CS 4F03 Distributed Systems Final Project:
Honey, I box and sphere folded the kids!

Brandon Byskov
1068517

Mitch Errygers
1137538

Ramy Samad
1152051

Kirisanth Subramaniam
1054839

April, 2015

Computed on Amazon Web Services EC2 C3 with 32 cores in 8 hours at 4K resolution

CS 4F03: Final Project
Department of Computing and Software
McMaster University

Instructed by:
Ned Nedialkov

Contents

1 Introduction

Our MandelBox video was created using two MandelBox programs. One paramater generator created 7200 *paramsXXXX.dat* files, and the main MandelBox program takes the params files and generates images in the form *fXXXX.bmp*. FFmpeg was used to create the video out of the images. The music of Jefferson Airplane was included in the video. This project is based on and incorporates some of the work of Thomas Gwosdz and Ned Nedialkov. For this, they should be acknowledged.

2 Running the Program

2.1 Running the Main Program

The main MandelBox program can be run on a set of params files. A *params.zip* file is provided in our repository, containing the params files that we used to generate the final video.

The program can be compiled by running:

```
$ make
```

The program can be run using the command:

```
$ ./mandelbox params/params%04d.dat images/f%04d.bmp 0 7199 3840 2160
```

There must be an *images* folder already present, or else the program will fail when trying to write the images. The first two parameters indicate the string format of the input params files and the output image files. The next two parameters indicate the first and last images in the range to be generated. The last two parameters indicate the width and height of the output images.

2.2 Running the Params Generator Program

This program will genereate a set of params files for the Main program to use. The source code is included in the repository in a subfolder. A *params* folder be present for the program to write it's output params files.

The program can be compiled by running:

```
$ make
```

The program can be run using the command:

```
$ ./mandelbox params.dat
```

Params.dat is the data file that will initailize all of the main parameters. The program is designed to generete the first 1200 frames in a pre-determined way, and randomize the rest of the frames. Because of certain intial paramaters are expected for the first 1200 framse, the included *params.dat* file should be used.

3 Video Description and Contents

The final video is 4 minute long display of the mandelbox we generated. It was created at 4K resolution (3840 x 2160).

3.1 The MandelBox

The MandelBox in this video constantly transforms throughout the video. This was done by adjusting the *rMin* and *rFixed* parameter for every frame. They were both modified using a *sin()* function. The *rMin* parameter varied between 0.25 and 0.75, with a wavelength of 15 seconds. *rFixed* varied between 1.0 and 2.0, with a wavelength of 30 seconds. This was the main creative aspect of the video, as well as the camera that had to be developed to properly move through this changing box. The colors of the box were also adjusted using some non-standard functions. Those modifications can be seen in the source code of the program.

3.2 The Camera

The development of the camera took a significant amount of effort. The movement path of the camera could not be pre-determined, because the layout of the internals of the MandelBox was constantly changing. The camera was set up to move along a random path through the MandelBox, and adjust it's look target randomly throughout the program. The movement speed of the camera was set to be mostly constant, save for some necessary situations. The Camera look direction change was also set to be slow and smooth. It would always look towards a random target inside the box.

3.2.1 Collision Detection

A collision detection system had to be set up in order for the camera to avoid moving through or getting stuck between any walls. Becasue of the nature of the changing MandelBox, objects could move or spawn in the pathway of the camera. Objects could also move or spawn at any instance to engulf, trap or hit the camera. The collision detection system we developed allowed the camera to detect objects nearby it every frame and avoid them as necessary.

