Image Blurring

Thomas Smale

5/20/2022

Gaussian Distribution aka Normal Distribution

The first step in detection edges is noise reduction To accomplish this we will use Gaussian filtering which is used to blur images and remove noise. σ is the standard deviation of the distribution, which measures how spread out values are from the mean. A large standard deviation means there is high variance and a low standard deviation indicates low variance. A key component of the standard deviation is that +-1 standard deviations away from the mean represent 68% of the values. While +-2 standard deviations away from the mean account for 95% of the set. Finally, +- 3 standard deviations contain 98% of the values. We will take advantage of this to fit a kernal that represents the shape of this. The bell curve is a common way to visualize this. The center of the curve is the mean and the width of the curve gives an idea for the standard deviation.

1 dimensional Gaussian function:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$$

When working with images, we will use a 2 dimensional Gaussian function

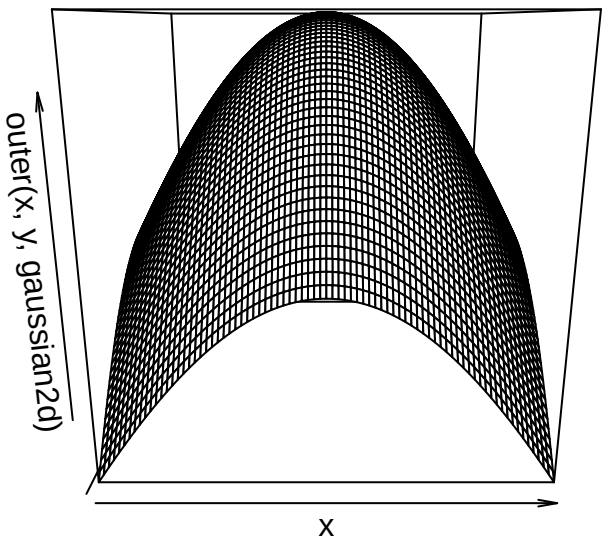
$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}}$$

The integral of the Gaussian Function also has importance.

$$I = \int_{-\infty}^{\infty} e(-x^2)dx = \sqrt{\pi}$$

* The square root of pi is the area under the graph of the Gaussian function * So the area under the curve will equal about 1.772.

```
gaussian2d <- function(x, y) {
  part1 <- 1 / (2*pi*sd(x*y)^2)
  part2 <- exp(1)^((-(x^2+y^2))/(2*(sd(x*y)^2)))
  return(part1*part2)
}
```

