CUDA Powered Mosaic Solving

By: Ryan Stemen & Shiraz Ahmad

CECS 590

Fall 2009

# Summary

Mosaic Mason is an application that allows the user to load images and create a mosaic by recreating a larger master image from a database of candidate images. This report outlines the algorithm for the Mosaic Mason, as well as the steps taken to transform it into a CUDA driven application.

# Introduction

The Mosaic Mason application is written to run in parallel. It was also initially written to execute its image processing on the CPU. The intent of this project was to modify the code to execute image processing on the GPU, which allows for faster computational performance. The algorithm outlined in this report alters the original code to implement CUDA principles using basic Matrix Multiplication concepts.

# The Algorithm

The Mosaic Mason implements an algorithm involving a multistep process that requires several components to be executed on the CPU. Each section of the algorithm has been outlined below. Through our analysis we have identified key sections of the algorithm that can be enhanced through the use of CUDA. The new algorithm will take advantage of CUDA principles by executing image processing on the GPU.

1. The first step in the algorithm is the selection of the master image—the image that is to be recreated by many small image cells. There are no explicit rules for the selecting master images but there are a few recommendations.
2. Select an image that has a common aspect ratio, typically 1.333 or 0.75. This is because the candidate images must be of the same aspect ratio as the master.
3. Select an image that is medium to large in size. Selecting an image that is too small can result in poor mosaic image detail.
4. Select an image that does not rely on small features to convey its message. Small features are typically lost in a mosaic unless you have a very large candidate image set.
5. Select an image that does not have large swathes of the same color. This makes the resulting mosaic more interesting generally.
6. Once the master image has been selected the candidate image pool must be selected. The user simply selects all images that he/she would like to be considered for placement into the mosaic. If the pool contains images that are not of the appropriate aspect ratio then they will be automatically excluded from the candidate image pool. Once again, there are a few recommendations for selecting images for the candidate pool.
7. Select only images that you would like to be in the mosaic, this typically means excluding blurred or improperly exposed images.
8. Select a wide variety of colors, brightness’s, and textures for your candidate pool wherever possible.
9. Try to select candidate images whose content is related in some way to the content of the master image. For example: if the master image is the Mona Lisa painting by Leonardo da Vinci, then a good choice for the candidate pool of images would be other paintings and sketches by da Vinci.
10. After the candidate image pool has been defined, the parameters of the mosaic creation process should be defined. There are three main factors that will define the quality of the mosaic at this point. The first factor is the number of candidate images per row (N), which defines the overall number of image cells that will be used to recreate the master image. There will be a total of images per mosaic due to the fact that the aspect ratio of the candidate images must match the aspect ratio of the master image. The value that the user selects for the number of candidate images per row has the largest impact on the mosaic’s overall ability to represent the master image.
11. The next factor is the subdivision level (S) of each candidate image. Each candidate image and corresponding image cell location in the master image is divided up into S by S regions for a total to regions. For each region the average color of the pixels is calculated and assigned to that block. During the comparison stage it is these regions that will be compared, not individual pixels, thereby making the algorithm partially scale invariant. Additionally, the subdivision level that the user selects influences both the maximum recognizable feature frequency in the image’s frequency domain, as well as the mosaic’s maximum theoretical resolution.
12. The last factor that the user controls in the mosaic creation process is the whether the master image should be interpreted as a grayscale image or a color image. If the user opts to interpret the image as a grayscale image then the entire candidate image pool will be interpreted as grayscale as well. In most cases it is beneficial with respect to the mosaic image quality to enable grayscale interpretation. This is due to the fact that it significantly reduces the complexity of the mosaic solving equations. Here are the general recommendations about picking the appropriate factor settings:
13. Try using a value around 32 for N as picking a high value of N (i.e. >48) will typically yield a high quality recreation of the master image, but will come at the cost of image file size. In order to see all the candidate images clearly the output mosaic image must be very large.
14. Select a value of S that is greater than two but less than ten. This is due to the fact that in selecting a higher subdivision the algorithm will give a preference to relatively high frequency features which the human eye is not particularly sensitive to.
15. Experiment with using grayscale and color modes of the mosaic creator in order to find the result that works best for you.
16. The next part of the algorithm must calculate the cost of the solution, which will be the number of candidate images being used multiplied by the number image cells in the master image (). In order to create a mosaic every one of the cells in the master image must be compared to every candidate image. This is where image processing comes into play; this is also the section of the algorithm that we believed could be accelerated through the use of CUDA.

This portion of the code was refactored to use the CUDA Driver API. The code is organized in such a way that 2D blocks are used. A block size of 16X16 is used for a total of 256 threads running at any given moment. The Manhattan Distance is used to calculate the distance of candidate image from the current block. Two processes are used to calculate the distance for each image:

1. For each 16X16 block, a thread is created for each image cell using its thread ID and block ID in a 2D array.
2. Within the thread for the current image cell loop through each sub-image in the current block and compare it to all candidate images. For each candidate image calculate the Manhattan distance from the master image. Keep track of all current best results.
3. After the all of the image calculations are done, each sub-image’s best candidate image is determined and the final mosaic is outputted to the user.

# Results

The following are the results gathered from running both the CPU and accelerated CUDA versions of Mosaic Mason. The leftmost column represents the NXS (Number of sub-images X subdivided cells). The CPU and GPU values, in milliseconds, are shown to the right. The rightmost column shows the GPU speed-up. The processor used was the AMD Phenom™ II X3 720 (2.80 GHz). The GPU used was the NVIDIA Tesla C1060.

|  |  |  |  |
| --- | --- | --- | --- |
|  | CPU/ms | GPU/ms | Speed-up |
| 128X3 | 33703.13 | 49.414 | 682.0563 |
| 64X3 | 8421.88 | 10.799 | 779.8759 |
| 32X3 | 2140.25 | 3.112 | 687.741 |
| 16X3 | 562.5 | 3.032 | 185.5211 |
| 128X2 | 16484.38 | 26.863 | 613.6461 |
| 64X2 | 4078.13 | 6.017 | 677.768 |
| 32X2 | 1031.25 | 1.924 | 535.9927 |
| 16X2 | 281.25 | 1.625 | 173.0769 |
| 128X1 | 5781.25 | 13.492 | 428.4947 |
| 64X1 | 1359.375 | 3.251 | 418.1406 |
| 32X1 | 359.375 | 1.126 | 319.1607 |
| 16X1 | 109.375 | 0.618 | 176.9822 |

# Conclusion

The results clearly show a dramatic speed-up in the computation time for the GPU version versus the CPU version. Our data shows a speed-up of 600 times or more for large mosaics. Therefore we can conclude that implementing CUDA in this situation is a worthwhile endeavor and can greatly improve the performance of this application. As we had hoped CUDA not only sped up the calculation time, but will also allow the application to run using a greater level of subdivided cells and as a result will provide more detailed mosaics.