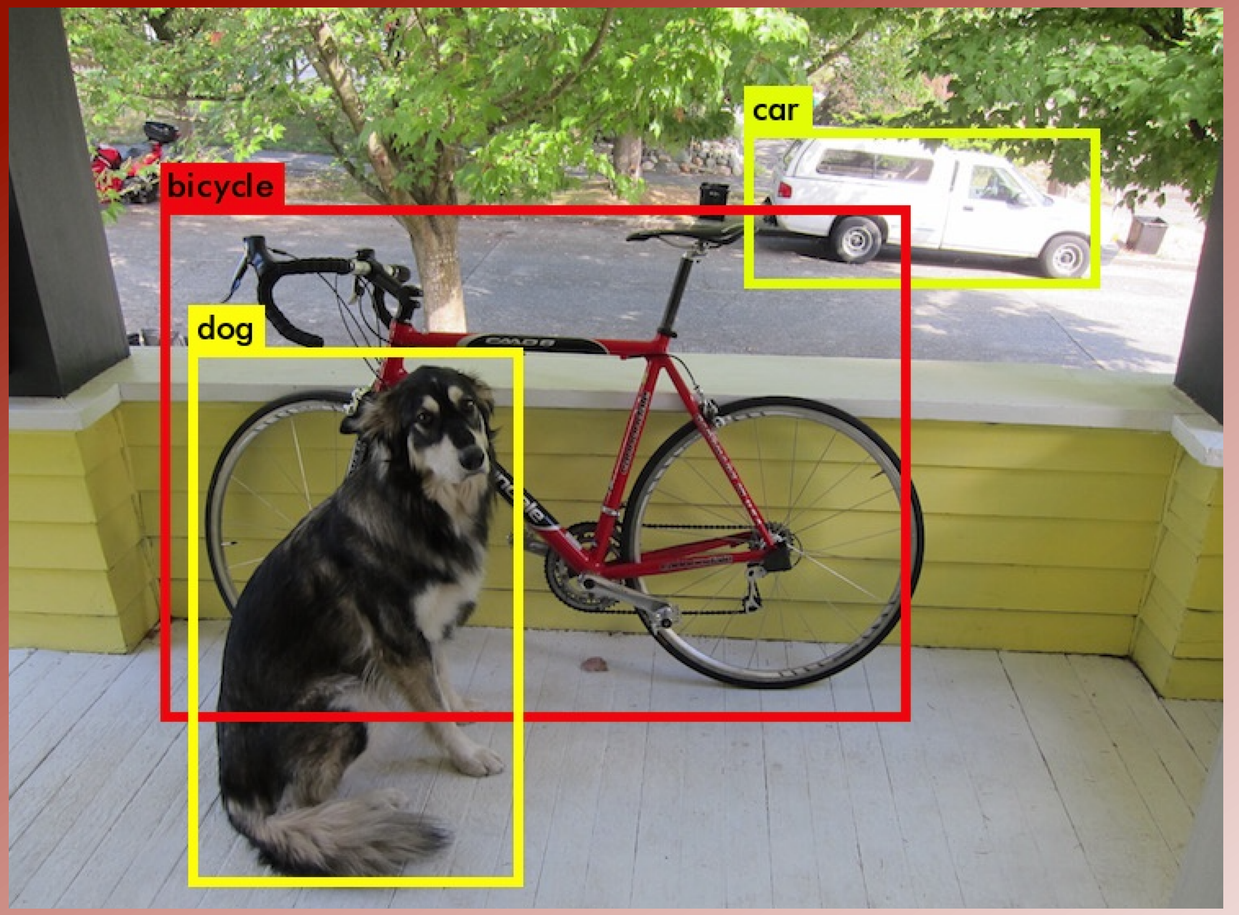


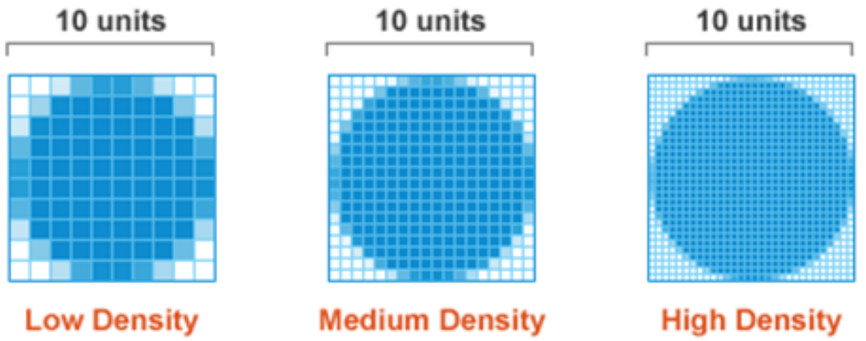
Lecture 12: Image Processing

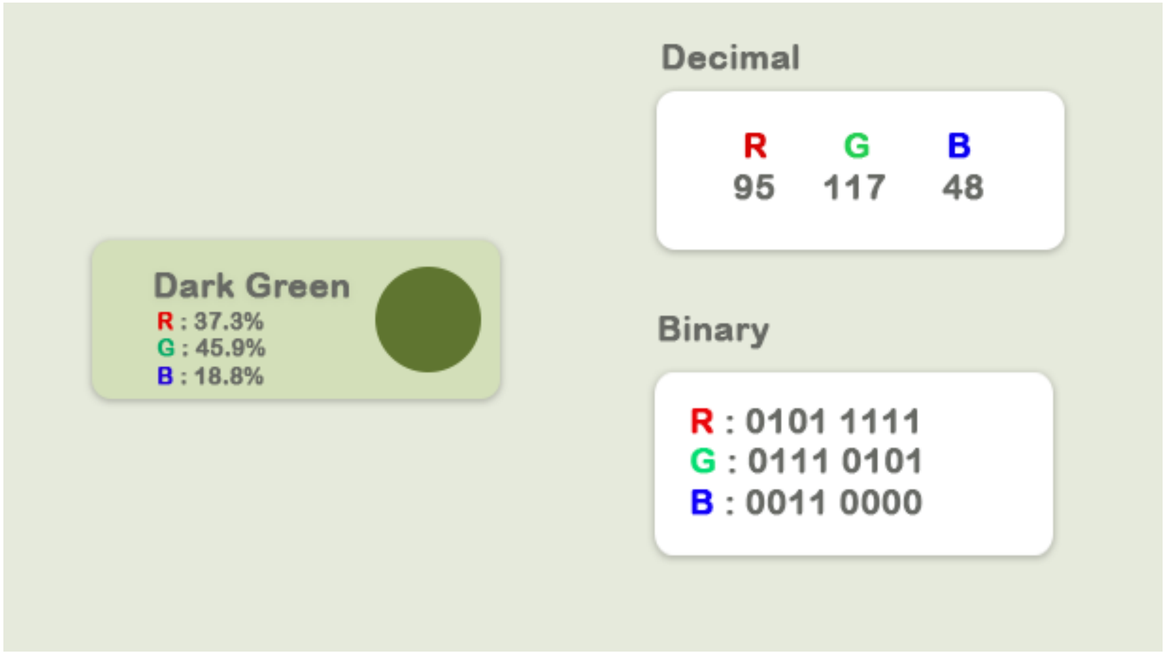
ENAE 380 Flight Software Systems
October 21, 2024

Examples

- Basic Concepts
- OpenCV Basics
- Optimal Mark Recognition
- Autostereogram
- Automatic segmentation

Pixels and Color





How to grayscale

- Get the red, green, and blue values of a pixel
- Use fancy math to turn those numbers into a single gray value
- Replace the original red, green, and blue values with the new gray value

```
Gray = (Red + Green + Blue) / 3
```

```
For Each Pixel in Image {

   Red = Pixel.Red
   Green = Pixel.Green
   Blue = Pixel.Blue

   Gray = (Red + Green + Blue) / 3

   Pixel.Red = Gray
   Pixel.Green = Gray
   Pixel.Blue = Gray

}
```

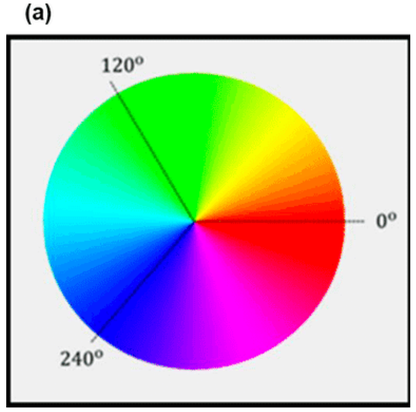
Method 1: Averaging

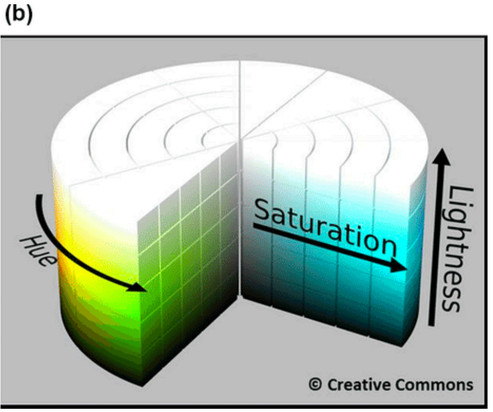
```
Gray = (Red + Green + Blue) / 3
```

Method 2: “Luminance”


```
Gray = (Red * 0.3 + Green * 0.59 + Blue * 0.11)
```

Method 3: Desaturation





HSL: hue, saturation, lightness

```
Gray = ( Max(Red, Green, Blue) + Min(Red, Green, Blue) ) / 2
```

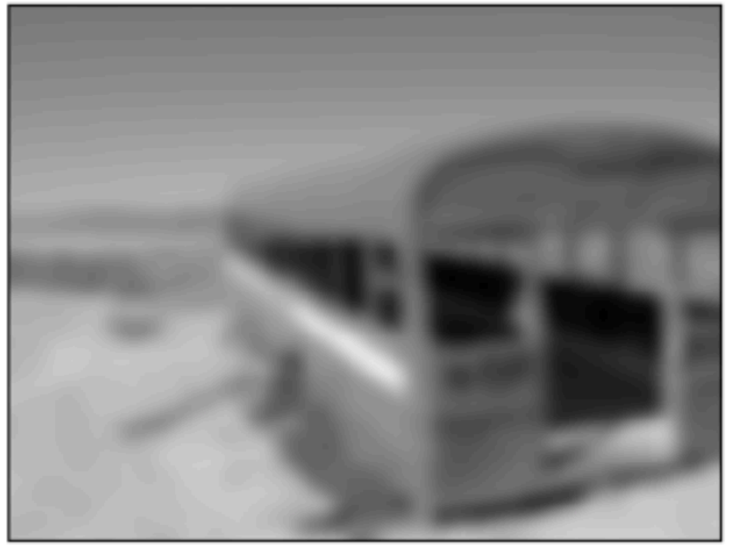
Averaging

Luminence

HSL

Filtering

- Point operations are limited
- They cannot accomplish tasks like sharpening or smoothing



Higher Dimensional Convolutions

0	0	0	0	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	1/9	1/9	1/9	0
0	0	0	0	0

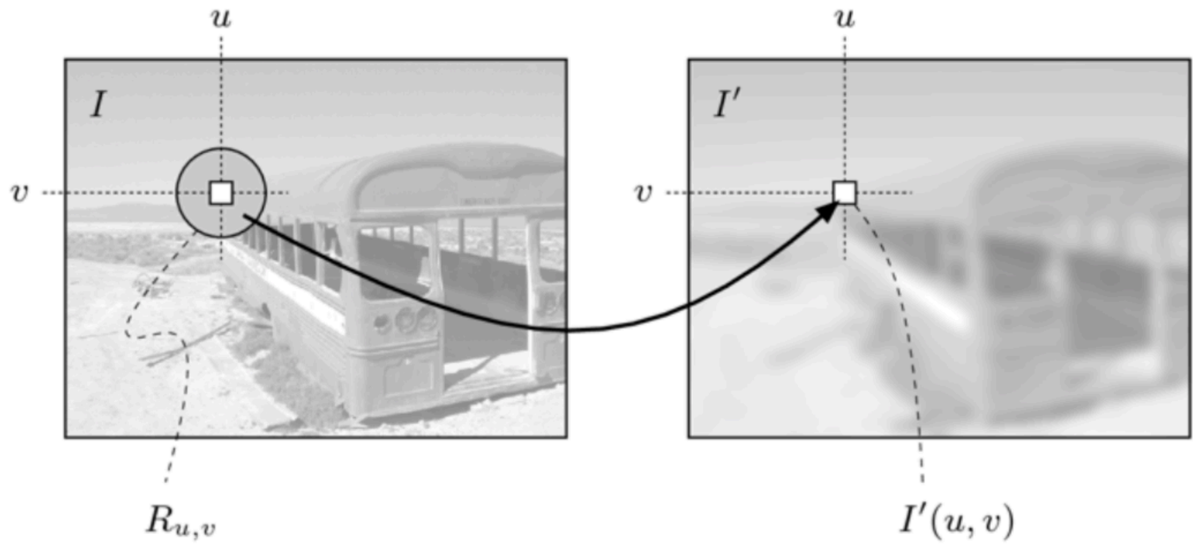
Higher Dimensional Convolutions

0	0	0	0	0
0	0	0	0	0
0	-1	1	0	0
0	0	0	0	0
0	0	0	0	0

Smoothing by averaging

- Replace each pixel by the average of its neighboring pixels
- Assume a 3x3 neighborhood:

$$I'(u,v) \leftarrow \frac{p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8}{9}$$



45	60	98	127	132	133	137	133
46	65	98	123	126	128	131	133
47	65	96	115	119	123	135	137
47	63	91	107	113	122	138	134
50	59	80	97	110	123	133	134
49	53	68	83	97	113	128	133
50	50	58	70	84	102	116	126
50	50	52	58	69	86	101	120

$f(x,y)$

$*$

0.1	0.1	0.1
0.1	0.2	0.1
0.1	0.1	0.1

$h(x,y)$

$=$

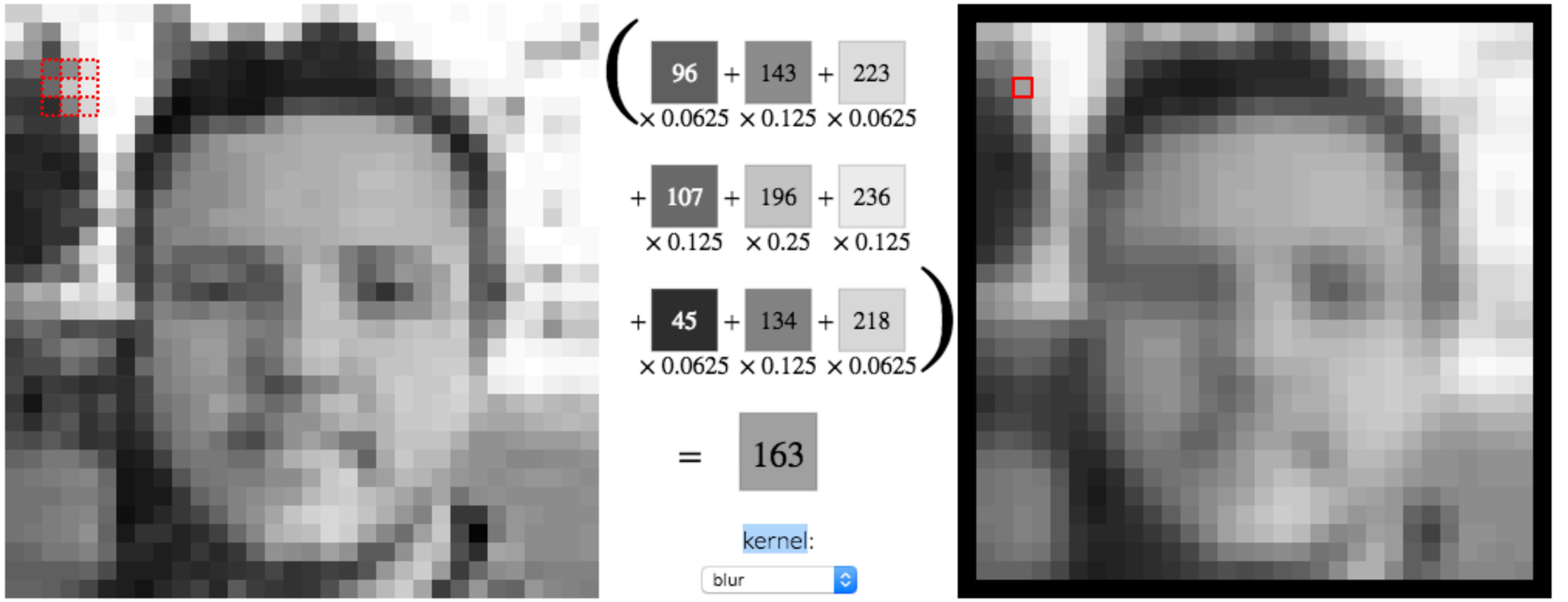
69	95	116	125	129	132
68	92	110	120	126	132
66	86	104	114	124	132
62	78	94	108	120	129
57	69	83	98	112	124
53	60	71	85	100	114

$g(x,y)$

Blur

$$\begin{pmatrix} 0.0625 & 0.125 & 0.0625 \\ 0.125 & 0.25 & 0.125 \\ 0.0625 & 0.125 & 0.0625 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.



input image

output image

Sharpen

sharpen

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

Below, for each 3x3 block of pixels in the image on the left, we multiply each pixel by the corresponding entry of the kernel and then take the sum. That sum becomes a new pixel in the image on the right. Hover over a pixel on either image to see how its value is computed.

input image

(? × 0 + 72 × -1 + 44 × 0
+ ? × -1 + 55 × 5 + 20 × -1
+ ? × 0 + 65 × -1 + 49 × 0)

= ?

