init widgets
        // create label[i] for Minimum
```

```
QLabel *labelMin = new QLabel;
labelMin->setText(QString("Min"));

// create label[i] for Maximum
QLabel *labelMax = new QLabel;
labelMax->setText(QString("Max"));

// create Min slider
m_sliderMin = new QSlider(Qt::Horizontal, m_ctrlGrp);
m_sliderMin->setTickPosition(QSlider::TicksBelow);
m_sliderMin->setTickInterval(25);
m_sliderMin->setMinimum(Min);
m_sliderMin->setMaximum(Max);
m_sliderMin->setValue (0);

// create Min spinbox
m_spinBoxMin = new QSpinBox(m_ctrlGrp);
m_spinBoxMin->setMinimum(Min);
m_spinBoxMin->setMaximum(Max);
m_spinBoxMin->setValue  (0);

// create Minautovalue checkbox
m_checkBoxMin = new QCheckBox(m_ctrlGrp);

// create Max slider
m_sliderMax = new QSlider(Qt::Horizontal, m_ctrlGrp);
m_sliderMax->setTickPosition(QSlider::TicksBelow);
m_sliderMax->setTickInterval(25);
m_sliderMax->setMinimum(Min);
m_sliderMax->setMaximum(Max);
m_sliderMax->setValue  (MaxGray);

// create Max spinbox
m_spinBoxMax = new QSpinBox(m_ctrlGrp);
m_spinBoxMax->setMinimum(Min);
m_spinBoxMax->setMaximum(Max);
m_spinBoxMax->setValue  (MaxGray);

// create Maxautovalue checkbox
m_checkBoxMax = new QCheckBox(m_ctrlGrp);

// init signal/slot connections for Min
connect(m_sliderMin , SIGNAL(valueChanged(int)), this, SLOT(changeMin (int)));
connect(m_spinBoxMin, SIGNAL(valueChanged(int)), this, SLOT(changeMin (int)));
connect(m_checkBoxMin, SIGNAL(stateChanged(int)), this, SLOT(changeMin (int)));

// init signal/slot connections for Max
connect(m_sliderMax , SIGNAL(valueChanged(int)), this, SLOT(changeMax (int)));
connect(m_spinBoxMax, SIGNAL(valueChanged(int)), this, SLOT(changeMax (int)));
connect(m_checkBoxMax, SIGNAL(stateChanged(int)), this, SLOT(changeMax (int)));
```

```
	//assemble dialog
	QGridLayout *layout = new QGridLayout;
	layout->addWidget(	labelMin		, 0, 0);
	layout->addWidget(	labelMax		, 1, 0);
	layout->addWidget(m_sliderMin , 0, 1);
	layout->addWidget(m_checkBoxMin, 0, 3);
	layout->addWidget(m_spinBoxMin, 0, 2);
	layout->addWidget(m_sliderMax , 1, 1);
	layout->addWidget(m_checkBoxMax, 1, 3);
	layout->addWidget(m_spinBoxMax, 1, 2);

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

	return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::changeMin
//
// Slot to process change in Min caused by moving the slider.
//
void HistogramStretching::changeMin(int m)
{
	m_sliderMin ->blockSignals(true);
	m_sliderMin ->setValue		(m	);
	m_sliderMin ->blockSignals(false);
	m_spinBoxMin->blockSignals(true);
	m_spinBoxMin->setValue		(m	);
	m_spinBoxMin->blockSignals(false);


// apply filter to source image; save result in destination Image
applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

// display Output
g_mainWindowP->displayOut();

}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::changeMax
//
// Slot to process change in Max caused by moving the slider.
//
void
HistogramStretching::changeMax(int m)
```

```
{
	m_sliderMax ->blockSignals(true);
	m_sliderMax ->setValue    (m   );
	m_sliderMax ->blockSignals(false);
	m_spinBoxMax->blockSignals(true);
	m_spinBoxMax->setValue    (m   );
	m_spinBoxMax->blockSignals(false);

	// apply filter to source image; save result in destination image
	applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

	// display output
	g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::histogramstretching:
//
// HistogramStretching I1 using the 2-level mapping shown below.  Output is in I2.
//! \brief		Mapping min-max to 0-255
//! \param[in]	I1  - Input image.
//! \param[in]	min - Minimum.
//! \param[in]	max - Maximum.
//! \param[out]	I2  - Output image.
//
void
HistogramStretching::histogramstretching(ImagePtr I1, int min, int max, ImagePtr I2)
{
	IP_copyImageHeader(I1, I2);
	int w = I1->width();
	int h = I1->height();
	int total = w * h;

	// compute lut[]
	// initialized lut of 256 entries
	// 1. subtract min from every pixel
	// 2. scale to [0, 1]
	// 3. map to [0, 255] range
	int i;
	for(i=0; i<MXGRAY; ++i)
	{Histogram[i] = CLIP((int)(MaxGray*(i- min)) / (max - min), 0, MaxGray) ;}


	// for each pixel intensity in I1, read its coresponding value from Histogram, and output it to I2
	// p1 is a pointer that points to current pixel in I1. p1++ is pointing to next pixel in I1
	// p2 is a pointer that points to current pixel in I2. p2++ is pointing to next pixel in I2
	int type;
	ChannelPtr<uchar> p1, p2, endd;
	for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
```

