Author: Cassidy Lamm
Email: clamm@clemson.edu

Project 01 ReadMe

Project Description: This project uses PyQt to view, manipulate and composite images. The operations include basic operations such as gamma, contrast and monochrome as well as the image compositing operations of mix, keyMix and over. It also includes edge-detect, blur, sharpen and median spatial filtering using the specific kernels given below. Finally, the matte creation and manipulation operations luma-key, chroma-key, and color-difference method are used.

Class Attributes:

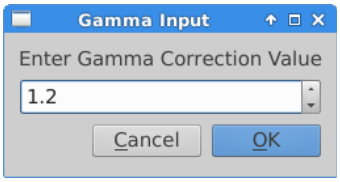
edgeArray: Array used for the kernel in edge detection function
blurArray: Array used for the kernel in blur function
sharpenArray: Array used for the kernel in sharpen function

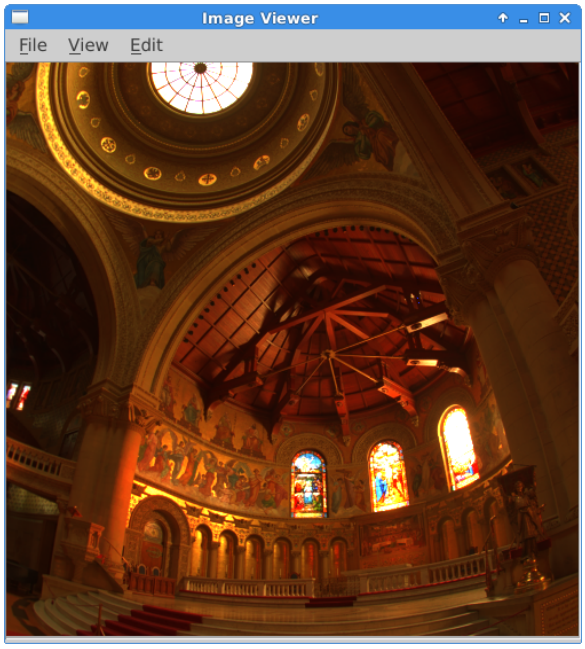
```
e = [[-1.0,-1.0,-1.0], [-1.0,8.0,-1.0], [-1.0,-1.0,-1.0]]
self.edgeArray = e

b = [[.111, .1111, .1111], [.1111, .1111, .1111], [.1111, .1111, .1111]]
self.blurArray = b

s = [[-1.0, -1.0, -1.0], [-1.0, 9.0, -1.0], [-1.0, -1.0, -1.0]]
self.sharpenArray = s
```

Common Functions:

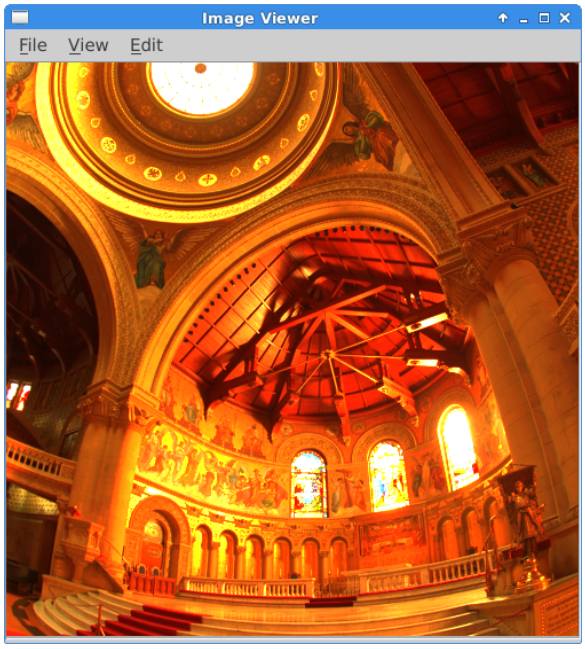
```
def clampInt(self, val):
    ''' Clamps values from 0 to 255

        Args:
            val: A float representing a rgb values
        Returns:
            The rgb value clamped between 0 and 255
    '''
    if(val > 255):
        return 255
    elif(val < 0):
        return 0
    else:
        return val
```

Gamma:





Original Image



Altered Image

Description: The gamma function raises each pixel to the power of (1 / gamma value). When the user clicks on Edit -> Gamma, a dialog box appears and allows the user to select his or her own gamma value.

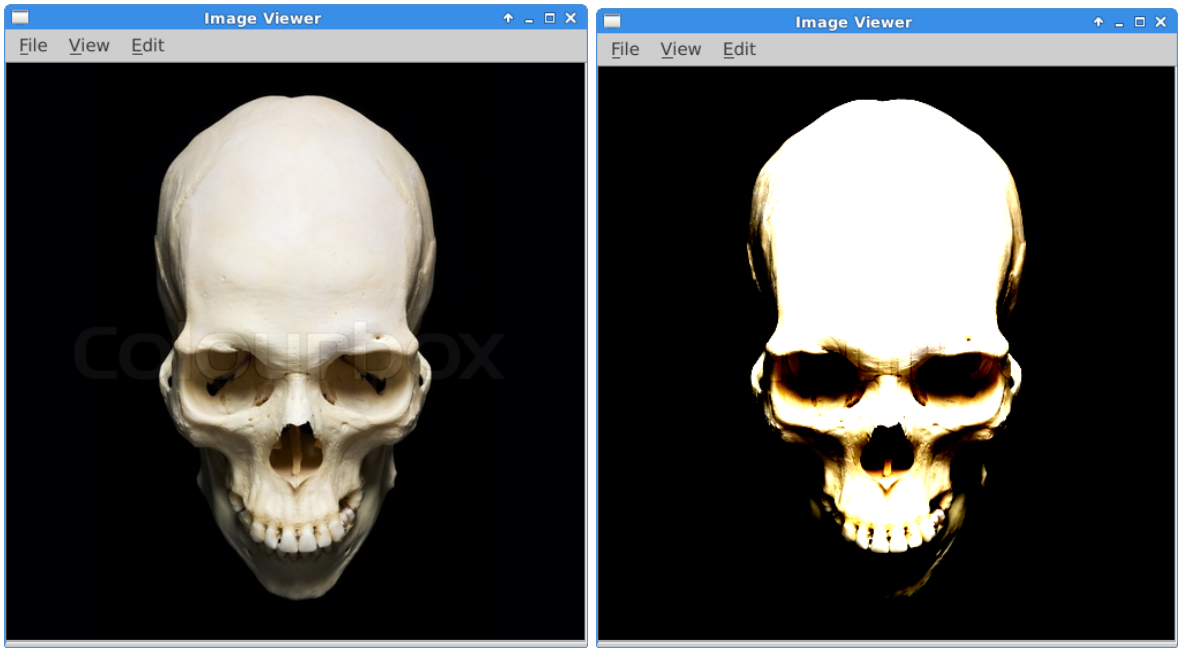
Code: imageViewer.py

```
    def gamma(self):
        '''Raises each pixel to the power of 1 divided by the gamma value supplied.
        This changes the midtones of the image.

        '''
        image = self.imageLabel.pixmap().toImage()
        outImage = image.copy()

        #Get user input Value from imageDialog class
        gVal = box.getGamma()

        for row in range(0,image.height()):
            for col in range(0,image.width()):

                curPixel = image.pixel(row,col)
                red = QtGui.qRed(curPixel) ** gVal
                green = QtGui.qGreen(curPixel) ** gVal
                blue = QtGui.qBlue(curPixel) ** gVal

                newPixel = QtGui.qRgb(self.clampInt(red), self.clampInt(green), self.clampInt(blue))
                outImage.setPixel(row,col,newPixel)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Code: imageDialog.py

```
    def getGamma(self):
        num,ok = QInputDialog.getDouble(self,"Gamma Input","Enter Gamma Correction Value")

        if ok:
            return num
```

Contrast:



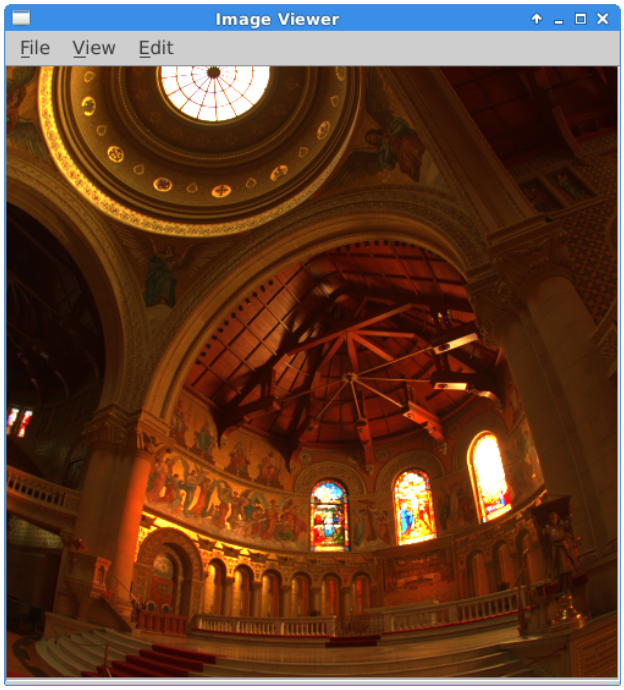
Original Altered

Description: This function changes brightness relationship between the upper and lower color ranges of an image. I use the function O = (I - ⅓)* 3 as specified by Brinkman.

Code: imageViewer.py

