1.4.8: Masking

Topic Progress: (https://gurus.pyimagesearch.com/topic/translation/) (https://gurus.pyimagesearch.com/topic/rotation/) (https://gurus.pyimagesearch.com/topic/resizing/) (https://gurus.pyimagesearch.com/topic/flipping/) (https://gurus.pyimagesearch.com/topic/cropping/) (https://gurus.pyimagesearch.com/topic/image-arithmetic/) (https://gurus.pyimagesearch.com/topic/bitwise-operations/) (https://gurus.pyimagesearch.com/topic/masking/) (https://gurus.pyimagesearch.com/topic/splitting-and-merging-channels/)

← Back to Lesson (https://gurus.pyimagesearch.com/lessons/basic-image-processing/)

In the previous section we explored bitwise operations (https://gurus.pyimagesearch.com/topic/bitwise-operations/), a very common technique that is heavily used in computer vision and image processing.

And as I hinted at previously, we can use a combination of both bitwise operations and masks to construct ROIs that are *non-rectangular*. This allows us to extract regions from images that are of *completely arbitrary shape*.

Put more simply, **a mask allows us to focus only on the portions of the image that interests us.**

For example, let's say that we were building a computer vision system to recognize faces. The only part of the image we are interested in finding and describing are the parts of the image that contain faces — we simply don't care about the rest of the content of the image. Provided that we could find the faces in the image, we might construct a mask to show *only* the faces in the image.

Objectives

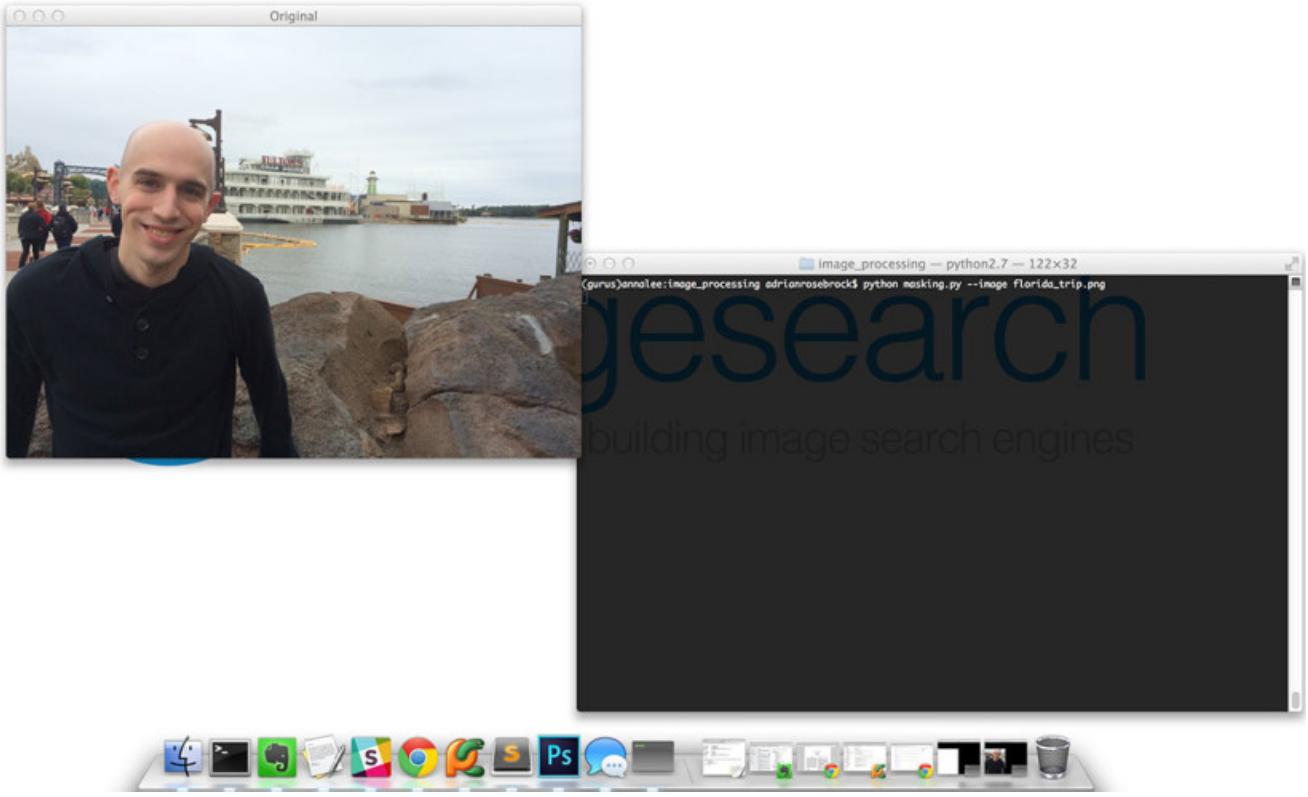
By the end of this topic you should be able to:

