Fractal Rendering as Implemented within Aleph One*

17th February 2005

Jason M^{c}Guiness

coder@hussar.demon.co.uk

Abstract

The rendering of fractals, such as the Mandelbrot set is an intractable problem in the time-domain. It is effectively NP in time complexity, if the classical technique of scanning each line to render the set is used. To increase the performance of this technique, a multi-threaded approach has been taken. A work-stealing algorithm has been applied to dynamically spread the work-load between each host processor, ensuring that a local minimum in time for rendering the set is discovered. This is a dynamic-programming approach, which is thus optimal for this problem domain. Therefore the reduction in total time to render the fractal is reduced in proportion to the number of host processors, plus a small additional time which is that taken for the work-stealing algorithm to operate. This additional time is not only small, but bounded by a maximum constant.

1 The Mandelbrot-Set Application.

In this section the reader will be reminded of the salient details of the Mandelbrot set and an informal algorithm will be given for generating the set. How this algorithm may be multi-threaded is presented. Alternative algorithms are also presented, but were not implemented. Finally the reader will be stepped through an example of the application running and the operation of the work-stealing algorithm.

*Based upon an early draft paper written in collaboration with the Universities of Delaware and Hertfordshire. The algorithms, programs, etc described within the paper were developed independently of each named organisation and prior to the author's involvement with those organisations.

1.1 An Introduction to the Mandelbrot Set.

The Mandelbrot set [9, 2] is so named after Professor B.B. Mandelbrot, who discovered the set in the 1960s. The Mandelbrot set is intimately related to the Julia set[1] [7], discovered in the 1910s. They are both mathematical entities called *fractals* relating to the fact that they do have a non-integer dimension, the details of this interesting fact are beyond the scope of this paper. Fractals are part of the branch of mathematical called *Chaos Theory*, which may be defined as the term for those theories relating to pseudo-random mappings and functions. The applications of Chaos Theory is widely varied and includes such applications as compression [11], cryptography [4], economics [10], seismology [12], the shape of naturally occurring objects [2] such as clouds, trees [1] and landscapes, medicine such as the modelling of fibrillation in the human heart [6], which is apart from the pure mathematical or aesthetic nature of the objects.

Both the Mandelbrot and Julia sets may be created by iteration of a very simple equation:

$$z_{n+1} = z_n^2 + c \tag{1}$$

In this equation, z_n is a complex number, where $z_0 = 0$. c is also a complex number, which is initialised to a value constant throughout the iterations. The iteration of equation 1 terminates when:

1. Either n reaches the so-called "maximum iteration" value, m, a fixed constant, greater than zero.
2. Or $\mid z_n \mid$ exceeds the so-called "bailout" value, a fixed constant, usually set to the real value 4, for efficiency reasons.

To generate the Mandelbrot set[2], algorithm 1 is used.

[1] Each point in the Mandelbrot set is an "index" into the Julia set for that point.

[2] Mathematically, the Mandelbrot set may be specified thus:
Define c, z_n to be complex numbers, n to be a positive integer, m to be a natural number and M to be the set of complex points that comprise the Mandelbrot set for which the following statement is true:
$\bigvee c \in M : -z_0 = 0, \bigvee n \in 0 \ldots m, z_{n+1} = z_n^2 + c, \mid z \mid < 2 \wedge n < m$

Algorithm 1 The classic algorithm used to generate the Mandelbrot set.

1. Set the value of m, the maximum iterations, greater than zero.
2. Select a point from the complex plane, and set c to that value.
3. Initialise $n = 0$, $z_0 = 0$.
4. Execute equation 1.
5. Increment n.
6. If $\mid z_n \mid \geq 2$ then that c is not in the set of points which comprise the Mandelbrot set. Go to 2.
7. If $n > m$ then that c is in the Mandelbrot set, i.e. $c \subset M$. Go to 2.
8. Go to 4.

Usually the selection of c is not random, but a "raster-scan" of the complex plane. Whole of the complex plane is not required to be scanned, as a property of the Mandelbrot set is that it is entirely contained within the circle of radius 2, centred on the origin of the complex plane. Another important property of conversion to floating-point arithmetic is that the distance between the successively selected points c is a finite number representable by a floating-point number, and obviously non-zero. In other terms, this distance is the resolution at which the set is created.