```
    def contrast(self):
        '''Changes brightness relationship between the upper and lower color
        ranges of an image. Uses the function O = (I - 1/3) * 3)

        '''
        image = self.imageLabel.pixmap().toImage()
        outImage = image.copy()

        for row in range(0,image.height()):
            for col in range(0,image.width()):

                curPixel = image.pixel(row,col)
                red = (QtGui.qRed(curPixel) - 85) * 3
                green = (QtGui.qGreen(curPixel) - 85) * 3
                blue = (QtGui.qBlue(curPixel) - 85) * 3

                newPixel = QtGui.qRgb( self.clampInt(red), self.clampInt(green), self.clampInt(blue))
                outImage.setPixel(row,col,newPixel)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Monochrome:



Original

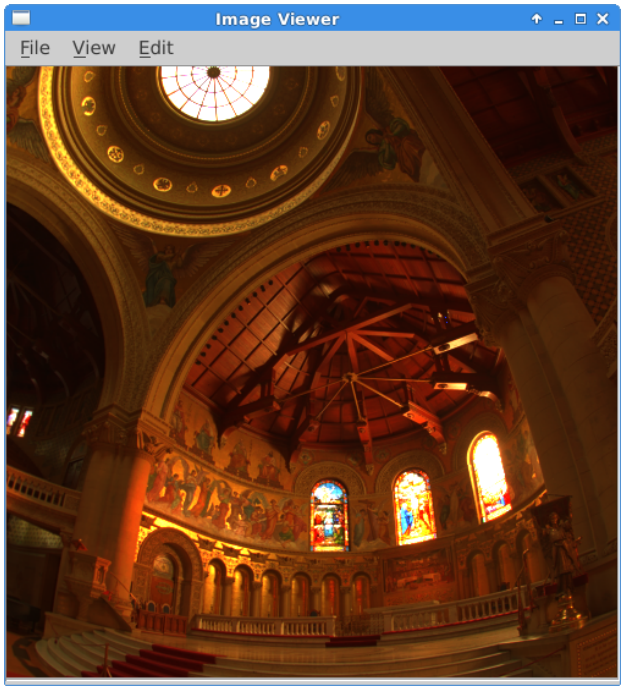
Altered

Description: Produces a monochromatic image by averaging the three channels together. Uses the function O = (R * 0.309) + (G * 0.609) + (B * 0.082) as specified by Brinkman.

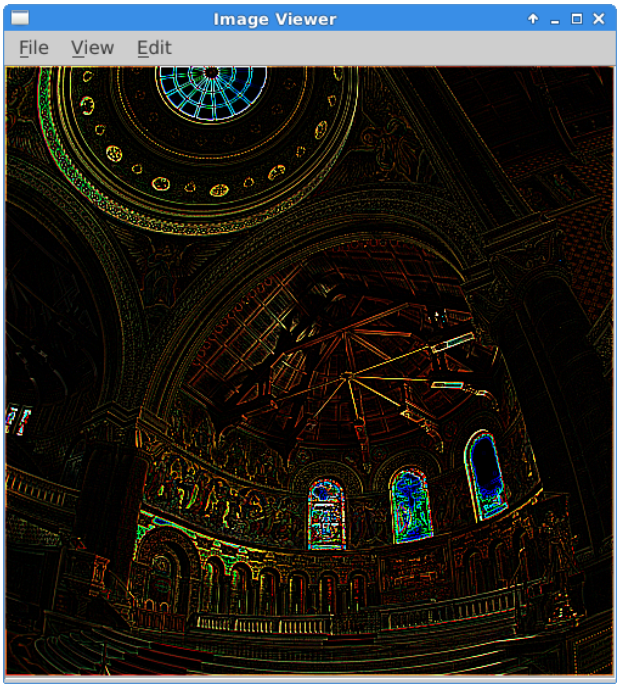
Code: imageViewer.py

```
def monochrome(self):
    '''Produces a monochromatic image by averaging the three channels together.
    Uses the function O = (R * 0.309) + (G * 0.609) + (B * 0.082)

    '''
    image = self.imageLabel.pixmap().toImage()
    outImage = image.copy()

    for row in range(0,image.height()):
        for col in range(0,image.width()):

            curPixel = image.pixel(row,col)
            r = QtGui.qRed(curPixel) * 0.309
            g = QtGui.qGreen(curPixel) * 0.609
            b = QtGui.qBlue(curPixel) * 0.082
            newValue = r + g + b

             newPixel = QtGui.qRgb( self.clampInt(newValue), self.clampInt(newValue),
                    self.clampInt(newValue))
            outImage.setPixel(row,col,newPixel)

    self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Edge-Detect:

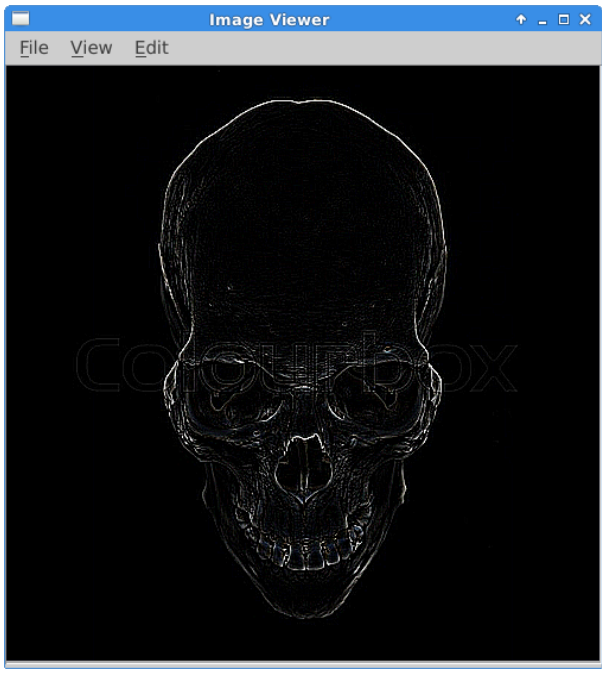


Original



Altered

Original



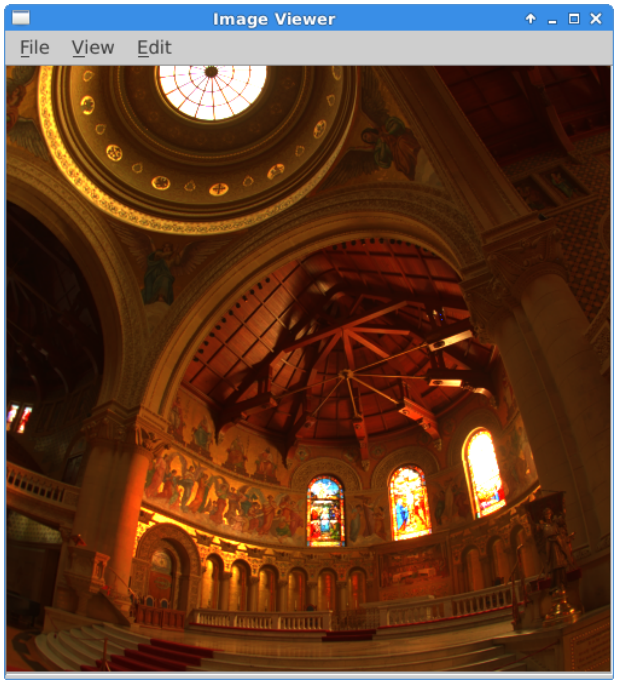
Altered

Description: A spatial filter that uses a specific kernel to detect edges in the image, producing bring pixels where the transition areas occur.

Code: imageViewer.py

```
    def edge(self):
        '''A spatial filter that uses a specified kernel to detect edges in the
        image, producing bright pixels where the transition areas occured.
            Kernel:
              [ [ -1, -1, -1], [ -1,  8, -1], [ -1, -1, -1] ]
        '''
        image = self.imageLabel.pixmap().toImage()
        outImage = image.copy()

        for row in range(0,image.height()):
            for col in range(0,image.width()):

                red = 0.0
                green = 0.0
                blue = 0.0

                imgPixel = image.pixel(row, col)

                #Loop -1 to 2 so that multiplication will start in the top left
                #index of the kernel
                for kRow in range(-1,2):
                    iRow = row + kRow
                    for kCol in range(-1,2):
                        iCol = col + kCol

                        #If the pixel aligned with the kernel is actually in the image
                        if(iRow >= 0 and iRow < image.height() and iCol >= 0 and iCol < image.width()):
                            curPixel = image.pixel(iRow, iCol)
                            red += (QtGui.qRed(curPixel) * self.edgeArray[kRow+1][kCol+1])
                            green += (QtGui.qGreen(curPixel) * self.edgeArray[kRow+1][kCol+1])
                            blue += (QtGui.qBlue(curPixel) * self.edgeArray[kRow+1][kCol+1])

                red = self.clampInt(red)
                green = self.clampInt(green)
                blue = self.clampInt(blue)

                newPixelColor = QtGui.qRgb(red,green,blue)
                outImage.setPixel(row,col,newPixelColor)