```
        IP_getChannel(I2, ch, p2, type);
        for(endd = p1 + total; p1<endd;) *p2++ = Histogram[*p1++];
        }
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramStretching::reset:
//
// Reset parameters.
//
void
HistogramStretching::reset() {}
```

HistogramMatching.h

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// HistogramMatching.h - HistogramMatching widget
//
// Written by: Khadeeja Din, 2016
// ========================================================================



#ifndef HISTOGRAMMATCHING_H
#define HISTOGRAMMATCHING_H

#include "ImageFilter.h"

class HistogramMatching : public ImageFilter {
        Q_OBJECT

public:
        HistogramMatching       (QWidget *parent = 0);          // constructor
        QGroupBox*      controlPanel    ();                     // create control panel
        bool            applyFilter(ImagePtr, ImagePtr);        // apply filter to input to init output
        void            reset           ();                     // reset parameters

protected:
        void histogrammatching(ImagePtr I1, int n, ImagePtr I2);
```

```
protected slots:
        void changeN(int);

private:

        QSlider         *m_sliderN ;                    // slider to read n
        QSpinBox        *m_spinBoxN;                    // spinbox to read n
        QLabel          *m_label;                   // label n
        QGroupBox       *m_ctrlGrp;
};

#endif                                  // HISTOGRAMMATCHING_H
```

HistogramMatching.cpp

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// HistogramMatching.cpp - HistogramMatching class
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#include "MainWindow.h"
#include "HistogramMatching.h"
#include <cmath>
#include <cstdio>

extern MainWindow *g_mainWindowP;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Min and Max for n range
int HMin = -100;
int HMax = 100;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramMatching::HistogramMatching:
//
// Constructor.
//
HistogramMatching::HistogramMatching(QWidget *parent) : ImageFilter(parent)
{}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramMatching::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
HistogramMatching::applyFilter(ImagePtr I1, ImagePtr I2)
{
        // error checking
        if (I1.isNull()) return 0;

        // get n value
        int n = m_sliderN->value();

        // error checking
        if(n < HMin || n > HMax) return 0;

        // apply filter
  histogrammatching(I1, n, I2);

        return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramMatching::createGroupBox:
//
// Create group box for control panel.
//
QGroupBox*
HistogramMatching::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("HistogramMatching");

        // init widgets
        // create label[i] for n
        QLabel *label = new QLabel;
        label->setText(QString("n"));


        // create slider for n
        m_sliderN = new QSlider(Qt::Horizontal, m_ctrlGrp);
        m_sliderN->setTickPosition(QSlider::TicksBelow);
        m_sliderN->setTickInterval(25);
        m_sliderN->setMinimum(HMin);
        m_sliderN->setMaximum(HMax);
        m_sliderN->setValue  (0);
```

```
        // create spinbox for n
        m_spinBoxN = new QSpinBox(m_ctrlGrp);
        m_spinBoxN->setMinimum(HMin);
        m_spinBoxN->setMaximum(HMax);
        m_spinBoxN->setValue  (0);

        // init signal/slot connections for HistogramMatching
        connect(m_sliderN , SIGNAL(valueChanged(int)), this, SLOT(changeN (int)));
        connect(m_spinBoxN, SIGNAL(valueChanged(int)), this, SLOT(changeN (int)));

        // assemble dialog
        QGridLayout *layout = new QGridLayout;
        layout->addWidget(  label  , 0, 0);
        layout->addWidget(m_sliderN , 0, 1);
        layout->addWidget(m_spinBoxN, 0, 2);

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

        return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramMatching::changeN:
//
// Slot to process change in n caused by moving the slider.
//
void
HistogramMatching::changeN(int n)
{
        m_sliderN ->blockSignals(true);
        m_sliderN ->setValue    (n );
        m_sliderN ->blockSignals(false);
        m_spinBoxN->blockSignals(true);
        m_spinBoxN->setValue    (n );
        m_spinBoxN->blockSignals(false);

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

```
// HistogramMatching::HistogramMatching:
//
// HistogramMatching. Output is in I2.
//! \brief          Mapping image to specified histogram.
//! \param[in]    I1  - Input image.
//! \param[in]    n - n.
//! \param[out]   I2  - Output image.
//
void
HistogramMatching::histogrammatching(ImagePtr I1, int n, ImagePtr I2)
{

        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        IP_copyImageHeader(I1, I2);

        int left[MXGRAY], right[MXGRAY];
        int Hsum;
        double Havg, scale, temp;
        // output histogram
        int h1[MXGRAY];
        // target histogram
        int h2[MXGRAY];

        // clear histogram h1
        for (int i = 0; i<MXGRAY; i++)
          {h1[i] = 0;}

        // evaluate histogram h1
        int type;
        ChannelPtr<uchar> p1, p2, endd; //p1 is a pointer that points to pixel. p1++ is pointing to next pixel
         for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++) {
           for(endd = p1 + total; p1<endd; p1++)
              { h1[*p1] ++; }
         }

           double average =  (double)total / MXGRAY;

           // If n = 0 : histogram equalization
           if (n == 0) {
                   for (int i = 0; i < MXGRAY - 1; ++i) {
                           h2[i] = (int)average;
                   }
                   h2[MXGRAY - 1] = total - (int) average*(MXGRAY - 1);
           }

           // if n > 0 exponentially increasing histogram
           else if (n > 0) {
                   for (int i=Havg = 0; i <MXGRAY; ++i) {
```

```
                temp = ROUND(pow((double) i/MXGRAY, (double) n) * MXGRAY);
                h2[i] = temp;
                Havg += h2[i];
        }
        scale = (double) total / Havg;
        if (scale != 1) {
                for (int i = 0; i < MXGRAY; ++i)
                        h2[i] *= scale;
        }
}

// if n <0 exponentially decreasing histogram
else if (n < 0) {
        Havg = 0;
        for (int i =0; i < MXGRAY; ++i) {
                temp = ROUND(pow(1 - (double) i/MXGRAY, (double) abs(n)) * MXGRAY);
                h2[i] =  temp;
                Havg += h2[i];
        }
        scale = (double) total / Havg;
        if (scale != 1) {
                for (int i = 0; i < MXGRAY; ++i)
                        h2[i] *= scale;
        }
}

int R = 0;
Hsum = 0;
int p;

// evaluate mapping of all input gray levels.
// each input gray value maps to an interval of valid output values.
// The endpoints of the intervals are left[] and right[]
for (int i =0; i < MXGRAY; ++i) {
        // left end of interval
        left[i] = R;
        // cumulative value for interval
        Hsum += h1[i];
        // compute width of interval in R and adjust Hsum ad the interval widens
        while (Hsum > h2[R] && R< (MXGRAY - 1)) {
                Hsum -= h2[R];
                R++;
        }
        // initializa right end of interval
        right[i] = R;
}

// clear h1 and reuse it below
for (int i =0; i < MXGRAY; ++i) {
        h1[i] = 0;
}
```

```
        // visit all input pixels and output the transformed pixel to I2
        for(int ch = 0; IP_getChannel(I1, ch, p1, type); ch++)
        {
                IP_getChannel(I2, ch, p2, type);
                for(endd = p1 + total; p1<endd; p1++)
          {
            p = left[*p1];
            if (h1[p] < h2[p])
              {
               *p2++ = p;
              }

            else
              {
               p = left[*p1] = MIN(p+1, right[*p1]);
                                                *p2++ = p;
              }
            h1[p]++;
          }
        }
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// HistogramMatching::reset:
//
// Reset parameters.
//
void
HistogramMatching::reset() {}
```