Usually the set of points M is displayed as an image, with those points in the set coloured to contrast with those that are not in the set. This gives the classic image in 1. The black region in 1 is a basin of stability of the algorithm 1. Those points of which it comprises remain within a finite distance of the origin, i.e. $\mid z_n \mid < \infty$. Those outside this region are unstable, and eventually $\mid z_n \mid \rightarrow \infty$.

More commonly, the points c have a value assigned to them that is derived from n, the iteration at which algorithm 1 terminated for that point. This gives a false-colour image 2, in which the points c of similar colour are termed "level-sets". These level-sets are not the Mandelbrot set, they are basins of stability identified by the algorithm that enclose the Mandelbrot set. The Mandelbrot set has many unusual properties and there are many programs for exploring these properties, but they are not relevant to this paper, the interested reader is directed to [9, 13, 8].

1.2 Threading and the Mandelbrot Set.

An important property of algorithm 1 to generate the Mandelbrot set is that the classification of each c in the com-

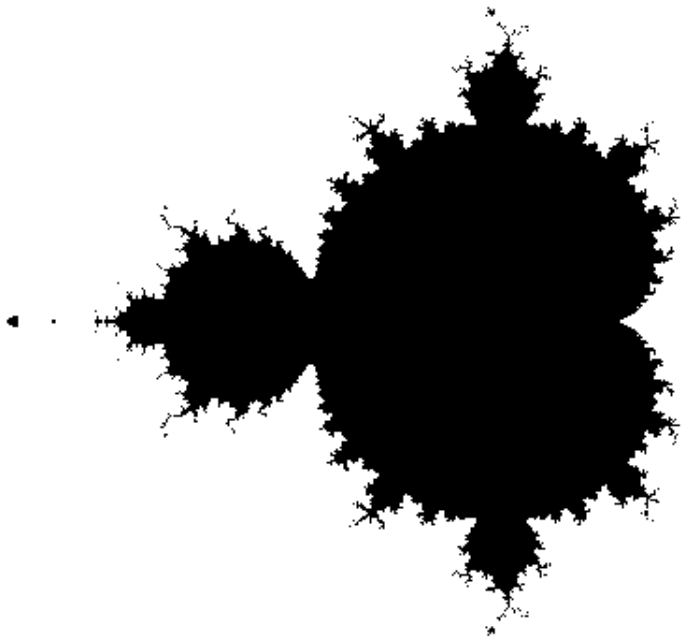

Figure 1: The classic Mandelbrot set image generated by "Fractint" [13]. Points coloured black are in M.

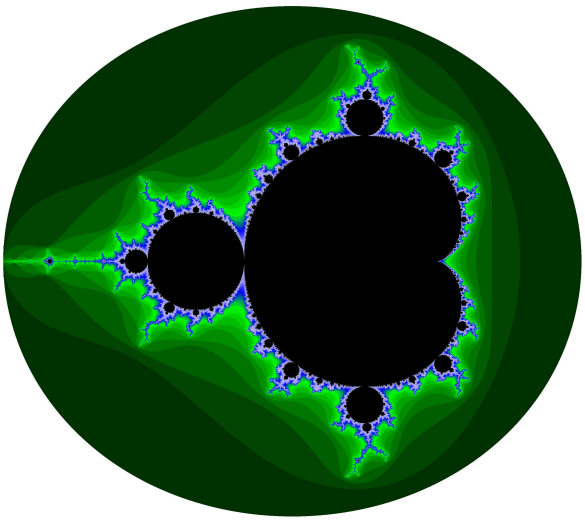

Figure 2: A false-colour image of the Mandelbrot set generated by "Aleph One" [8].

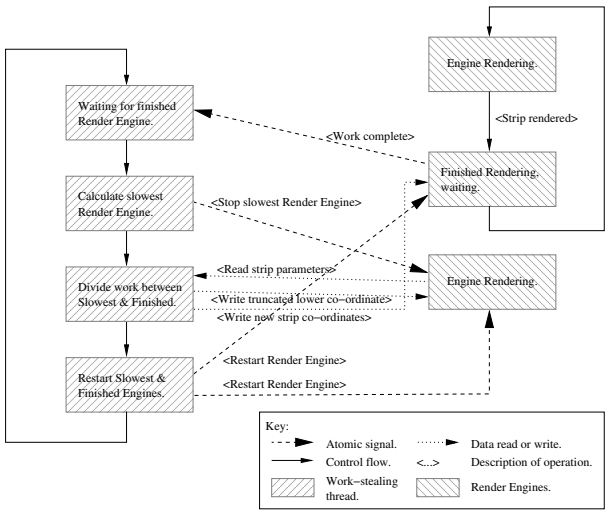

Figure 3: The simplified data & control flows and state transitions of the work-stealing algorithm and render thread algorithms.

plex plane is independent of the classification of any other c. Thus the Mandelbrot set may be implemented as a massively parallel application, thus potentially suited to cellular architectures. Studies of alternative implementations for different architectures, such as fine-grain threaded-architectures [5] and NUMA architectures [3] have already been done.