        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Blur:



Original

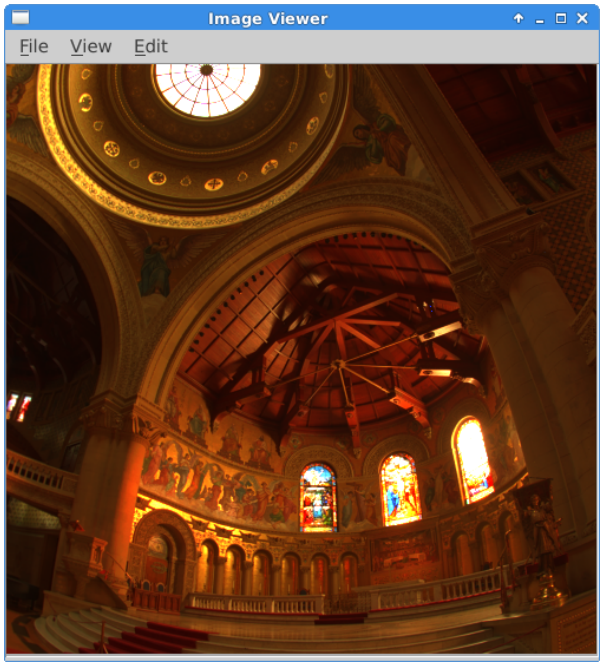
Altered (You can really tell in the dome glass)

Description: A spatial filter that uses a specified kernel to blur the image by averaging the neighboring pixels with the current pixel.

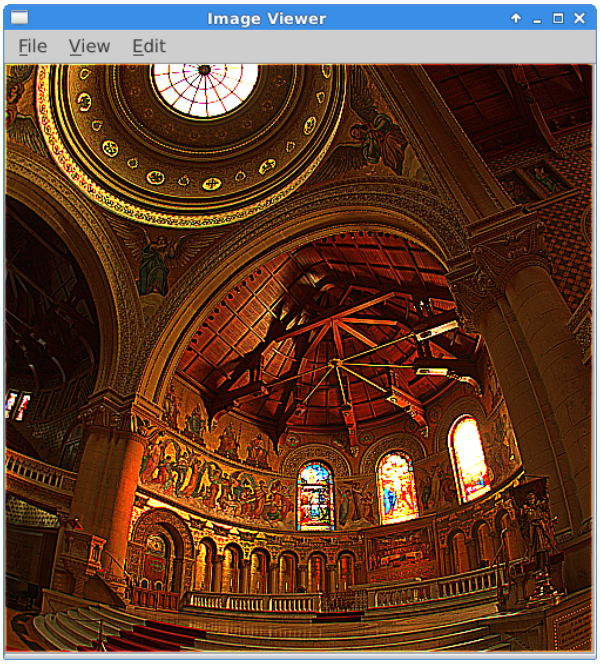
Code: imageViewer.py

```
def blur(self):
    '''A spatial filter that uses a specified kernel to blur the image by
    averaging the neighboring pixels with the current pixel.

        Kernel:
            [ [ 1/9, 1/9, 1/9], [ 1/9, 1/9, 1/9], [ 1/9, 1/9, 1/9] ]
    '''
    image = self.imageLabel.pixmap().toImage()
    outImage = image.copy()

    for row in range(0,image.height()):
        for col in range(0,image.width()):

            red = 0.0
            green = 0.0
            blue = 0.0

            imgPixel = image.pixel(row, col)

            #Loop -1 to 2 so that multiplication will start in the top left
            #index of the kernel
            for kRow in range(-1,2):
                iRow = row + kRow
                for kCol in range(-1,2):
                    iCol = col + kCol

                    #If the pixel aligned with the kernel is actually in the image
                    if(iRow >= 0 and iRow < image.height() and iCol >= 0 and iCol < image.width()):
                        curPixel = image.pixel(iRow, iCol)
                        red += (QtGui.qRed(curPixel) * self.blurArray[kRow+1][kCol+1])
                        green += (QtGui.qGreen(curPixel) * self.blurArray[kRow+1][kCol+1])
                        blue += (QtGui.qBlue(curPixel) * self.blurArray[kRow+1][kCol+1])

            red = self.clampInt(red)
            green = self.clampInt(green)
            blue = self.clampInt(blue)

            newPixelColor = QtGui.qRgb(red,green,blue)
            outImage.setPixel(row,col,newPixelColor)

    self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Sharpen:



Original



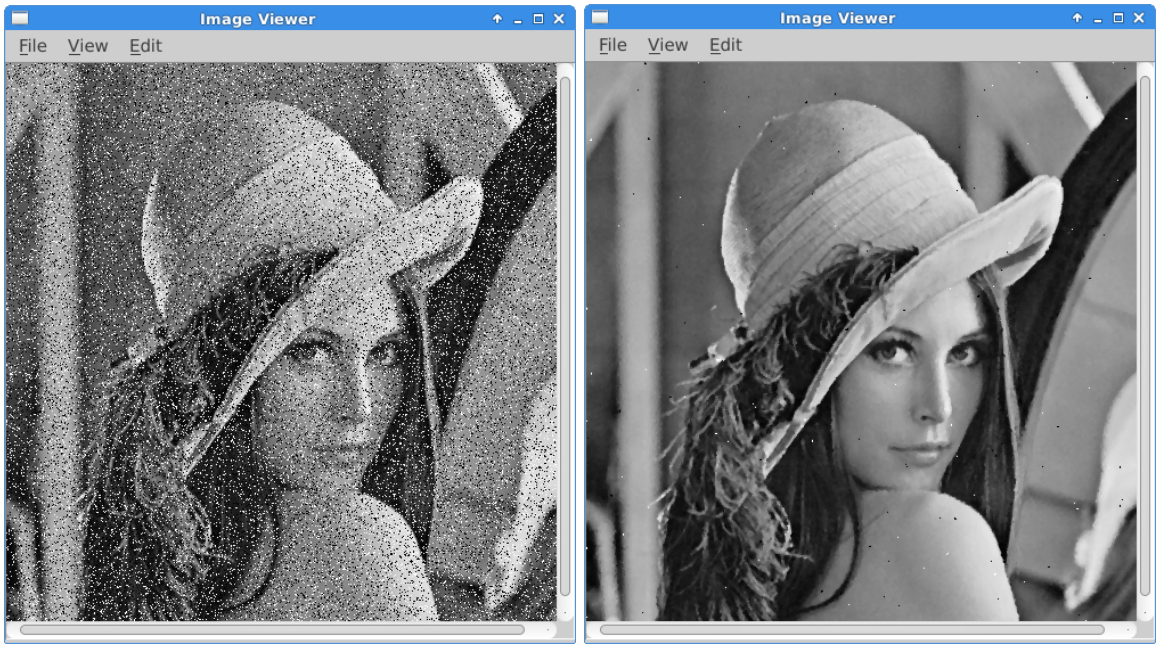
Altered

Description: A spatial filter that uses a specified kernel to sharpen the image by increasing the contrast between areas of transition in the image

Code: imageViewer.py

```
def sharpen(self):
    '''A spatial filter that uses a specified kernel to sharpen the image by
    increasing the contrast between areas of transition in an image.

        Kernel:
          [ [ -1, -1, -1], [ -1,  9, -1], [ -1, -1, -1] ]
    '''
    image = self.imageLabel.pixmap().toImage()
    outImage = image.copy()

    for row in range(0,image.height()):
        for col in range(0,image.width()):

            red = 0.0
            green = 0.0
            blue = 0.0

            imgPixel = image.pixel(row, col)

            #Loop -1 to 2 so that multiplication will start in the top left
            #index of the kernel
            for kRow in range(-1,2):
                iRow = row + kRow
                for kCol in range(-1,2):
                    iCol = col + kCol

                    #If the pixel aligned with the kernel is actually in the image
                    if(iRow >= 0 and iRow < image.height() and iCol >= 0 and iCol < image.width()):
                        curPixel = image.pixel(iRow, iCol)
                        red += (QtGui.qRed(curPixel) * self.sharpenArray[kRow+1][kCol+1])
                        green += (QtGui.qGreen(curPixel) * self.sharpenArray[kRow+1][kCol+1])
                        blue += (QtGui.qBlue(curPixel) * self.sharpenArray[kRow+1][kCol+1])

            red = self.clampInt(red)
            green = self.clampInt(green)
            blue = self.clampInt(blue)

            newPixelColor = QtGui.qRgb(red,green,blue)
            outImage.setPixel(row,col,newPixelColor)

    self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Median:



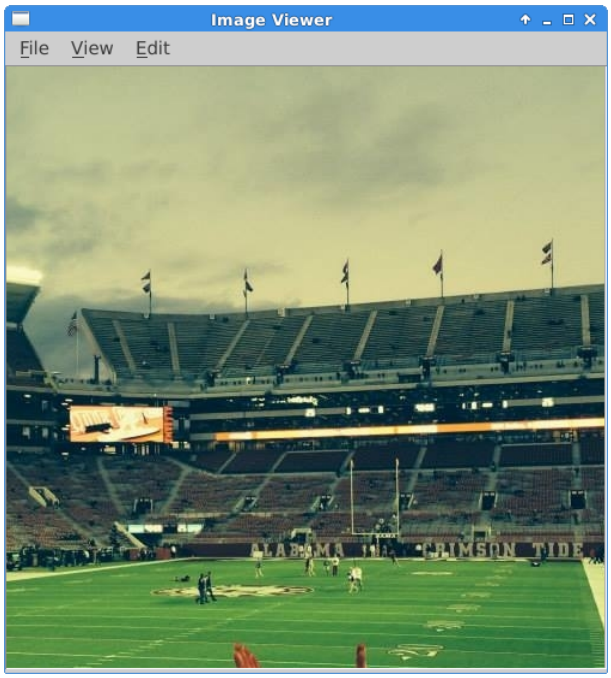
Description: A spatial filter that ranks the kernel pixels in terms of brightness and then changes the value to be the same as the median.