Blur.h

```
// ==========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Blur.h - Blur widget
//
// Written by: Khadeeja Din, 2016
// ==========================================================================

#ifndef BLUR_H
#define BLUR_H
```

```
#include "ImageFilter.h"

class Blur : public ImageFilter {
	Q_OBJECT

public:
	Blur		(QWidget *parent = 0);		// constructor
	QGroupBox*	controlPanel	();				// create control panel
	bool		applyFilter(ImagePtr, ImagePtr);	// apply filter to input to init output
	void		reset		();				// reset parameters

protected:
	void blur(ImagePtr I1, int xsz, int ysz, ImagePtr I2);
	template <class T>
	void IP_blur1D (ChannelPtr<T> src, int len, int stride, double ww, ChannelPtr<T> dst);

protected slots:
	void	changeWidth (int);
	void	changeHeight (int);
	void	changeBoth (int);

private:
	// blur controls
	QSlider		*m_sliderW ;	// filter width slider
	QSlider		*m_sliderH ;	// filter height slider
	QSpinBox	*m_spinBoxW;	// filter width spinbox
	QSpinBox	*m_spinBoxH;	// filter height spinbox
	QCheckBox *m_checkBox;	// checkbox for sz * sz filter

	// widgets and groupbox
	QGroupBox	*m_ctrlGrp;	// groupbox for panel
};

#endif	// BLUR_H
```