One might choose to implement the algorithm per thread unit within the machine. This approach would work well for massive clusters of computing nodes. (Remember that for an image of 100×100 points, c, 10,000 thread units would be required with this technique.) Moreover, the classification of any randomly selected c may take between 1 and m iterations of the algorithm. In general it is not possible to know in advance how long such a c will take to classify. Therefore the computation time would take approximately m times the time per iteration loop in algorithm 1.

This implementation, taken from the implementation used in [8], the complex plane is divided into a series of horizontal strips. These strips may be calculated or *rendered* independently of each other, using separate *render threads*, as the classification of the points c within each strip is independent of such classification on other render threads. A diagram of the simplified data & control flows and state transitions is given in 3. Therefore each render thread implements a slightly modified version of algorithm 1, which is given in algorithm 2. Only the co-ordinates for the bounding rectangle are inter-related between the render threads. However, each strip will, in general, take a different amount of time to render, thus the render threads will complete their assigned portion of

Algorithm 2 The render-thread algorithm.

1. Set the value of m, the maximum iterations, greater than zero. Set the estimated completion-time, t, to ∞.
2. Set $c = x$, where x is the top-left of the strip to be rendered.
3. Initialise $n = 0$, $z_0 = 0$.
   (a) Execute equation 1.
   (b) Increment n.
   (c) If $\mid z_n \mid \geq 2$ then that c is not in the set of points which comprise the Mandelbrot set. Go to 4.
   (d) If $n > m$ then that c is in the Mandelbrot set, i.e. $c \subset M$. Go to 4.
   (e) Go to 3a.
4. Increment the real part of c. If the real part of c is less than the width of the strip to be rendered, go to 3.
5. Calculate the average of t and the time it took to render that line.
6. Set the real part of c to the left-hand of the strip. Increment the complex part of c. If the complex part of c is less than the height of the strip, go to 3.
7. Signal work completed, set $t = 0$ (thus this thread is guaranteed not to be selected by the work-stealing algorithm 3).
8. Suspend.

Algorithm 3 The work-stealing algorithm.

1. Monitor render threads for a work-completed signal. That thread that completes we shall denote as T_c.
2. Find that render thread with the longest estimated completion-time, t, note that each render thread updates this time upon completion of a line. Call this thread T_l.
3. Stop T_l when it completes the current line it is rendering.
4. Split the remaining work to be done in the strip equally between the two render threads T_c and T_l.
5. Restart the render threads T_c and T_l.
6. Go to 1.

work at different times. This lead to the addition of a load-balancing algorithm to move uncompleted work to threads that have completed their assigned work. Thus a work-stealing algorithm 3 was added to perform the load-balancing between the render threads. As far as the author is aware, the application of work-stealing to the Mandelbrot set as described above is original, originating from work done in [8]. Alternative implementations of the Mandelbrot set using a work-stealing algorithm [3] or fine-grain threaded algorithm [5] exist.

The updates to the start, x, and finish points of the strips for the render threads T_c and T_l are performed atomically - the threads are suspended whilst these updates are done, either because T_c is stopped or because T_l is stopped by using a mutex. (A mutex is required as the data to be updated is a two complex numbers, x and the finish point, these must both be updated as a pair, atomically.) It is clear that this is a dynamic-programming solution to the load-balancing problem of work distribution between the render threads. Due to the selection of the slowest render thread, this algorithm may been see to be optimal. Moreover, it may be seen that the algorithm is robust: if the estimated completion-time, t, has an error, which it is very likely to have, the algorithm merely performs excessive work-stealing operations, but automatically tunes to find a local minimum in the total completion time curve. Indeed experiments with [8] have shown that the algorithm can accommodate errors of over 100% in the estimated completion-times, and rapidly corrects to the new local minimum.