Code: imageViewer.py

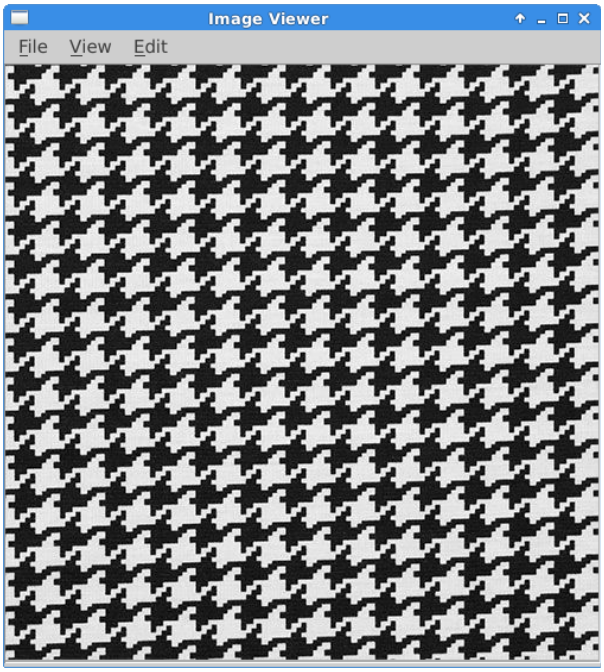
```
    def medianArray(self, array):
        '''Finds the median index of an unsorted array
            Args:
                array: unsorted array of hsv values from 0 to 1
            Returns:
                The median index of an unsorted array
        '''
        sortList = sorted(array)
        return len(sortList)/2

    def median(self):
        '''Filter that ranks the kernel pixels in terms of brightness and then
        changes the value to be the same as the median.
        '''
        image = self.imageLabel.pixmap().toImage()
        outImage = image.copy()

        for row in range(0,image.height()):
            for col in range(0,image.width()):
```

```
        #load rgb values into a list so that the median index can be retrieved
        red = []
        green = []
        blue = []
        value = []

        imgPixel = image.pixel(row, col)

        for kRow in range(-1,2):
            for kCol in range(-1,2):
                iRow = row + kRow
                iCol = col + kCol

                if(iRow >= 0 and iRow < image.height() and iCol >= 0 and iCol < image.width()):
                    curPixel = image.pixel(iRow, iCol)

                    #append each values to its respective list
                    red.append(QtGui.qRed(curPixel))
                    green.append(QtGui.qGreen(curPixel))
                    blue.append(QtGui.qBlue(curPixel))

                    #convert to hsv and load resulting values into a new array
                    hsvPixel = self.hsv(QtGui.qRed(curPixel), QtGui.qGreen(curPixel),
                            QtGui.qBlue(curPixel))
                    value.append(hsvPixel[2])

        #find the median of the values
        medianIndex = self.medianArray(value)

        sortRed = sorted(red)
        sortGreen = sorted(green)
        sortBlue = sorted(blue)

        #retrieve the correct rgb values
        redVal = self.clampInt(sortRed[medianIndex])
        greenVal = self.clampInt(sortGreen[medianIndex])
        blueVal = self.clampInt(sortBlue[medianIndex])

        newPixelColor = QtGui.qRgb(redVal,greenVal,blueVal)
        outImage.setPixel(row,col,newPixelColor)

self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

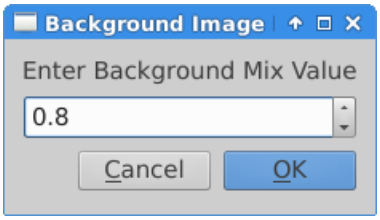
Mix:

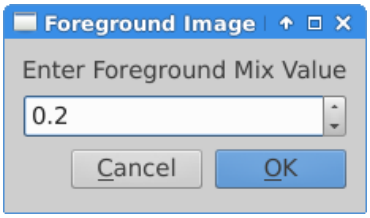


Background

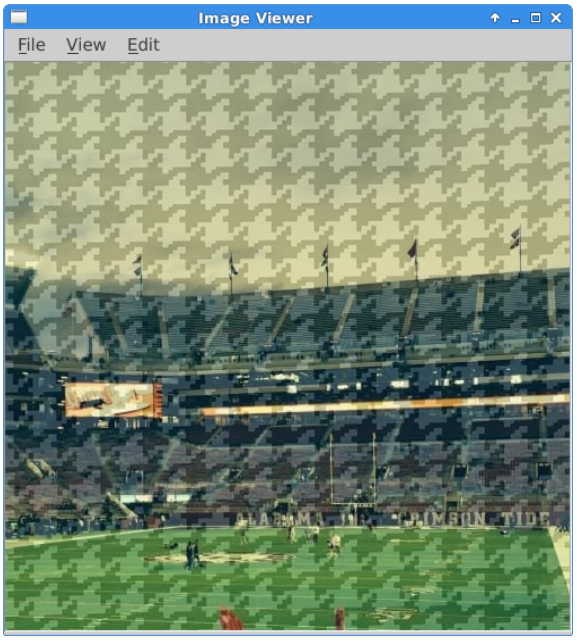


Foreground





User Input for Background and Foreground Images



Description: Calculates the normalized addition of two images. Uses the formula O = (A * aMix) + (B * bMix) where aMix and bMix are inputs from the user.

Code: imageViewer.py

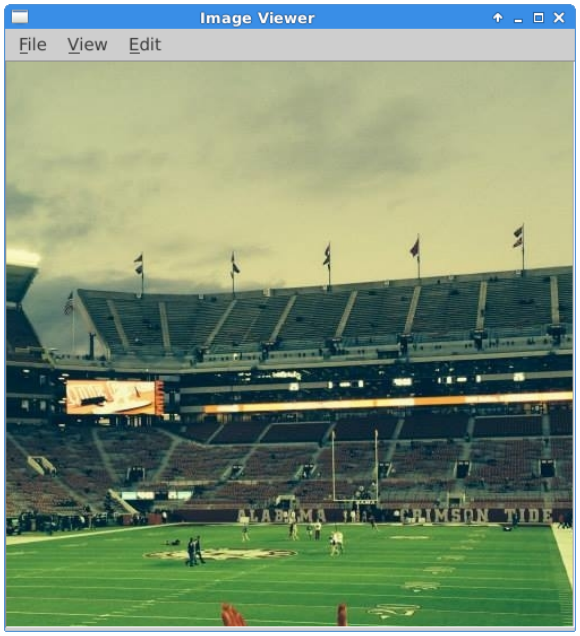
```
    def mix(self):
        '''Calculates the normalized addition of two images. Uses the formula:
        O = (A * aMix) + (B * bMix)
        '''
        #store the imageLabel as the background image
        bImage = self.imageLabel.pixmap().toImage()
        outImage = bImage.copy()

        #open a new image as a foreground image
        fileName = QtGui.QFileDialog.getOpenFileName(self, "Open Foreground Image",
                                            QtCore.QDir.currentPath())
        if fileName:
            aImage = QtGui.QImage(fileName)
            if aImage.isNull():
                QtGui.QMessageBox.information(self, "New Image cannot load",
                                    "Cannot load %s." % fileName)
                return

        #Ensure both images are the same size
        if(aImage.height()!= bImage.height()):
            QtGui.QMessageBox.information(self, "Images do not have same height",
                    "Cannot load %s." % fileName)
            return
        if(aImage.width()!= bImage.width()):
            QtGui.QMessageBox.information(self, "Images do not have same width",
                     "Cannot load %s." % fileName)
            return