1. Leverage masks to extract *rectangular* regions from images, similar to cropping.
2. Leverage masks to extract *non-rectangular* and *arbitrarily shaped* regions from images, which basic cropping cannot accomplish.

Masking

Let's make the notion of extracting *rectangular* vs. *non-rectangular* regions a little more concrete.

In **Figure 1** below we have our original image:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/masking_original.jpg)

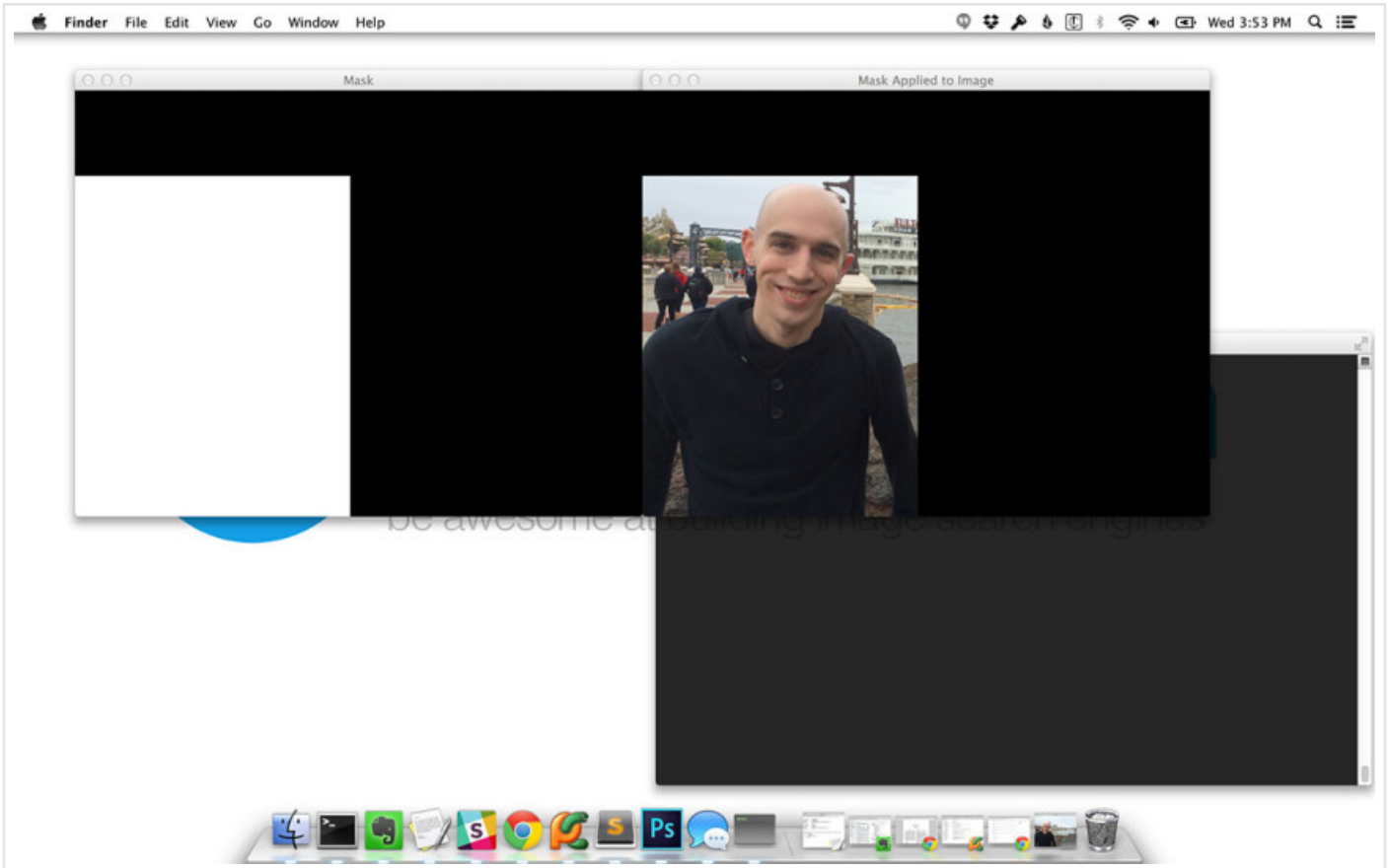
FIGURE 1: OUR ORIGINAL IMAGE.