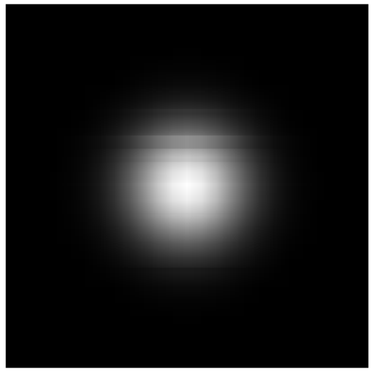
kernel:

sharpen

output image

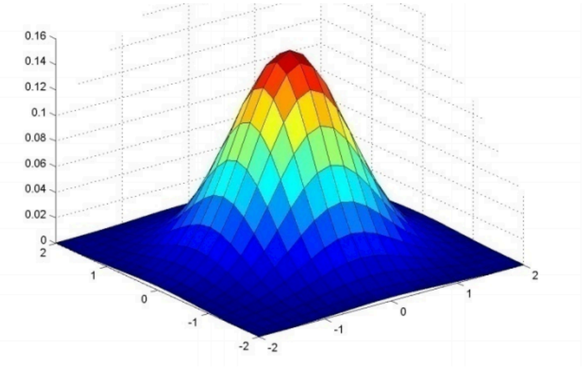
Smoothing with a Gaussian

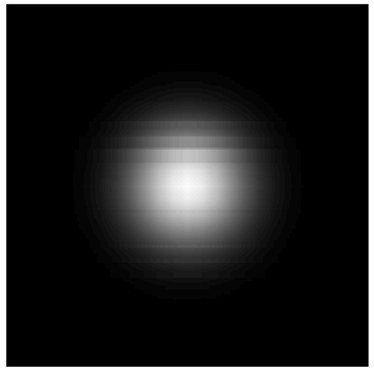
- Single point of light looks like a fuzzy blob, but averaging process would give a little square
- To eliminate edge effects, weight contribution of neighborhood pixels according to their closeness to center



Gaussian kernel

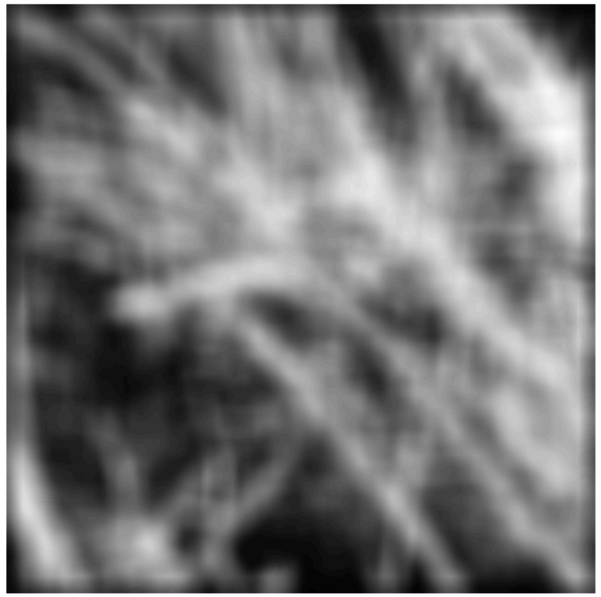
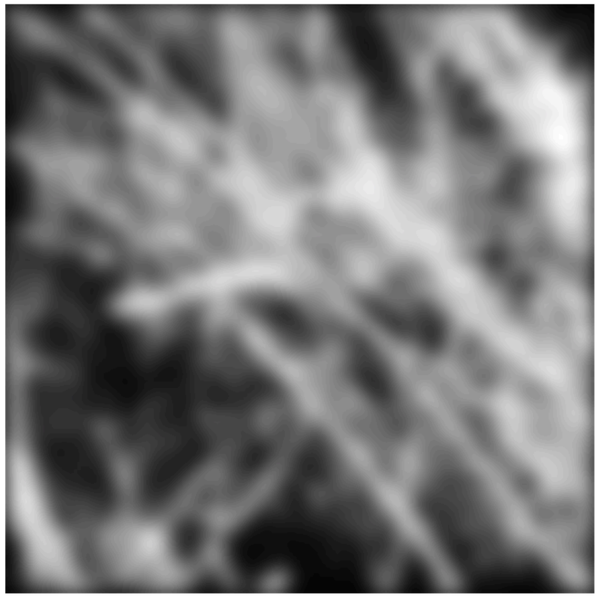
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$



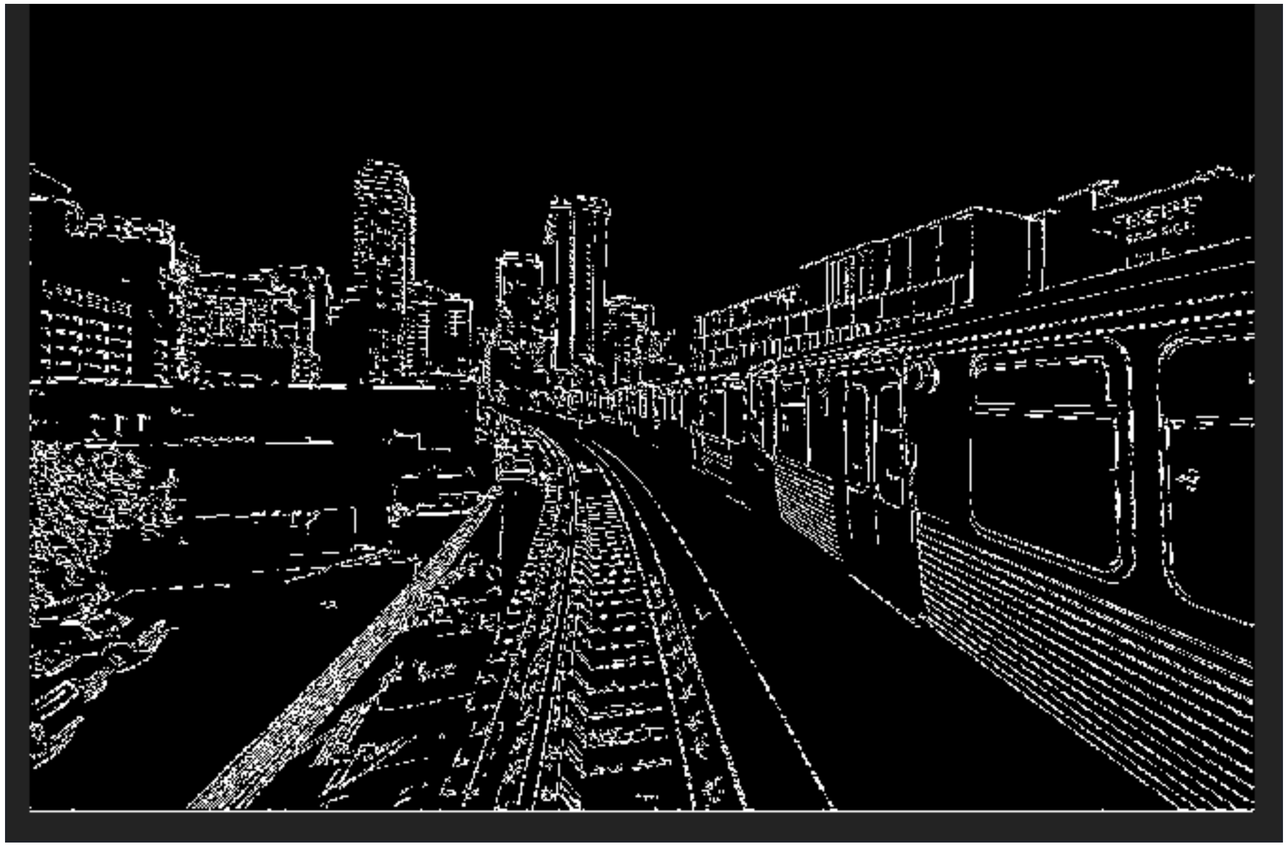


0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

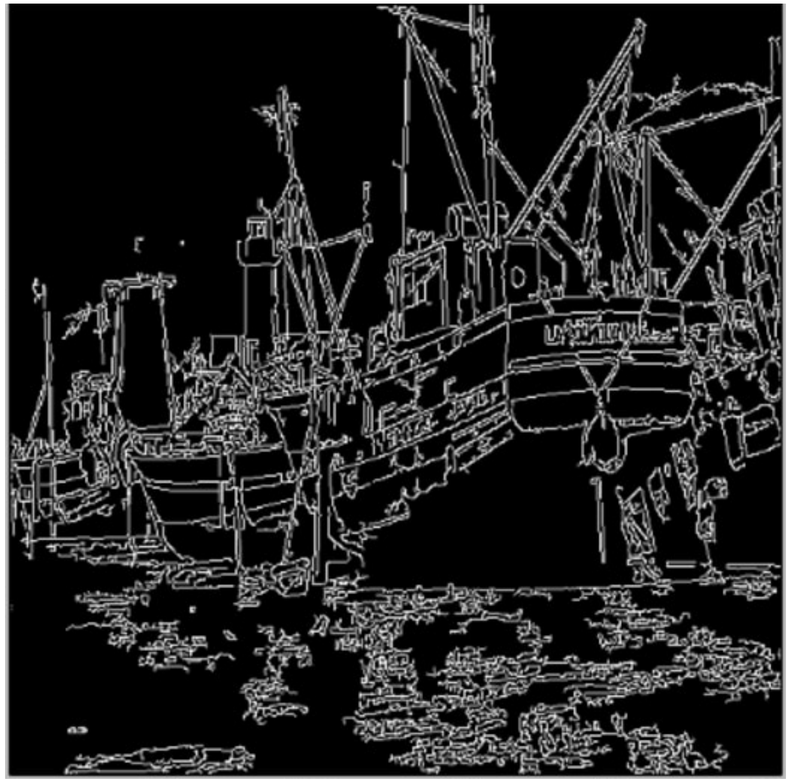




Edge Detection: Canny Edge Detector



Edge Detection: Canny Edge Detector

- Gaussian filter to remove noise
- Find intensity gradients
- Apply non-maximum suppression to remove spurious responses
- Apply double threshold to determine potential edges
- Track edge by hysteresis



```python
import cv2
import matplotlib.pyplot as plt

im = cv2.imread('boat.png')
edges = cv2.Canny(im,25,255,L2gradient=False)
plt.imshow(edges,cmap='gray')
plt.show()
```

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$

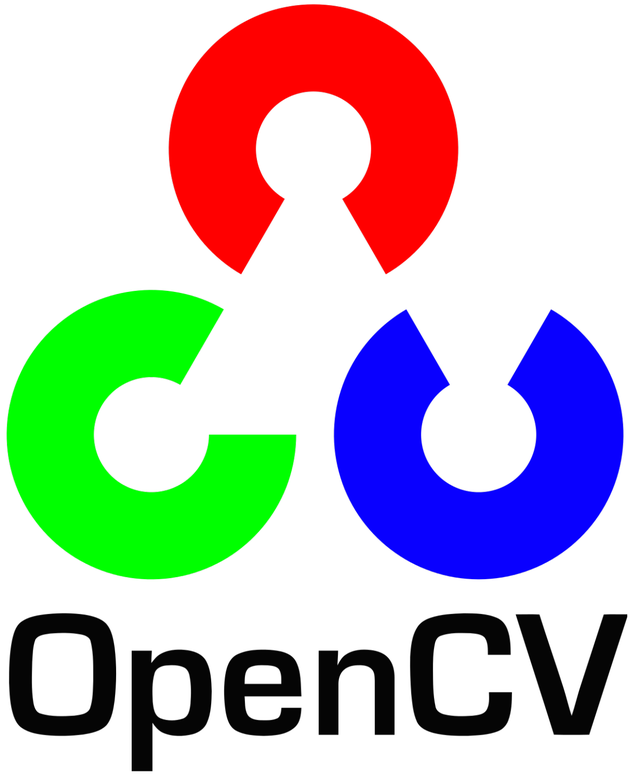
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

$$Edge\_Gradient\ (G) = |G_x| + |G_y|$$

OpenCV

Installation

- OpenCV
  - https://docs.opencv.org/master/da/df6/tutorial_py_table_of_contents_setup.html
  - Macs:
    - pip install opencv-python
- Imutils, scikit-image
  - Used for this lecture (for convenience)



OpenCV

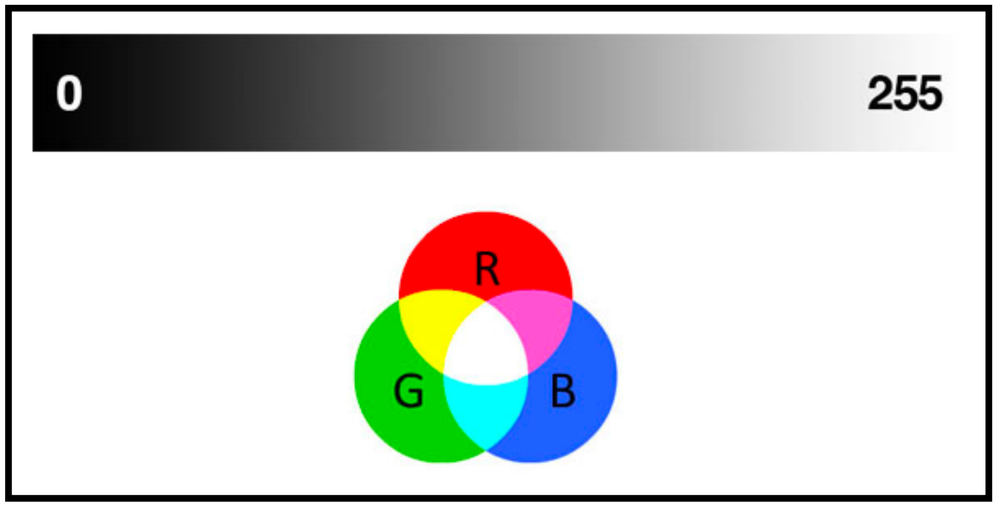
Loading and Displaying an Image

```python
# import the necessary packages
import imutils
import cv2

# load the input image and show its dimensions, keeping in mind that
# images are represented as a multi-dimensional NumPy array with
# shape no. rows (height) x no. columns (width) x no. channels (depth)
image = cv2.imread("jurassicpark.jpg")
(h, w, d) = image.shape
print("width={}, height={}, depth={}".format(w, h, d))

# display the image to our screen -- we will need to click the window
# open by OpenCV and press a key on our keyboard to continue execution
cv2.imshow("Image", image)
cv2.waitKey(0)
```

OpenCV

Accessing Individual Pixels

- OpenCV color images in (B,G,R)
  - Each value has range [0,255]
  - 256*256*256 = 16777216 colors




```
17  # access the RGB pixel located at x=50, y=100, keepind in mind that
18  # OpenCV stores images in BGR order rather than RGB
19  (B, G, R) = image[100, 50]
20  print("R={}, G={}, B={}".format(R, G, B))
```

OpenCV

Array Slicing and Cropping

```
22  # extract a 200x200 pixel square ROI (Region of Interest) from the
23  # input image
24  roi = image[150:350, 730:930]
25  cv2.imshow("ROI", roi)
26  cv2.waitKey(0)
```

OpenCV

Resizing Images

```
28  # resize the image to 200x200px, ignoring aspect ratio
29  resized = cv2.resize(image, (200, 200))
30  cv2.imshow("Fixed Resizing", resized)
31  cv2.waitKey(0)
```

OpenCV

Resizing Images

```
33  # fixed resizing and distort aspect ratio so let's resize the width
34  # to be 300px but compute the new height based on the aspect ratio
35  r = 300.0 / w
36  dim = (300, int(h * r))
37  resized = cv2.resize(image, dim)
38  cv2.imshow("Aspect Ratio Resize", resized)
39  cv2.waitKey(0)
```

OpenCV

Rotating Images

```
47  # let's rotate an image 45 degrees clockwise using OpenCV by first
48  # computing the image center, then constructing the rotation matrix,
49  # and then finally applying the affine warp
50  center = (w // 2, h // 2)
51  M = cv2.getRotationMatrix2D(center, -45, 1.0)
52  rotated = cv2.warpAffine(image, M, (w, h))
53  cv2.imshow("OpenCV Rotation", rotated)
54  cv2.waitKey(0)
```

OpenCV

Rotating Images

```
61  # OpenCV doesn't "care" if our rotated image is clipped after rotation
62  # so we can instead use another imutils convenience function to help
63  # us out
64  rotated = imutils.rotate_bound(image, 45)
65  cv2.imshow("Imutils Bound Rotation", rotated)
66  cv2.waitKey(0)
```

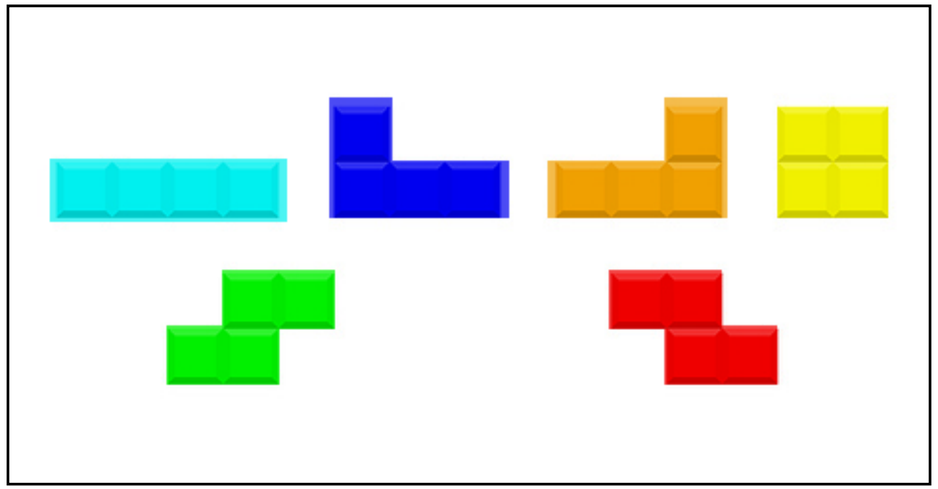
OpenCV

Smoothing Images

```
68  # apply a Gaussian blur with a 11x11 kernel to the image to smooth it,
69  # useful when reducing high frequency noise
70  blurred = cv2.GaussianBlur(image, (11, 11), 0)
71  cv2.imshow("Blurred", blurred)
72  cv2.waitKey(0)
```

