APPLIED ROBOT TECHNOLOGY

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Image Aquisition: Assignment 1

Students :
Nourane BOUZAD
Alvaro ESTEBAN MUNOZ

Teacher :
Karl SKRETTING

October 26, 2021

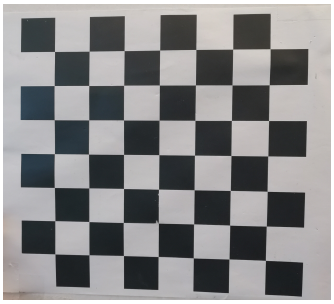
Contents

1 Introduction

In this assignment, we will see in a first part how to characterize an image and then a second part different way of coding. We will manage how to plot an image, filter it or even extract its histogram. These tools are very useful for image processing and allows to detect edges and some details of a picture.

2 Capture a test image

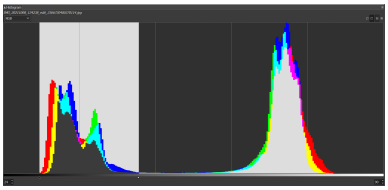
a) In this report, we will study the following image:



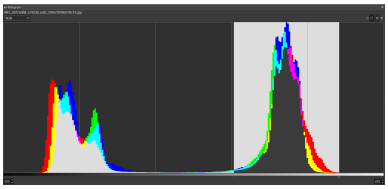
b) We can summarize the meta information in this table:

IMAGE	
Name	Value
Dimensions	2512*2266
Width	2512 pixels
Height	2266 pixels
Horizontal resolution	96 dpi
Vertical resolution	96 dpi
Bit depth	24
Resolution unit	
Color representation	sRGB
Compressed bits/pixels	0.95
CAMERA	
Exposure time	1/100 sec
Focal length	5mm
Max aperture	1.69
Flash mode	No flash

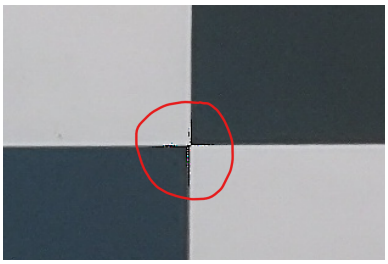
c) The value of the black pixels in RGB is in a range between 24 and 95 (using 8 bits per pixel). See histogram of the image below.



d) The value of the white pixels in RGB is in a range between 155 and 255 (using 8 bits per pixel). See histogram below.



e) We can assess the sharpness of an image by reducing the blur. In fact, to make an image sharper we can play on different parameters such as the image content, the spatial resolution, and the contrast of the image. We can see below the use of a sharping tool, it appears some colored pixels because by using this tool we need a more precise resolution.



f) Yes, because it doesn't need to be sharper in the center of the square but on the edges of each square, so we must have different values for black or white areas and the sharpness of the image.

g) Not all the straight lines on the scene need to be also straight on the image, normally, straight lines become sharp lines because of discretization effect, to represent a perfect straight line we would need an infinite amount of pixels to represent it.

h) The true side length of a black square is 2.24 cm and around 273 pixels. Yes they should be all the same side length, but it depends on how we take the picture and the resolution of the camera that took the picture.

i) To estimate the distance from camera (lens center) to the chessboard (center) we can use the Thales Theorem:

$$\frac{f}{d} = \frac{a}{i} \implies d = \frac{f \cdot i}{a}$$

- d: Distance between the image and the focal point
- f: Focal length
- a: Aperture of the camera
- i: Image size (vertically)

With the meta information we can estimate the following result:

$$d = \frac{0.5 \cdot 17.92}{0.277} = 32.35cm$$

3 Install and check Python

Everything is ok for this section.