But I'm not interested in the *entire* image — I'm only interested in the part of the image that contains myself (narcissistic, I know).

To extract myself from this image we could simply apply a cropping to extract that region of the image.

Feedback

Or, we could instead apply a mask to the image:



(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/masking_rectangular.jpg)

FIGURE 2: UTILIZING A RECTANGULAR MASK TO EXTRACT MYSELF FROM AN IMAGE.

Feedback

The image on the *Left* is our mask — a white rectangle at the bottom-left region of the image. By applying our mask to our original image, we arrive at the image on the *Right*. By using our rectangle mask, we have focused only on the region of image that contains the person.

Let's examine the code to accomplish the masking in **Figure 2**:

masking.py Python

```
# import the necessary packages
import numpy as np
import argparse
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-i", "--image", required=True, help="Path to the image")
args = vars(ap.parse_args())

# load the image and display it
image = cv2.imread(args["image"])
cv2.imshow("Original", image)

# Masking allows us to focus only on parts of an image that interest us.
# A mask is the same size as our image, but has only two pixel values,
# 0 and 255. Pixels with a value of 0 are ignored in the orignal image,
# and mask pixels with a value of 255 are allowed to be kept. For example,
# let's construct a rectangular mask that displays only the person in
# the image
mask = np.zeros(image.shape[:2], dtype="uint8")
cv2.rectangle(mask, (0, 90), (290, 450), 255, -1)
cv2.imshow("Mask", mask)

# Apply our mask -- notice how only the person in the image is cropped out
masked = cv2.bitwise_and(image, image, mask=mask)
cv2.imshow("Mask Applied to Image", masked)
cv2.waitKey(0)
```

On **Lines 1-13** we import the packages we need, parse our arguments, and load our image.

Feedback

We then construct a NumPy array, filled with zeros, with the same width and height as our original image on **Line 21**. As I mentioned in **Module 1.4.5** (https://gurus.pyimagesearch.com/topic/cropping/) on Cropping, we'll be exploring computer vision and machine learning methods to *automatically* detect objects/people in images later in this course, but for the time being we'll be using our *a priori* knowledge of our example image: we know that the region we want to extract is in the bottom-left corner of the image. **Line 22** draws a white rectangle on our mask, which corresponds to the region that we want to extract from our original image.

Remember reviewing the `cv2.bitwise_and` function in the **Section 1.4.7** (https://gurus.pyimagesearch.com/topic/bitwise-operations/)? Well, it turns out that this function is used extensively when applying masks to images.

We apply our mask on **Line 26** using the `cv2.bitwise_and` function. The first two parameters are the image itself (i.e., the images we want to apply the bitwise operation to). However, the important part of this function is the `mask` keyword. When supplied, the bitwise AND is `True` when the pixel values of the input images are equal *and* the mask is non-zero at each *(x, y)*-coordinate. In this case, only pixels that

After applying our mask, we display the output on **Line 27 and 28**, which you can see in **Figure 2** above. Using our rectangular mask we were able to extract only the region of the image that contains the person and ignore rest.