Blur.cpp

```
// ============================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Blur.cpp - Blur class
//
// Written by: Khadeeja Din, 2016
// ============================================================================
```

```
#include "MainWindow.h"
#include "Blur.h"

extern MainWindow *g_mainWindowP;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
	// min and max variables for filter size range
int minfilter = 1;
int maxfilter = 99;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::Blur
//
// Constructor.

Blur::Blur(QWidget *parent) : ImageFilter(parent)
{}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
Blur::applyFilter(ImagePtr I1, ImagePtr I2)
{
	// error checking
	if(I1.isNull()) return 0;

	// get filter width
	int xsz = m_sliderW->value();

	// get filter height
	int ysz = m_sliderH->value();

	// error checking
	if (xsz < minfilter || xsz > maxfilter || ysz < minfilter || ysz > maxfilter)
	   return 0;

	// apply filter
	blur (I1, xsz, ysz, I2);
	return 1;
}
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::controlPanel:
//
// Create group box for control panel.
//
QGroupBox*
Blur::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("Blur");

        // init widgets
        // create label[i] for width
        QLabel *wlabel = new QLabel;
        wlabel->setText(QString("Width"));

        // create label[i] for height
 QLabel *hlabel = new QLabel;
 hlabel->setText(QString("Height"));

        // create width slider
        m_sliderW = new QSlider(Qt::Horizontal, m_ctrlGrp);
        m_sliderW->setTickPosition(QSlider::TicksBelow);
        m_sliderW->setTickInterval(25);
        m_sliderW ->setSingleStep(2);
 m_sliderW->setMinimum(minfilter);
        m_sliderW->setMaximum(maxfilter);
        m_sliderW->setValue  (minfilter);

        // create width spinbox
        m_spinBoxW = new QSpinBox(m_ctrlGrp);
        m_spinBoxW->setMinimum(minfilter);
        m_spinBoxW->setMaximum(maxfilter);
        m_spinBoxW->setValue  (minfilter);
        m_spinBoxW->setSingleStep(2);

        // create height slider
 m_sliderH = new QSlider(Qt::Horizontal, m_ctrlGrp);
 m_sliderH->setTickPosition(QSlider::TicksBelow);
 m_sliderH->setTickInterval(25);
        m_sliderH ->setSingleStep(2);
 m_sliderH->setMinimum(minfilter);
 m_sliderH->setMaximum(maxfilter);
 m_sliderH->setValue  (minfilter);

 // create height spinbox
 m_spinBoxH = new QSpinBox(m_ctrlGrp);
 m_spinBoxH->setMinimum(minfilter);
 m_spinBoxH->setMaximum(maxfilter);
 m_spinBoxH->setValue  (minfilter);
        m_spinBoxH->setSingleStep(2);
```

```
        // create checkbox
        m_checkBox = new QCheckBox(m_ctrlGrp);
        m_checkBox->setChecked(true);

        // init signal/slot connections for Width
        connect(m_sliderW , SIGNAL(valueChanged(int)), this, SLOT(changeWidth (int)));
        connect(m_spinBoxW, SIGNAL(valueChanged(int)), this, SLOT(changeWidth (int)));

        // init signal/slot connections for Height
  connect(m_sliderH , SIGNAL(valueChanged(int)), this, SLOT(changeHeight (int)));
  connect(m_spinBoxH, SIGNAL(valueChanged(int)), this, SLOT(changeHeight (int)));

        // init signal/slot connections for checkbox
        connect(m_checkBox, SIGNAL(stateChanged(int)), this, SLOT(changeBoth (int)));

        // assemble dialog
  QGridLayout *layout = new QGridLayout;
  layout->addWidget(     wlabel              , 0, 0);
  layout->addWidget(     hlabel              , 1, 0);
  layout->addWidget(m_sliderW , 0, 1);
  layout->addWidget(m_spinBoxW, 0, 2);
  layout->addWidget(m_sliderH , 1, 1);
  layout->addWidget(m_spinBoxH, 1, 2);
        layout->addWidget(m_checkBox, 1, 3);

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

        return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::changeWidth:
//
// Slot to process change in width caused by moving the slider.
// only dealing with odd size filter boxes
//
void
Blur::changeWidth(int value)
{
        // if width value from slider is even add 1 to make it odd
  value += !(value %2);
        m_sliderW ->blockSignals(true);
        m_sliderW ->setValue    (value);
        m_sliderW ->blockSignals(false);
        m_spinBoxW->blockSignals(true);
        m_spinBoxW->setValue    (value);
        m_spinBoxW->blockSignals(false);
```

```
        // if checkbox is checked then set the height slider to same value as width slider
        if(m_checkBox->isChecked())
        {
           m_sliderH ->  setValue(value);
        }

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::changeHeight:
//
// Slot to process change in height caused by moving the slider.
// only dealing with odd size filter boxes
//
void
Blur::changeHeight(int value)
{
        // if height value from slider is even add 1 to make it odd
  value += !(value %2);
        m_sliderH ->blockSignals(true);
        m_sliderH ->setValue    (value);
        m_spinBoxH->blockSignals(true);
        m_spinBoxH->setValue    (value);
        m_spinBoxH->blockSignals(false);
        m_sliderH ->blockSignals(false);

        // if checkbox is checked then set the width slider to same value as height slider
        if (m_checkBox->isChecked())
  {
           m_sliderW ->  setValue(value);
  }

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::changeBoth:
```

```
//
// Slot to process change in height and width equally.
//
void
Blur::changeBoth(int value)
{
        // checks if the checkbox is checked
        if (value)
        {
           // copy the value from width slider to height slider
           m_sliderH->setValue (m_sliderW->value());
        }

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}




// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::blur:
//
//! \brief    Blur sends all rows first to IP_blur1D then it sends all columns to IP_blur1D
//! \details          First apply blur horizontally and then apply blur vertically and output to I2.
//! \param[in]     I1  - Input image.
//! \param[in]     xsz - filter Width.
//! \param[in]     ysz - filter Height.
//! \param[out]   I2  - Output image.
//
void
Blur::blur(ImagePtr I1, int xsz, int ysz, ImagePtr I2) {
        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // error checking
        // If window width is greater than image width than we divide it by 2 and subtract 1 to make it odd
        if (xsz > w) {
           IP_copyImage(I1, I2);
           return;
        }

        // If window height is greater than image height than we divide it by 2 and subtract 1 to make it odd
        if (ysz > h) {
           IP_copyImage(I1, I2);
           return;
        }
```

```
// trivial case:
// if Width and Height are 1 the window size is 0 and no blurring needs to be done
// we simple copy input image to output image
if (xsz <= 1 && ysz<= 1){
   if (I1 != I2)
            IP_copyImage(I1, I2);
   return;
}

IP_copyImageHeader(I1, I2);
int type;
ChannelPtr<uchar> src, dst;
ChannelPtr<float> fsrc, fdst;

for(int ch = 0; IP_getChannel(I1, ch, src, type); ch++) {
   // type is uchar pixel
   if (type == UCHAR_TYPE) {
            if (xsz > 1.) {
                     dst = I2[ch];
                     // process all rows first
                     for (int y =0; y<h; y++) {
                              // send one row at a time to IP_blur1D
                              IP_blur1D(src, w, 1, xsz, dst);
                              src += w;
                              dst += w;
                     }
                     src = I2[ch];
            }

    if (ysz > 1.) {
                     dst = I2[ch];
                     // process all columns second
                     for (int x =0; x<w; x++) {
                              // send one column at a time to IP_blur1D
                              IP_blur1D(src, h, w, ysz, dst);
                              src += 1;
                              dst += 1;
                     }
            }
   }

   // float type from image copying
   else {
            IP_castChannel(I1, ch, I2, ch, FLOAT_TYPE);
            if (xsz > 1.) {
                     fdst = I2[ch];
                     fsrc = I2[ch];
                     // process all rows first
                     for (int y =0; y<h; y++) {
                              // send one row at a time to IP_blur1D
```

```
                                IP_blur1D(fsrc, w, 1, xsz, fdst);
                                fsrc += w;
                                fdst += w;
                        }
                        fsrc = I2[ch];
                }

                if (ysz > 1.) {
                        fdst = I2[ch];
                        // process all columns second
                        for (int x =0; x<w; x++) {
                                // send one column at a time to IP_blur1D
                                IP_blur1D(fsrc, h, w, ysz, fdst);
                                fsrc += 1;
                                fdst += 1;
                        }
                }
        }
    }
}



//IP_blur1D applies blur in 1 direction only
//len is the length of the Image
//stride is the distance from 1 element to next
//ww is the width of the filter
//src is the pointer to input Image
//dst is the pointer to output Image
template <class T>
void
Blur::IP_blur1D(ChannelPtr<T> src, int len, int stride, double ww, ChannelPtr<T> dst) {

    // buffer size is length of image + filter width - 1
    size_t buf_size = len + ww -1;

    // padding is the extra spaces to pad around the image
    int padding = (ww - 1)/2;

    // error checking
    if (ww > len) {
       return;
    }

    // trivial  case
    if (ww <= 1) {
       if (src != dst) {
                for (int i =0; i<len; i++) {
                        *dst = *src;
                        dst += stride;
```

```
                    src += stride;
                }
    }
    return;
}

// creating buffer
uint16_t *buffer;
buffer = (uint16_t*)malloc(sizeof(uint16_t*) * buf_size);
if (buffer == NULL) {
    exit(0);
}

int i = 0;
// filling up left padded spaces
for (; i<padding; ++i) {
    buffer[i] = (*src);
}

// incrementing length of image by left padding
len += i;

// reading the remaining pixels in row from src to buffer
for (; i<len; ++i) {
    buffer[i] = (*src);
    src += stride;
}

// decrementing src pixel back to the last pixel in the current row.
src -= stride;

// filling up right padded spaces
padding += i;
for (; i<padding; ++i) {
    buffer[i] = (*src);
    }

 // SUM //
 // initialize sum to 0.0
 double sum = 0;
 int j = 0;

 // for each pixel in filter width add its value to sum
 for (; j < ww; j++) {
     sum += buffer[j];
 }

 // output sum/ww to dst(current output pixel)
 *dst = sum/ww ;

 // move output image pointer to next pixel
```

```
        dst += stride;

        // process all the pixels remaining in the row
        // move filter along the row by removing the left most pixel in filter and adding the next pixel after
the filter
        for (; j< buf_size; j++, dst += stride) {
           int last = j - ww;
           sum += (buffer[j] - buffer[last]);
           *dst = sum/ww;
        }

        // empty the buffer after 1 row is processed
        free (buffer);
}




// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Blur::reset:
//
// Reset parameters.
//
void
Blur::reset() {}
```