4 Image Processing using Python and Qt

All sections are solved in the next code:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# ../ELE610/py3/appSimpleImageViewer.py
#
#  Simple program that uses Qt to display an image, only basic
   options.
#    File menu: Open File, and Quit
#    Scale menu: Scale Up, and Scale down
```

```
#  No status line at bottom.
#
# Karl Skretting, UiS, September - November 2018, November 2020

# Example on how to use file:
# (C:\...\Anaconda3) C:\..\py3> activate py38
# (py38) C:\..\py3> python appSimpleImageViewer.py

_appFileName = "appSimpleImageViewerj"
_author = " lvaro  Esteban Mu oz & Nourane Bouzad"
_version = "2021.00.01"

import sys
import numpy as np
import cv2
import qimage2ndarray
from PyQt5.QtCore import (QT_VERSION_STR, QRect, Qt)
from PyQt5.QtGui import QPixmap, QTransform, QImage
from PyQt5.QtWidgets import (QApplication, QWidget, QMainWindow,
        QGraphicsScene, QGraphicsView, QGraphicsPixmapItem,
           QAction, QFileDialog, QInputDialog)
try:
    from myTools import DBpath # my DropBox
    myPath = DBpath( 'm610' )  # path where KS have some images
except:
    myPath = './images'                # path where you may have
       some images
#end try

class MainWindow(QMainWindow):
    """MainWindow class for a simple image viewer."""
    def __init__(self, parent = None):
        """Initialize the main window object with title, location
            and size,
        an empty image (pixmap), empty scene and empty view
        """
        super().__init__(parent)
        self.setWindowTitle('Simple Image Viewer')
        self.setGeometry(150, 50, 1400, 800)  # initial window
           position and size
        #
        self.curItem = None
        self.pixmap = QPixmap()  # a null pixmap
        self.scene = QGraphicsScene()
        self.view = QGraphicsView(self.scene, parent=self)
        self.view.setGeometry(0, 20, self.width(), self.height()
           -20)
        self.initMenu()
        #
        return
```

```
    def initMenu(self):
        """Set up the menu for main window: File with Open and
           Quit, Scale with + and -."""
        qaOpenFile = QAction('openFile', self)
        qaOpenFile.setShortcut('Ctrl+O')
        qaOpenFile.setStatusTip('Open (image) File using dialog
           box')
        qaOpenFile.triggered.connect(self.openFile)
        qaSaveFile = QAction('saveFile', self)
        qaSaveFile.setShortcut('Ctrl+S')
        qaSaveFile.setStatusTip('Save image to a file')
        qaSaveFile.triggered.connect(self.saveFile)
        qaCloseWin = QAction('closeWin', self)
        qaCloseWin.setShortcut('Ctrl+Q')
        qaCloseWin.setStatusTip('Close and quit program')
        qaCloseWin.triggered.connect(self.closeWin)
        qaScaleUp = QAction('scaleUp', self)
        qaScaleUp.setShortcut('Ctrl++')
        qaScaleUp.triggered.connect(self.scaleUp)
        qaScaleDown = QAction('scaleDown', self)
        qaScaleDown.setShortcut('Ctrl+-')
        qaScaleDown.triggered.connect(self.scaleDown)
        qaCropImg = QAction('cropImage', self)
        qaCropImg.setShortcut('Ctrl+R')
        qaCropImg.setStatusTip('Crop image to a selected area')
        qaCropImg.triggered.connect(self.cropImg)
        qaGrayScale = QAction('grayScale', self)
        qaGrayScale.setShortcut('Ctrl+G')
        qaGrayScale.setStatusTip('Turn the image into a gray
           scale image')
        qaGrayScale.triggered.connect(self.grayScale)
        #
        mainMenu = self.menuBar()
        fileMenu = mainMenu.addMenu('&File')
        fileMenu.addAction(qaOpenFile)
        fileMenu.addAction(qaCloseWin)
        fileMenu.addAction(qaSaveFile)
        scaleMenu = mainMenu.addMenu('&Scale')
        scaleMenu.addAction(qaScaleUp)
        scaleMenu.addAction(qaScaleDown)
        editMenu = mainMenu.addMenu('&Edit')
        editMenu.addAction(qaCropImg)
        editMenu.addAction(qaGrayScale)
        return

# Methods for File menu
    def openFile(self):
        """Use the Qt file open dialog to select an image to open
            as a pixmap,
```