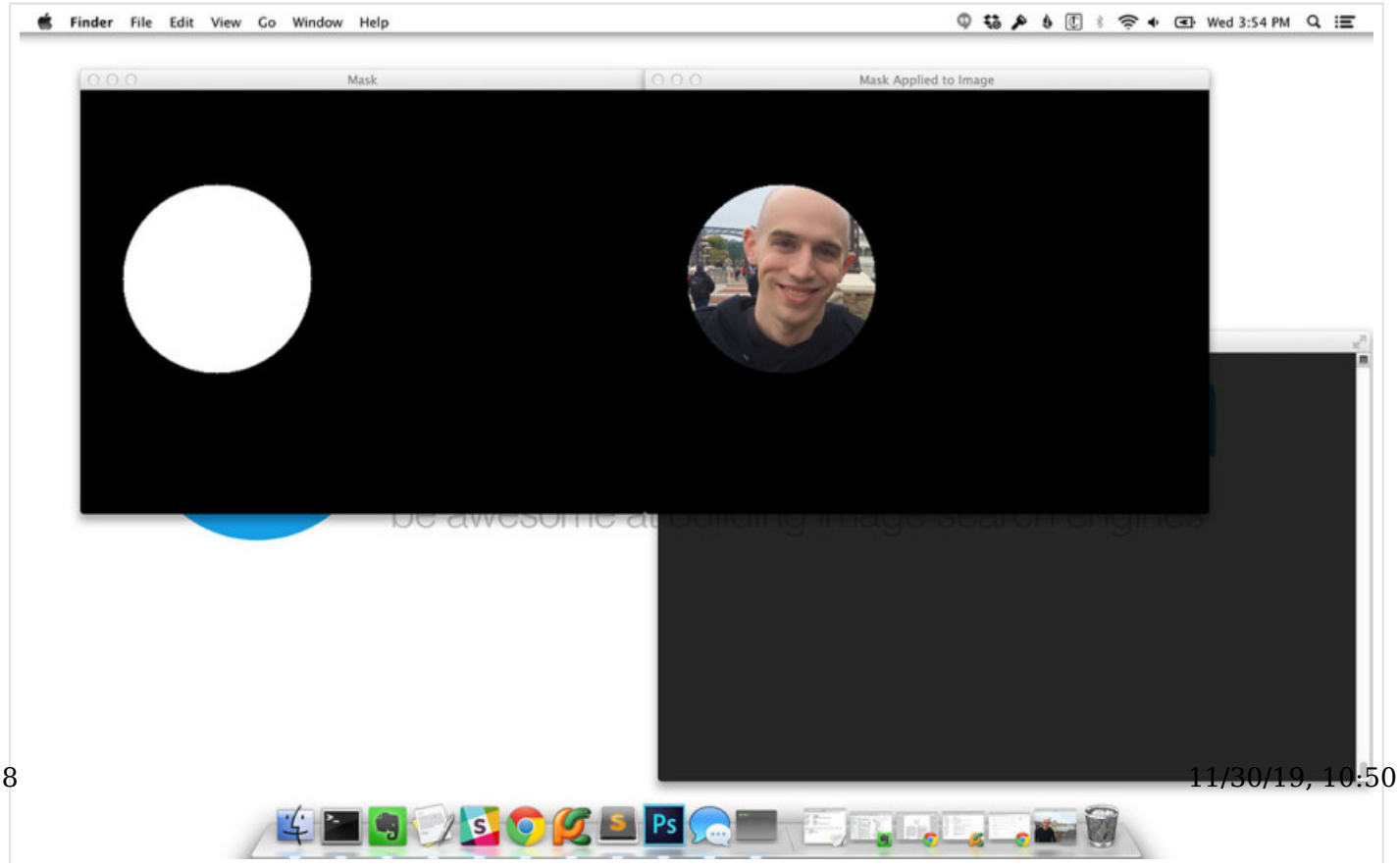
Let's look at another example, but this time using a *non-rectangular* mask:

masking.py Python

```
30 # Now, let's make a circular mask with a radius of 100 pixels and apply the
31 # mask again
32 mask = np.zeros(image.shape[:2], dtype="uint8")
33 cv2.circle(mask, (145, 200), 100, 255, -1)
34 masked = cv2.bitwise_and(image, image, mask=mask)
35 cv2.imshow("Mask", mask)
36 cv2.imshow("Mask Applied to Image", masked)
37 cv2.waitKey(0)
```