1.3 A Discussion of the Work-Stealing Algorithm 3.

The algorithm 3 has some important features:

- The bandwidth of the single thread that implements that algorithm is the limiting factor in it's ability to scale. Conversely this algorithm is able to tolerate failures in render threads. If a render thread stops responding, eventually it will be the slowest, unfinished render thread, and it's work will be stolen. It is possible to scale this work-stealing algorithm, if one observes that the work-stealing algorithm operates upon a slice of the complex plane. This clue demonstrates that the work-stealing algorithm is recursive. It is possible to assign strips $s_{0\ldots j}$ of the plane to independent sets of render threads, governed by their own work-stealing thread. These s_i strips are in turn monitored by a work-stealing thread in turn, those strips returning an aggregate estimated completion time. But this has an ultimate limit: Once the number of render threads becomes of the order of the vertical resolution of the image, the completion time is bounded by the maximum time it takes a render thread to generate a single line. This line for the Mandelbrot set in images 1 and 2 is the line $(-2, 0)$ to $(2, 0)$, which has the most points within the set. These points take m time to classify. As the unit of work in the work-stealing algorithm is a line, this is the slowest line, and thus the ultimate limit of this algorithm, unless the resolution is increased. This discussion leads to the following algorithm:
- If robustness is not required, then the image generated may be viewed as an array values. Each of these values is the classification of c. Thus if one has $p_{0\ldots q}$ threads, each p_n thread initially classifies a point in the array offset by n, and once completed, moves along the array using a stride of q. This allows the use of a number of threads that is bounded by the number of points within the image. For more thread units, the image resolution would need to be increased. Unfortunately this algorithm does not have a natural ability to tolerate failures in thread units, unlike the work-stealing algorithm, 3.

1.4 Execution details of the Mandelbrot-set application.

A description, with images taken during program execution, will be given regarding the operation of the render threads and work-stealing algorithm. In this example there are three threads for simplicity:

1. On program start, the render threads start to execute and perform their assigned work. The assigned work is initially equal $\frac{1}{3}$ portions of the total image, arranged in horizontal strips. The top render thread is denoted by T_0, the middle by T_1 and the lower by T_2, although this relative position will change later.

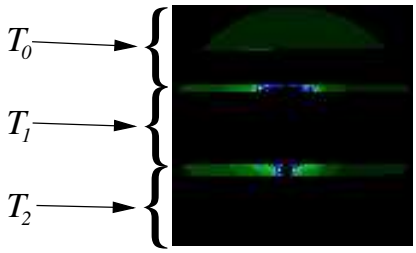

Figure 4: The image generated shortly after program start-up.

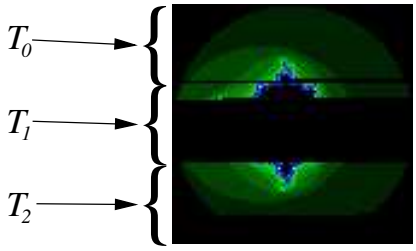

Figure 5: Image generation has progressed, shortly before a work-stealing event.

The operation of the render threads may be seen in figure 4. No work-stealing has occurred, so there are just three strips, one per render thread, scanning from left to right, top to bottom.

2. As the image generation proceeds, the T_0 and T_2 threads progress faster than T_1, as seen in figure 5. Note how T_0 has calculated more than T_2 - the lighter areas take longer to calculate, and the strip generated by T_0 is black at the top, and white at the bottom, but the converse is true for T_2.

3. The first work-stealing operation has just occurred. T_0 finished and T_1, the slowest (mainly white) has had the remaining work divided between it and T_0, see figure 6. Note how the end-point of T_1 was assigned to be the new end point of T_0 and the new end-point of T_1 is the start-point of T_0.

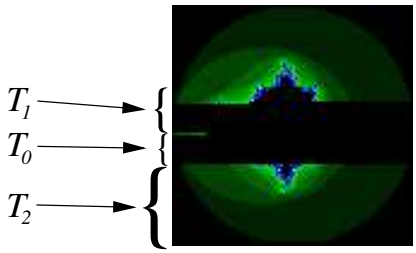

Figure 6: Just after the first work-stealing operation.

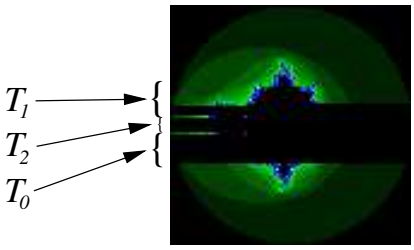

Figure 7: The second work-stealing operation.