Each frame, the Camera would detect the distance from it to other objects in 14 different directions. Six directions were parallel to the x, y, and z axis in both positive and negative directions. The other eight were each set to be equally spaced between the first six axis directions ([sqrt(1/3), sqrt(1/3), sqrt(1/3)], etc.). We set the camera to move away from any objects that got too close. It would change it's pathway to be oppositedirection of the objects that were very close to it. This created and interesting *bounce* effect. If, by chance, the camera became enclosed by objects and couldn't *bounce* out. Then it would move in the opposite direction of it's last pathway until in reached an open area. We set the minimum distance that was allowed between the camera and an object to be 0.05. This was a good value to keep the camera safe from getting stuck inside an object and having to find a way out. The minimum allowed distance combined with the collision

directions checked essentially resulted in a virtual sphere around the camera that no object was allowed in. It would be this sphere that moves through the MandelBox, with a camera inside it looking in various directions.

This method can make frames appear jittery when the camera is very close to two opposing objects, but not stuck inbetween them. It can bounce back and forth between them, but this is usually only for a short time, as the objects will continue to move and transform. The will either come together and push the camera out, or simply move apart or disappear. This occurs only for a small portion of the duration of the video, and is not significantly detrimental to the viewing experience.

3.3 Format of Video

The first 30 seconds were spent by the camera in a stationary position outside the MandelBox, observing the full range of changes it can make. Then, It moves towards one corner of the box and passes inside of it. The final duration of the video is spent by the camera moving around randomly, but mostly bouncing between objects that appear and disappear. The Camera looks around various parts of the box.

4 Parallelization and Optimization

4.1 Multi-threading with OpenMP

OpenMP was used to distribute the calculation of pixel values between threads. For each frame that was generated, each thread would calculate every $(p + kn)^{th}$ row of pixels, where p is the thread id, n is the total number of threads, and k is an variable integer. This allowed a fairly even distribution of work across threads. Alternating rows has the advantage of forcing each thread to calculate regions all across the image from top to bottom. This will balance the workload in the event that some particular region is easier to compute than another, since threads will likely work on both the hard and easy parts of the image.

OpenMP was chosen over more highly distributed options such as MPI. The shared memory when using OpenMP allowed all threads to write to the same array of pixel results and get all initialization parameters without transfering much data across networks. There is also much less overhead in spawning new threads, which occurs once for each frame that is generated.

This method has the disadvantage that only one thread can work during the saving of the images and initialization of the parameters for each frame. We are also limited in computer power by the size of compute nodes that are available. The ones we had access to were limited to 32 cores per node. Since all params files were generated ahead of time, it would be possible to runs separate jobs on separate nodes.

4.2 Code Optimizations

Many aspects of the code were optimized. We implemented many optimizations, including inlining functions, eliminating redundant or repeated operations, approximating calculations through simpler estimation methods, reducing the number of operations required for some calculations, reducing new stack allocation from function calls, and eliminating any uneccessary parts of the program. Most of the effort was spent on the inner loops of the program, namely the distance estimator and ray marcher. Notable approximation calulations that we change to achieve a slightly less accurate but quicker approximation were with the calculation of the surface normal of a point of the MandelBox and faster convergence to the total distance to the point, both methods being used in the ray marcher.

The -O3 optimization flag was used as a minimum, and -Ofast was used when the server's compiler supported it. The compiler flag -funroll-loops was also used to unroll loops in the program.

We used a loop in the main function to loop over all the frames that needed to be created. This allowed us to create all of the frames for an entire video with only a single process being created.

5 Performance

5.1 Server Hardware

The following performance benchmarks were run on an Amazon Web Services (AWS) EC2 C3 instance. This was a compute focused server node with 32 cores, and enough storage to save approximately 175 GB of image data. The server was the c3.8xlarge node. It made available 32 virtual cores of Intel Xeon E5-2680 v2 processors, which are 10 core, 20 thread processors with 2.8Ghz base frequency, 3.6 Ghz turbo frequency, and 25 MB of top level cache. They are based on the previous generation Ivy Bridge architecture. There was made available 60 GB of memory and 320 GB of SSD storage.

