Distributed Fractal Generation Across a Piconet

Daniel C Doolan, Sabin Tabirca*
University College Cork,
Ireland.

Abstract

This paper explores the distributed generation of fractal images on mobile phones using Bluetooth (JSR-82) as the inter-device communications mechanism. Mobile phones inherently have limited resources (such as memory and processing power). The aim of this paper is to demonstrate a technique, where the resources of several mobile devices can be used in conjunction. This will allow for the computation of a processor intensive task in a short period of time.

CR Categories: I3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Geometric algorithms, languages and systems.; D1.7 [Programming Techniques]: Visual Programming; C2.4 [Computer Communication Networks]: Distributed Systems—Client/Server.

Keywords: Fractal Image Generation, Distributed Computing, Bluetooth, Piconet

1 Introduction

Fractals have been a classical topic ever since what is considered to be the discovery of the first fractal in 1872 by Georg Cantor, (typically called "Cantor Dust"). The present interest in Fractal geometry is mainly due to the work of Benoit Mandelbrot, who in the mid 1970's was successful in the creation of the an index (Mandelbrot Set) for all the possible Julia Sets. Since then their have been many further developments such as Fractal Terrain Generation [Meyer 1982] [Lucas and Marquand 1983], Fractal Image Compression [Lu 1997], Fractal Encryption and Fractal Music. However, the process of generating such images requires significant computational resources [Doolan and Tabirca 2005]. In this paper we present a study parallelizing the generation of Mandelbrot images on mobile phones using Bluetooth as the communication mechanism.

Over the past couple of years both Sony-Ericsson and Nokia Developer websites have published invaluable training material on the development of J2ME applications with Bluetooth technology.

Some interesting literature has been published of late, dealing with Bluetooth technology. One thesis [Long 2004] "A Study of Java Games in Bluetooth Wireless Networks" gives an overview of the Bluetooth API and shows how such technology can be employed in the development of wireless networked games. The thesis gives a simple example of a game designed in a Point to Point Piconet Configuration (Figure 1). It also includes some useful references to various sources dealing with both game programming and networked communications.

*e-mail: {d.doolan, tabirca}@cs.ucc.ie

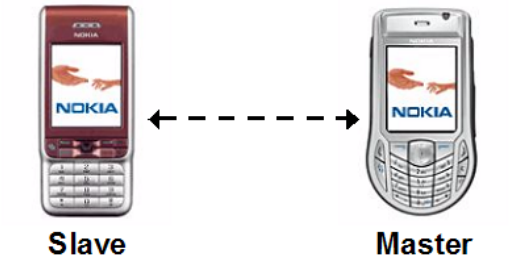

Figure 1: Simple Point to Point Piconet Configuration.

A far more in-depth look at Bluetooth technology with J2ME may be found in the MSc thesis of [Klingsheim 2004]. The thesis centered around the development of two Bluetooth applications and the testing of same on several devices, namely Nokia 6600 to Nokia 6600, Sony Ericsson P900 and PC to Nokia 6600. The applications dealt with device discovery and bench-marking data transfer.

The generation of the Mandelbrot Set is often referred to as an "embarrassingly parallel computation". It can be easily divided into a number of completely independent parts, each of which can be executed by a separate processor. The parallel computation of the Mandelbrot Set usually relies on the C based Message Passing Interface (MPI) library [MPI][Book][O'Mahony 2004].

More and more mobile devices are shipping with Bluetooth technology as standard. Typically all Smartphones fall within this remit, examples include the Nokia Series 60 2^{nd} and 3^{rd} Edition phones. Sony-Ericsson so too have a significant number of Bluetooth enabled devices (K750, D750, W800, K608). The Java capabilities of these devices are that of levels 5 and 6 of Sony-Ericssons Java Platforms [Sony-Ericsson 2005].

2 Fractal Generation

The generation algorithms for the Mandelbrot Set are quite simple, but as the generating function is iterated repeatedly this gives rise to complexity and results in a highly complex fractal image. Fractals are usually obtained when the generating function $f(z)$ is non linear. For $f(z) = z^2 + c$ the classical Mandelbrot Set is obtained. Mandelbrot-like sets are also obtained when the generating function has the form of $f(z) = Z^u + C^v$.