```
100             The pixmap is added as an item to the graphics scene
                    which is shown in the graphics view.
                The view is scaled to unity.
                """
                options = QFileDialog.Options()
                options |= QFileDialog.DontUseNativeDialog      # make
                    dialog appear the same on all systems
105             flt = "All jpg files (*.jpg);;All bmp files (*.bmp);;All
                    png files (*.png);;All files (*)"
                (fName, used_filter) = QFileDialog.getOpenFileName(parent
                    =self, caption="Open image file",
                      directory=myPath, filter=flt, options=options)
                #
                if (fName != ""):
110                   if self.curItem:
                            self.scene.removeItem(self.curItem)
                            self.curItem = None
                      #end if
                      self.pixmap.load(fName)
115                   # If the file does not exist or is of an unknown
                          format, the pixmap becomes a null pixmap.
                      if self.pixmap.isNull():
                            self.setWindowTitle('Image Viewer (error for
                                file %s)' % fName)
                            self.view.setGeometry( 0, 20, self.width(),
                                self.height()-20 )
                      else: # ok
120                         self.curItem = QGraphicsPixmapItem(self.pixmap)
                            self.scene.addItem(self.curItem)
                            self.setWindowTitle('Image Viewer: ' + fName)
                            self.view.setTransform(QTransform())  #
                                identity (for scale)
                      #end if
125             #end if
                return

          def closeWin(self):
                """Quit program."""
130             print("Close the main window and quit program.")
                self.close()
                return

          def saveFile(self):
135             options = QFileDialog.Options()

                flt = "All jpg files (*.jpg);;All bmp files (*.bmp);;All
                    png files (*.png);;All files (*)"
                (fName, used_filter) = QFileDialog.getSaveFileName(self,
                    caption="Save image file as",
                      directory=myPath, filter=flt, options=options)
```

```
140
            if (fName != ""):
                if self.pixmap.save(fName):
                    print(f"Saved image into file {fName}")
                else:
145                 print("Failed to save the image")

            return

    # Methods for Scale menu
150     def scaleUp(self):
            """Scale up the view by factor 2"""
            if not self.pixmap.isNull():
                self.view.scale(2,2)
            return
155
        def scaleDown(self):
            """Scale down the view by factor 0.5"""
            if not self.pixmap.isNull():
                self.view.scale(0.5,0.5)
160         return

    # Methods for Edit menu
        def cropImg(self):

165         # Ask for the parameters to crop the image
            x = QInputDialog.getInt(self, 'Crop area', 'Introduce x
               coordinate for the left top corner of the cropped area
               ')
            y = QInputDialog.getInt(self, 'Crop area', 'Introduce y
               coordinate for the left top corner of the cropped area
               ')
            height = QInputDialog.getInt(self, 'Crop area', '
               Introduce height for the area to crop')
            width = QInputDialog.getInt(self, 'Crop area', 'Introduce
                widht for the area to crop')
170
            # Create the new form for the image
            rect = QRect(x[0], y[0], width[0], height[0])
            if not self.pixmap.isNull():
                # Clear the scene
175             self.scene.removeItem(self.curItem)
                self.curItem = None

                # Copy the original image in the cropping rectangle
                cropped = self.pixmap.copy(rect)
180
                # Set the cropped image as the actual image
                self.pixmap = cropped
                self.curItem = QGraphicsPixmapItem(cropped)
```

```
                self.scene.addItem(self.curItem)

            return

        def grayScale(self):

            if not self.pixmap.isNull():
                image = self.pixmap.toImage()

                npImage = qimage2ndarray.rgb_view(image)
                npImage = cv2.cvtColor(npImage, cv2.COLOR_RGB2GRAY)
                image = qimage2ndarray.array2qimage(npImage)

                self.pixmap = QPixmap.fromImage(image)
                self.scene.removeItem(self.curItem)
                self.curItem = QGraphicsPixmapItem(self.pixmap)
                self.scene.addItem(self.curItem)

            return


# methods for 'slots'
        def resizeEvent(self, arg1):
            """Make the size of the view follow any changes in the
               size of the main window.
            This method is a 'slot' that is called whenever the size
               of the main window changes.
            """
            self.view.setGeometry( 0, 20, self.width(), self.height()
               -20 )
            return
#end class MainWindow