        #get user input for mix values
        aMix = box.getAMix()
        bMix = box.getBMix()

        for row in range(0,aImage.height()):
            for col in range(0,aImage.width()):

                aPixel = aImage.pixel(row,col)
                bPixel = bImage.pixel(row,col)
```

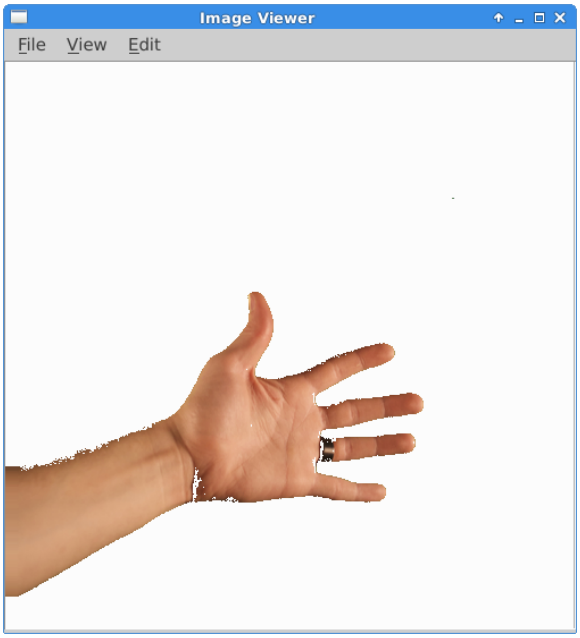
```
            red = (QtGui.qRed(aPixel) * aMix) + (QtGui.qRed(bPixel) * bMix)
            green = (QtGui.qGreen(aPixel) * aMix) + (QtGui.qGreen(bPixel) * bMix)
            blue = (QtGui.qBlue(aPixel) * aMix) + (QtGui.qBlue(bPixel) * bMix)

            newPixelColor = QtGui.qRgb(self.clampInt(red),self.clampInt(green),
                  self.clampInt(blue))
            outImage.setPixel(row,col,newPixelColor)

      self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Code: imageDialog.py

```
  def getAMix(self):
    num,ok = QInputDialog.getDouble(self,"Foreground Image Input",
        "Enter Foreground Mix Value")

    if ok:
      return num

  def getBMix(self):
    num,ok = QInputDialog.getDouble(self,"Background Image Input",
        "Enter Background Mix    Value")

    if ok:
      return num
```

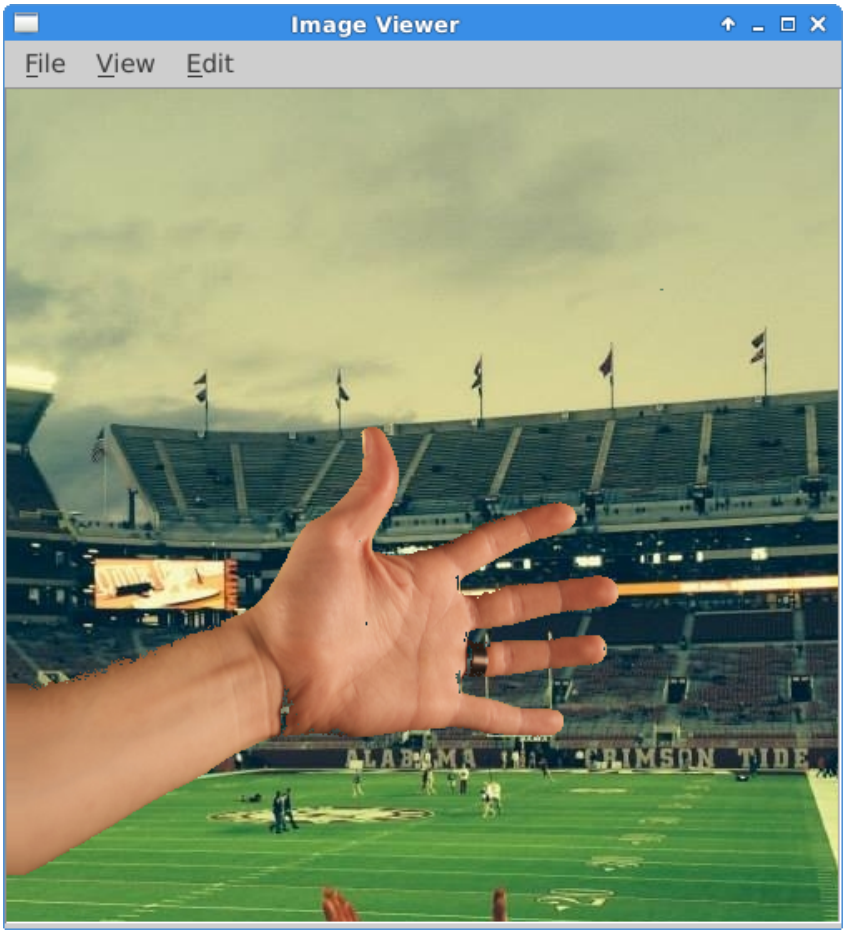
Key-Mix:





Background

Foreground (A hand with blue screen keyed out)



Description: Uses a matte as a key to determine how two images mix together on a pixel by pixel basis. Uses the formula: O = (A x M) + [(1-M) * A]. Note: In this function, the alpha of the foreground (second image) is used as the matte. Thus, the foreground image must be a straight image.

Code: imageViewer.py

```
    def keyMix(self):
        '''Uses a matte as a key to determine how two images mix together on a pixel
        by pixel basis. Uses the formula: O = (A x M) + [(1-M) * A]

            Note: In this function, the alpha of the foreground (second image) is used
            as the matte. Thus, the foreground image must be a straight image.
        '''

        #store the imageLabel as the background image
        bImage = self.imageLabel.pixmap().toImage()
        outImage = bImage.copy()

        #open a new image as a foreground image
        fileName = QtGui.QFileDialog.getOpenFileName(self, "Open Foreground Image",
                                            QtCore.QDir.currentPath())
        if fileName:
            aImage = QtGui.QImage(fileName)
            if aImage.isNull():
                QtGui.QMessageBox.information(self, "Foreground Image cannot load",
                                    "Cannot load %s." % fileName)
                return

        #Ensure both images are the same size
        if(aImage.height()!= bImage.height()):
            QtGui.QMessageBox.information(self, "Images do not have same height",
                    "Cannot load %s." % fileName)
            return

        if(aImage.width()!= bImage.width()):
            QtGui.QMessageBox.information(self, "Images do not have same width",
            "Cannot load %s." % fileName)
            return

        for row in range(0,aImage.height()):
            for col in range(0,aImage.width()):

                aPixel = aImage.pixel(row,col)
```

```
        bPixel = bImage.pixel(row,col)

        red = (QtGui.qRed(aPixel) * (QtGui.qAlpha(aPixel)/255)) +
                        ((1 - (QtGui.qAlpha(aPixel)/255)) * QtGui.qRed(bPixel))

        green = (QtGui.qGreen(aPixel) * (QtGui.qAlpha(aPixel)/255)) +
                        ((1 - (QtGui.qAlpha(aPixel)/255)) * QtGui.qGreen(bPixel))

        blue = (QtGui.qBlue(aPixel) * (QtGui.qAlpha(aPixel)/255)) +
                        ((1 - (QtGui.qAlpha(aPixel)/255)) * QtGui.qBlue(bPixel))

        newPixelColor = QtGui.qRgb(self.clampInt(red), self.clampInt(green),
                        self.clampInt(blue))
        outImage.setPixel(row,col,newPixelColor)

self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Over:

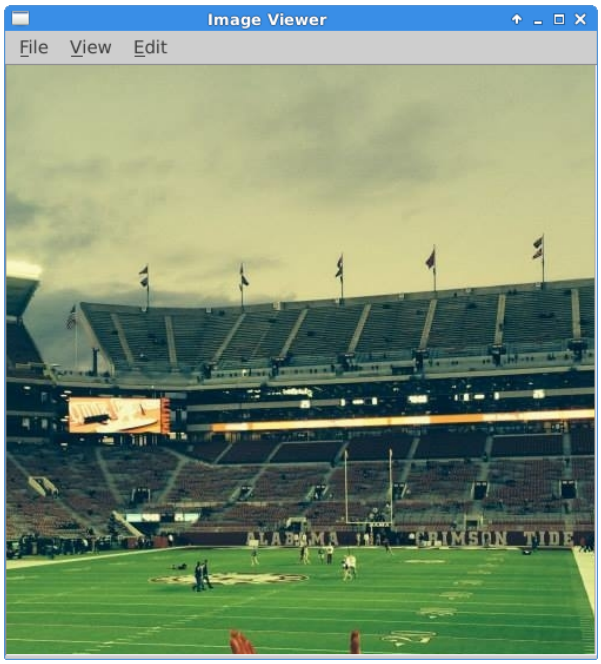


Image Viewer
File View Edit


Image Viewer
File View Edit


Image Viewer
File View Edit

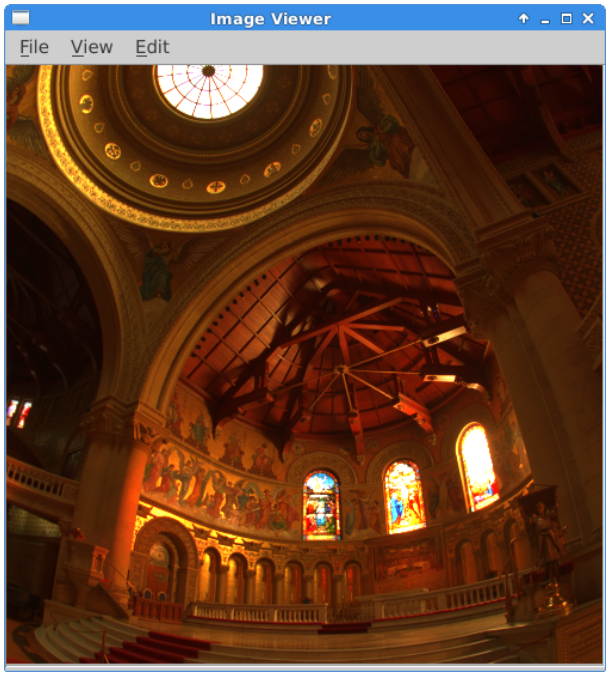
Description: Layers a four channel image over another image. Uses the formula
O = A + (alphaA) * B)

Code: imageViewer.py