On **Line 32** we re-initialize our mask to be filled with zeros and the same dimensions as our beach image. Then, we draw a white circle on our mask image, starting at the center of my face with a radius of 100 pixels. Applying the circular mask is then performed on **Line 34**, again using the `cv2.bitwise_and` function.

The results of our circular mask can be seen in **Figure 3** below:



Feedback

(https://gurus.pyimagesearch.com/wp-content/uploads/2015/03/masking_circular.jpg)

FIGURE 3: EXTRACTING THE FACE FROM AN IMAGE USING A CIRCULAR MASK INSTEAD OF A RECTANGULAR ONE.

Here we can see that our circle mask on shown the *Left*, and the application of the mask on the *Right*. Unlike the output from **Figure 2** above when we extracted a rectangular region, this time we have extracted a circular region that corresponds to only my face in the image.

Furthermore, this approach can be used to extract regions from an image of arbitrary shape.

Summary

Right now masking may not seem very interesting. But we'll return to it once we start computing histograms in **Module 1.11** (https://gurus.pyimagesearch.com/lessons/1-11-histograms/)and **Module 4**, where we start to explore machine learning and image classification.

Again, the key point of masks is that they allow us to focus our computation *only* on regions of the image that interest us. Being able to focus our computations on regions that interest us has *dramatic impacts* when we explore topics such as machine learning, image classification, and building image search engines.

For example, let's assume that we wanted to build a system to classify the species of the flower. In reality, we are probably only interested in the color and texture of the flower petals to perform the classification. But since we are capturing the photo in a natural environment we'll also have many other regions in our image, including dirt from the ground, insects, and other flowers crowding our view — how are we going to quantify and classify *just* the flower we are interested in? As we'll see, the answer is ***masks***.

Downloads:

Download the Code (https://gurus.pyimagesearch.com/protected/code/computer_vision_basics/masking.zip)

Quizzes	Status

```
1  Masking Quiz (https://gurus.pyimagesearch.com/quizzes/masking-quiz/)
```

← Previous Topic (https://gurus.pyimagesearch.com/topic/bitwise-operations/) Next Topic → (https://gurus.pyimagesearch.com/topic/splitting-and-merging-channels/)

Course Progress

Ready to continue the course?

Click the button below to **continue your journey to computer vision guru**.

I'm ready, let's go! (/pyimagesearch-gurus-course/)

Resources & Links

- PyImageSearch Gurus Community (https://community.pyimagesearch.com/)
- PyImageSearch Virtual Machine (https://gurus.pyimagesearch.com/pyimagesearch-virtual-machine/)
- Setting up your own Python + OpenCV environment (https://gurus.pyimagesearch.com/setting-up-your-python-opencv-development-environment/)
- Course Syllabus & Content Release Schedule (https://gurus.pyimagesearch.com/course-syllabus-content-release-schedule/)
- Member Perks & Discounts (https://gurus.pyimagesearch.com/pyimagesearch-gurus-discounts-perks/)
- Your Achievements (https://gurus.pyimagesearch.com/achievements/)
- Official OpenCV documentation (http://docs.opencv.org/index.html)

Your Account

- Account Info (https://gurus.pyimagesearch.com/account/)
- Support (https://gurus.pyimagesearch.com/contact/)
- Logout (https://gurus.pyimagesearch.com/wp-login.php?action=logout&redirect_to=https%3A%2F%2Fgurus.pyimagesearch.com%2F&_wpnonce=5736b21cae)

Search

Feedback