OpenCV

Counting Objects

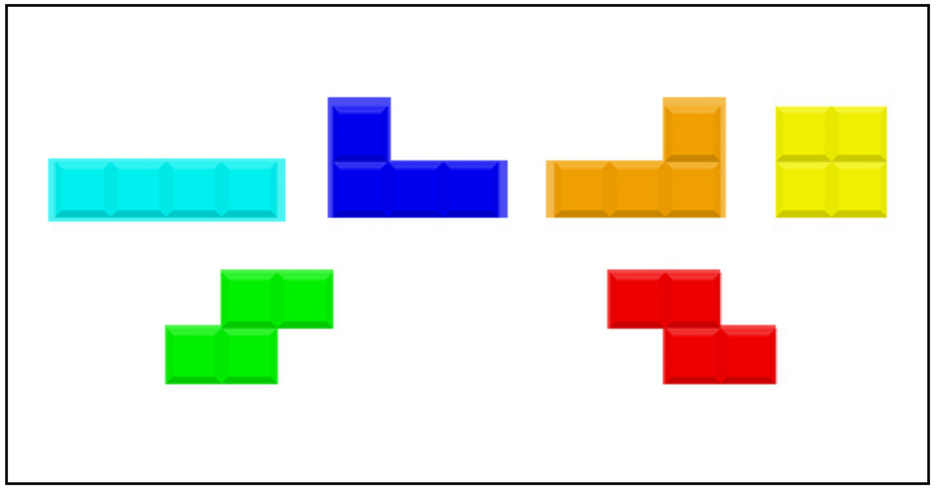


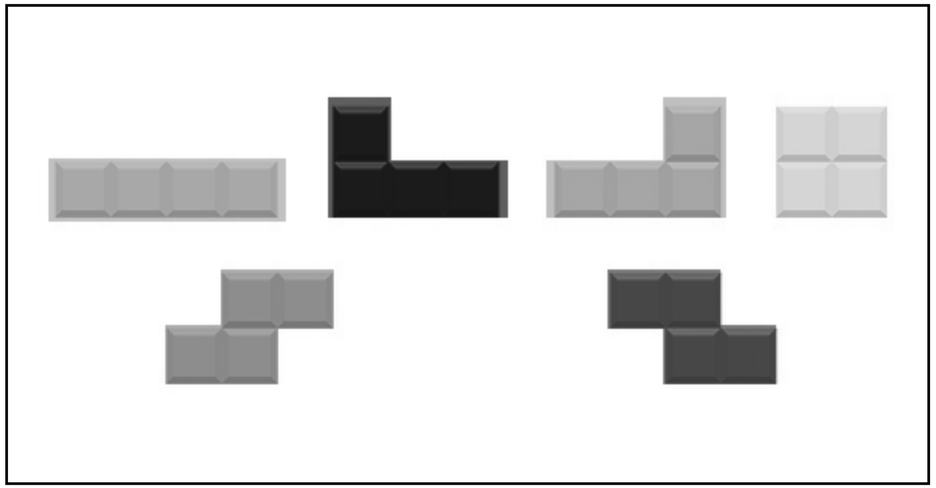
```python
# import the necessary packages
import argparse
import imutils
import cv2

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

OpenCV

Converting to Grayscale

```
12  # load the input image (whose path was supplied via command line
13  # argument) and display the image to our screen
14  image = cv2.imread(args["image"])
15  cv2.imshow("Image", image)
16  cv2.waitKey(0)
17
18  # convert the image to grayscale
19  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
20  cv2.imshow("Gray", gray)
21  cv2.waitKey(0)
```

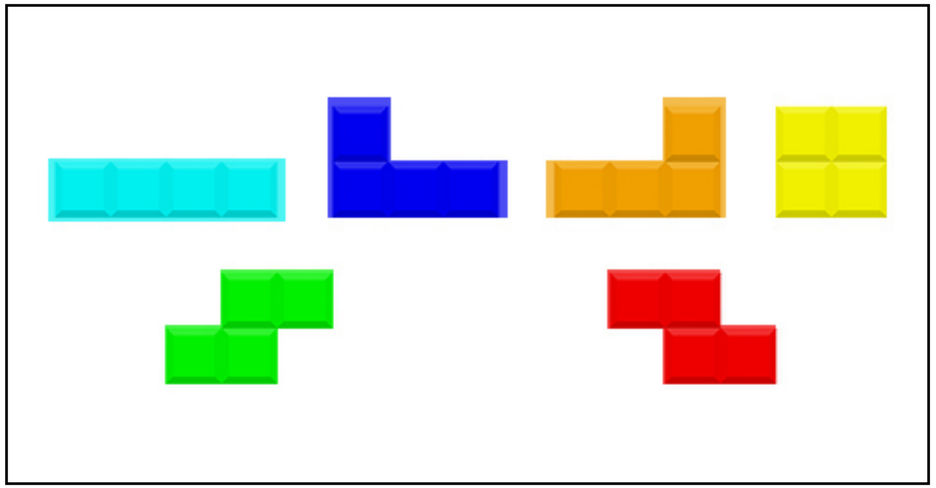


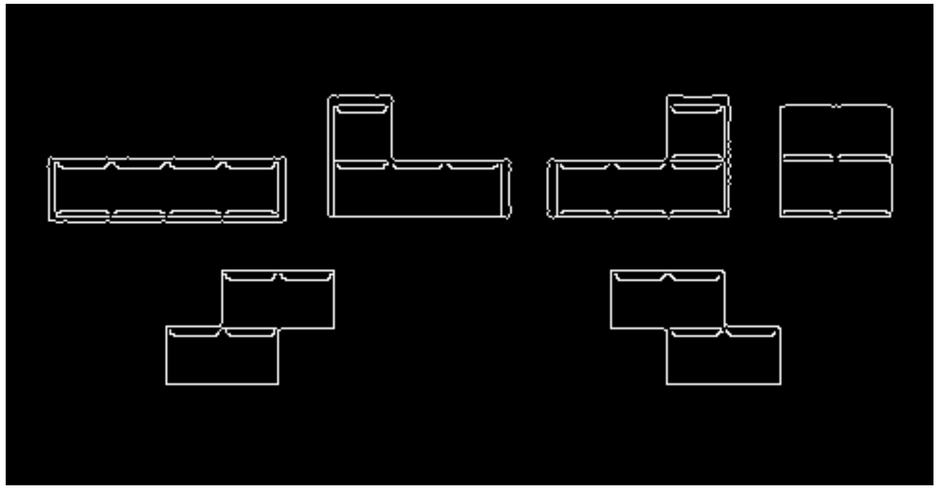


OpenCV

Edge Detection

```
23  # applying edge detection we can find the outlines of objects in
24  # images
25  edged = cv2.Canny(gray, 30, 150)
26  cv2.imshow("Edged", edged)
27  cv2.waitKey(0)
```

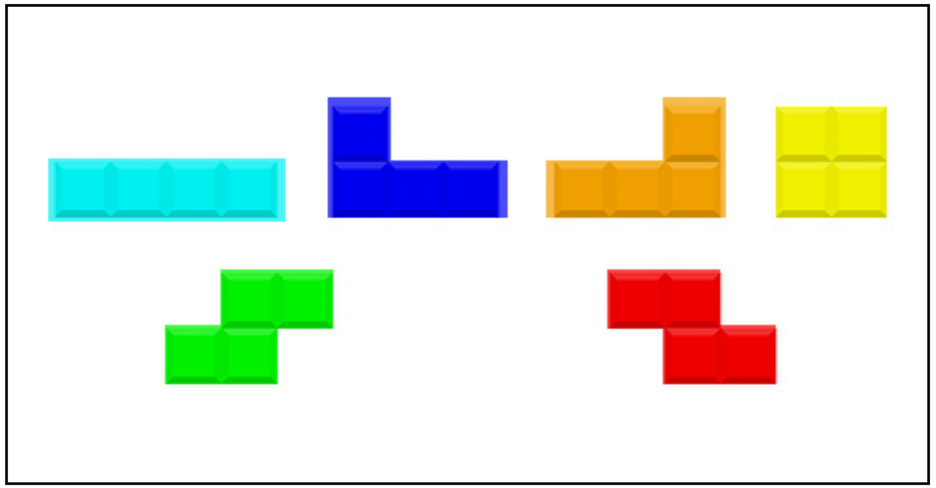


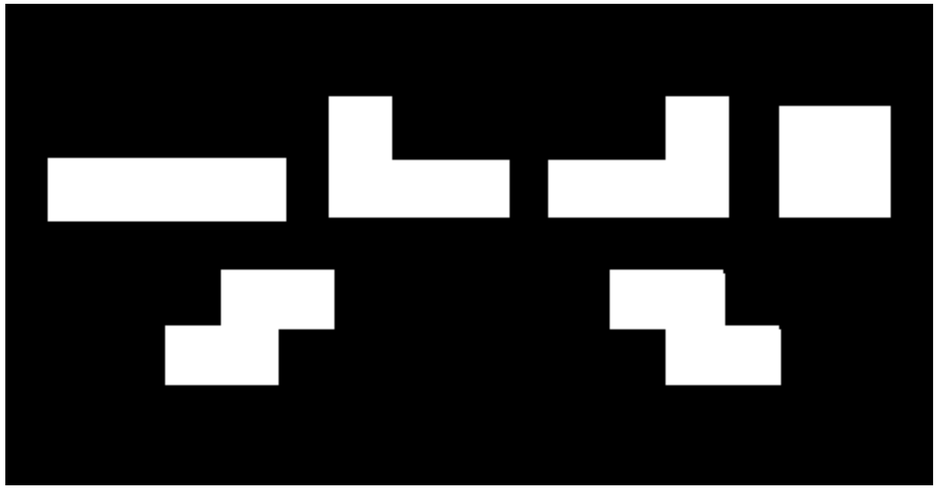


OpenCV

Thresholding

```
29  # threshold the image by setting all pixel values less than 225
30  # to 255 (white; foreground) and all pixel values >= 225 to 255
31  # (black; background), thereby segmenting the image
32  thresh = cv2.threshold(gray, 225, 255, cv2.THRESH_BINARY_INV)[1]
33  cv2.imshow("Thresh", thresh)
34  cv2.waitKey(0)
```

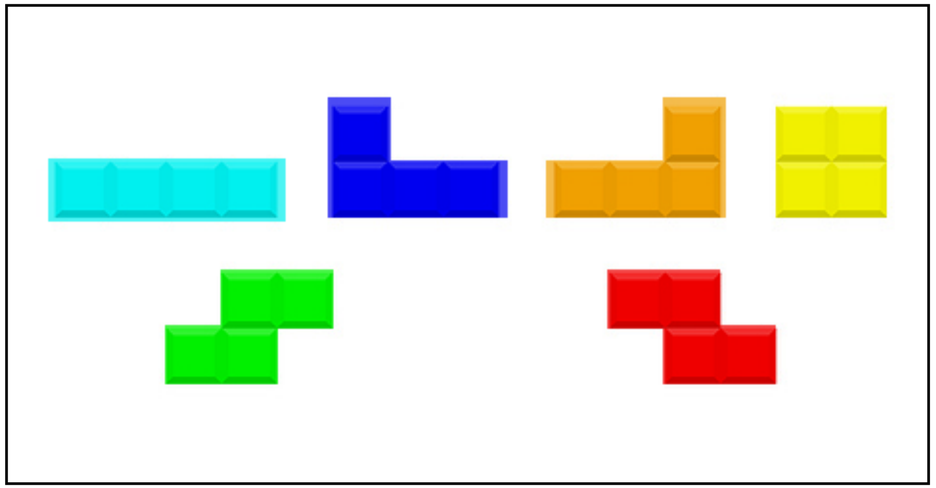


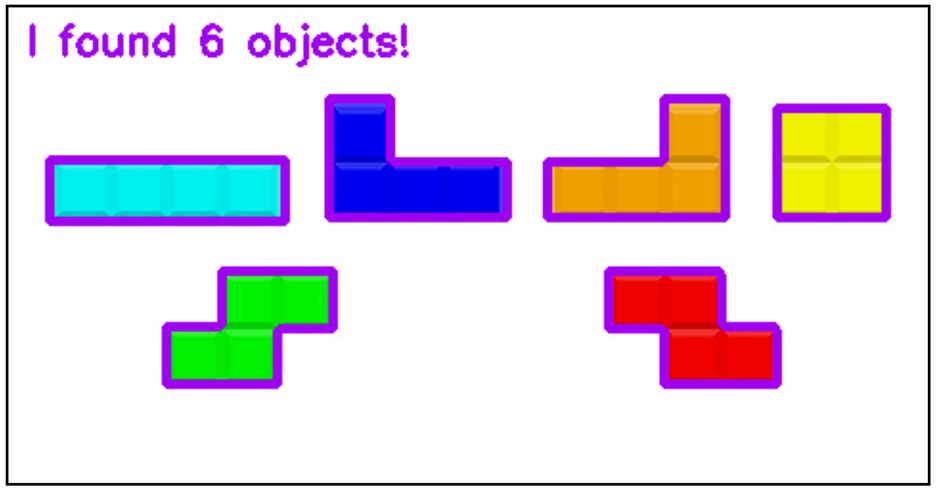


OpenCV

Detecting/Drawing Contours

```
36  # find contours (i.e., outlines) of the foreground objects in the
37  # thresholded image
38  cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
39      cv2.CHAIN_APPROX_SIMPLE)
40  cnts = imutils.grab_contours(cnts)
41  output = image.copy()
42  # loop over the contours
43  for c in cnts:
44      # draw each contour on the output image with a 3px thick purple
45      # outline, then display the output contours one at a time
46      cv2.drawContours(output, [c], -1, (240, 0, 159), 3)
47      cv2.imshow("Contours", output)
48      cv2.waitKey(0)
```

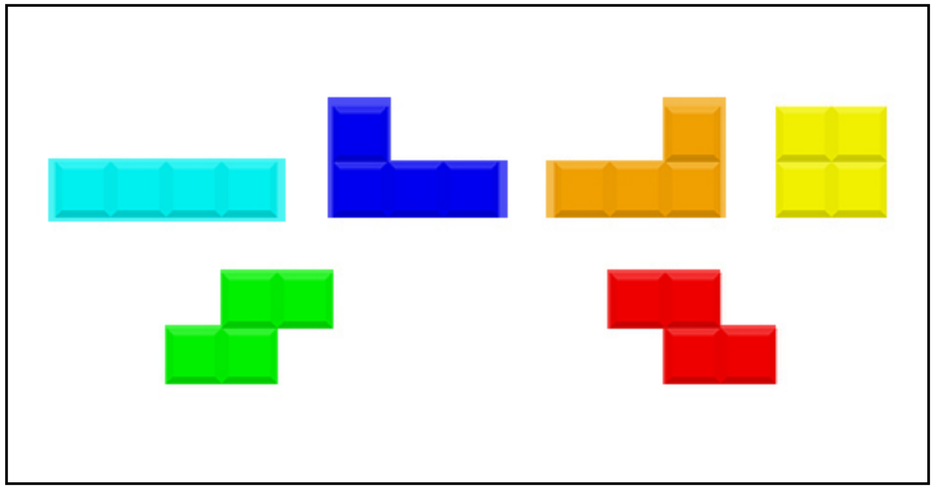


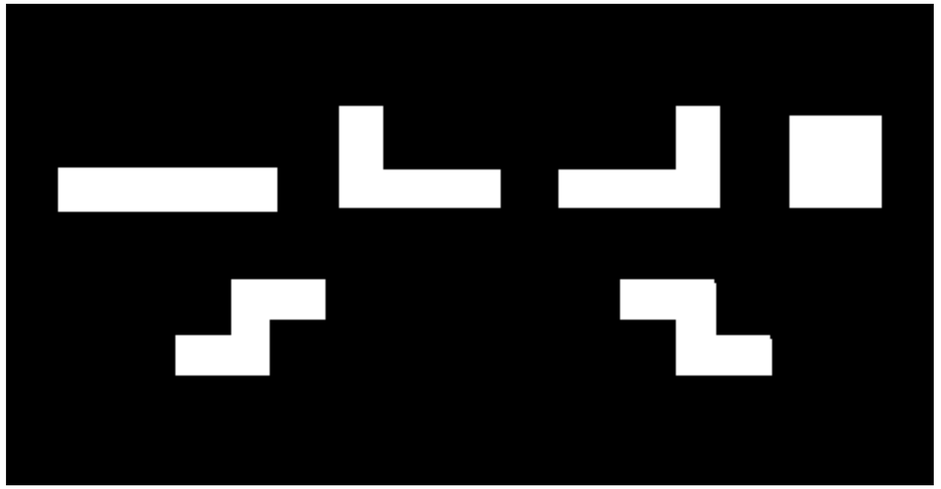


OpenCV

Erosions and Dilations

```
58  # we apply erosions to reduce the size of foreground objects
59  mask = thresh.copy()
60  mask = cv2.erode(mask, None, iterations=5)
61  cv2.imshow("Eroded", mask)
62  cv2.waitKey(0)
```

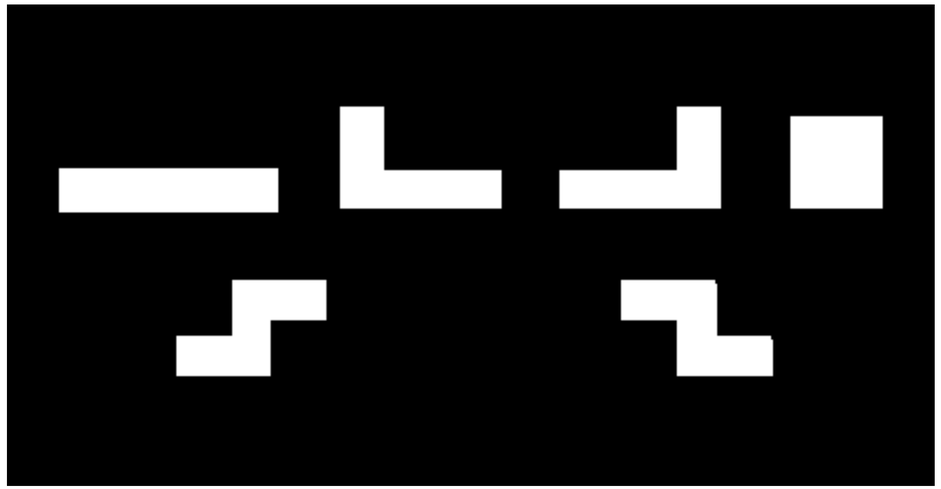


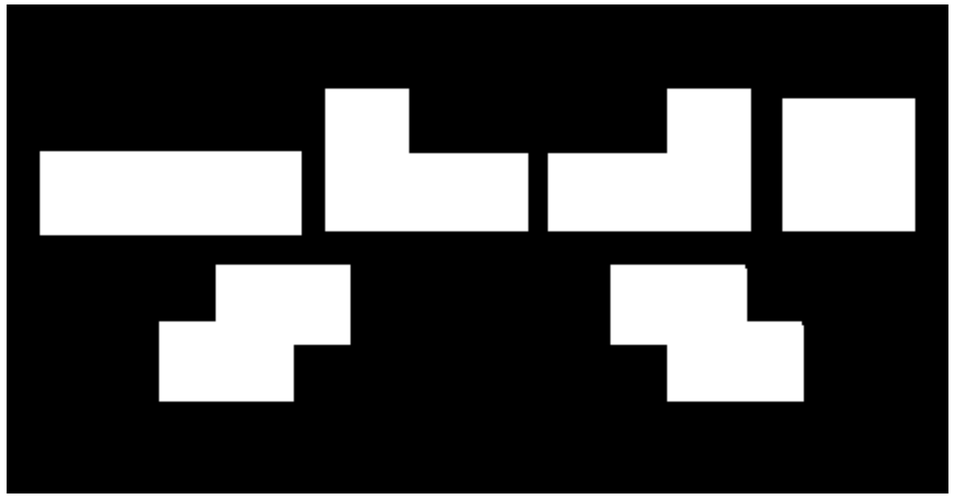


OpenCV

Erosions and Dilations