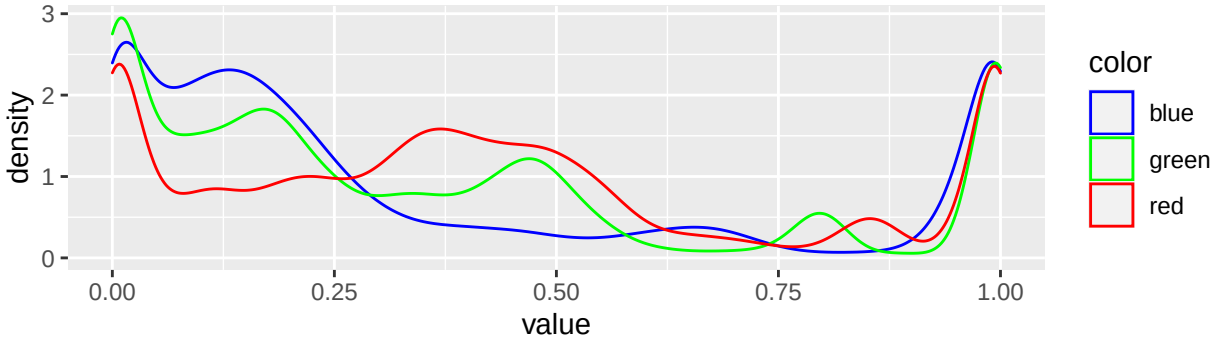


Gaussian Smoothing

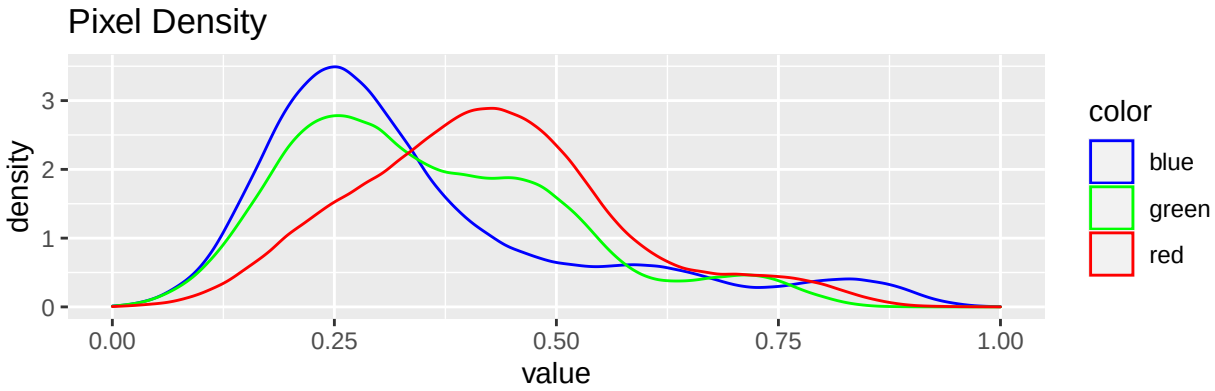
Here we use an extra noisy image to see what the pixel density looks like. The pixel density shows a distribution of the pixels. It is really a histogram that has been smoothed with a Gaussian curve. Before the image has been smoothed, its distribution is very hectic. There are all sorts of peaks and different skew's. Once we smooth the image with a Gaussian kernel, the image distribution looks more normal.

A Noisy Image

Pixel Density



The Noisy Image Smoothed



Importance of Standard Deviation

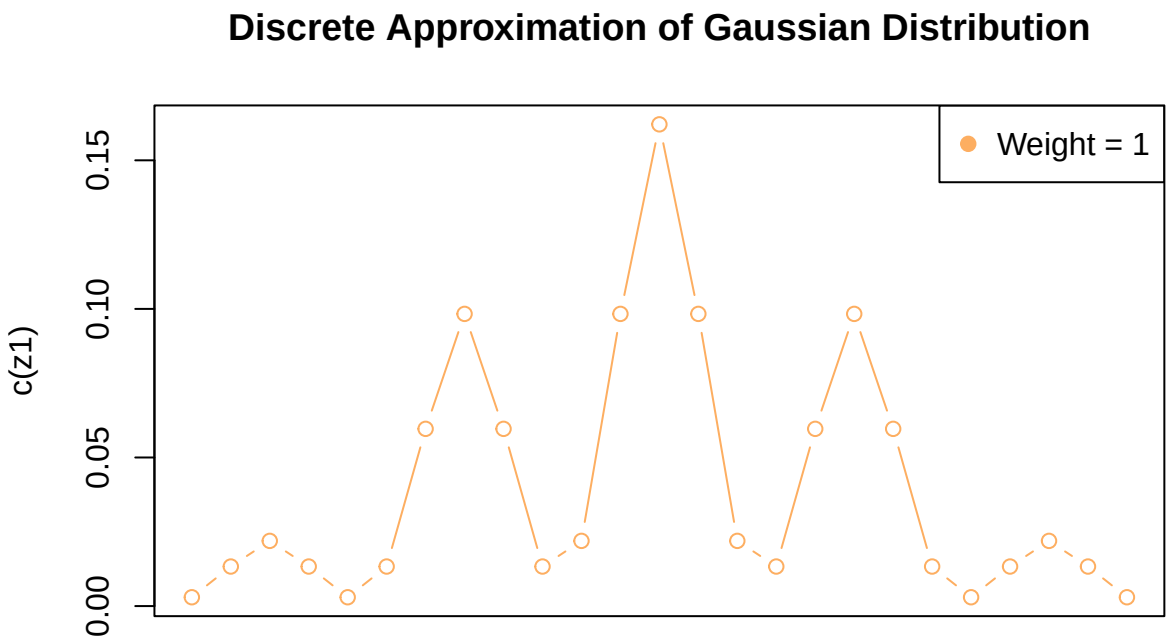
The standard deviation measures on average how far away observations are from the mean. The mean is the center of the Gaussian distribution. We can see that as we increase the standard deviation, we get more smoothing. However, increase it too much and the image goes dark!