The area of the Fractal image that is displayed is dependent on the xmin,ymin, xmax,ymax values. Figure 2 shows a typical Mandelbrot Image and the corresponding coordinates for it.

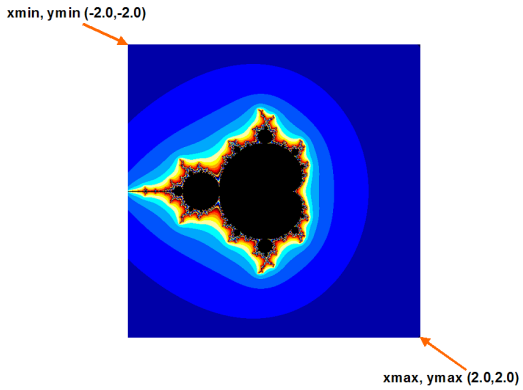

Figure 2: Typical Mandelbrot Image for xmin,ymin = -2.0, xmax,ymax = 2.0.

A Code sample (Listing 1) of the Mandelbrot Set generation function shows a direct relationship with the Mandelbrot Set Algorithm. One can clearly see that if the absolute value of the complex number lies outside the threshold R, that the pixel at the current coordinates of the iteration through the image will be drawn in a specific colour.

```
1  for(int i=0;i<SIZEX;i++)for(int j=0;j<SIZEY;j++){
2      Complex c = new Complex((XMIN+i*STEPX),(YMIN+j*STEPY));
3      Complex z = new Complex();
4      for (k=0;k<NR_ITER;k++){
5          z=f(z,c);
6          if(z.getAbs()>R){
7              r = c[k%l][0]; g = c[k%l][1]; b = c[k%l][2];
8              color = b + (g<<8) + (r<<16) + alpha;
9              pixels[(j*SIZEX) + i] = color;
10             break;
11         }
12     }
13 }
```

Listing 1: Code listing of Mandelbrot Set Function

The execution times for generating a fractal image on a mobile phone are quite long (Table 1), especially as the number of iterations required increases. Execution on a Nokia 6630 Phone requires 55,657ms to generate a 200 x 200 pixel image using 500 iterations, running at 1000 iterations the processing time increases to 98,250ms (almost double). Similar tests using Sun's WTK emulator yields results far in excess of the execution times for the Nokia 6630 Phone. The Sony-Ericsson Emulator demonstrated the fastest computation.

Device	500	750	1000
SE WTK 2.2	30,079 ms	42,516 ms	56,141 ms
Nokia 6630	55,657 ms	75,266 ms	98,250 ms
Sun WTK 2.2	140,875 ms	205,078 ms	269,734 ms

Table 1: Image Generation Times for the Mandelbrot Set (Image Size 200x200 pixels, xmin,ymin -2.0, xmax,ymax 2.0) at varying number of iterations.

3 Distributed Fractal Generation

The application was developed using Sun Java Studio Mobility 6 2004Q3 [Sun-Microsystems a] and the J2ME Wireless Toolkit (WTK) Emulators [Sun-Microsystems b].

The application has been designed to work with several fractal formulas and not just fractals of the form $f(z) = Z^u + C^v$. The other implemented forms include $f(z) = Z^u - C^v$ and $f(z) = Z^u + C^v + Z$. A more comprehensive Math's Class would allow for a wider diversity of possible fractal formulas. The MathFP class is a efficient integer based alternative to this.

The design of the system is that of a Point to Multi-Point configuration (Figure 3). The Point to Multi-Point configuration is significantly more complex than the simpler Point to Point configuration. For the Point to Multi-Point Configuration to work the Master Device must keep a list of all clients connected to it.

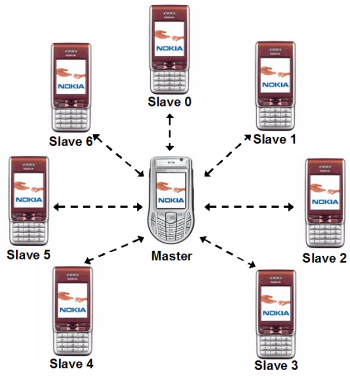

Figure 3: Point to Multi-Point Piconet Configuration.