```
    def over(self):
        '''Layers a four channel image over another image. Uses the formula:
            O = A + [(1- alphaA) * B]

            Note: In this function the foreground image is assumed to be a premultiplied image.
        '''
        #store the imageLabel as the background image
        bImage = self.imageLabel.pixmap().toImage()
        outImage = bImage.copy()

        #open a new image as a foreground image
        fileName = QtGui.QFileDialog.getOpenFileName(self, "Open Foreground Image",
                                            QtCore.QDir.currentPath())
        if fileName:
            aImage = QtGui.QImage(fileName)
            if aImage.isNull():
                QtGui.QMessageBox.information(self, "Foreground Image cannot load",
                                    "Cannot load %s." % fileName)
                return

        #Ensure both images are the same size
        if(aImage.height()!= bImage.height()):
            QtGui.QMessageBox.information(self, "Images do not have same height",
                    "Cannot load %s." % fileName)
            return

        if(aImage.width()!= bImage.width()):
            QtGui.QMessageBox.information(self, "Images do not have same width",
                    "Cannot load %s." % fileName)
            return


        for row in range(0,aImage.height()):
            for col in range(0,aImage.width()):

                aPixel = aImage.pixel(row,col)
                bPixel = bImage.pixel(row,col)

                red = QtGui.qRed(aPixel) + ((1 - (QtGui.qAlpha(aPixel)/255)) * QtGui.qRed(bPixel))
```

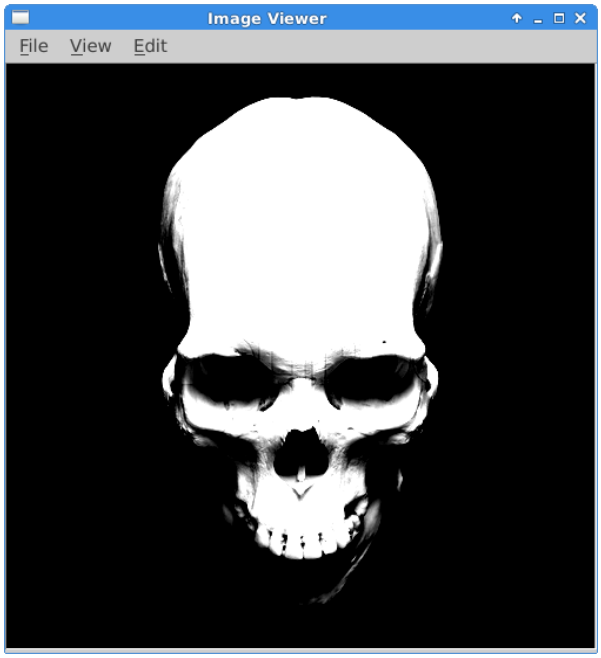
```
        green = QtGui.qGreen(aPixel) + ((1 - (QtGui.qAlpha(aPixel)/255)) *
                QtGui.qGreen(bPixel))
        blue = QtGui.qBlue(aPixel) + ((1 - (QtGui.qAlpha(aPixel)/255)) * QtGui.qBlue(bPixel))
        alpha = QtGui.qAlpha(aPixel) + ((1 - (QtGui.qAlpha(aPixel)/255)) *
                QtGui.qAlpha(bPixel))

        newPixelColor = QtGui.qRgba(self.clampInt(red), self.clampInt(green),
                self.clampInt(blue), self.clampInt(alpha))
        outImage.setPixel(row,col,newPixelColor)

self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Luma-Key:

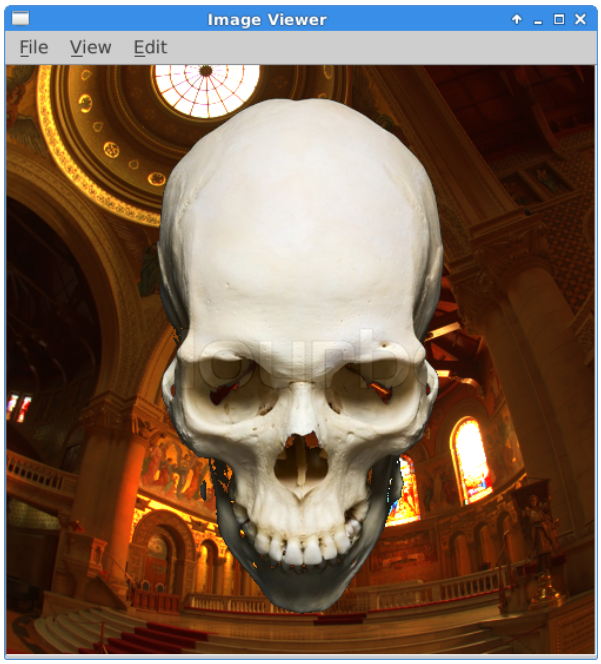


Background

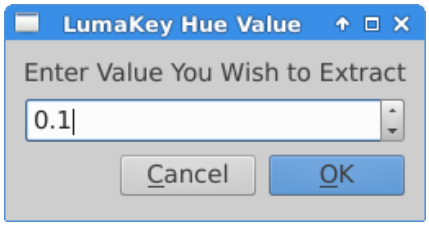
Original Foreground



Foreground after monochrome and contrast



Final image after extracting the matte



User input dialogue box

Description: Extracts a matte based on manipulating luminance values. This is done by converting to a monochrome image, applying a contrast to the resulting image and then setting that alpha channel of the darker values to 0. Note: To further demonstrate the lumaKey, I automatically place the matte over a second image.

Code: imageViewer.py

```
    def hsv(self, r, g, b):
        '''Converts rbg values to thier corresponding hsv values

            Note: This code was given to us by Dr. Levine in 6040. I think it
                works really well, so I incorporated it.

            Args:
                r: float of red color value
                g: float of green color value
                b: float of blue color value

            Returns:
                An array with the hsv values: [H, S, V]

        '''
        red = r / 255.0
        green = g / 255.0
        blue = b / 255.0

            maxc = max(max(red, green), blue)
            minc = min(min(red, green), blue)

        # value is maximum of r, g, b
            v = maxc

        #saturation and hue 0 if value is 0
            if(maxc == 0):
            s = 0
            h= 0

        #saturation is color purity on scale 0 - 1
            else:
            s = (maxc - minc) / maxc
            delta = maxc - minc

            #hue doesn't matter if saturation is 0
```

```
        if(delta == 0):
            h = 0

            #otherwise, determine hue on scale 0 - 360
        else:
                if(red == maxc):
                        h = (green - blue) / delta
                elif(green == maxc):
                        h = 2.0 + (blue - red) / delta

            #(blue == maxc)
                else:
                        h = 4.0 + (red - green) / delta

                h = h * 60.0
                if(h < 0):
                        h = h + 360.0

    #put new hsv values into an array and return it
    hsvPix = []
        hsvPix.append(h)
        hsvPix.append(s)
        hsvPix.append(v)
        return hsvPix

def lumaKey(self):
    '''Extracts a matte based on manipulating luminance values. This is done
    by converting to a monochrome image, applying a contrast to the resulting
    image and then setting that alpha channel of the darker values to 0.

        Note: To further demonstrate the lumaKey, I automatically place the
            matte over a second image.
    '''
    #store the imageLabel as the background image
    bImage = self.imageLabel.pixmap().toImage()
    outImage = bImage.copy()

    #open a new image as a foreground image
    fileName = QtGui.QFileDialog.getOpenFileName(self, "Open Foreground Image",
                                        QtCore.QDir.currentPath())
    if fileName:
        aImage = QtGui.QImage(fileName)
        if aImage.isNull():
```

