Finding the Center of a Circle

Paul Pearson

December 19, 2015

Finding the center of a circle

Here is the tiny 20 pixel by 15 pixel image that we will analyze:

We now read this image into R as an array (with three indices: two for pixel location and one for color) and flatten it to a grayscale image by averaging the RGB color channels.

```
# read the image into an array
library(jpeg)
img <- readJPEG("find_the_center_example.jpg",native=FALSE)
str(img) # what is the structure of img?
```

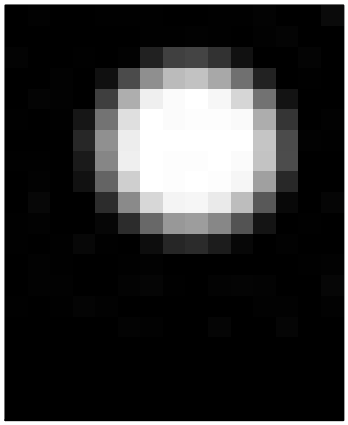
```
##  num [1:20, 1:15, 1:3] 0 0 0.0118 0 0 ...
```

```
img.gray <- (img[,,1] + img[,,2] + img[,,3])/3 # flatten to grayscale
str(img.gray) # what is the structure of img.gray?
```

```
##  num [1:20, 1:15] 0 0 0.0118 0 0 ...
```

Let's plot the `img.gray` matrix in grayscale and large, so that we can see individual pixels.

```
image( t(img.gray[20:1,]),
       col = grey(seq(0, 1, length = 256)), # make it grayscale
       xaxt= "n", yaxt= "n" # do not plot axes
       )
```



The image is now just a matrix

When we print the `img.gray` object, we notice that the values are all between 0 and 1, with black pixels taking the value 0 and white pixels taking the value 1.

```
img.gray # print the values in the matrix
```

```
##              [,1]        [,2]        [,3]        [,4]        [,5]
##  [1,] 0.000000000 0.007843137 0.000000000 0.000000000 0.003921569
##  [2,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##  [3,] 0.011764706 0.000000000 0.000000000 0.003921569 0.000000000
##  [4,] 0.000000000 0.000000000 0.007843137 0.000000000 0.062745098
##  [5,] 0.000000000 0.015686275 0.007843137 0.000000000 0.247058824
##  [6,] 0.003921569 0.000000000 0.003921569 0.086274510 0.450980392
##  [7,] 0.000000000 0.000000000 0.000000000 0.152941176 0.564705882
##  [8,] 0.003921569 0.003921569 0.000000000 0.101960784 0.525490196
##  [9,] 0.000000000 0.007843137 0.000000000 0.066666667 0.392156863
## [10,] 0.000000000 0.027450980 0.000000000 0.000000000 0.172549020
## [11,] 0.003921569 0.007843137 0.000000000 0.000000000 0.023529412
## [12,] 0.003921569 0.000000000 0.003921569 0.027450980 0.003921569
## [13,] 0.000000000 0.007843137 0.003921569 0.000000000 0.000000000
## [14,] 0.000000000 0.007843137 0.007843137 0.000000000 0.000000000
## [15,] 0.003921569 0.000000000 0.003921569 0.015686275 0.007843137
## [16,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [17,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [18,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [19,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