5.2 Image Generation for Video

The images were generated by a single process that computed the entire range of frames from 0 to 7199. The program was provided with previously generated params files. The entire run took 7 hours and 56 minutes. This equates to just under 4 seconds per frame. Due to an error with code that reported the cpu time of each image during the image generation process, a seperate benchmark was run on a corrected version of the program to test the performance. We will provide an analysis this main run of the program after we present the performance benchmarks below.

5.3 Benchmarks

We ran three test on two versions of the code: one jog with the original code provided by the instuctor, and the final optimized version in both serial (one thread) and parallel (32 threads) jobs. The programs were timed running the job of generating 50 images over the range from 2000 to 2049. This region of the video would be when the camera is inside the MandelBox. The real time was calculated by the computer, using the Unix *time* command. The cpu time of generating each of the images was printed for both tests of the optimized versions of the program, and those are attached to the end of this report.

5.3.1 Original Code

This code provided by the instructor was run unchaged by our group. The code is not parallelized.

```
Benchmark Results:
Machine:            AWS EC2 c3.8xlarge
Cores/Threads:      1
Real Time           66m 41.579s
Total Frames        50
Seconds per Frame   80.03s
```

5.3.2 Serial Optimized Code

This program is identical to the final version of the code, containing all optimizations, but run on only a single thread.

```
Benchmark Results:
Machine:            AWS EC2 c3.8xlarge
Cores/Threads:      1
Real Time           50m 49.890s
Total Frames        50
Seconds per Frame   61.00s
```

5.3.3 Parallel Optimized Code

This program is identical to the final version of the code, containing all optimizations, run with 32 threads.

```
Benchmark Results:
Machine:            AWS EC2 c3.8xlarge
Cores/Threads:      32
Real Time           2m 22.450s
Total Frames        50
Seconds per Frame   2.85s
```

5.4 Analysis

As previously stated, the cpu time of each image was not recorded for the run of the 7200 image frames for the video, but only for the benchmark runs. We see that based on this presented data that our code optimiziation improved performance by 31.2%. Parallelization over 32 threads further improved performace by 21.4x. This is an improvement from the origianal code of 28.1x. We can attribute the lack of perfect efficiency scaling with cores to the serial portions of the code, notably large storage transfers of about 24 MB that must be performed after every frame. There may also be a small difference in the number of calculations that are performed in each row of pixels that each thread recieves to calculate.

Our run of the code over 7200 frames was performed in 7 hours 56 minutes. If we exprapolate the data from our benchmark run of 50 frames in 2 minutes 22.45 seconds, we should have an expected time for 7200 frames of only 5 hours 42 minutes. This realization caused us to look more closely at the benchmark data to reveal the culprit of this divergence.

We have attached the cpu times generated from the benchmark runs at the end of this report. One can see that in both runs of the final code, the cpu time steadily increases over the course of the run. This is most likely due to a reduction in processor frequency throughout the duration of the run. Intel's processors will use a higher frequency than their base frequency when temperature and power usage conditions allow it. As the temperature of the processors increase over the run, the processors will spend less time

at their turbo frequency, and more time at their base frequency. This is a large difference in frequency, with 2.8 GHz at the base clock and 3.6 GHz at the turbo clock. The server would likely generate a large amount of heat when being fully utilized over a long run, and the dynamic frequency scaling as a result of this can explain the differences in expected and actual runtime.