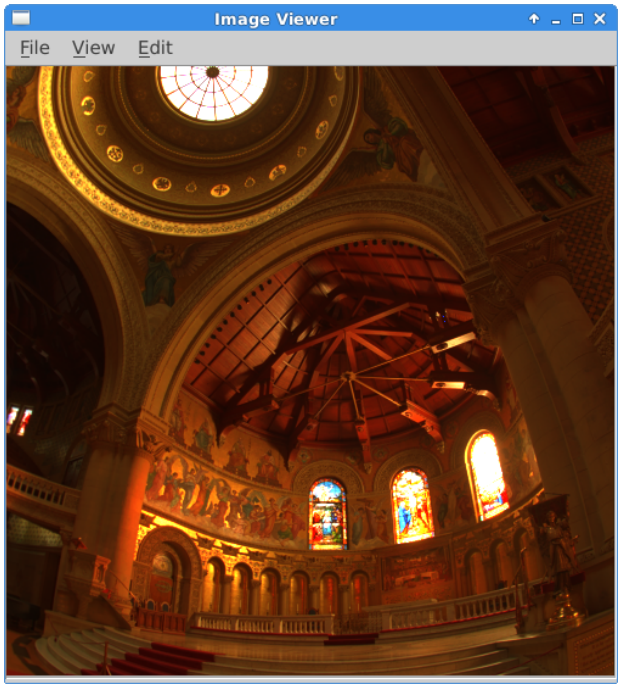
```
        QtGui.QMessageBox.information(self, "Foreground Image cannot load",
                                      "Cannot load %s." % fileName)
        return

#Ensure both images are the same size
if(aImage.height()!= bImage.height()):
    QtGui.QMessageBox.information(self, "Images do not have same height",
              "Cannot load %s." % fileName)
    return

if(aImage.width()!= bImage.width()):
    QtGui.QMessageBox.information(self, "Images do not have same width",
               "Cannot load %s." % fileName)
    return

#set the foreground image as the ImageLabel
self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(aImage))

#apply monochrome and contrast to the foreground image
self.monochrome()
self.contrast()

#get user input
lumValue = box.getLum()

for row in range(0,aImage.height()):
    for col in range(0,aImage.width()):

        aPixel = aImage.pixel(row,col)
        bPixel = bImage.pixel(row,col)

        red = QtGui.qRed(aPixel)
        green = QtGui.qGreen(aPixel)
        blue = QtGui.qBlue(aPixel)
        alpha = 255

        #convert pixel to hsv to compare values
        hsvPix = self.hsv(red,green,blue)

        #clear lowest value pixels
        if(hsvPix[2] <= lumValue):
            alpha = 0
```

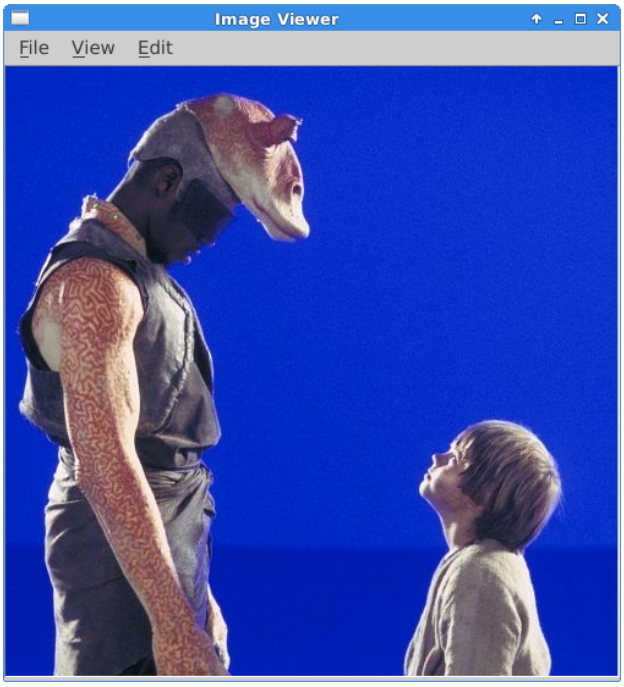
```
            #lPixel = new luma pixel
            lPixel = QtGui.qRgba(red,green,blue,alpha)
            aImage.setPixel(row,col,lPixel)

            #Layer the foreground matte over the background image
            red = (QtGui.qRed(lPixel) * (QtGui.qAlpha(lPixel)/255)) +
                  ((1 - (QtGui.qAlpha(lPixel)/255)) * QtGui.qRed(bPixel))

            green = (QtGui.qGreen(lPixel) * (QtGui.qAlpha(lPixel)/255))
                  + ((1 - (QtGui.qAlpha(lPixel)/255)) * QtGui.qGreen(bPixel))

            blue = (QtGui.qBlue(lPixel) * (QtGui.qAlpha(lPixel)/255))
                  + ((1 - (QtGui.qAlpha(lPixel)/255)) * QtGui.qBlue(bPixel))

            newPixelColor = QtGui.qRgba(self.clampInt(red), self.clampInt(green),
                  self.clampInt(blue), self.clampInt(alpha))
            outImage.setPixel(row,col,newPixelColor)

      self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

Code: imageDialogue.py

```
  def getLum(self):
    num,ok = QInputDialog.getDouble(self,"LumaKey Hue Value",
        "Enter Value You Wish to Extract")

    if ok:
      return num
```

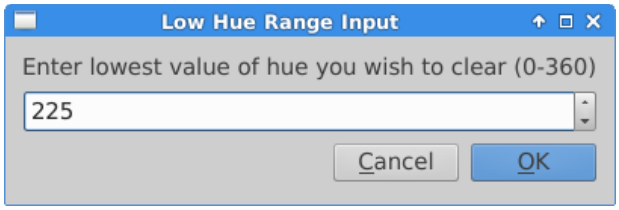
Chroma-Key:

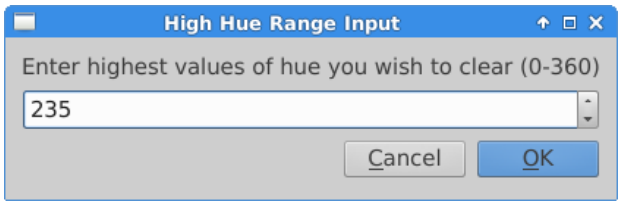


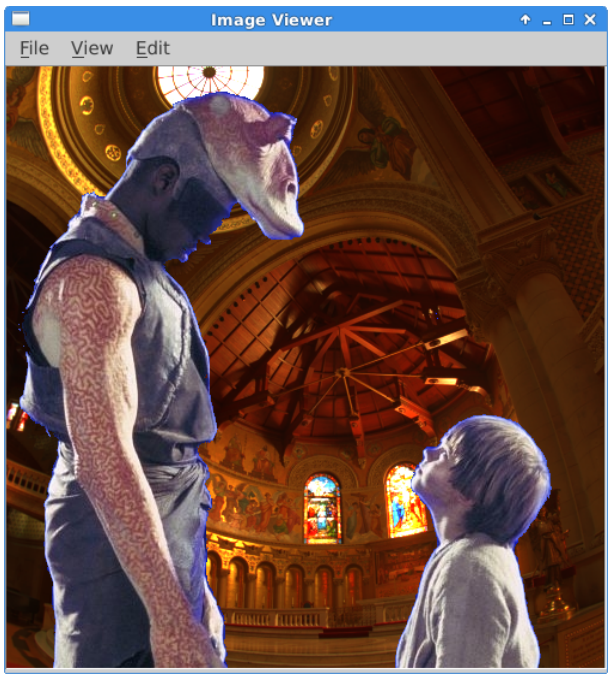


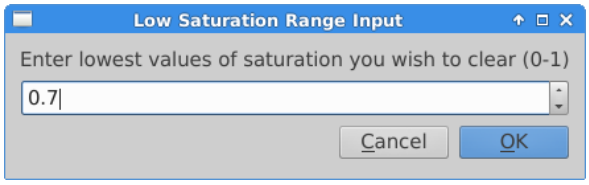
Background

Foreground









Description: Extracts a matte based on a range of hue and saturation values. This is done by converting to hsv values and picking a range of both hue and saturation to take out the alpha. Note: To further demonstrate the chromaKey, I automatically place the matte over a second image.

Code: imageViewer.py

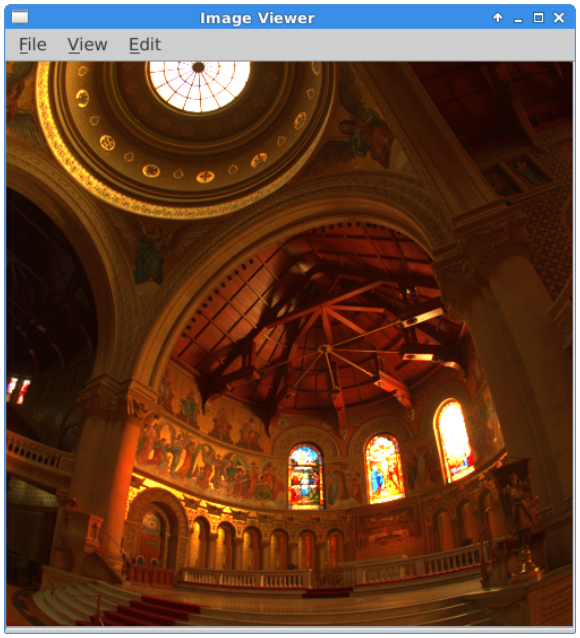
```
    def chromaKey(self):
        '''Extracts a matte based on a range of hue and saturation values. This
        is done by converting to hsv values and picking a range of both hue and
        saturation to take out the alpha.