if __name__ == '__main__':
        print("%s: (version %s), path for images is: %s" % (
           _appFileName, _version, myPath))
        print("%s: Using Qt %s" % (_appFileName, QT_VERSION_STR))
        mainApp = QApplication(sys.argv)
        mainWin = MainWindow()
        mainWin.show()
        sys.exit(mainApp.exec_())
```

5 More Image Processing using Python and OpenCV

[1]:
```
# Import the packages
import cv2
import numpy as np
from math import floor
```

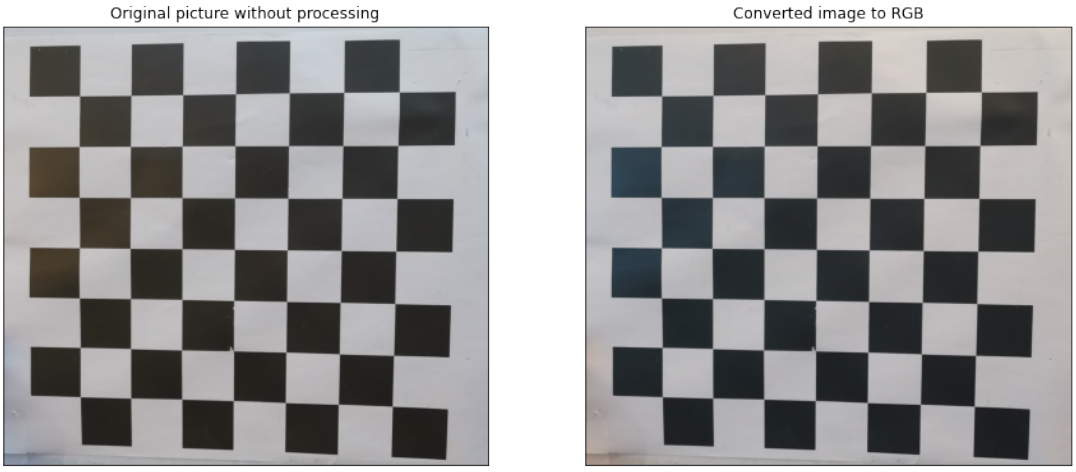
```
import matplotlib.pyplot as plt
```

5.1 Question a

[2]:
```
img = cv2.imread('./images/chessboard.jpg')
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

5.2 Question B

[3]:
```
plt.figure(figsize=[15,15])
plt.subplot(121),plt.imshow(img),plt.title('Original picture without␣
 ↪processing')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(img_rgb), plt.title('Converted image to␣
 ↪RGB')
plt.xticks([]), plt.yticks([])
plt.show()
```



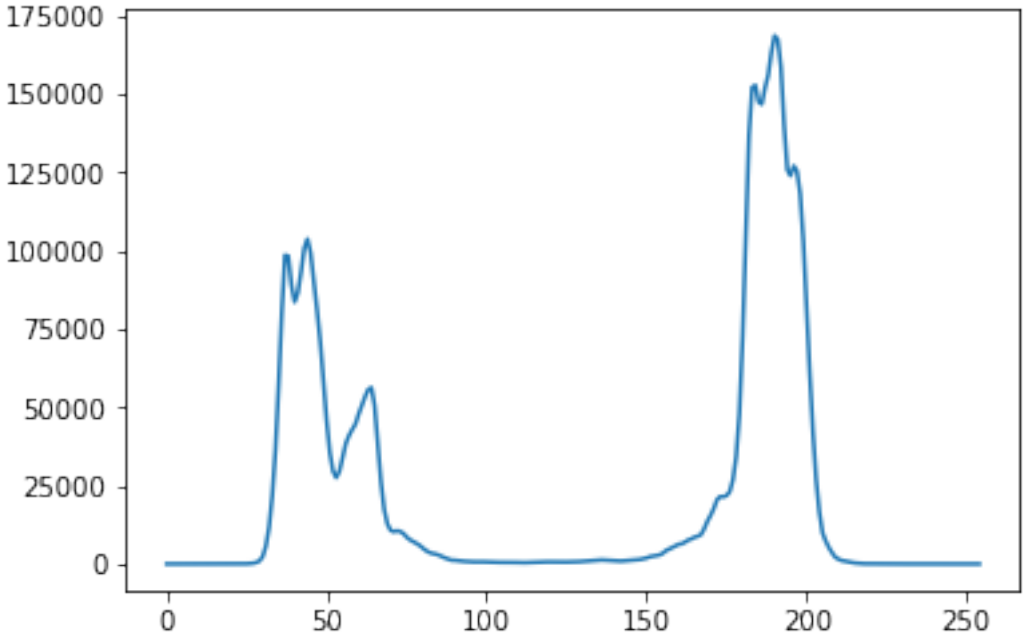
5.3 Question c

Open cv reads an image in BGR but matplotlib reads it in RGB so that's we need to preprocess the picture before and convert the image BGR to RGB.