```
64  # similarly, dilations can increase the size of the ground objects
65  mask = thresh.copy()
66  mask = cv2.dilate(mask, None, iterations=5)
67  cv2.imshow("Dilated", mask)
68  cv2.waitKey(0)
```

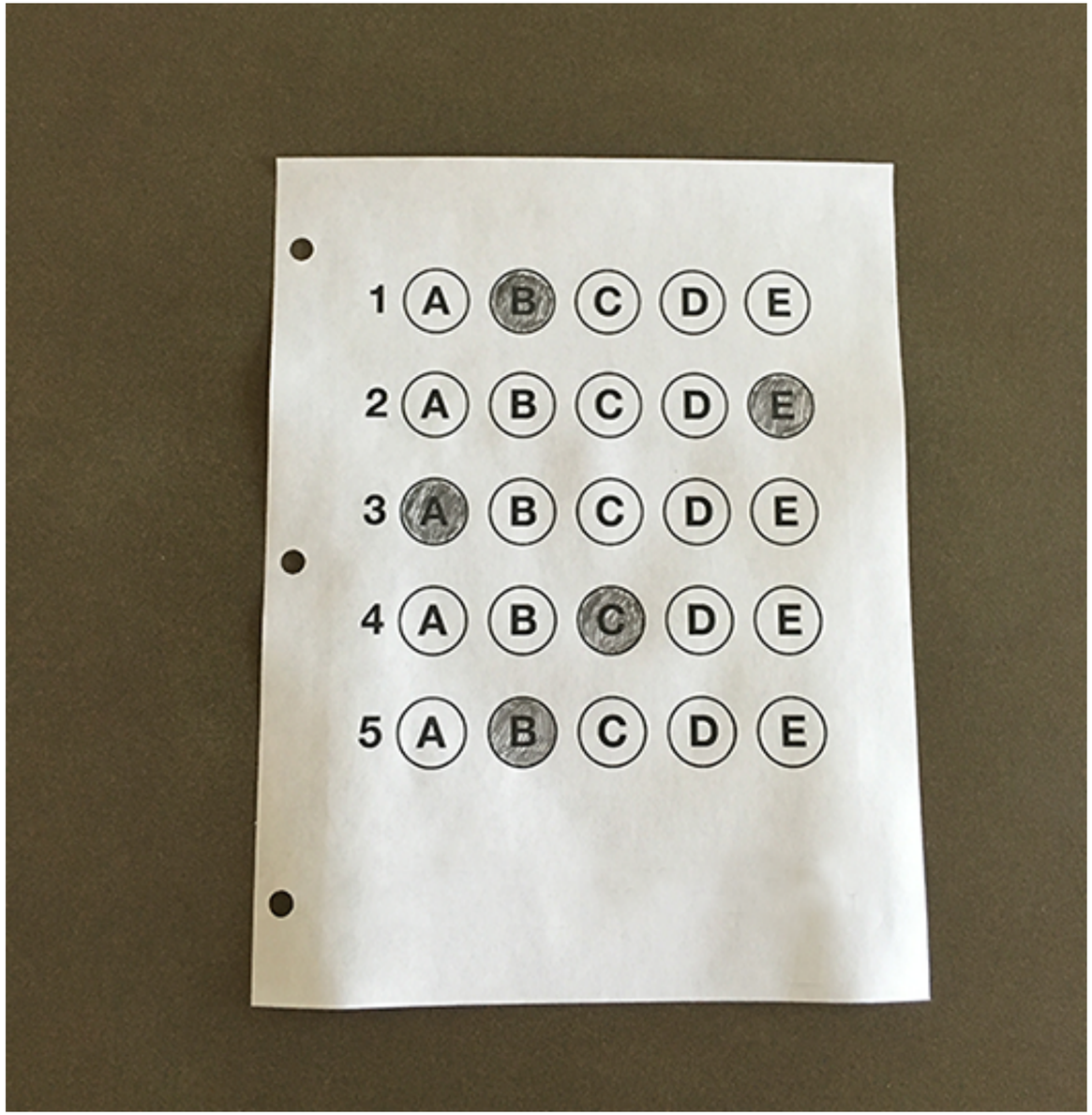




OMR

Bubble Sheet Scanner

- Detect exam in an image
- Apply transform to extract top-down view of exam
- Extract set of bubbles
- Sort questions/bubbles
- Determine the marked bubble in each row
- Lookup correct answer
- Repeat for all questions in exam



OMR

Test Grader

```python
# import the necessary packages
from imutils.perspective import four_point_transform
from imutils import contours
import numpy as np
import argparse
import imutils
import cv2

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

# define the answer key which maps the question number
# to the correct answer
ANSWER_KEY = {0: 1, 1: 4, 2: 0, 3: 3, 4: 1}

```

Question #1: B

Question #2: E

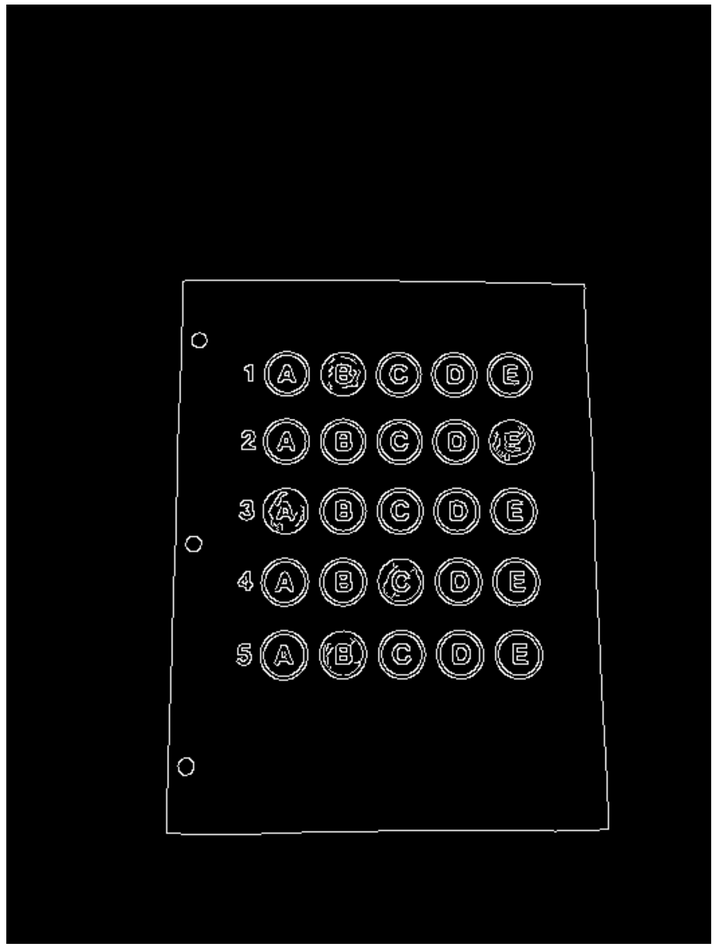
Question #3: A

Question #4: D

Question #5: B

OMR

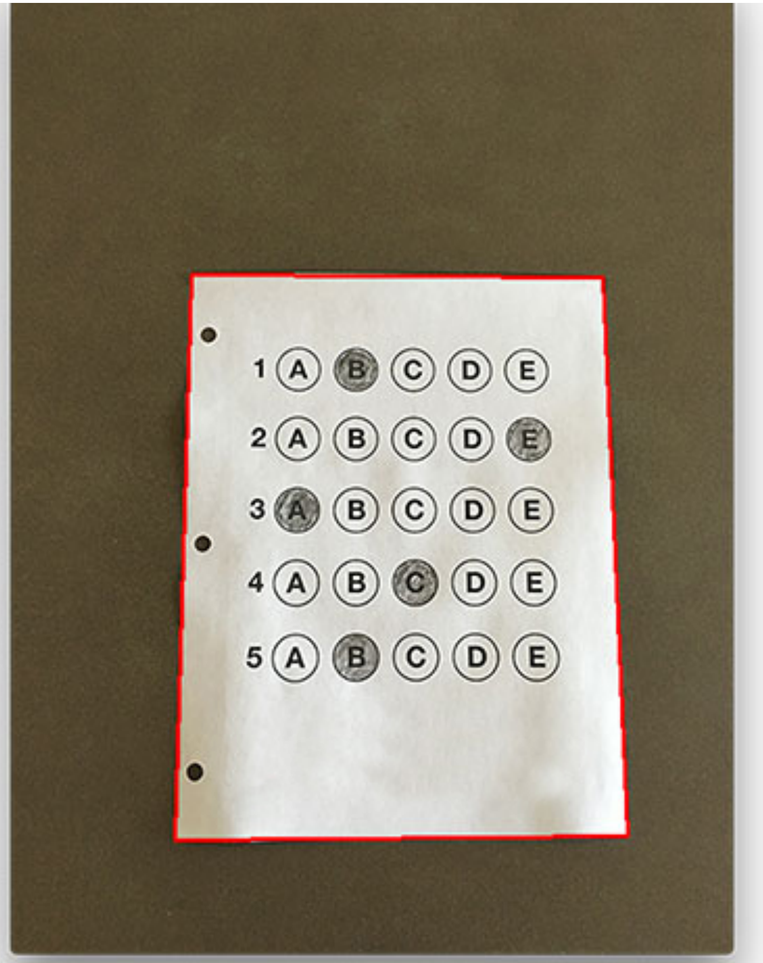
Pre-Process Image




```
19  # load the image, convert it to grayscale, blur it
20  # slightly, then find edges
21  image = cv2.imread(args["image"])
22  gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
23  blurred = cv2.GaussianBlur(gray, (5, 5), 0)
24  edged = cv2.Canny(blurred, 75, 200)
25
```

OMR

Find the exam

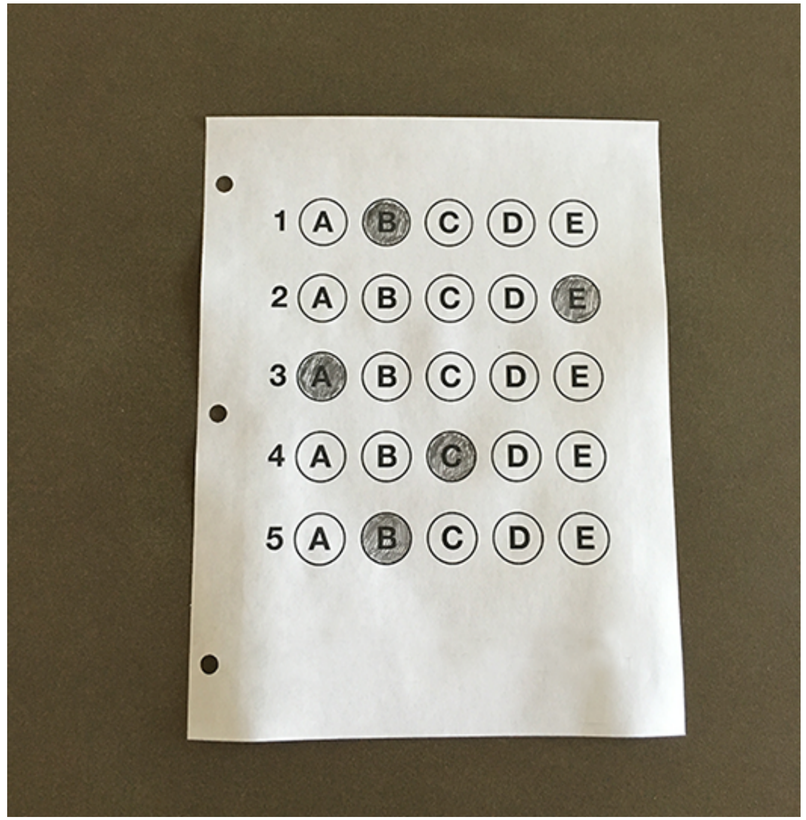



```python
# find contours in the edge map, then initialize
# the contour that corresponds to the document
cnts = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
docCnt = None

# ensure that at least one contour was found
if len(cnts) > 0:
    # sort the contours according to their size in
    # descending order
    cnts = sorted(cnts, key=cv2.contourArea, reverse=True)

    # loop over the sorted contours
    for c in cnts:
        # approximate the contour
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)

        # if our approximated contour has four points,
        # then we can assume we have found the paper
        if len(approx) == 4:
            docCnt = approx
            break

```

OMR

Top-down view of document

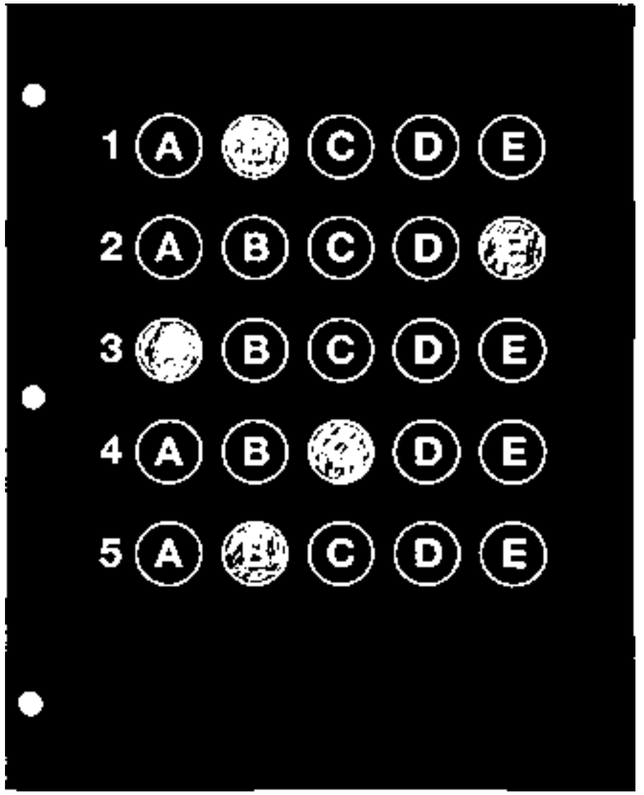
```
51  # apply a four point perspective transform to both the
52  # original image and grayscale image to obtain a top-down
53  # birds eye view of the paper
54  paper = four_point_transform(image, docCnt.reshape(4, 2))
55  warped = four_point_transform(gray, docCnt.reshape(4, 2))
56
```



OMR

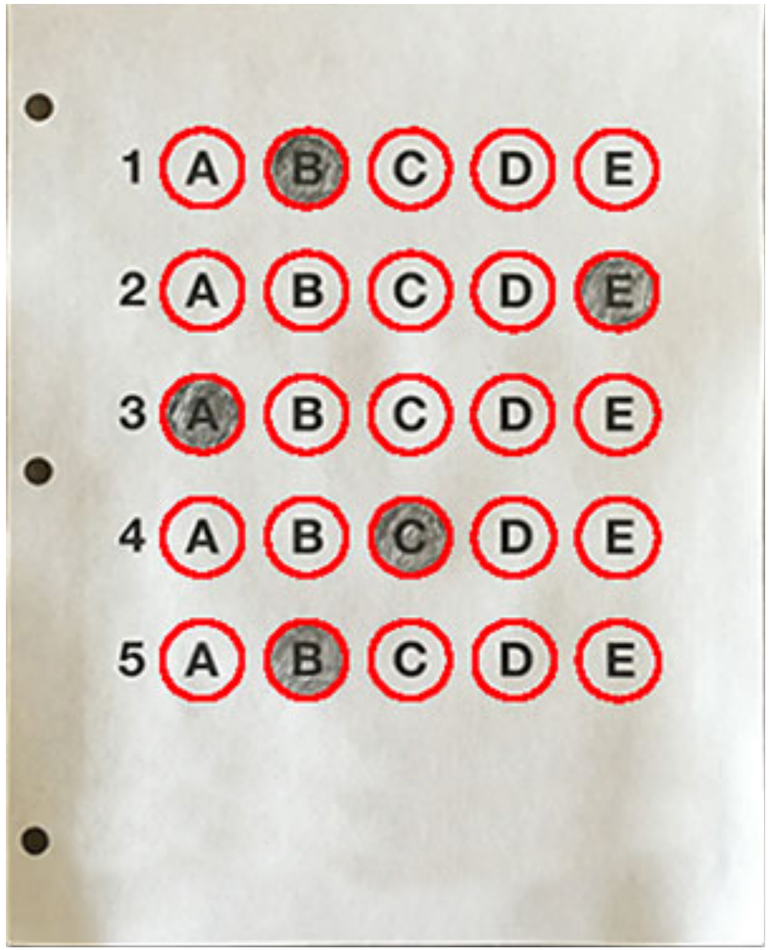
Top-down view of document