Creating a kernel

The Gaussian curve is a continuous function, and we are working in a discrete space. Computer's do not have the ability to store an infinitely large image. So we need to take an approximation of the Gaussian function. We do this by creating a kernel.

A kernel is a square matrix of coefficients with an anchor point in the center. We refer to the center point aka anchor, as (0, 0). The rest of the points represent the distance from the anchor using pixels as a unit. We plug these values into the 2 dimensional Gaussian function to initialize the values of the kernel. The sum of the values in the kernel must equate to 1. The original output from the 2 dimensional Gaussian function will not equal 1. So we take the sum of the values outputted, set it as the denominator, and divide by 1. Then we multiply that, (1 / sum(x)) by all the values from the output. The sum of the values in the kernel will now equal 1. We use the kernel to apply convolutions to the image to produce edge detection.

$$\begin{matrix} (1,1) & (0,1) & (1,1) \\ (1,0) & (0,0) & (1,0) \\ (1,1) & (0,1) & (1,1) \end{matrix}$$



2 important concepts conveyed in the graph above is that the kernel is symmetric and the overall shape resembles a bell curve. The different peaks represent 1, 2, and 3 standard deviations from the mean (center) of the bell curve. As the weight changes, the y-axis becomes smaller and smaller.

Convolution

Convolution is the process of applying the kernel over the original image. We place the anchor of kernel over every pixel in the image. We perform a matrix multiplication with the kernel and the pixel's neighbors. Then, compute the sum of the products. The output of this value is the new pixel value in the transformed image.

When applying a kernel to an image, it may result in a smaller size. The output of the image can be calculated using this formula.

```
# Calculate the output size of image after applying filter and convolution
# dim is width or height of image
# ksize is size of kernal
# padding is if we want a border or something like that
# stride is how much the filter increments by
new_size <- function(dim, ksize, padding, stride) {
  return(((dim-ksize+2*padding)/1)+1)
}
```

However, in this case we will be using a technique called zero padding. This pads the image with zero's so the result from the convolution is the original image size. The number of rows and columns of zero's we pad

the original image with is calculated by the number of rows/columns in the kernel - 1. Then we apply the convolution, it's mathematical formula is given below.

2D convolution forumla

$$y[i,j] = \sum_{m=-\infty}^{\infty} \sum_{n=-\infty}^{m=\infty} h[m,n] * x[i-m, j-n]$$

Here x is the input image and y is the output which is the new image. H is the kernel matrix. I and j iterate over the image while m and n deal with that of the kernel.

When applying a 2d convolution using a Gaussian filter it reduces the noise in the image. This gives the effect of blurring the image.

Results from convolution

To see my comparison to an openCV comparison run the blur.py file.

Sequential vs Pthread

Making programs run efficiently is one of the most important jobs of a computer scientist. Programs must take advantage of ever improving changes to the hardware. Some ways to do this are by using pthreads, MPI, or openMP. For this project I used Pthread's because of how compatible it is. A potential problem with using pthread's is that it can require a mutex lock which serializes the code. Convolution can be considered an embarrassingly parallel program since it is not very difficult to get speed up. What I did was divide the work among the threads. So each thread convoluted a part of the image. It resulted in very successful results!