5.4 Question d

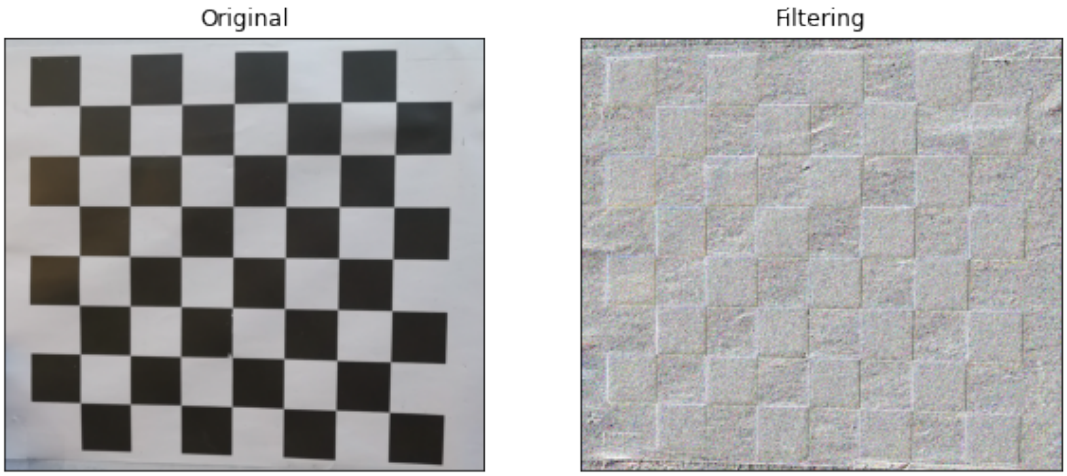
[4]:
```
img_gray = cv2.cvtColor(img, cv2.IMREAD_GRAYSCALE)

hist = cv2.calcHist([img_gray], [0], None, [255], [0, 255])
plt.plot(hist)
```

[4]: [<matplotlib.lines.Line2D at 0x27538a408b0>]



5.5 Question e

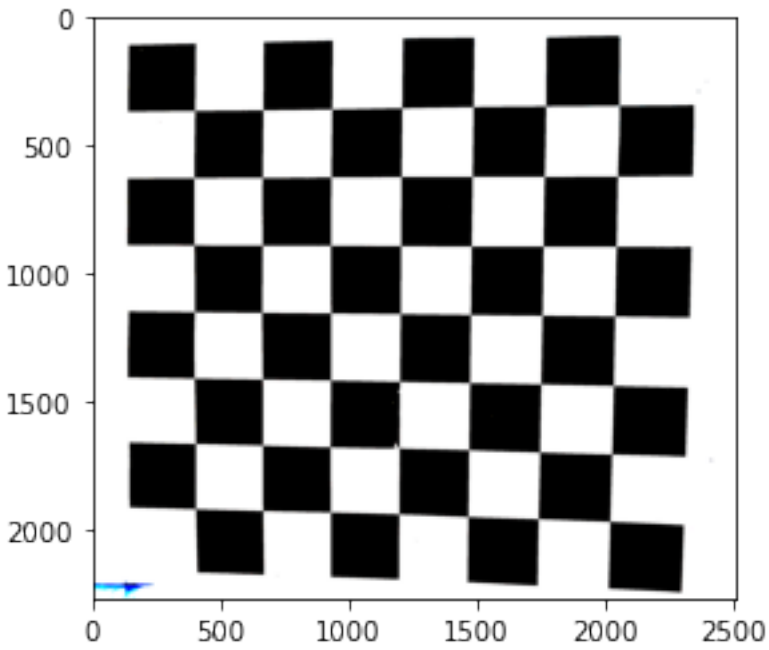
[5]:
```
# Sobel filter

#Introducing the kernel on each directions
kernelx = np.array([[1, 0, -1],[2, 0, -2],[1, 0, -1]])
kernely = np.array([[1, 2, 1],[0, 0, 0],[-1, -2, -1]])

# Filtering with the different kernels
conv_x = cv2.filter2D(img, -1, kernelx)
conv_y = cv2.filter2D(img, -1, kernely)

#Calculationg the gradient
sobelfilter = (conv_x*conv_x+conv_y*conv_y)**(1/2)

#Plot of the results
plt.figure(figsize=[10,10])
plt.subplot(121),plt.imshow(img,cmap='gray'),plt.title('Original')
plt.xticks([]), plt.yticks([])
plt.subplot(122),plt.imshow(sobelfilter,cmap='gray'),plt.
 ↪title('Filtering')
plt.xticks([]), plt.yticks([])
plt.show()
```