```
57  # apply Otsu's thresholding method to binarize the warped
58  # piece of paper
59  thresh = cv2.threshold(warped, 0, 255,
60      cv2.THRESH_BINARY_INV | cv2.THRESH_OTSU)[1]
61
```



OMR

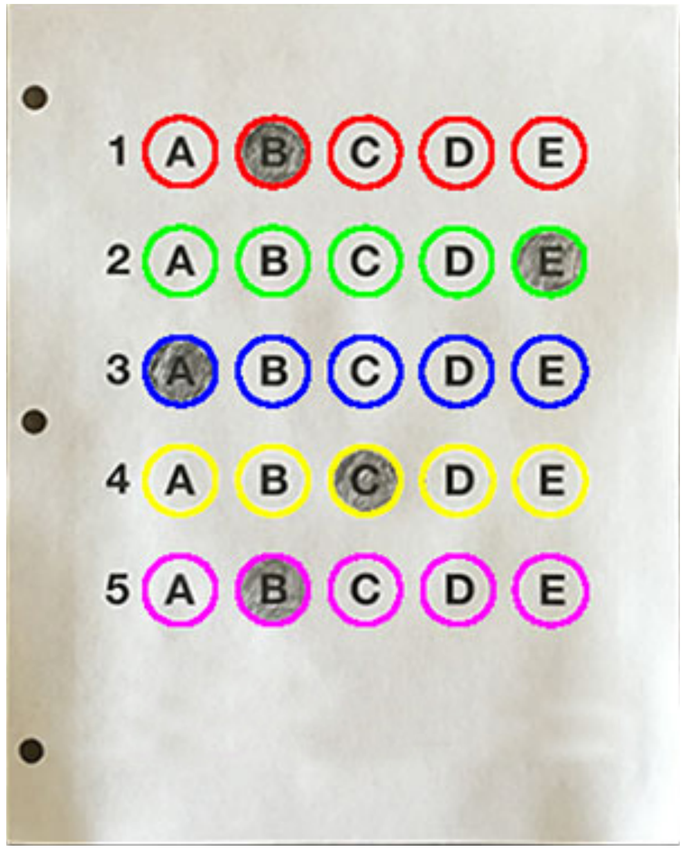
Finding Bubbles




```
62  # find contours in the thresholded image, then initialize
63  # the list of contours that correspond to questions
64  cnts = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
65      cv2.CHAIN_APPROX_SIMPLE)
66  cnts = imutils.grab_contours(cnts)
67  questionCnts = []
68  # loop over the contours
69  for c in cnts:
70      # compute the bounding box of the contour, then use the
71      # bounding box to derive the aspect ratio
72      (x, y, w, h) = cv2.boundingRect(c)
73      ar = w / float(h)
74      # in order to label the contour as a question, region
75      # should be sufficiently wide, sufficiently tall, and
76      # have an aspect ratio approximately equal to 1
77      if w >= 20 and h >= 20 and ar >= 0.9 and ar <= 1.1:
78          questionCnts.append(c)
79
```

OMR

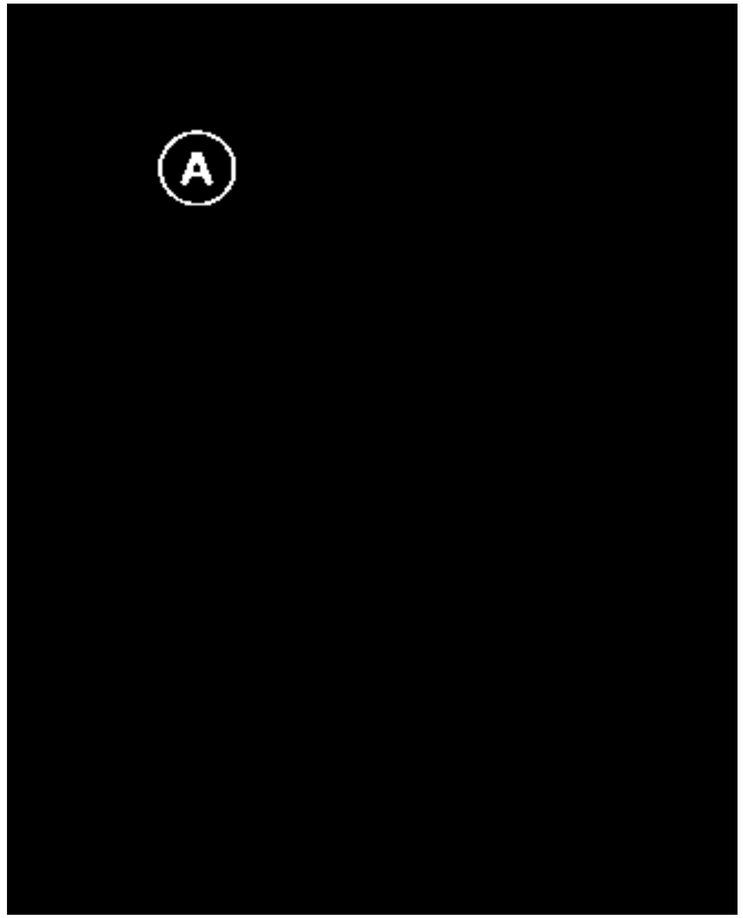
Grading




```
82  # sort the question contours top-to-bottom, then initialize
83  # the total number of correct answers
84  questionCnts = contours.sort_contours(questionCnts,
85      method="top-to-bottom")[0]
86  correct = 0
87  # each question has 5 possible answers, to loop over the
88  # question in batches of 5
89  for (q, i) in enumerate(np.arange(0, len(questionCnts), 5)):
90      # sort the contours for the current question from
91      # left to right, then initialize the index of the
92      # bubbled answer
93      cnts = contours.sort_contours(questionCnts[i:i + 5])[0]
94      bubbled = None
```

OMR

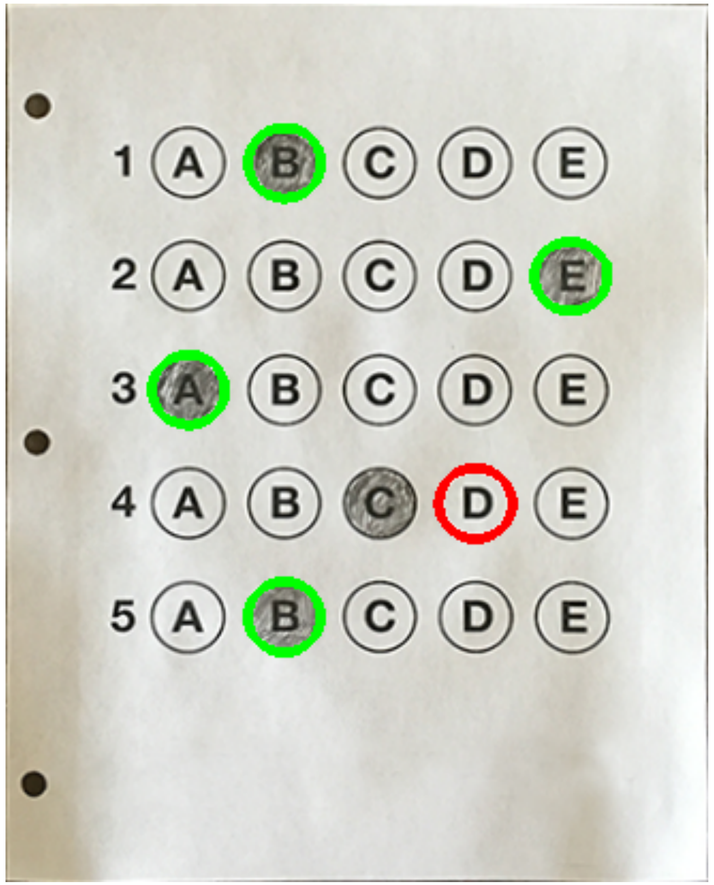
What's Filled In




```
    # loop over the sorted contours
    for (j, c) in enumerate(cnts):
        # construct a mask that reveals only the current
        # "bubble" for the question
        mask = np.zeros(thresh.shape, dtype="uint8")
        cv2.drawContours(mask, [c], -1, 255, -1)

        # apply the mask to the thresholded image, then
        # count the number of non-zero pixels in the
        # bubble area
        mask = cv2.bitwise_and(thresh, thresh, mask=mask)
        total = cv2.countNonZero(mask)

        # if the current total has a larger number of total
        # non-zero pixels, then we are examining the currently
        # bubbled-in answer
        if bubbled is None or total > bubbled[0]:
            bubbled = (total, j)

```

OMR

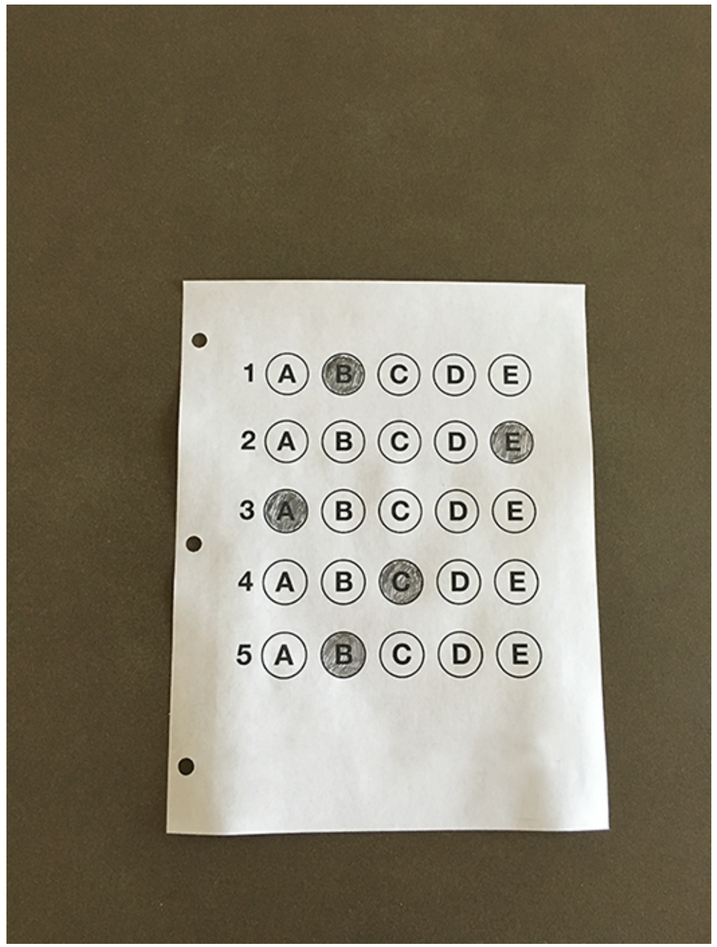
Answer Key



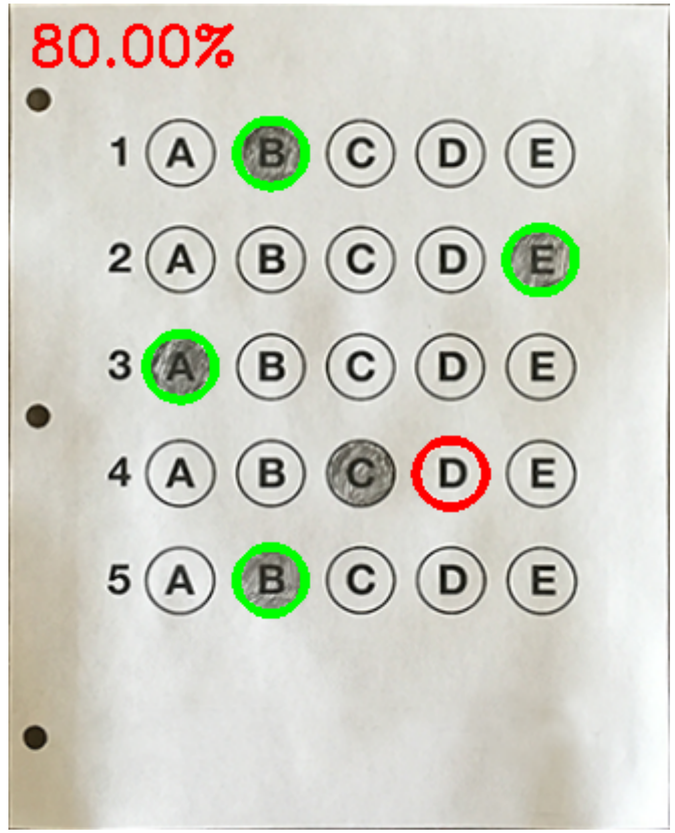

```
116     # initialize the contour color and the index of the
117     # *correct* answer
118     color = (0, 0, 255)
119     k = ANSWER_KEY[q]
120
121     # check to see if the bubbled answer is correct
122     if k == bubbled[1]:
123         color = (0, 255, 0)
124         correct += 1
125
126     # draw the outline of the correct answer on the test
127     cv2.drawContours(paper, [cnts[k]], -1, color, 3)
```

OMR

Scoring




```
129  # grab the test taker
130  score = (correct / 5.0) * 100
131  print("[INFO] score: {:.2f}%".format(score))
132  cv2.putText(paper, "{:.2f}%".format(score), (10, 30),
133      cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
134  cv2.imshow("Original", image)
135  cv2.imshow("Exam", paper)
136  cv2.waitKey(0)
```



Autostereograms

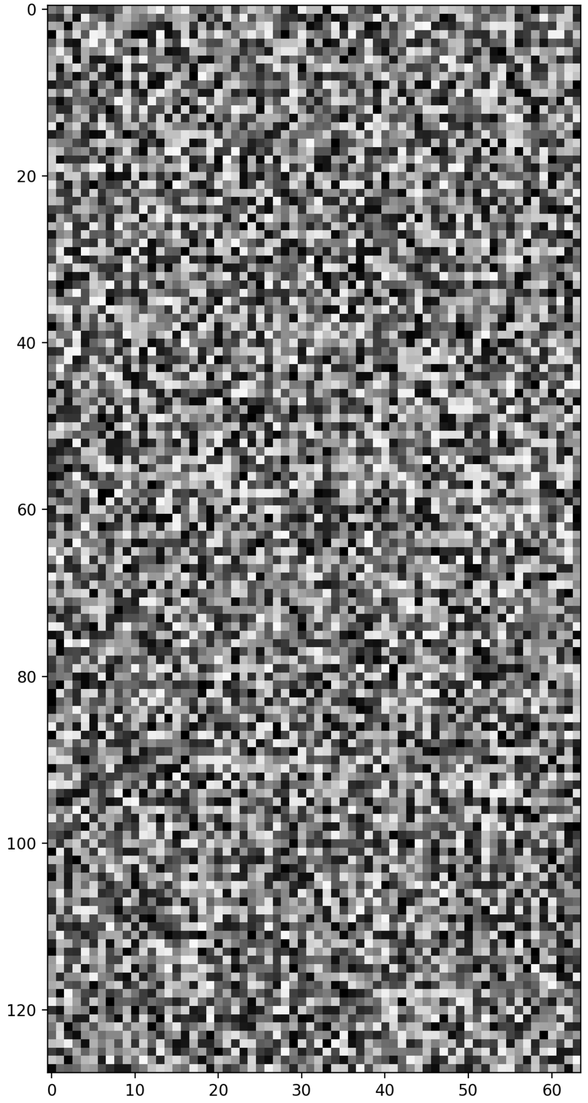
Depth Map Based Stereograms

1. Pattern

```
def make_pattern(shape=(16, 16), levels=64):
    "Creates a pattern from gray values."
    return np.random.randint(0, levels - 1, shape) / levels
```

```
def display(img, colorbar=False):
    "Displays an image."
    plt.figure(figsize=(10, 10))
    if len(img.shape) == 2:
        i = skimage.io.imshow(img, cmap='gray')
    else:
        i = skimage.io.imshow(img)
    if colorbar:
        plt.colorbar(i, shrink=0.5, label='depth')
    plt.tight_layout()
    plt.show()
```

```
59  pattern = make_pattern(shape=(128,64))
60  display(pattern)
```

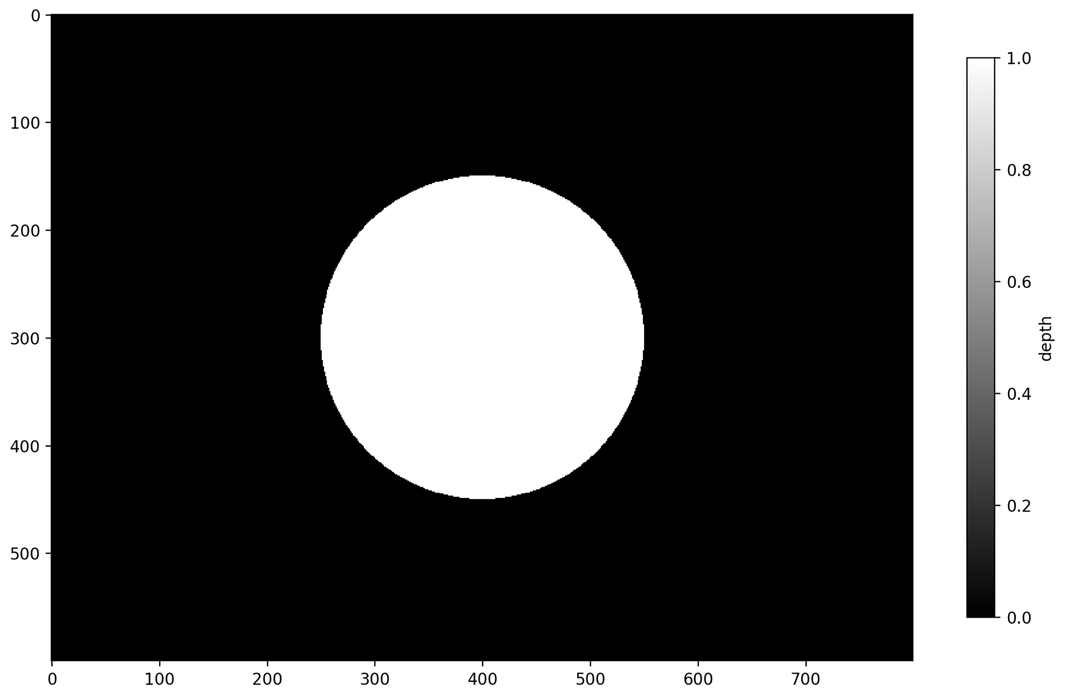


Depth Map Based Stereograms

2. Depth Map

```python
def create_circular_depthmap(shape=(600, 800), center=None, radius=100):
    "Creates a circular depthmap, centered on the image. "
    depthmap = np.zeros(shape, dtype=float)
    r = np.arange(depthmap.shape[0])
    c = np.arange(depthmap.shape[1])
    R, C = np.meshgrid(r, c, indexing='ij')
    if center is None:
        center = np.array([r.max() / 2, c.max() / 2])
    d = np.sqrt((R - center[0])**2 + (C - center[1])**2)
    depthmap += (d < radius)
    return depthmap
```