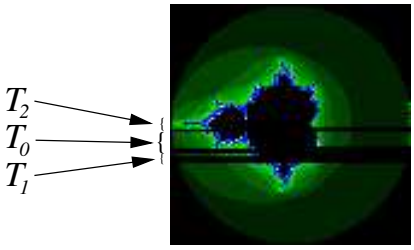

Figure 8: The third work-stealing operation.

4. Shortly after this first work-stealing operation occurs, T_2 completes it's assigned work. A second work-stealing operation occurs, see figure 7. In this case work was again stolen from T_1, and assigned to T_2.

5. After a pause T_1 completes it's assigned work, and another work-stealing operation occurs, this time with T_0, which may be seen in figure 8.

6. Finally the set is completed, see figure 9 , with no further work-stealing operations, as the number of uncompleted lines for any render thread is less than 2, and a line is the minimum unit of work for this algorithm.

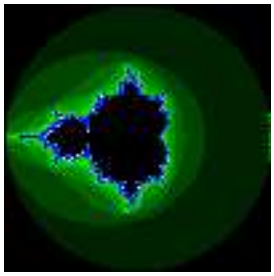

Figure 9: The completed Mandelbrot set.

2 Conclusion/Discussion.

The work-stealing algorithm is an optimal and scalable approach to the problem of multi-thread rendering of the Mandelbrot set. (Indeed this solution can be (and has been) applied to any other fractal rendering problem.)

The current design works well with "heavy-weight" NT or pthread style threads, but has also been run on alternative "light-weight" threaded archtectures, such as the CyclopsE chip.

For massive scalability the nesting of render computers, each controlled by a work-stealing algorithm into a render farm in which each computer within the farm is controlled by an overall work-stealing thread would need to be implemented and tested. Moreover the size of a render farm would need to be evaluated with respect to the power of the machine performing the work-stealing amongst render computers.

References

[1] Aono, M., Kunil, T.L., "Botanical Tree Image Generation.", IEEE Computer Graphics and Applications 4,5 (1984) 10-33.

[2] Barnsley, M. F., Devaney, R.L., Mandelbrot, B.B., Peitgen, H.-O., Saupe, D., Voss, R.F., "The Science of Fractal Images.", Springer-Verlag, 1988.

[3] Cavalherio, G. G. H., Doreille, M., Galilée, F., Gautier, T., Roch, J-L., "Scheduling Parallel Programs on Non-Uniform Memory Architectures.", HPCA Conference – Workshop on Parallel Computing for Irregular Applications WPCIA1, Orlando, USA, January 1999.

[4] Dachselt, F., Kelber, K. and Schwarz, W., "Chaotic Coding and Cryptanalysis.", Proceedings of 1997 IEEE International Symposium on Circuits and Systems Circuits and Systems in the Information Age (New York, USA), vol. 4, May 1998, pp. 518-21.

[5] Engler, D. R., Andrews, G. R., Lowenthal, D, K., "Filaments: Efficient Support for Fine-Grain Parallelism.",TR 93-13a, Dept. of Computer Science, University of Arizona, Tucson, 1993.

[6] Garfinkel, A., Chen, P. S., Walter, D. O., Karagueuzian, H. S., Kogan, B., Evans, S. J., Karpoukhin, M., Hwang, C., Uchida, T., Gotoh, M., Nwasokwa, O., Sager, P. and Weiss, J. N., "Quasiperiodicity and chaos in cardiac fibrillation.", Journal of Clinical Investigation. 99(2), pp. 305-14, January 1997.

[7] Julia, G., "Sur l'iteration des Functions Rationelles.", Journal de Math. Pure et Appl. 8 (1918) 47-245.

[8] M^{c}Guiness, J. M., "Aleph One", `http://aleph1.sourceforge.net`.

[9] Mandelbrot, B.B., "The Fractal Geometry of Nature.", W.H.Freeman & Co., Sept., 1982.

[10] Peters, E. E., "Fractal Market Analysis : Applying Chaos Theory to Investment and Economics.", John Wiley & Sons, 1994.

[11] Reghbati, H.K., "An Overview of Data Compression Techniques.", Computer 14,4 (1981) 71-76.

[12] Turcotte, D. L., "Fractals and Chaos in Geology and Geophysics.", Cambridge University Press, 1992, pp. 3550.

[13] Wegner, T., Osuch, J., Martin, G., Bussell, B., "Fractint", `http://www.fractint.org`