Sharpen.h

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Sharpen.h - Sharpen widget
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#ifndef SHARPEN_H
#define SHARPEN_H

#include "ImageFilter.h"

class Sharpen : public ImageFilter {
        Q_OBJECT

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

```
        QGroupBox*      controlPanel    ();                         // create control panel
        bool            applyFilter(ImagePtr, ImagePtr);    // apply filter to input to init output
        void            reset           ();                         // reset parameters

protected:
  void sharpblur(ImagePtr I1, int xsz, int ysz, ImagePtr I2);
        template <class T>
        void sharpIP_blur1D (ChannelPtr<T> src, int len, int stride, double ww, ChannelPtr<T> dst);
        void sharpen(ImagePtr I1, int sz, double fctr, ImagePtr I2);

protected slots:
        void    changeSize (int);
        void    changeFactor (int);

private:
        // blur and sharpen sliders
        QSlider         *m_sliders ;    // filter size slider
        QSlider         *m_sliderf ;    // sharpen factor slider
        QSpinBox        *m_spinBoxs;    // filter size spin box
        QSpinBox        *m_spinBoxf;    // sharpen factor spin box

        // widgets and groupbox
        QGroupBox       *m_ctrlGrp;     // groupbox for panel
};

#endif   // SHARPEN_H
```