```python
depthmap = create_circular_depthmap(radius=150)
display(depthmap, colorbar=True)
```

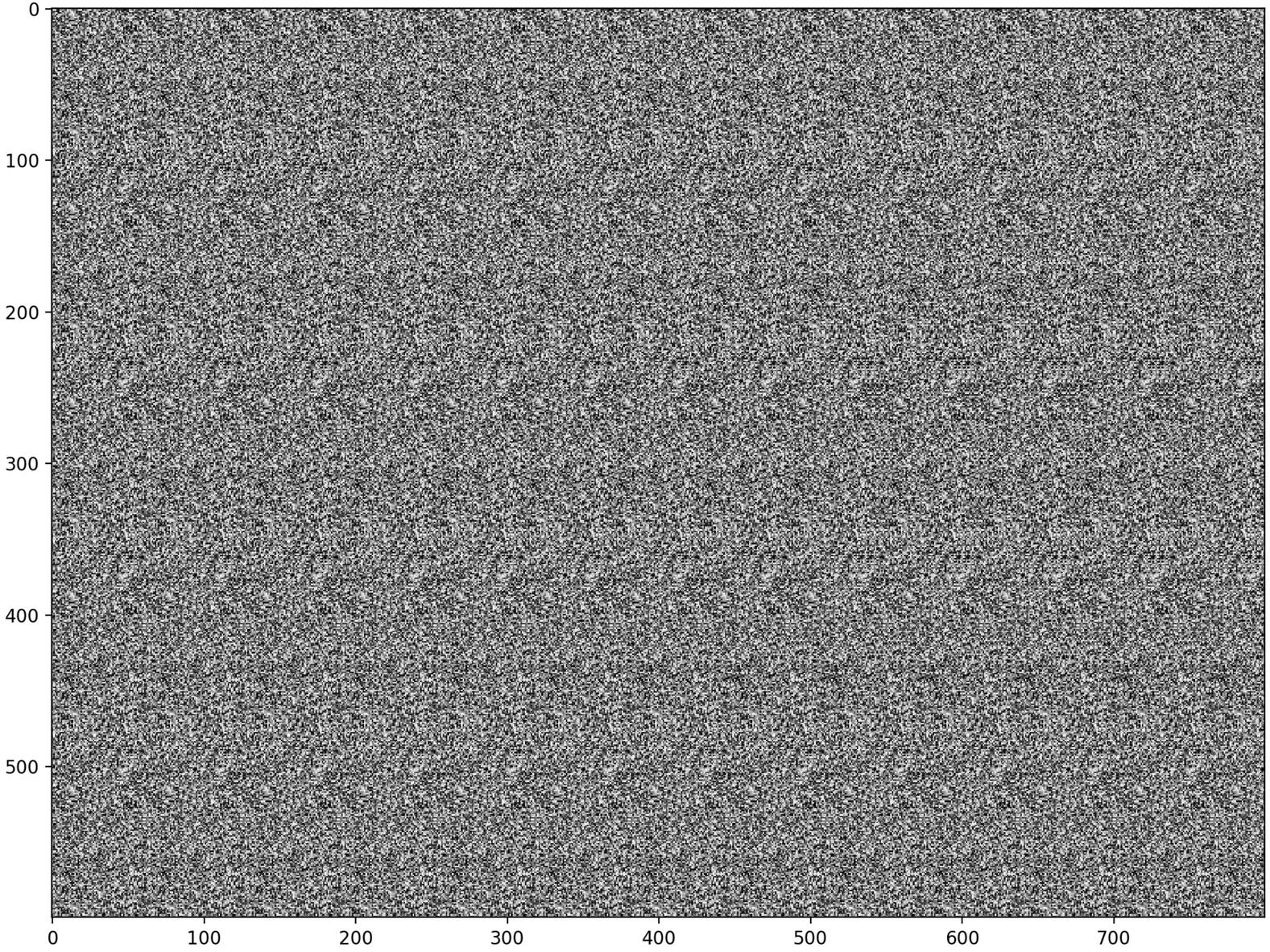


Depth Map Based Stereograms

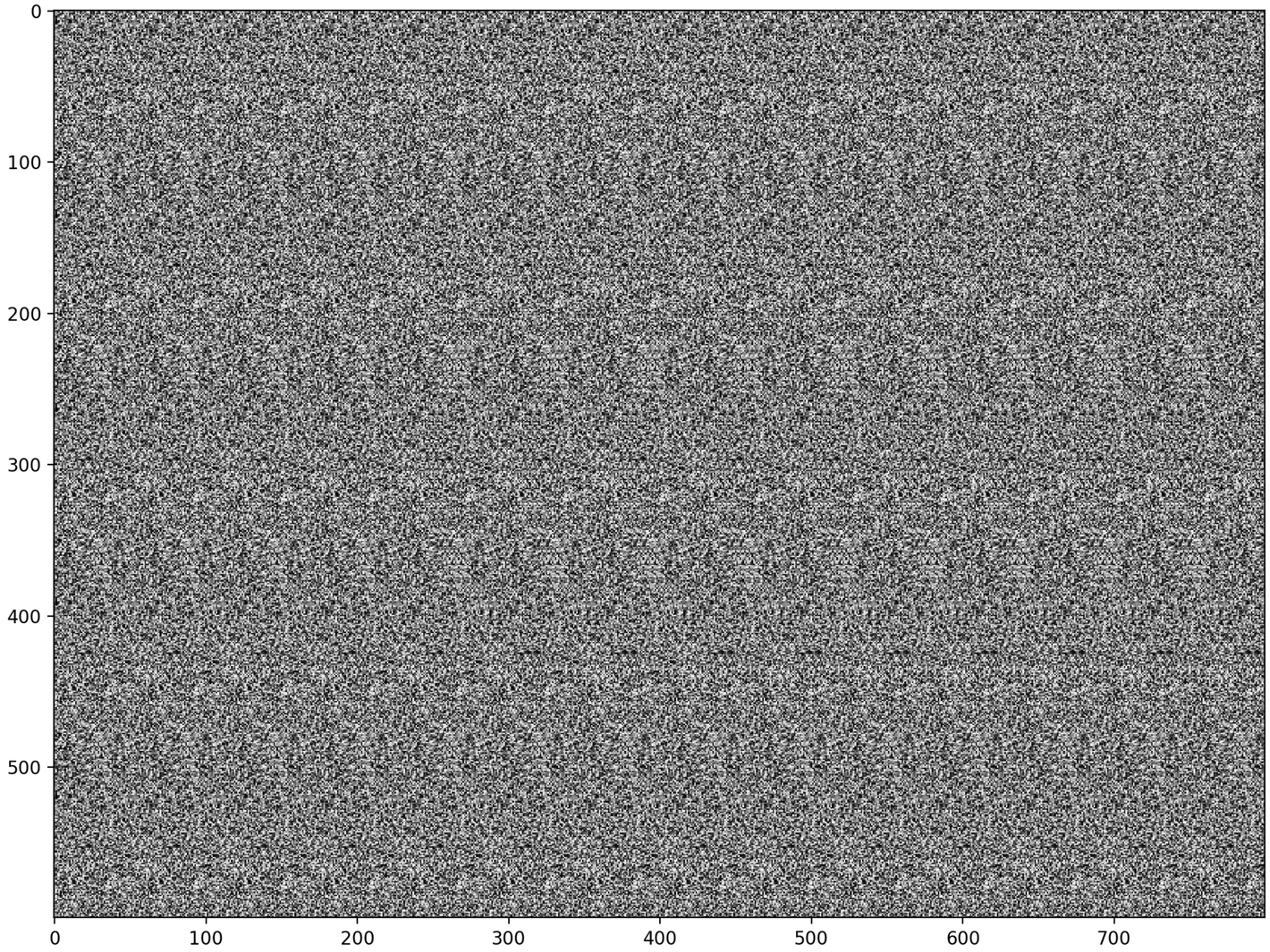
3. Autostereogram

```
def normalize(depthmap):
    "Normalizes values of depthmap to [0, 1] range."
    if depthmap.max() > depthmap.min():
        return (depthmap - depthmap.min()) / (depthmap.max() - depthmap.min())
    else:
        return depthmap

def make_autostereogram(depthmap, pattern, shift_amplitude=0.1, invert=False):
    "Creates an autostereogram from depthmap and pattern."
    depthmap = normalize(depthmap)
    if invert:
        depthmap = 1 - depthmap
    autostereogram = np.zeros_like(depthmap, dtype=pattern.dtype)
    for r in np.arange(autostereogram.shape[0]):
        for c in np.arange(autostereogram.shape[1]):
            if c < pattern.shape[1]:
                autostereogram[r, c] = pattern[r % pattern.shape[0], c]
            else:
                shift = int(depthmap[r, c] * shift_amplitude * pattern.shape[1])
                autostereogram[r, c] = autostereogram[r, c - pattern.shape[1] + shift]
    return autostereogram
```

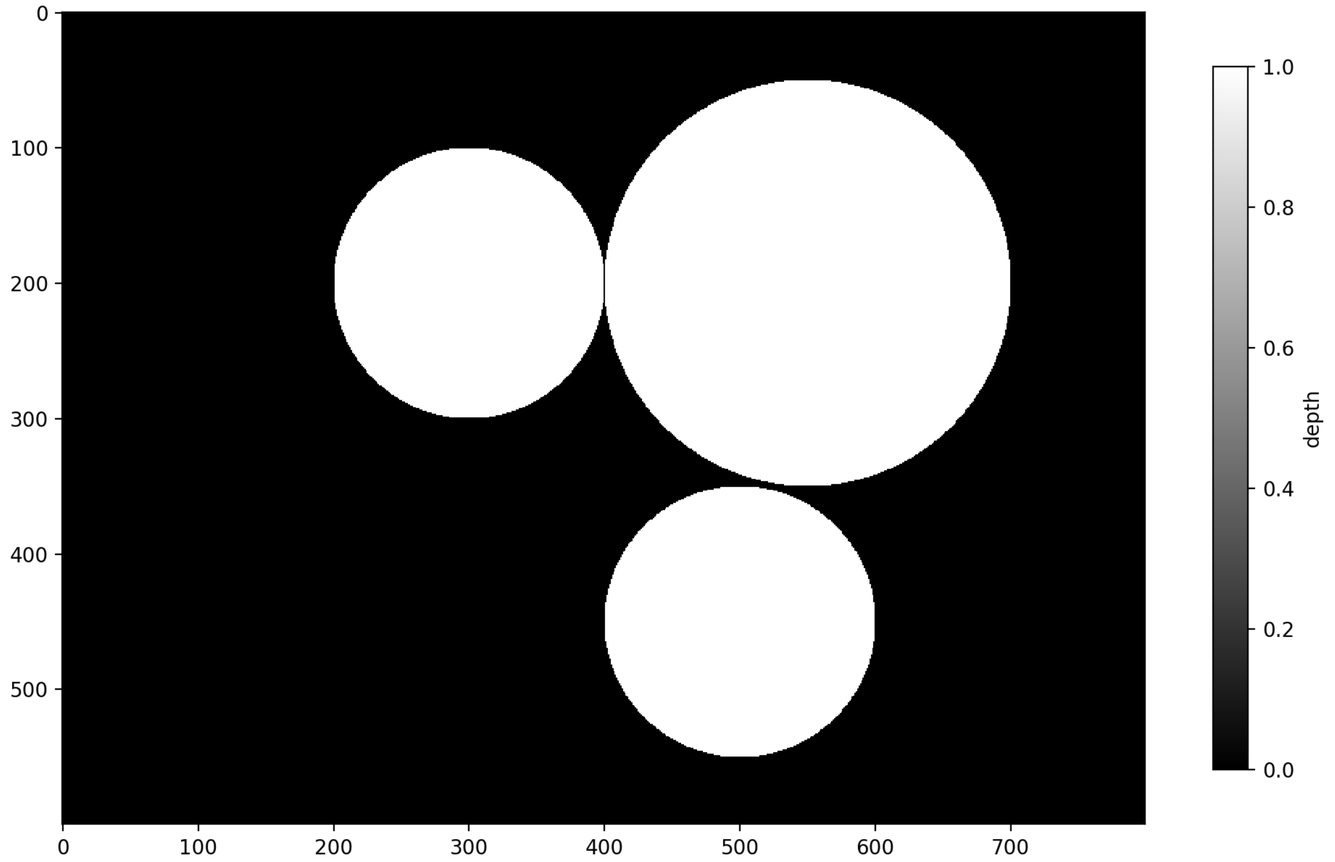
```
autostereogram = make_autostereogram(depthmap, pattern)
display(autostereogram)
```


0
100
200
300
400
500
0
100
200
300
400
500
600
700

```
autostereogram = make_autostereogram(depthmap, pattern, invert=True)
display(autostereogram)
```



```
depthmap = create_circular_depthmap(center=(200, 300), radius=100) + \
           create_circular_depthmap(center=(450, 500), radius=100) + \
           create_circular_depthmap(center=(200, 550), radius=150)
depthmap = normalize(depthmap)
display(depthmap, colorbar=True)
autostereogram = make_autostereogram(depthmap, pattern)
display(autostereogram)
```



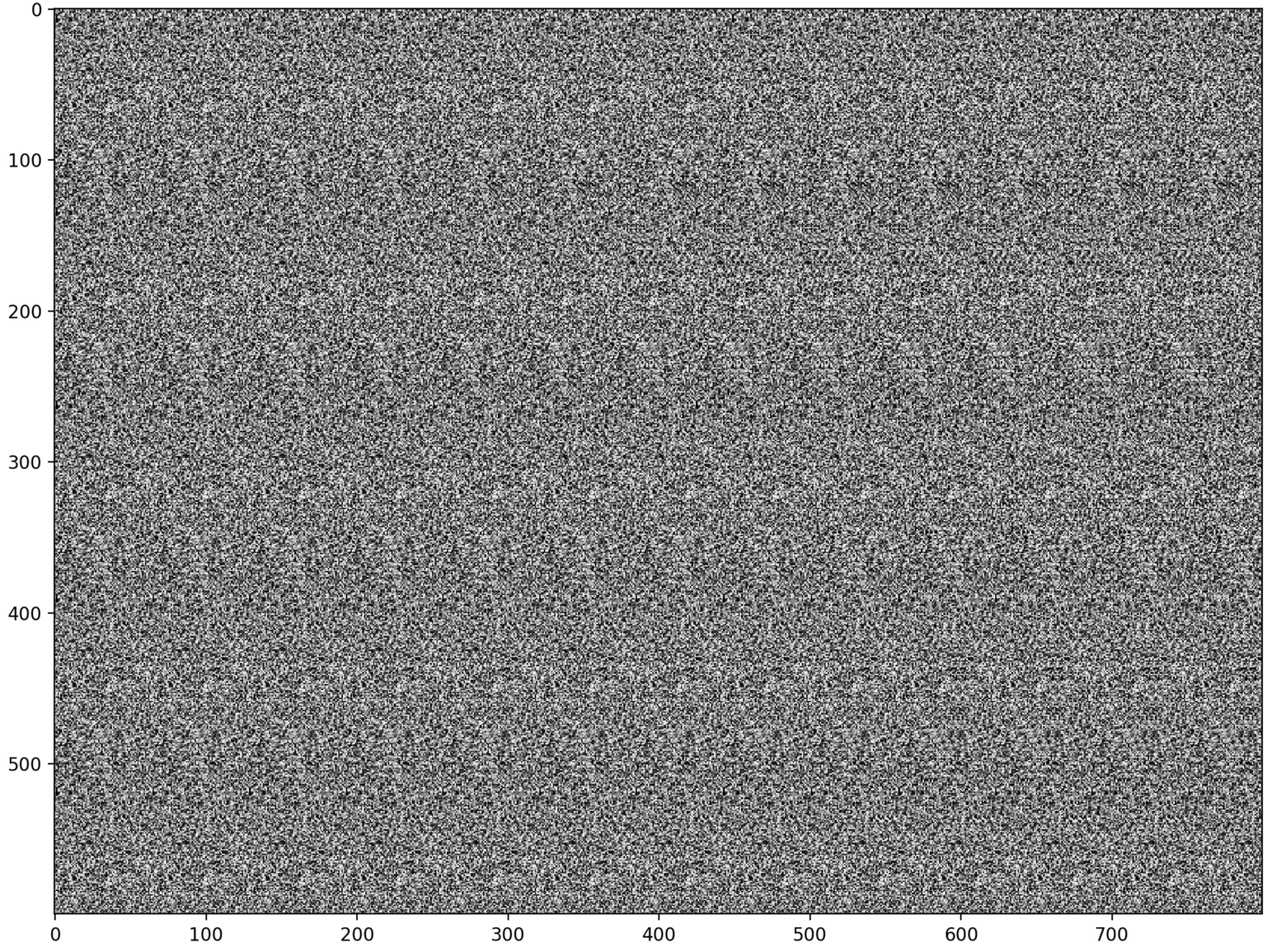
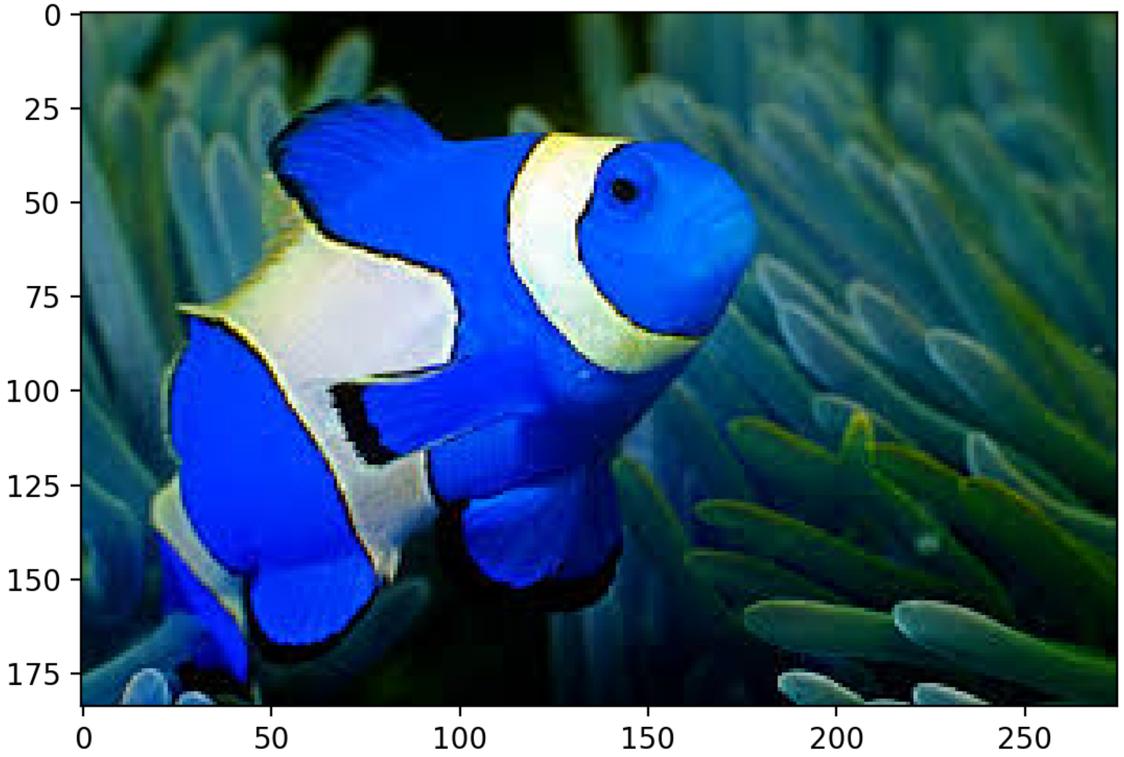

0
100
200
300
400
500
0
100
200
300
400
500
600
700

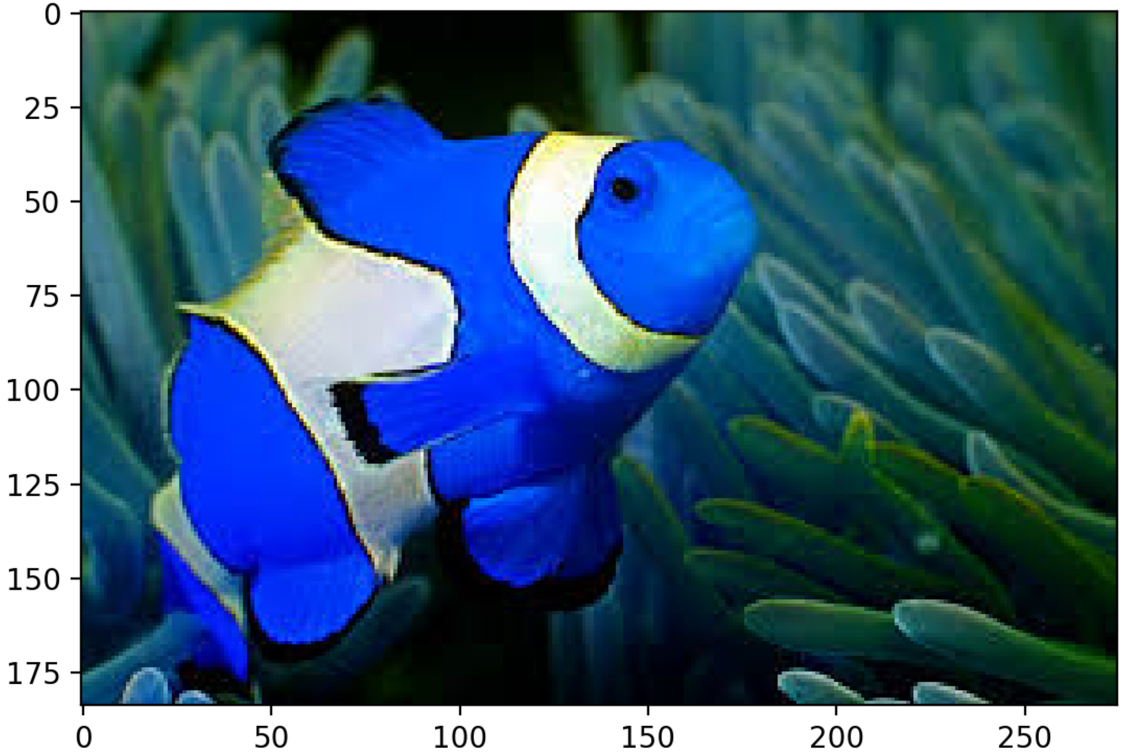
Image Segmentation

Image Segmentation

