Assignment 1: Mean Shift Segmentation

Daniel Sawyer
University of South Florida
danielsawyer@mail.usf.edu

Abstract

*Naive implementation of the Mean Shift algorithm using L*a*b* color space as well as gray scale images. Computational implementation is using the Cuda JIT compiler built into the Numba library for python. The Cuda computations reduced the time for each iteration/step for a 229x229 gray scale image from 20-25 minutes an iteration/step to 3-5 seconds an iteration/step. This led to a significant increase in performance compared to the standard naive CPU implementation.*

1. Algorithm

The mean shift algorithm is fairly simple. It uses a Gaussian kernel that calculates a the spatial and color distance from the current pixel, X. This pixel X is a vector that contains values for intensity or three channel color, depending on if it is gray scale or color, and also includes the position of the pixel in a euclidean plane. You calculate the color and euclidean distances from the current pixel to every other pixel in the image. This is naive, but optimal. It is extremely slow due to its naive nature, in that it reads every other pixel even though the pixel is obviously further than the radius of the distance to the other pixel, but is guaranteed to converge with the optimal amount of steps/iterations. Since the naive implementation runs at (rows x cols) squared, with large and complex images, which can easily become millions of comparisons. The easy solution to this problem is implementing the naive algorithm into a naive Cuda kernel, which will take advantage of the massively parallel nature of Cuda. The Gaussian kernel is exp(0.5 * M((X-XI) / H)$\hat{2}$ and is easily implemented using nested for loops.

2. Related Work

I used the Gaussian kernel Dr. Sarkar used in class along with the basic CUDA tutorial for Numba in python[1]. Besides that I did not use any code from any other sources and you can see my python code implementation in smcImgCuda.py located in the same directory as this report file.

3. Examples

I initially ran the algorithm with Hr=8, Hs=7, and M=40 along with the number of standard deviations for Hr, called SDR, and Hs, called SDS of 3. The following images are comparisons between the differnt parameters set for Hr, Hs, and M. fig:baboon

Figure 1. Baboon Original

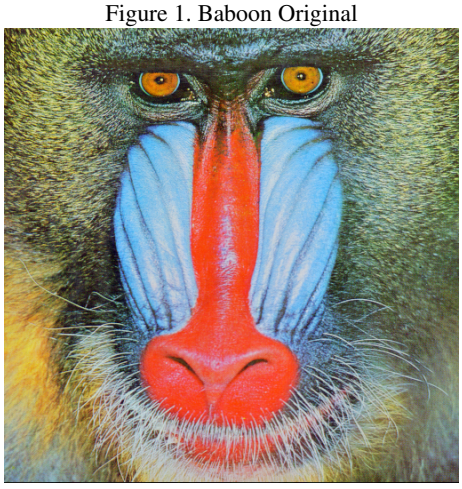

Figure 2. Baboon(8, 7, 40) 50 Steps

Figure 3. Baboon(8, 7, 40) 1000 Steps

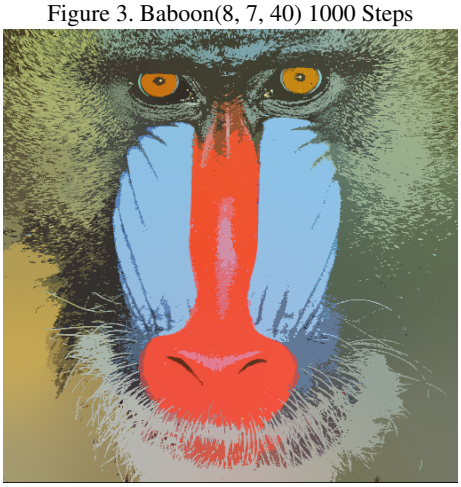

Figure 4. MM Original

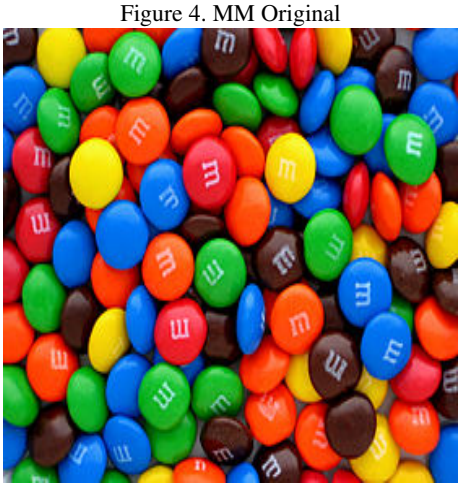

Figure 5. MM (8, 7, 40) 100 Steps

Figure 6. MM (8, 7, 40) 2000 Steps

4. Lessons Learned

I learned quite a few lessons with this project. The major lesson is how you can use just intensity/color values combined with spatial location in order to segment an image. It is quite amazing how you can use a Gaussian kernel in order to make the pixels cluster and shift to a relative mean that converges everytime. I was quite suprised on how well it converges. In figure 2 and 3, the only difference is that figure 2 is 50 steps and 3 is 1000 steps. You can tell that it basically converged at 50 steps and by 1000 it had completely converged with very minimal differences between the two. It was the same for the MM images. Figure 5 was 100 steps where figure 6 was 2000. I dont think there is hardly any difference between the two, so after a certain amount of steps, there is no noticable difference in intensity values, hence the image has converged. I also learned how valuable parallel processing is, I would have never been able to complete this project on time if it was not for the CUDA implementation of the algorithm since the naive CPU version is very, very slow.

References

[1] Numba cuda programming. 2019.