3.1 User Interface

One of the most important parts of the Server Program is the User Interface for modifying the Fractal Image settings. The most important options include the Image Size, Number of Iterations, Fractal Co-ordinates and Fractal Equation Type (Figure 4). The Client Application has a simple interface that displays whether an image is currently being generated or not, the processing time for the generation of an image and the Image Settings the Client is currently using to generate the image (Figure 4).

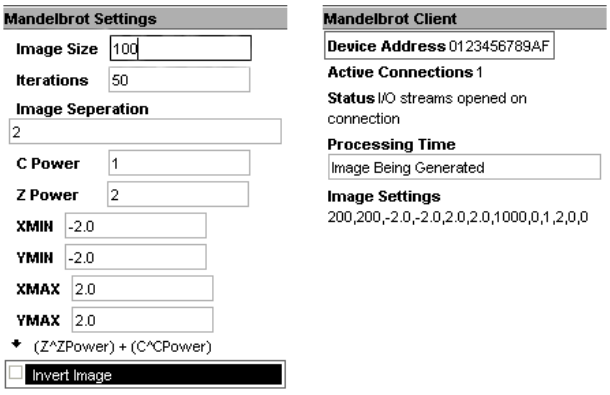

Figure 4: Fractal Image Settings Form, (Server). Client Info Form.

3.2 Work Load Balancing

There are several ways in which the image can be divided among the client's nodes for processing. These include Uniform Block, Cyclic and Dynamic Load Balancing.

3.2.1 Uniform Block Balancing

This method of load balancing was the first scheme to be implemented in the application. In this implementation the image is divided into *nrClients* vertical (or horizontal) strips of equal size (Figure 5). The client i receives for computations the image chunk with the consecutive columns $i \times w/nrClients, i \times w/nrClients + 1, ..., (i+1) \times w/nrClients - 1$. This division of the image into similar vertical strips would yield to similar computation on each client. It is a simple scheme where a single message is sent to each client which results in a corresponding message being returned to the master with the results of the computation. A similar method [Murty 2004] where the image is divided into a grid was used for the generation of Fractal Images using .NET Web Services.

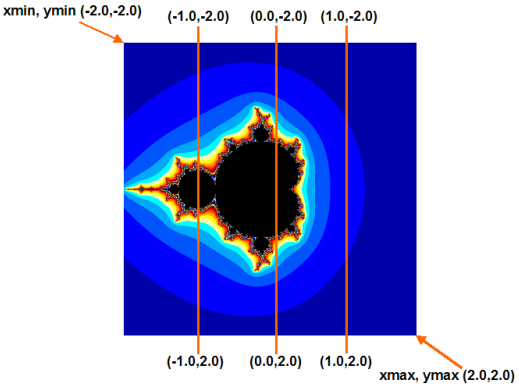

Figure 5: Division of Image into Vertical Strips.

For example dividing the image into four vertical strips yields a matrix of coordinates that may be sent to the clients connected to the Master. In the case of 4 vertical strips with xmin, ymin starting at -2.0 and xmax, ymax starting at 2.0, the following matrix is generated (Table 2). In the case that 4 clients were used using a grid of sub-squares the resulting matrix would center around the value 0.0 for a Mandelbrot Image where xmin, ymin = -2.0 and xmax, ymax = 2.0.

xmin	ymin	xmax	ymax
-2.0	-2.0	-1.0	2.0
-1.0	-2.0	0.0	2.0
0.0	-2.0	1.0	2.0
1.0	-2.0	2.0	2.0

Table 2: Co-ordinate Matrix for 4 Vertical Strips: xmin,ymin = -2.0, xmax,ymax = 2.0

Several other elements are essential for the distributed processing of the image. This size of the image each client is to generate. An identifier for which section of the image was generated by each client. This is necessary so the image can be reassembled on the Server in the correct order. The order in which images are returned to the Server is random as some sections will require greater processing time than others. The other parameters include whether the image should be inverted or not, the powers for Z and C, the number of iterations to be executed and finally a flag for the type of fractal formula to use.

