ELEC 5727 Comp Vis & Image Proc Acc

Julian Abbott-Whitley

Fall 2019

Assignment 1

Analysis:

This assignment was completed by utilizing functionality from the OpenCV library, Numpy library, and pythons native math library. Using OpenCV, it was possible to load a test color image, (Figure 1), into a 3 dimensional Numpy array, (see Figure 2 below).

Figure 1: Test Image

```
In [127]: img = cv2.imread('tetons_color.jpg')
     ...: # Show original image
     ...: cv2.imshow("Grand Tetons", img)
     ...: # Wait for user to hit key
     ...: cv2.waitKey(0)
     ...: # Destroy current image window
     ...: cv2.destroyAllWindows()

In [128]: np.shape(img)
Out[128]: (380, 676, 3)
```

Figure 2: Sample Code

Once completed, an array of all zeros was created using the dimensions of the test image, (see figure 3). This array of zeros will be used to create a mask for clipping a circle from the original test image. Doing this required a few static variables to define the center point of the circle as well as the circle radius.

```
In [130]: shape = np.shape(img)
     ...: mask = np.zeros(shape, dtype = np.uint8)
     ...:
     ...: # Radius value for circle
     ...: r = 175
     ...: # Center x variable of circle
     ...: centerX = 180
     ...: # Center Y variable of circle
     ...: centerY = 300
```

Figure 3: Sample Code

With the zero array generated and the circle defined, the next step was to iterate through the mask array and set each pixel value based on its location within or outside of the defined circle. Pixels outside the mask were given a value of zero while values inside the mask were given a value of 1, see Figure 4.

```
In [131]: for i in range(np.shape(img)[0]):
     ...:     for j in range(np.shape(img)[1]):
     ...:         # Calculate the current pixel distance from the circle center point
     ...:         d = int(math.sqrt(((i-centerX)**2) + ((j-centerY)**2)))
     ...:         # If the current pixel is inside of the circle, set pixel value to 1
     ...:         if d < r:
     ...:             # Create a circle mask
     ...:             # Pixels outside of the mask will be given a value of 0
     ...:             # Values inside the mask will be given a value of 1
     ...:             mask[i, j, :] = 1
```

Figure 4: Original Test Image

The results from this process can be visualize in Figure 5. This output png file was generated by multiplying the entire mask by a value of 255. Values outside of the mask remained zeros, (i.e. black) while values inside the mask were given a value of 255, (i.e. white). However, this array was not used to clip the final output. The mask with values of zero and one was utilized to create the clipped output.

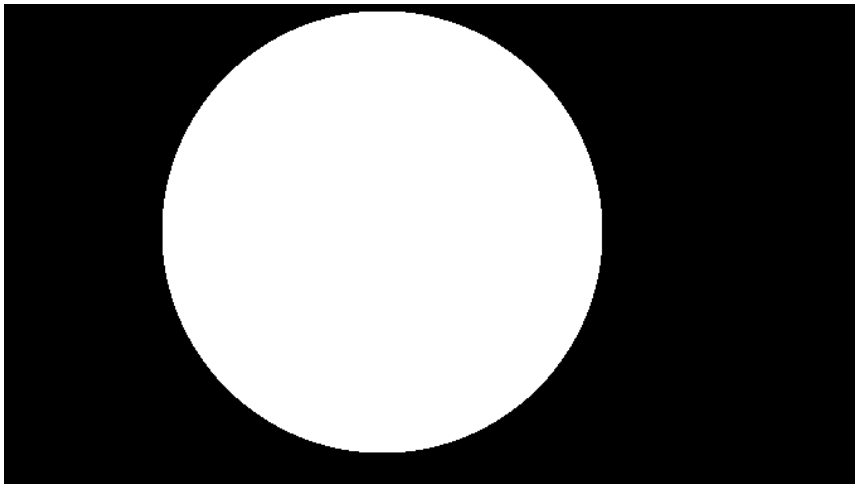

Figure 5: Mask Image

Finally, the mask containing zeros and ones was used to clip the final image by simply multiplying the test image by the mask matrix. Pixels within the circle were multiplied by a value of one while pixels outside the circle were multiplied by a value of zero. See source code shown in figure 6. The final clipped image is show in figure 7.

```
In [152]: cv2.imshow("Grand Tetcons", img * mask)
     ...: cv2.waitKey(0)
     ...: cv2.destroyAllWindows()
     ...: cv2.imwrite('tetons_clipped.png',img * mask)
Out[152]: True
```

Figure 6

Figure 7: Clipped Test Image

Conclusion:

This assignment demonstrates how an image is essentially a 2 or 3-dimensional array, where by each pixels value is represented by each element within the array. Using a few programming tools, we can easily manipulate the pixel values of an image in order to edit the output image. In addition to a few custom libraries, this assignment utilized For Loops and matrix multiplication in order to generate the final output.