            Note: To further demonstrate the chromaKey, I automatically place the
                matte over a second image.
        '''
        #store the imageLabel as the background image
        bImage = self.imageLabel.pixmap().toImage()
        outImage = bImage.copy()

        #open a new image as a foreground image
        fileName = QtGui.QFileDialog.getOpenFileName(self, "Open Foreground Image",
                                        QtCore.QDir.currentPath())
        if fileName:
            aImage = QtGui.QImage(fileName)
            if aImage.isNull():
                QtGui.QMessageBox.information(self, "Foreground Image cannot load",
                                    "Cannot load %s." % fileName)
                return

        #Ensure both images are the same size
        if(aImage.height()!= bImage.height()):
            QtGui.QMessageBox.information(self, "Images do not have same height",
                    "Cannot load %s." % fileName)
            return

        if(aImage.width()!= bImage.width()):
            QtGui.QMessageBox.information(self, "Images do not have same width",
                    "Cannot load %s." % fileName)
            return

        #set the foreground image as the ImageLabel
        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(aImage))

        #get user input
```

```
hueLow = box.getHueLow()
hueHigh = box.getHueHigh()
satLow = box.getSatLow()

for row in range(0,aImage.height()):
   for col in range(0,aImage.width()):

      aPixel = aImage.pixel(row,col)
      bPixel = bImage.pixel(row,col)

      red = QtGui.qRed(aPixel)
      green = QtGui.qGreen(aPixel)
      blue = QtGui.qBlue(aPixel)
      alpha = 255

      #convert to HSV
      hsvPix = self.hsv(red,green,blue)

      if(hsvPix[0] <= hueHigh and hsvPix[0] >= hueLow and hsvPix[1] >= satLow):
         alpha = 0

      #cPixel = new chormaKey pixel
      cPixel = QtGui.qRgba(red,green,blue,alpha)
      aImage.setPixel(row,col,cPixel)

      #place over background image
      red = (QtGui.qRed(cPixel) * (QtGui.qAlpha(cPixel)/255)) +
            ((1 - (QtGui.qAlpha(cPixel)/255)) * QtGui.qRed(bPixel))

      green = (QtGui.qGreen(cPixel) * (QtGui.qAlpha(cPixel)/255)) +
            ((1 - (QtGui.qAlpha(cPixel)/255)) * QtGui.qGreen(bPixel))

      blue = (QtGui.qBlue(cPixel) * (QtGui.qAlpha(cPixel)/255)) +
            ((1 - (QtGui.qAlpha(cPixel)/255)) * QtGui.qBlue(bPixel))

      newPixelColor = QtGui.qRgba(self.clampInt(red), self.clampInt(green),
            self.clampInt(blue), self.clampInt(alpha))

      outImage.setPixel(row,col,newPixelColor)

self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```

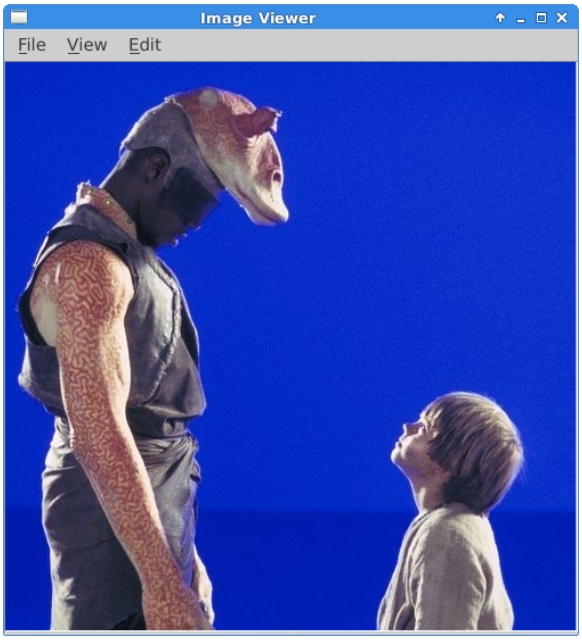
Code: imageDialogue.py

```
  def getHueLow(self):
    num,ok = QInputDialog.getInt(self,"Low Hue Range Input",
        "Enter lowest value of hue you wish to clear (0-360)")

    if ok:
      return num

  def getHueHigh(self):
    num,ok = QInputDialog.getInt(self,"High Hue Range Input",
        "Enter highest values of hue you wish to clear (0-360)")

    if ok:
      return num

  def getSatLow(self):
    num,ok = QInputDialog.getDouble(self,"Low Saturation Range Input",
        "Enter lowest values of saturation you wish to clear (0-1)")

    if ok:
      return num
```

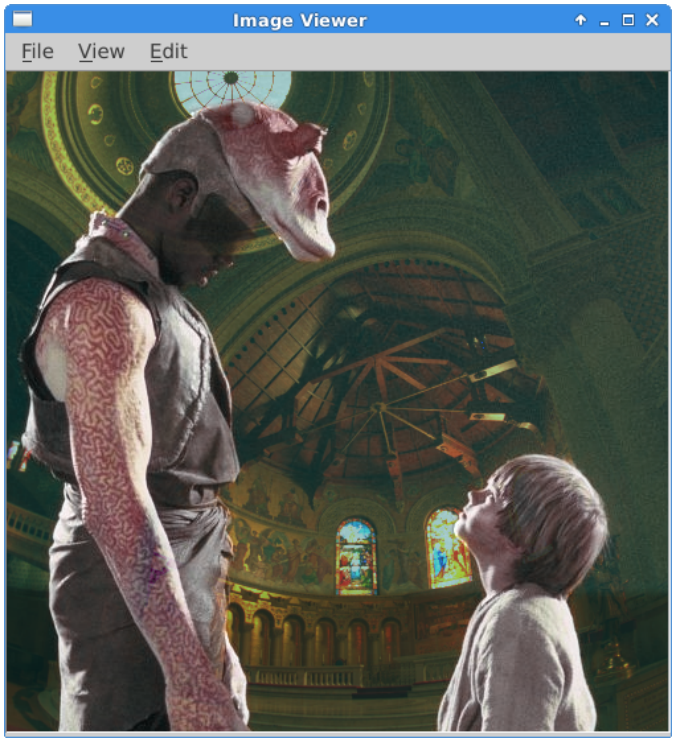
Color-Difference:



Background



Foreground



Final Image

Description: Extracts a matte from a blue-screen image using spill-suppression, color correction and composites it over a second image.

Code: imageViewer.py

```
    def colorDiff(self):
        '''Extracts a matte from a blue-screen image, color corrects it, and
        composites it over a second image.
        '''
        #store the imageLabel as the background image
        bImage = self.imageLabel.pixmap().toImage()
        outImage = bImage.copy()

        #open a new image as a foreground image
        fileName = QtGui.QFileDialog.getOpenFileName(self, "Open Foreground Image",
                                            QtCore.QDir.currentPath())
        if fileName:
            aImage = QtGui.QImage(fileName)
            if aImage.isNull():
                QtGui.QMessageBox.information(self, "Foreground Image cannot load",
                                    "Cannot load %s." % fileName)
                return

        #Ensure both images are the same size
        if(aImage.height()!= bImage.height()):
            QtGui.QMessageBox.information(self, "Images do not have same height",
                    "Cannot load %s." % fileName)
            return

        if(aImage.width()!= bImage.width()):
            QtGui.QMessageBox.information(self, "Images do not have same width",
                    "Cannot load %s." % fileName)
            return

        #set the foreground image as the ImageLabel
        self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(aImage))

        for row in range(0,aImage.height()):
            for col in range(0,aImage.width()):

                aPixel = aImage.pixel(row,col)
                bPixel = bImage.pixel(row,col)

                red = QtGui.qRed(aPixel)
```

```
            green = QtGui.qGreen(aPixel)
            blue = QtGui.qBlue(aPixel)
            alpha = 255

            #spill suppression
            if(blue > green):
               newBlue = green
            else:
               newBlue = blue

            #matte creation
            mAlpha = QtGui.qBlue(aPixel) - max(QtGui.qGreen(aPixel), QtGui.qRed(aPixel))
            mAlpha /= 255.0

            lPixel = QtGui.qRgba(red,green,newBlue,alpha)
            aImage.setPixel(row,col,lPixel)

            #image composite
            red = (mAlpha * QtGui.qRed(bPixel)) + QtGui.qRed(lPixel)
            green = (mAlpha * QtGui.qGreen(bPixel)) + QtGui.qGreen(lPixel)
            blue = (mAlpha * QtGui.qBlue(bPixel)) + QtGui.qBlue(lPixel)
            alpha = (mAlpha * QtGui.qAlpha(bPixel)) + QtGui.qAlpha(lPixel)

            newPixelColor = QtGui.qRgba(self.clampInt(red), self.clampInt(green),
                  self.clampInt(blue), self.clampInt(alpha))
            outImage.setPixel(row,col,newPixelColor)

      self.imageLabel.setPixmap(QtGui.QPixmap.fromImage(outImage))
```