Time was measured using the time library from C. This was wrapped around the function call to convolute. That way, measurements like file io do not make my results bias. However, the program as a whole has room for speedup. The program is able to read a directory recursively. File IO can be made faster by allocating multiple threads to read and write to files.

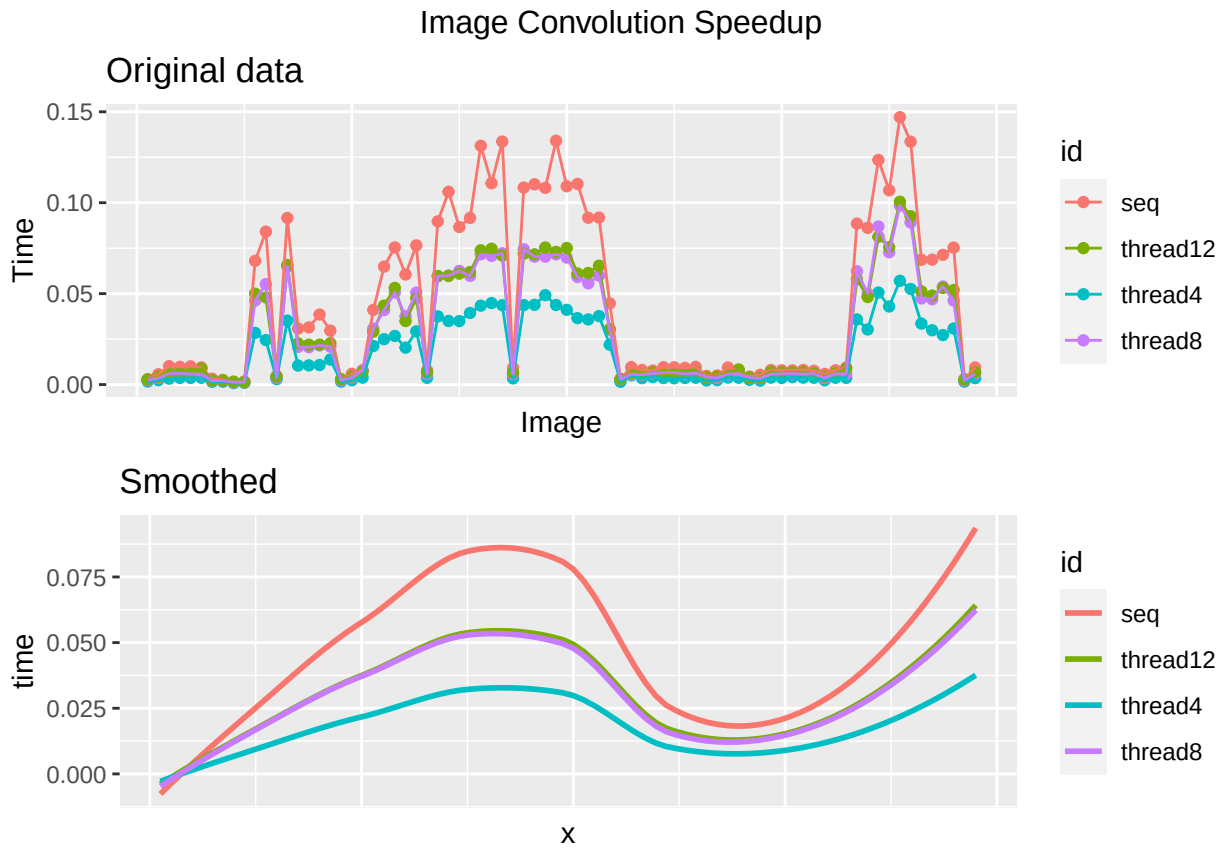


Image Convolution Speedup
Original data
0.15
0.10
0.05
0.00
Time
Image
id
seq
thread12
thread4
thread8
Smoothed
0.075
0.050
0.025
0.000
time
x
id
seq
thread12
thread4
thread8