Sharpen.cpp

```
// ============================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Sharpen.cpp - Sharpen class
//
// Written by: Khadeeja Din, 2016
// ============================================================================

#include "MainWindow.h"
#include "Sharpen.h"

extern MainWindow *g_mainWindowP;


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
        // min and max variables for filter size and factor range
int s_min = 1;
int s_max = 99;
int f_max = 5;
```

```
// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::Sharpen:
//
// Constructor.

Sharpen::Sharpen(QWidget *parent) : ImageFilter(parent)
{
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
Sharpen::applyFilter(ImagePtr I1, ImagePtr I2)
{
        // error checking
        if(I1.isNull()) return 0;

        // get filter size
        int size = m_sliders->value();

        // get factor value
        double factor = m_sliderf->value();

        // error checking
        if(size < s_min || size > s_max) return 0;

        // apply filter
        sharpen(I1, size, factor, I2);
        return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::controlPanel:
//
// Create group box for control panel.
//
QGroupBox*
Sharpen::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("Sharpen");
```

```
        // init widgets
        // create label[i] for size
        QLabel *labels = new QLabel;
        labels->setText(QString("Size"));

        // create label[i] for factor
QLabel *labelf = new QLabel;
labelf->setText(QString("Factor"));

        // create size slider
        m_sliders = new QSlider(Qt::Horizontal, m_ctrlGrp);
        m_sliders->setTickPosition(QSlider::TicksBelow);
        m_sliders->setTickInterval(25);
        m_sliders->setSingleStep(2);
        m_sliders->setMinimum(s_min);
        m_sliders->setMaximum(s_max);
        m_sliders->setValue  (s_min);

        // create size spinbox
m_spinBoxs = new QSpinBox(m_ctrlGrp);
        m_spinBoxs->setSingleStep(2);
        m_spinBoxs->setMinimum(s_min);
        m_spinBoxs->setMaximum(s_max);
        m_spinBoxs->setValue  (s_min);

        // create factor slider
m_sliderf = new QSlider(Qt::Horizontal, m_ctrlGrp);
m_sliderf->setTickPosition(QSlider::TicksBelow);
m_sliderf->setTickInterval(2);
        m_sliderf->setSingleStep(2);
m_sliderf->setMinimum(s_min);
m_sliderf->setMaximum(f_max);
m_sliderf->setValue  (s_min);

        // create factor spinbox
m_spinBoxf = new QSpinBox(m_ctrlGrp);
        m_spinBoxf->setSingleStep(2);
m_spinBoxf->setMinimum(s_min);
m_spinBoxf->setMaximum(f_max);
m_spinBoxf->setValue (s_min);


        // init signal/slot connections for size
        connect(m_sliders , SIGNAL(valueChanged(int)), this, SLOT(changeSize (int)));
        connect(m_spinBoxs, SIGNAL(valueChanged(int)), this, SLOT(changeSize (int)));

        // init signal/slot connections for factor
connect(m_sliderf , SIGNAL(valueChanged(int)), this, SLOT(changeFactor (int)));
connect(m_spinBoxf, SIGNAL(valueChanged(int)), this, SLOT(changeFactor (int)));
```

```
	// assemble dialog
	QGridLayout *layout = new QGridLayout;
  layout->addWidget(     labels                 , 0, 0);
  layout->addWidget(     labelf                 , 1, 0);
  layout->addWidget(m_sliders , 0, 1);
  layout->addWidget(m_spinBoxs, 0, 2);
  layout->addWidget(m_sliderf , 1, 1);
  layout->addWidget(m_spinBoxf, 1, 2);

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

	return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::changeSize:
//
// Slot to process change in size caused by moving the slider.
//
void
Sharpen::changeSize(int value)
{
  value += !(value %2);
  m_sliders ->blockSignals(true);
  m_sliders ->setValue    (value);
  m_sliders ->blockSignals(false);
  m_spinBoxs->blockSignals(true);
  m_spinBoxs->setValue    (value);
  m_spinBoxs->blockSignals(false);

	// apply filter to source image; save result in destination image
	applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

	// display output
	g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::changeFactor:
//
// Slot to process change in factor caused by moving the slider.
//
void
Sharpen::changeFactor(int value)
{
  m_sliderf ->blockSignals(true);
```

```
  m_sliderf ->setValue    (value);
  m_sliderf ->blockSignals(false);
  m_spinBoxf->blockSignals(true);
  m_spinBoxf->setValue    (value);
  m_spinBoxf->blockSignals(false);

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::sharpen:
//
//! \brief    Sharpen blurs the input image and save it to temp
//! \details          Subtracts temp from I1 and output it to I2
//! \param[in]    I1  - Input image.
//! \param[in]    sz - filter size.
//! \param[in]    fctr - sharpen factor.
//! \param[out]   I2  - Output image.
//

void
Sharpen::sharpen(ImagePtr I1, int sz, double fctr, ImagePtr I2) {
        ImagePtr temp;
        IP_copyImageHeader(I1, temp);
        IP_copyImageHeader(I1, I2);
        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // send I1 to sharpblur. The blurred image is stored in temp
        sharpblur (I1, sz, sz, temp);

        // src is pointer to I1, tempPointer is pointer to temp, dst is pointer to I2
        int type;
  ChannelPtr<uchar> src, tempPointer, dst, end;
  for(int ch = 0; IP_getChannel(I1, ch, src, type); ch++) {
           IP_getChannel(temp, ch, tempPointer, type);
        IP_getChannel(I2, ch, dst, type);
        for(end = src + total; src<end;)
     {  *dst = CLIP(*src + CLIP((*src - *tempPointer), 0, 255) * fctr, 0, 255);
                ++src;
                ++tempPointer;
                ++dst;
           }
  }
```

```
}


void
Sharpen::sharpblur(ImagePtr I1, int xsz, int ysz, ImagePtr I2) {
        int w = I1->width();
        int h = I1->height();
        int total = w * h;

        // error checking
        // If window width is greater than image width than we divide it by 2 and subtract 1 to make it odd
        if (xsz > w) {
           IP_copyImage(I1, I2);
           return;
        }

        // If window height is greater than image height than we divide it by 2 and subtract 1 to make it odd
        if (ysz > h) {
           IP_copyImage(I1, I2);
           return;
        }

        // trivial case:
        // if Width and Height are 1 the window size is 0 and no blurring needs to be done
        // we simple copy input image to output image
        if (xsz <= 1 && ysz<= 1){
           if (I1 != I2)
                   IP_copyImage(I1, I2);
           return;
        }

        IP_copyImageHeader(I1, I2);
        int type;
        ChannelPtr<uchar> src, dst;
        ChannelPtr<float> fsrc, fdst;

        for(int ch = 0; IP_getChannel(I1, ch, src, type); ch++) {
           // type is uchar pixel
           if (type == UCHAR_TYPE) {
                   if (xsz > 1.) {
                           dst = I2[ch];
                           // process all rows first
                           for (int y =0; y<h; y++) {
                                   // send one row at a time to IP_blur1D
                                   sharpIP_blur1D(src, w, 1, xsz, dst);
                                   src += w;
                                   dst += w;
                           }
                           src = I2[ch];
                   }
```

```
            if (ysz > 1.) {
                        dst = I2[ch];
                        // process all columns second
                        for (int x =0; x<w; x++) {
                                // send one column at a time to IP_blur1D
                                sharpIP_blur1D(src, h, w, ysz, dst);
                                src += 1;
                                dst += 1;
                        }
                }
        }

        // float type from image copying
        else {
                IP_castChannel(I1, ch, I2, ch, FLOAT_TYPE);
                if (xsz > 1.) {
                        fdst = I2[ch];
                        fsrc = I2[ch];
                        // process all rows first
                        for (int y =0; y<h; y++) {
                                // send one row at a time to IP_blur1D
                                sharpIP_blur1D(fsrc, w, 1, xsz, fdst);
                                fsrc += w;
                                fdst += w;
                        }
                        fsrc = I2[ch];
                }

                if (ysz > 1.) {
                        fdst = I2[ch];
                        // process all columns second
                        for (int x =0; x<w; x++) {
                                sharpIP_blur1D(fsrc, h, w, ysz, fdst);
                                // send one column at a time to IP_blur1D
                                fsrc += 1;
                                fdst += 1;
                        }
                }
        }
    }
}

//sharpIP_blur1D applies blur in 1 direction only
//len is the length of the Image
//stride is the distance from 1 element to next
//ww is the width of the filter
//src is the pointer to input Image
//dst is the pointer to output Image
template <class T>
void
Sharpen::sharpIP_blur1D(ChannelPtr<T> src, int len, int stride, double ww, ChannelPtr<T> dst) {
```

```
// buffer size is length of image + filter width - 1
size_t buf_size = len + ww -1;

// padding is the extra spaces to pad around the image
int padding = (ww - 1)/2;

// error checking
if (ww > len) {
        return;
}

// trivial  case
if (ww <= 1) {
        if (src != dst) {
        for (int i =0; i<len; i++) {
                *dst = *src;
                dst += stride;
                src += stride;
        }
}
return;
}

// creating buffer
uint16_t *buffer;
buffer = (uint16_t*)malloc(sizeof(uint16_t*) * buf_size);
if (buffer == NULL) {
        exit(0);
}


int i = 0;
// filling up left padded spaces
for (; i<padding; ++i) {
        buffer[i] = (*src);
}

// incrementing length of image by left padding
len += i;

// reading the remaining pixels in row from src to buffer
for (; i<len; ++i) {
        buffer[i] = (*src);
        src += stride;
}

// decrementing src pixel back to the last pixel in the current row.
src -= stride;

// filling up right padded spaces
```

```
        padding += i;
        for (; i<padding; ++i) {
                buffer[i] = (*src);
                }

        // SUM //
        // initialize sum to 0.0
        double sum = 0;
        int j = 0;

        // for each pixel in filter width add its value to sum
        for (; j < ww; j++) {
                sum += buffer[j];
        }

        // output sum/ww to dst(current output pixel)
        *dst = sum/ww ;

        // move output image pointer to next pixel
        dst += stride;

        // process all the pixels remaining in the row
        // move filter along the row by removing the left most pixel in filter and adding the next pixel
after the filter
        for (; j< buf_size; j++, dst += stride) {
                int last = j - ww;
                sum += (buffer[j] - buffer[last]);
                *dst = sum/ww;
        }

        // empty the buffer after 1 row is processed
        free (buffer);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Sharpen::reset:
//
// Reset parameters.
//
void
Sharpen::reset() {}
```