5.6 Question f

[6]:
```
thresh = 128
im_bin = cv2.threshold(img_gray, thresh, 255, cv2.THRESH_BINARY)[1]
plt.imshow(im_bin)
```

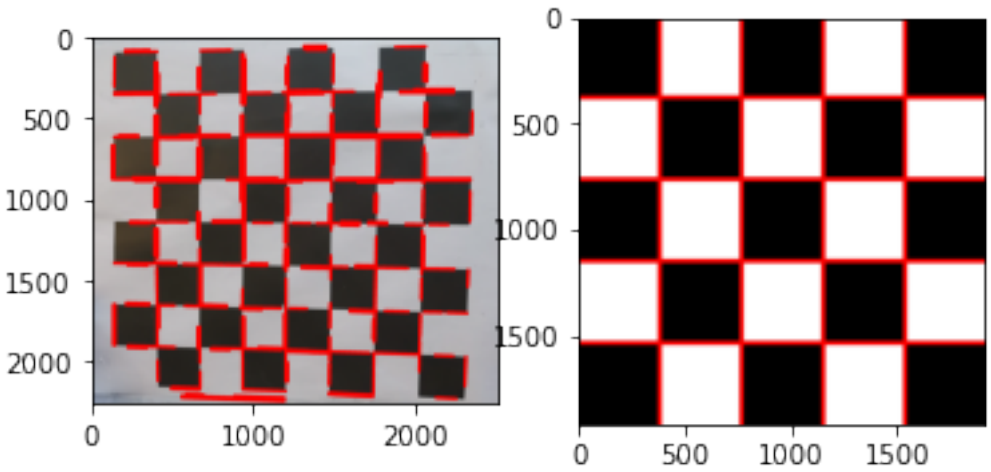
[6]:



5.7 Question g

[9]:
```
img_gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img_gray,50,150,apertureSize = 3)

lines = cv2.HoughLinesP(edges,1,np.pi/
 ↪180,100,minLineLength=100,maxLineGap=10)
for line in lines:
    x1,y1,x2,y2 = line[0]
    cv2.line(img,(x1,y1),(x2,y2),(255,0,0),20)
cv2.imwrite('linedetected.jpg',img)

clearchessboard = cv2.imread('./images/chessboard.png')
img_gray = cv2.cvtColor(clearchessboard,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(img_gray,50,150,apertureSize = 3)

lines = cv2.HoughLinesP(edges,1,np.pi/
 ↪180,100,minLineLength=100,maxLineGap=10)
for line in lines:
    x1,y1,x2,y2 = line[0]
    cv2.line(clearchessboard,(x1,y1),(x2,y2),(255,0,0),20)
cv2.imwrite('linedetected2.jpg',clearchessboard)

res = cv2.imread('linedetected.jpg')
res2 = cv2.imread('linedetected2.jpg')

plt.figure(figsize=(10,10))
plt.subplot(121),plt.imshow(res)
plt.subplot(122),plt.imshow(res2)
```

[9]:
```
(<AxesSubplot:>, <matplotlib.image.AxesImage at 0x275388a01f0>)
```



Here, we used the chessboard picture that we took manually and a clear image of a chessboard. We can see that with our image the lines could be a bit blurred and sometimes not really on the edges. It is because the image taken is not really centered and the pixel intensity less contrasted than in the 'clear' chessboard where the lines are straight and well put on the edges.