6 Appendix

6.1 Benchmark Console Logs

6.1.1 Original Code

```
$ time ./test.sh 2000 2049

real    66m41.579s
user    66m21.487s
sys     0m11.431s
```

6.1.2 Serial Optimized Code

```
$ export OMP_NUM_THREADS=1
$ time ./mandelbox params/params%04d.dat images/f%04d.bmp 2000 2049 3840 21
images/f2000.bmp        58.859931
images/f2001.bmp        58.915108
images/f2002.bmp        58.961249
images/f2003.bmp        59.009561
images/f2004.bmp        59.052111
images/f2005.bmp        59.099790
images/f2006.bmp        59.152422
images/f2007.bmp        59.228614
images/f2008.bmp        59.308087
images/f2009.bmp        59.391639
images/f2010.bmp        59.474239
images/f2011.bmp        59.557885
images/f2012.bmp        59.642323
images/f2013.bmp        59.730457
images/f2014.bmp        59.827304
images/f2015.bmp        59.929745
images/f2016.bmp        60.029850
images/f2017.bmp        60.127764
images/f2018.bmp        60.229152
images/f2019.bmp        60.327453
images/f2020.bmp        60.418212
images/f2021.bmp        60.509756
images/f2022.bmp        60.604855
images/f2023.bmp        60.691216
images/f2024.bmp        60.776975
images/f2025.bmp        60.873055
images/f2026.bmp        60.984522
images/f2027.bmp        61.097920
images/f2028.bmp        61.211037
images/f2029.bmp        61.338695
images/f2030.bmp        61.474028
images/f2031.bmp        61.608057
```

```
images/f2032.bmp        61.744315
images/f2033.bmp        61.874070
images/f2034.bmp        62.005184
images/f2035.bmp        62.132838
images/f2036.bmp        62.255072
images/f2037.bmp        62.366804
images/f2038.bmp        62.476404
images/f2039.bmp        62.583244
images/f2040.bmp        62.677205
images/f2041.bmp        62.776510
images/f2042.bmp        62.863018
images/f2043.bmp        62.951509
images/f2044.bmp        63.042364
images/f2045.bmp        63.139561
images/f2046.bmp        63.236667
images/f2047.bmp        63.335232
images/f2048.bmp        63.428827
images/f2049.bmp        63.518970

real    50m49.890s
user    50m48.954s
sys     0m0.904s
```

6.1.3 Parallel Optimized Code

```
$ export OMP_NUM_THREADS=32
$ time ./mandelbox params/params%04d.dat images/f%04d.bmp 2000 2049 3840 21
images/f2000.bmp        86.382063
images/f2001.bmp        85.612220
images/f2002.bmp        85.155386
images/f2003.bmp        85.313662
images/f2004.bmp        85.825485
images/f2005.bmp        86.243552
images/f2006.bmp        85.786434
images/f2007.bmp        86.089892
images/f2008.bmp        86.497078
images/f2009.bmp        86.315106
images/f2010.bmp        86.392990
images/f2011.bmp        86.750021
images/f2012.bmp        86.535518
images/f2013.bmp        86.577063
images/f2014.bmp        87.537914
images/f2015.bmp        87.639876
images/f2016.bmp        87.541581
images/f2017.bmp        87.863582
images/f2018.bmp        87.809541
images/f2019.bmp        88.223599
images/f2020.bmp        87.936544
```

```
images/f2021.bmp        87.937748
images/f2022.bmp        88.093698
images/f2023.bmp        88.627627
images/f2024.bmp        88.283351
images/f2025.bmp        88.377168
images/f2026.bmp        88.408821
images/f2027.bmp        88.581168
images/f2028.bmp        88.318609
images/f2029.bmp        88.440072
images/f2030.bmp        89.043256
images/f2031.bmp        89.197899
images/f2032.bmp        89.595845
images/f2033.bmp        89.493680
images/f2034.bmp        89.896815
images/f2035.bmp        89.794378
images/f2036.bmp        90.077492
images/f2037.bmp        90.006695
images/f2038.bmp        90.315489
images/f2039.bmp        90.433613
images/f2040.bmp        90.733051
images/f2041.bmp        90.686932
images/f2042.bmp        91.004530
images/f2043.bmp        90.838531
images/f2044.bmp        90.995758
images/f2045.bmp        91.198265
images/f2046.bmp        91.145057
images/f2047.bmp        91.433315
images/f2048.bmp        91.591040
images/f2049.bmp        91.111057

real    2m22.450s
user    73m42.614s
sys     0m1.088s
```