```
nemo = cv2.imread("nemo0.jpg")
plt.imshow(nemo)
plt.show()
```

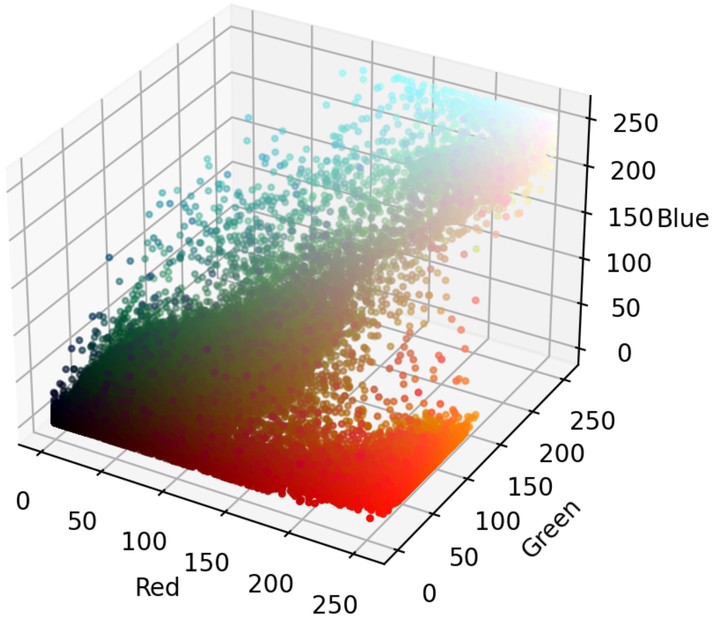


```
nemo = cv2.cvtColor(nemo, cv2.COLOR_BGR2RGB)

plt.imshow(nemo)
plt.show()
```



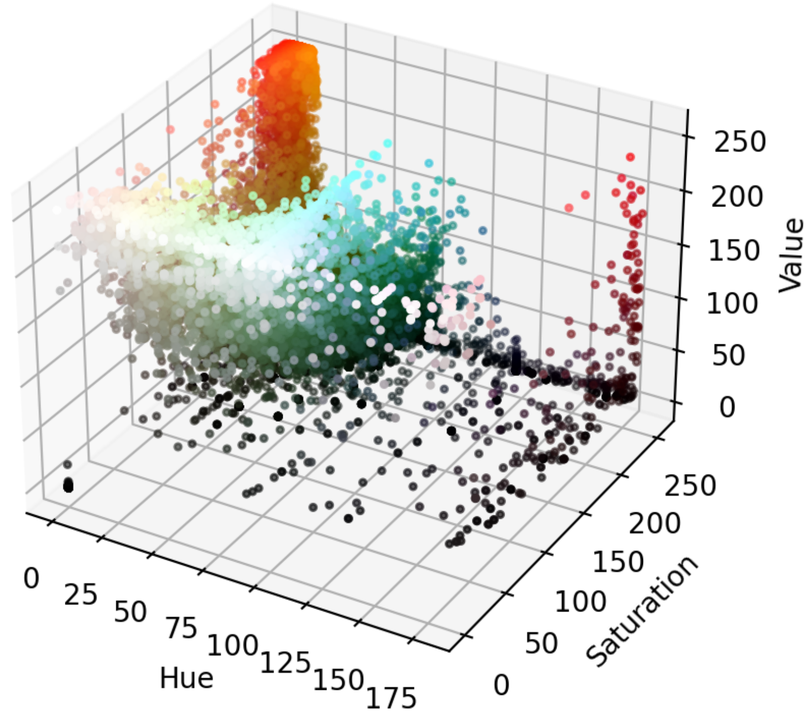
Visualizing in RGB Color Space

```
r, g, b = cv2.split(nemo)

fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")
pixel_colors = nemo.reshape((np.shape(nemo)[0] * np.shape(nemo)[1], 3))
norm = colors.Normalize(vmin=-1.0, vmax=1.0)
norm.autoscale(pixel_colors)
pixel_colors = norm(pixel_colors).tolist()

axis.scatter(
    r.flatten(), g.flatten(), b.flatten(), facecolors=pixel_colors, marker="."
)
axis.set_xlabel("Red")
axis.set_ylabel("Green")
axis.set_zlabel("Blue")
plt.show()
```

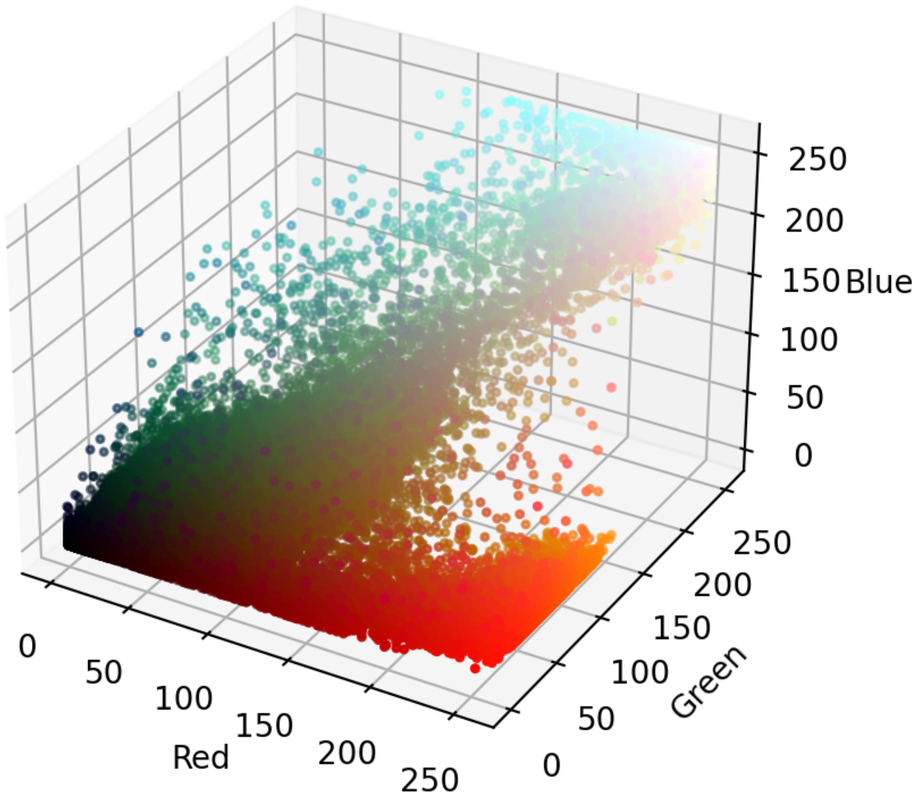


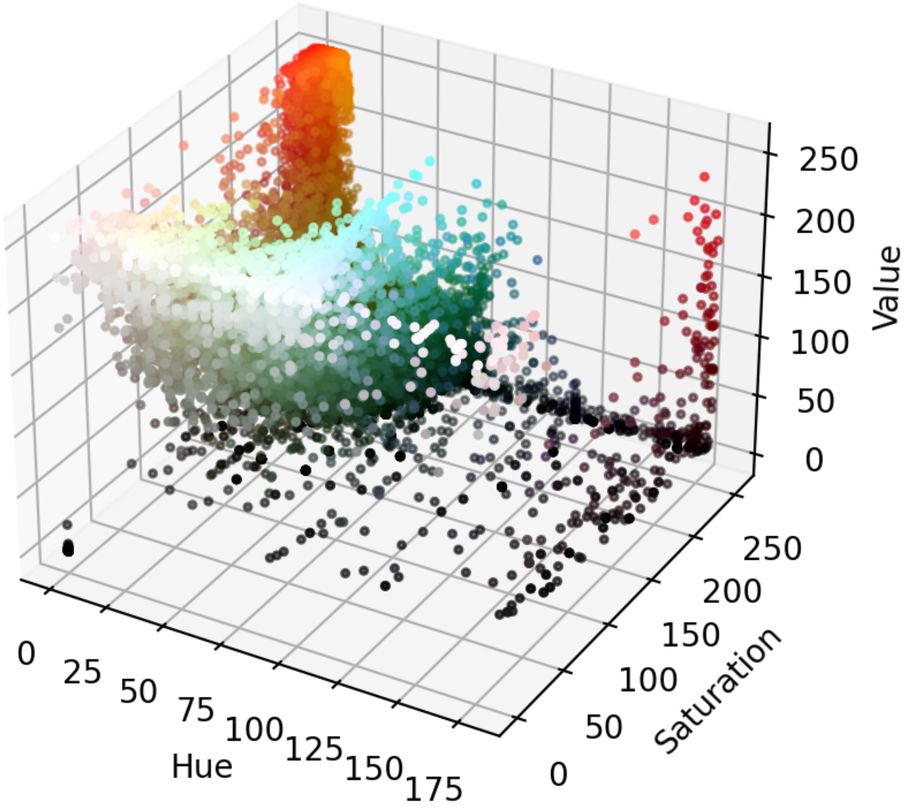
Visualizing in HSV Color Space

```
hsv_nemo = cv2.cvtColor(nemo, cv2.COLOR_RGB2HSV)

h, s, v = cv2.split(hsv_nemo)

fig = plt.figure()
axis = fig.add_subplot(1, 1, 1, projection="3d")

axis.scatter(
    h.flatten(), s.flatten(), v.flatten(), facecolors=pixel_colors, marker="."
)
axis.set_xlabel("Hue")
axis.set_ylabel("Saturation")
axis.set_zlabel("Value")
plt.show()
```



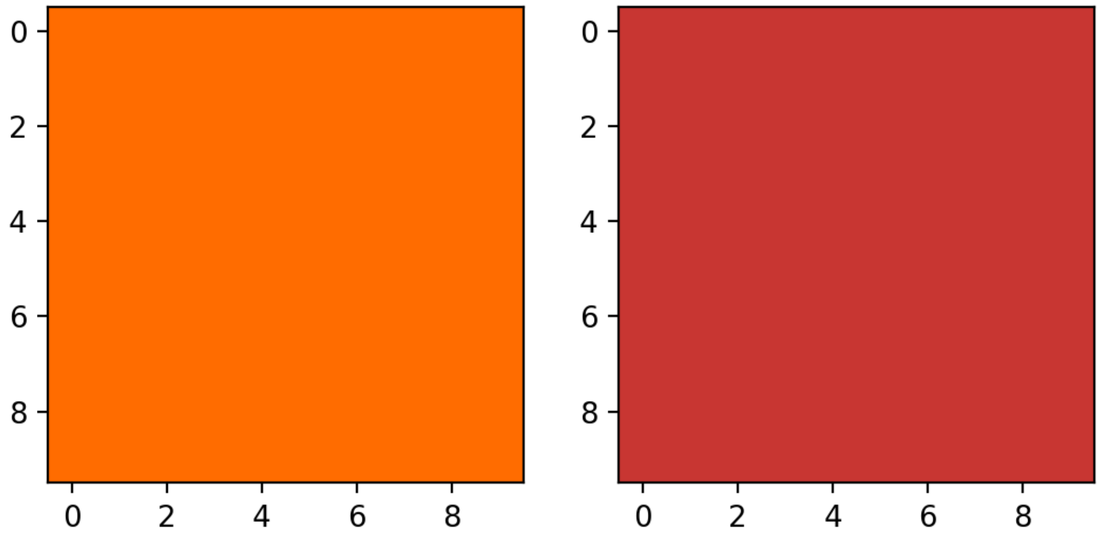
RBG vs HSL

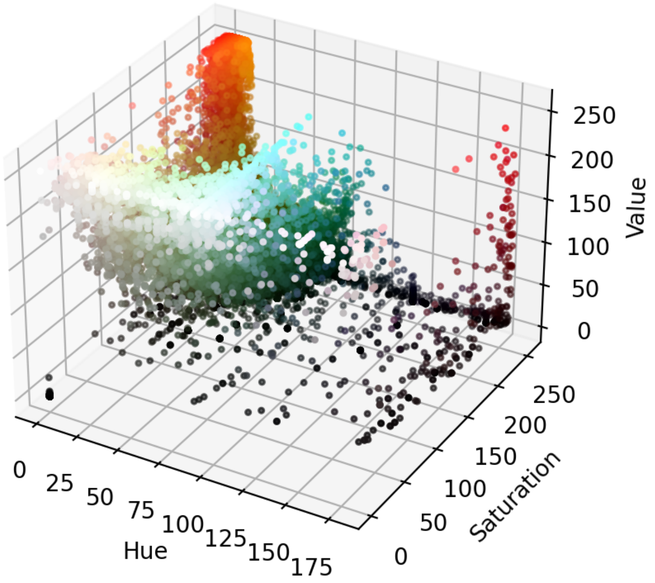

Red
Green
Blue
0
50
100
150
200
250


Hue
Saturation
Value
0
25
50
75
100
125
150
175
200
250

Picking Out a Range

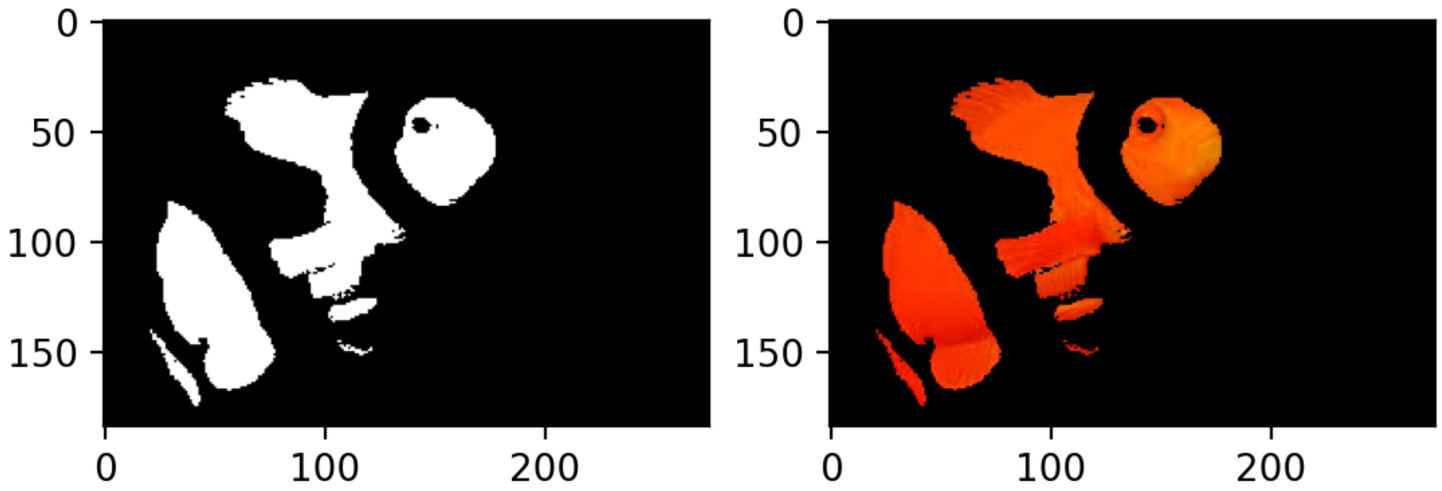
```
light_orange = (1, 190, 200)
dark_orange = (18, 255, 255)