Median.h

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Median.h - Median widget
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#ifndef MEDIAN_H
#define MEDIAN_H

#include "ImageFilter.h"

class Median : public ImageFilter {
	Q_OBJECT

public:
	Median	(QWidget *parent = 0);			// constructor
	QGroupBox*	controlPanel	();				// create control panel
	bool		applyFilter(ImagePtr, ImagePtr);	// apply filter to input to init output
	void		reset		();					// reset parameters

protected:
	void median(ImagePtr I1, int sz, int avg_nbrs, ImagePtr I2);

protected slots:
	void	changeSize (int);
	void	changeAvg_nbrs (int);

private:
	// median controls
	QSlider		*m_slidersz ;		// kernel size slider
	QSlider		*m_slideravg ;		// slider for no. of neighborhoods to average
	QSpinBox	*m_spinBoxsz;		// kernel size spinbox
	QSpinBox	*m_spinBoxavg;			// spin box for no. of neighborhoods to
average
	QLabel		*szlabel;
	QLabel		*avglabel;

	// widgets and groupbox
	QGroupBox	*m_ctrlGrp;	// groupbox for panel
};

#endif  // MEDIAN_H
```

Median.cpp

```
// ========================================================================
// IMPROC: Image Processing Software Package
// Copyright (C) 2016 by George Wolberg
//
// Median.cpp - Median class
//
// Written by: Khadeeja Din, 2016
// ========================================================================

#include "MainWindow.h"
#include "Median.h"

extern MainWindow *g_mainWindowP;



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
//

int s_minkernel = 1;
int s_maxkernel = 99;

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::Median
//
// Constructor.

Median::Median(QWidget *parent) : ImageFilter(parent)
{}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::applyFilter:
//
// Run filter on the image, transforming I1 to I2.
// Overrides ImageFilter::applyFilter().
// Return 1 for success, 0 for failure.
//
bool
Median::applyFilter(ImagePtr I1, ImagePtr I2)
{
        // error checking
        if(I1.isNull()) return 0;

        // get kernel size
        int size = m_slidersz->value();
```

```
        // get average neighborhoods count
        int avg_nbrs = m_slideravg->value();

        // max neighborhood that can be averaged
        int max_avg_nbrs = ((size * size) - 1) >> 1;

        // error checking
        if (size < s_minkernel || size > s_maxkernel || avg_nbrs < 0 || avg_nbrs > max_avg_nbrs)
           return 0;

        median (I1, size, avg_nbrs, I2);

        return 1;
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::controlPanel:
//
// Create group box for control panel.
//
QGroupBox*
Median::controlPanel()
{
        // init group box
        m_ctrlGrp = new QGroupBox("Median");

        // init widgets
        // create label[i] for size
        QLabel *szlabel = new QLabel;
        szlabel->setText(QString("Size"));

        // create label[i] for average numbers
   QLabel *avglabel = new QLabel;
   avglabel->setText(QString("Avg_nbrs"));

        // create slider for size
        m_slidersz = new QSlider(Qt::Horizontal, m_ctrlGrp);
        m_slidersz->setTickPosition(QSlider::TicksBelow);
        m_slidersz->setTickInterval(25);
        m_slidersz ->setSingleStep(2);
   m_slidersz->setMinimum(s_minkernel);
        m_slidersz->setMaximum(s_maxkernel);
        m_slidersz->setValue  (s_minkernel);