```
## [20,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##              [,6]        [,7]        [,8]        [,9]       [,10]
##  [1,] 0.003921569 0.003921569 0.000000000 0.000000000 0.003921569
##  [2,] 0.000000000 0.003921569 0.003921569 0.007843137 0.003921569
##  [3,] 0.007843137 0.129411765 0.250980392 0.266666667 0.196078431
##  [4,] 0.294117647 0.560784314 0.733333333 0.760784314 0.658823529
##  [5,] 0.682352941 0.937254902 0.984313725 1.000000000 0.968627451
##  [6,] 0.874509804 1.000000000 0.984313725 0.992156863 0.992156863
##  [7,] 0.929411765 1.000000000 1.000000000 1.000000000 1.000000000
##  [8,] 0.937254902 1.000000000 0.992156863 0.992156863 1.000000000
##  [9,] 0.811764706 1.000000000 0.992156863 1.000000000 0.992156863
## [10,] 0.541176471 0.843137255 0.956862745 0.964705882 0.917647059
## [11,] 0.172549020 0.407843137 0.596078431 0.615686275 0.529411765
## [12,] 0.000000000 0.050980392 0.149019608 0.168627451 0.109803922
## [13,] 0.003921569 0.007843137 0.000000000 0.000000000 0.000000000
## [14,] 0.011764706 0.019607843 0.003921569 0.000000000 0.011764706
## [15,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [16,] 0.011764706 0.007843137 0.000000000 0.000000000 0.019607843
## [17,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [18,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [19,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [20,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
##              [,11]       [,12]       [,13]       [,14]       [,15]
##  [1,] 0.003921569 0.011764706 0.000000000 0.000000000 0.039215686
##  [2,] 0.000000000 0.007843137 0.015686275 0.000000000 0.000000000
##  [3,] 0.066666667 0.000000000 0.000000000 0.015686275 0.000000000
##  [4,] 0.439215686 0.133333333 0.000000000 0.003921569 0.000000000
##  [5,] 0.839215686 0.447058824 0.074509804 0.000000000 0.000000000
##  [6,] 0.980392157 0.678431373 0.207843137 0.000000000 0.007843137
##  [7,] 1.000000000 0.772549020 0.294117647 0.007843137 0.003921569
##  [8,] 0.984313725 0.764705882 0.298039216 0.000000000 0.000000000
##  [9,] 0.964705882 0.627450980 0.156862745 0.000000000 0.000000000
## [10,] 0.741176471 0.345098039 0.031372549 0.000000000 0.019607843
## [11,] 0.325490196 0.078431373 0.000000000 0.003921569 0.000000000
## [12,] 0.027450980 0.000000000 0.007843137 0.000000000 0.000000000
## [13,] 0.000000000 0.000000000 0.000000000 0.003921569 0.007843137
## [14,] 0.015686275 0.011764706 0.000000000 0.000000000 0.023529412
## [15,] 0.000000000 0.007843137 0.007843137 0.000000000 0.007843137
## [16,] 0.000000000 0.000000000 0.019607843 0.000000000 0.000000000
## [17,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [18,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [19,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
## [20,] 0.000000000 0.000000000 0.000000000 0.000000000 0.000000000
```

Examining the properties of the matrix

When we print the 20 row sums of the matrix, we notice that row 7 has the largest row sum.

```
rowSums(img.gray)
```

```
##  [1] 0.07843137 0.04313725 0.94901961 3.65490196 6.20392157 7.26274510
##  [7] 7.72549020 7.60392157 7.01176471 5.56078431 2.76470588 0.55294118
```

```
## [13] 0.03529412 0.11372549 0.05490196 0.05882353 0.00000000 0.00000000
## [19] 0.00000000 0.00000000
```

```
which.max( rowSums(img.gray) ) # return the index of the row that has the max row sum
```

```
## [1] 7
```

Similarly, when we print the 15 column sums for this matrix, we notice that column 9 has the largest column sum.

```
colSums(img.gray)
```

```
##  [1] 0.03137255 0.08627451 0.03921569 0.45490196 2.45490196 5.28235294
##  [7] 6.97254902 7.64705882 7.76862745 7.40392157 6.38823529 3.88627451
## [13] 1.11372549 0.03529412 0.10980392
```

```
which.max( colSums(img.gray) ) # return the index of the column that has the max column sum
```

```
## [1] 9
```

So, a good estimate for the center of the white region in the image is row 7 and column 9. Why does this work? Because the more white pixels there are in a row (or a column), the larger the sum will be. When the row sum (or column sum) is largest, we have found a diameter for the white disk-shaped region. We are exploiting the fact that the region outside of the white disk is uniformly black.

Thought question: Will this method always work well to find the center of the white region? When will it work well? When will it fail?