# Normalise to 0 - 1 range for viewing

lo_square = np.full((10, 10, 3), light_orange, dtype=np.uint8) / 255.0
do_square = np.full((10, 10, 3), dark_orange, dtype=np.uint8) / 255.0

plt.subplot(1, 2, 1)
plt.imshow(hsv_to_rgb(do_square))
plt.subplot(1, 2, 2)
plt.imshow(hsv_to_rgb(lo_square))
plt.show()
```





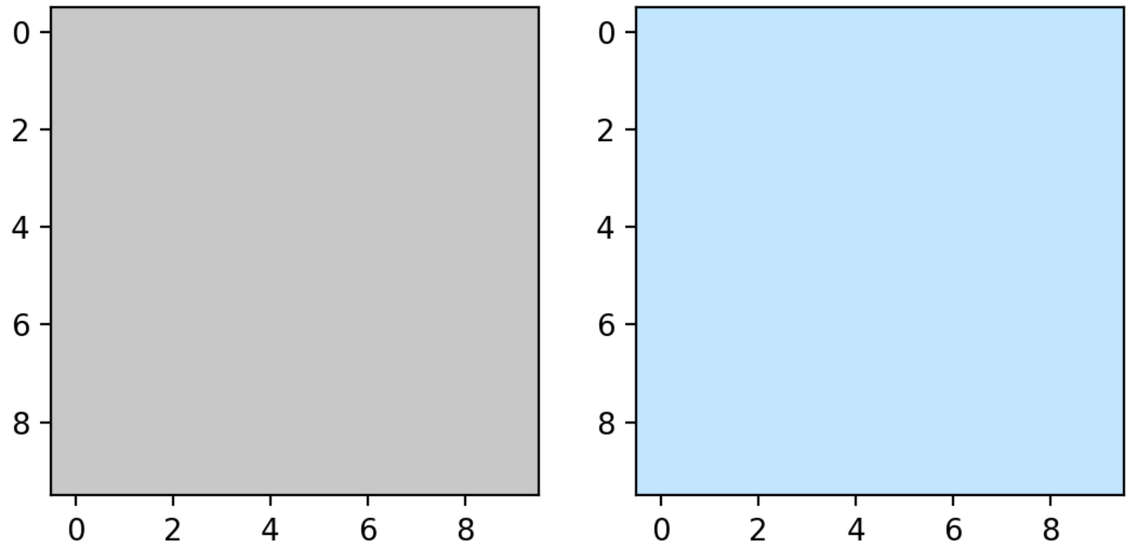
Picking Out a Range

```
mask = cv2.inRange(hsv_nemo, light_orange, dark_orange)

# Bitwise-AND mask and original image

result = cv2.bitwise_and(nemo, nemo, mask=mask)

# Convert back to RGB in order to plot using `matplotlib.pyplot`

plt.subplot(1, 2, 1)
plt.imshow(mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result)
plt.show()
```



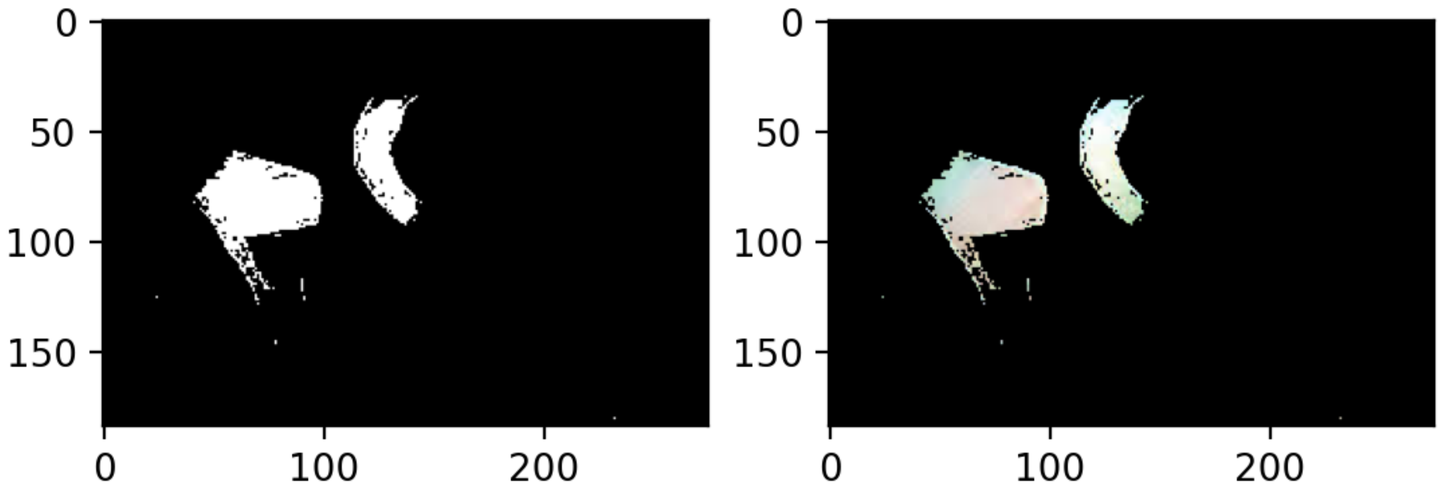
Adding a Second Mask

```
light_white = (0, 0, 200)
dark_white = (145, 60, 255)

lw_square = np.full((10, 10, 3), light_white, dtype=np.uint8) / 255.0
dw_square = np.full((10, 10, 3), dark_white, dtype=np.uint8) / 255.0

plt.subplot(1, 2, 1)
plt.imshow(hsv_to_rgb(lw_square))
plt.subplot(1, 2, 2)
plt.imshow(hsv_to_rgb(dw_square))
plt.show()
```



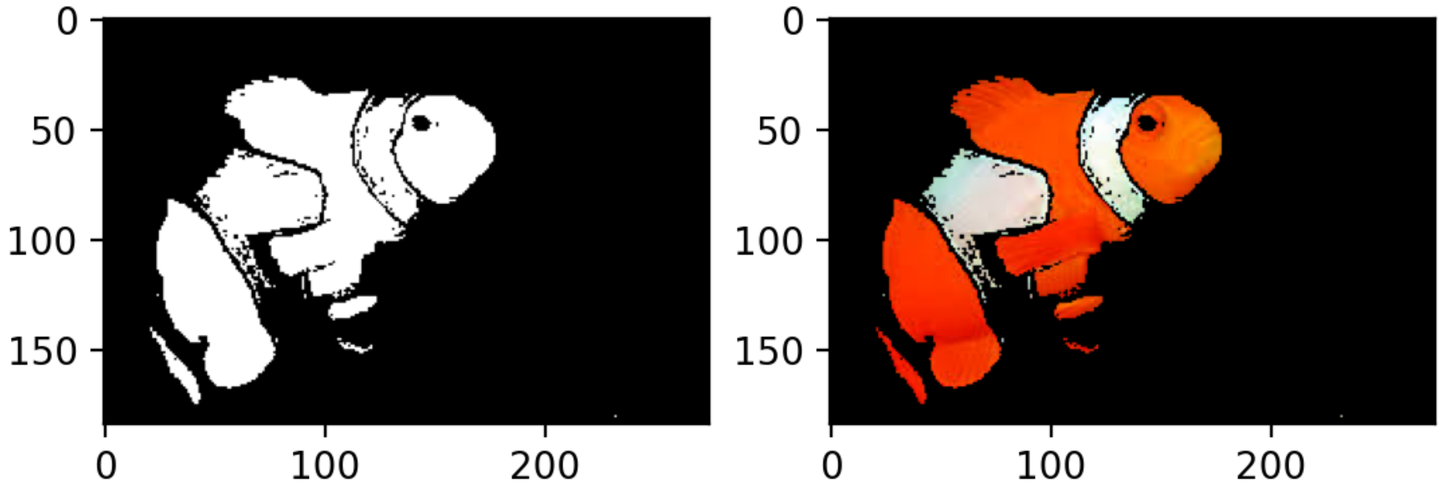
Adding a Second Mask

```
mask_white = cv2.inRange(hsv_nemo, light_white, dark_white)
result_white = cv2.bitwise_and(nemo, nemo, mask=mask_white)

plt.subplot(1, 2, 1)
plt.imshow(mask_white, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(result_white)
plt.show()
```

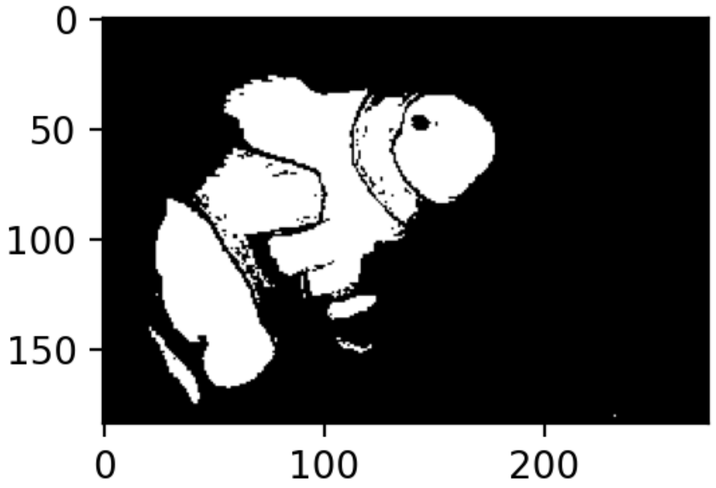


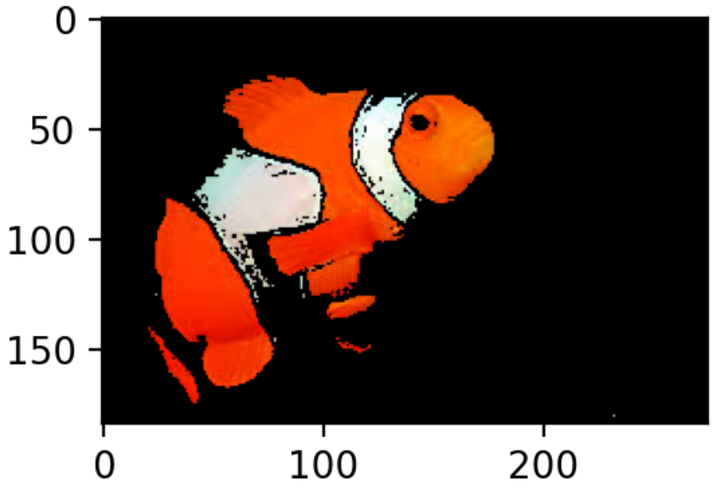
Combining Masks

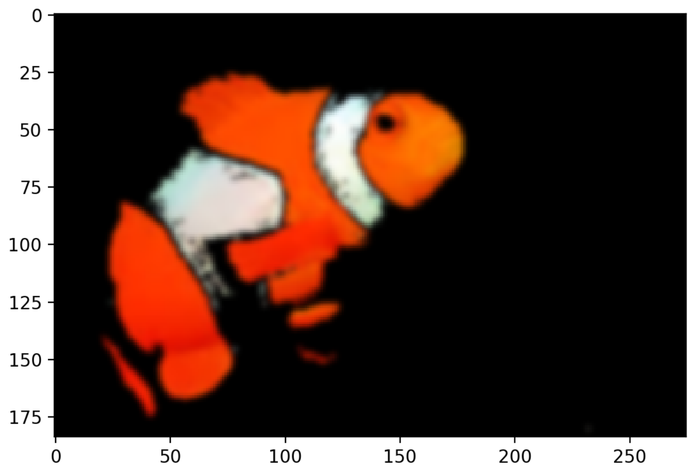
```
final_mask = mask + mask_white

final_result = cv2.bitwise_and(nemo, nemo, mask=final_mask)

plt.subplot(1, 2, 1)
plt.imshow(final_mask, cmap="gray")
plt.subplot(1, 2, 2)
plt.imshow(final_result)
plt.show()
```



Gaussian Blur







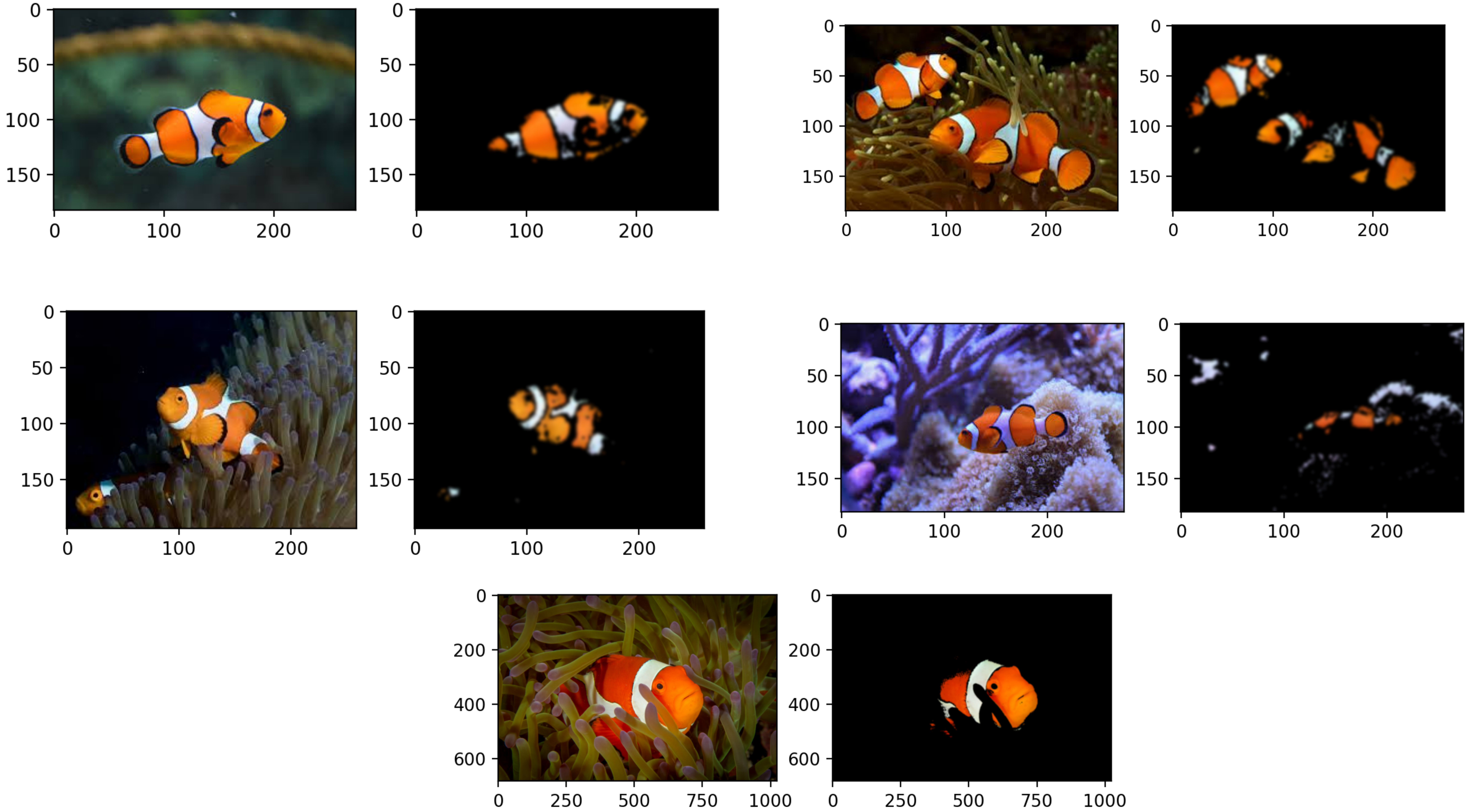

```
blur = cv2.GaussianBlur(final_result, (7, 7), 0)

plt.imshow(blur)
plt.show()
```

Generalizing the Segmentation

```
path = "nemo"

nemos_friends = []
for i in range(6):
    friend = cv2.cvtColor(
        cv2.imread(path + str(i) + ".jpg"), cv2.COLOR_BGR2RGB
    )
    nemos_friends.append(friend)
```

```
def segment_fish(image):
    """Attempts to segment the clown fish out of the provided image."""
    hsv_image = cv2.cvtColor(image, cv2.COLOR_RGB2HSV)
    light_orange = (1, 190, 200)
    dark_orange = (18, 255, 255)
    mask = cv2.inRange(hsv_image, light_orange, dark_orange)
    light_white = (0, 0, 200)
    dark_white = (145, 60, 255)
    mask_white = cv2.inRange(hsv_image, light_white, dark_white)
    final_mask = mask + mask_white
    result = cv2.bitwise_and(image, image, mask=final_mask)
    result = cv2.GaussianBlur(result, (7, 7), 0)
    return result
```

```
results = [segment_fish(friend) for friend in nemos_friends]

for i in range(1, 6):
    plt.subplot(1, 2, 1)
    plt.imshow(nemos_friends[i])
    plt.subplot(1, 2, 2)
    plt.imshow(results[i])
    plt.show()
```


0
50
100
150
0
100
200
0
200
400
600
0
250
500
750
1000