        // create spinbox for size
        m_spinBoxsz = new QSpinBox(m_ctrlGrp);
        m_spinBoxsz ->setSingleStep(2);
        m_spinBoxsz->setMinimum(s_minkernel);
```

```
        m_spinBoxsz->setMaximum(s_maxkernel);
        m_spinBoxsz->setValue  (s_minkernel);

        // create slider for average numbers
    m_slideravg = new QSlider(Qt::Horizontal, m_ctrlGrp);
    m_slideravg->setTickPosition(QSlider::TicksBelow);
    m_slideravg->setTickInterval(25);
        m_slideravg ->setSingleStep(2);
    m_slideravg->setMinimum(s_minkernel);
    m_slideravg->setMaximum(s_maxkernel);
    m_slideravg->setValue  (s_minkernel);

    // create spinbox for average numbers
    m_spinBoxavg = new QSpinBox(m_ctrlGrp);
        m_spinBoxavg ->setSingleStep(2);
    m_spinBoxavg->setMinimum(s_minkernel);
    m_spinBoxavg->setMaximum(s_maxkernel);
    m_spinBoxavg->setValue  (s_minkernel);


        // init signal/slot connections for kernel size
        connect(m_slidersz , SIGNAL(valueChanged(int)), this, SLOT(changeSize (int)));
        connect(m_spinBoxsz, SIGNAL(valueChanged(int)), this, SLOT(changeSize (int)));

        // init signal/slot connections for avg_nbrs
    connect(m_slideravg , SIGNAL(valueChanged(int)), this, SLOT(changeAvg_nbrs (int)));
    connect(m_spinBoxavg, SIGNAL(valueChanged(int)), this, SLOT(changeAvg_nbrs (int)));

        // assemble dialog
    QGridLayout *layout = new QGridLayout;
    layout->addWidget(   szlabel  , 0, 0);
    layout->addWidget(   avglabel, 1, 0);
    layout->addWidget(m_slidersz , 0, 1);
    layout->addWidget(m_spinBoxsz, 0, 2);
    layout->addWidget(m_slideravg , 1, 1);
    layout->addWidget(m_spinBoxavg, 1, 2);

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

        return(m_ctrlGrp);
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::changeSize:
//
// Slot to process change in kernel size
//
void
```

```
Median::changeSize(int value)
{
        // if kernel size value is even add 1 to make it odd
  value += !(value %2);

        m_slidersz ->blockSignals(true);
        m_slidersz ->setValue    (value);
        m_slidersz ->blockSignals(false);
        m_spinBoxsz->blockSignals(true);
        m_spinBoxsz->setValue    (value);
        m_spinBoxsz->blockSignals(false);

        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::changeAvg_nbrs:
//
// Slot to process change in avergae neighborhood numbers
//
void
Median::changeAvg_nbrs(int value)
{
        int sz = m_slidersz->value();


        m_slideravg ->blockSignals(true);
        m_slideravg ->setValue    (value);
        m_slideravg ->blockSignals(false);
        m_spinBoxavg->blockSignals(true);
        m_spinBoxavg->setValue    (value);
        m_spinBoxavg->blockSignals(false);


        // apply filter to source image; save result in destination image
        applyFilter(g_mainWindowP->imageSrc(), g_mainWindowP->imageDst());

        // display output
        g_mainWindowP->displayOut();
}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::median:
```

```
//
//! \brief
//! \details
//! \param[in]    I1  - Input image.
//! \param[in]    sz - kernel size.
//! \param[in]    avg_nbrs - average neighbors to blur with
//! \param[out]   I2  - Output image.
//
void
Median::median(ImagePtr I1, int sz, int avg_nbrs, ImagePtr I2) {

        int w = I1->width();
        int h = I1->height();
        int total = w * h;
        int mid = ((sz*sz) / 2) + 1;

        int i, x, y, xx, yy, t, sum, ww;

        IP_copyImageHeader(I1, I2);

        // p is temporary uchar type to hold uchar pixel values
        // p1 is initially first pixel in I1
        ChannelPtr<uchar> p1, p2, p;
        for(int ch = 0; IP_getChannel(I1, ch, p1, t); ch++) {
          IP_getChannel(I2, ch, p2, t);

        int Histogram[MXGRAY];

        // process each row
        for (y =0; y<h; ++y) {

          //initialize histo with 0's
          for (int k = 0; k<MXGRAY; k++)
                  {Histogram[k] = 0;}

          // fill kernel
          for (yy = 0; yy<sz; yy++) {

                  // p is the pixel to read
                  // p is the pixel intensity we read. Increase frequency of p in Histogram by 1.
                  // p = 0 on first turn
                  p = p1 + (yy * w);

                  // add values to histogram.
                  for (xx = 0; xx<sz; ++xx) {
                          Histogram[*p++]++;
                  }
          }

          //process remaining points in that row
          for (x=0; x<w; ++x) {
```

```
                //find median
                for (i=sum=0; i<MXGRAY; ++i) {
                        sum += Histogram[i];
                        if (sum >= mid)
                                break;
                }
                //copy median (i) into output
                *p2++ = i;

                //decrement
                p = p1 + x;
                for (yy = 0; yy < sz; yy++) {
                        Histogram[*p]--;
                        p += ww;
                }

                //increment
                p = p1 + x + sz;
                for (yy = 0; yy < sz; yy++) {
                        Histogram[*p]++;
                        p += w;
                }
          }

          p1 += ww;
          }
        }

}



// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Median::reset:
//
// Reset parameters.
//
void
Median::reset() {}
```