3.2.2 Cyclic Load Balancing

This is a follow on approach from the Uniform Block Load Balancing example. The image is still divided into equal sized vertical strips $\{S_0, S_1, ..., S_{p-1}\}$ each of them containing only a few columns. The partition of this p strips onto the clients is performed into a cyclic matter so that the client i would receive the strips $\{S_{i+j \times nrClients} : j = 0, 1, ..., \frac{p}{nrClients} - 1\}$. Figure 6 shows an example of this division. In this example each client will receive four small sections of the image. The image sections a single client will receive are evenly distributed through out the image to be generated (e.g. Figure 6 clearly shows that the first client will receive grid sections 1,5, 9, 13). In this case the computation of the regions with significant details is evenly distributed amongst processors.

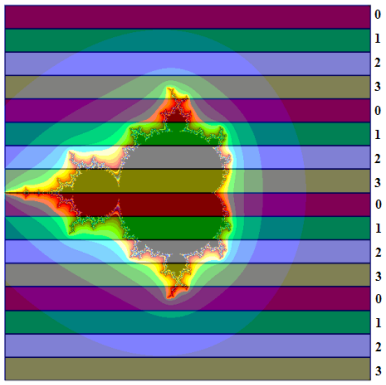

Figure 6: Cyclic Division of Image Area.

3.2.3 Dynamic Load Balancing

The Server maintains a Work Pool of jobs that can be sent out to clients (Figure 7). Initially every connected Client will receive a work unit to process, as soon as a Client returns a result another work unit is issued to the Client. This process is carried out until all the jobs in the Servers Work Pool have been completed. For this procedure the image area is typically divided into a grid structure. There is of course a slight increase in communication costs compared with that of the Uniform Block method, but the distribution of processing should be far more even across all connected Clients.

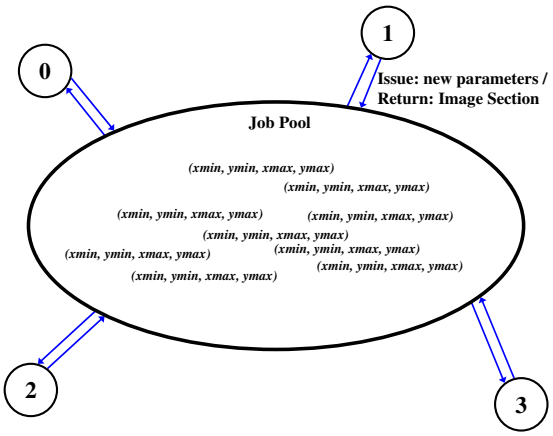

Figure 7: Dynamic Division of Image Area.

The division of the image into a grid (Figure 8) is carried out by the user choosing a particular granularity g which is the width of each grid block. Figure 8 is a 200 x 200 pixel image divided into 16 areas (Granularity of 50 pixels). The granularity by which the image is divided greatly effects the number of messages that are sent between Server and Clients (Figure 9). The area (in pixels) of each grid block is given by g^2 so that the number of grid blocks is given by $\frac{w \times h}{g^2}$. The total number of exchanges between Server and all the Clients has the form $2\left(\frac{w \times h}{g^2}\right)$ as a request must be sent to the client and a result returned. The total number of exchanges between the Server and any connected Client is $2\left(\frac{w \times h}{g^2 \times nrClients}\right)$. It is clear that the granularity has an important effect on both computation and communication. For computation the finer the granularity is the better the computation load balance becomes amongst processors. With a fine granularity the grid blocks have a small area so that the clients compute them in short time. In the case that one client becomes idle (work pool is empty) this will wait only little time for the other clients to finish the computation. Moreover, a fine granularity will evenly distribute the regions of high details amongst all the processors. However, fine granularity increases the amount of communication between the Server and the Clients. This would imply more overheads to start up the communications as well as more information that is sent between the Server and the Clients.

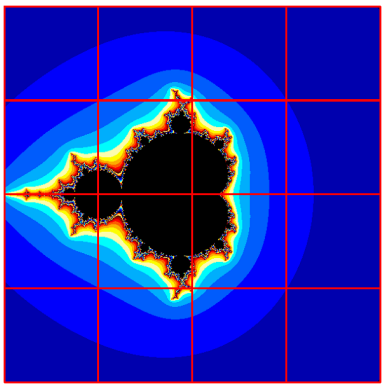

Figure 8: Dynamic Division of Image Area.

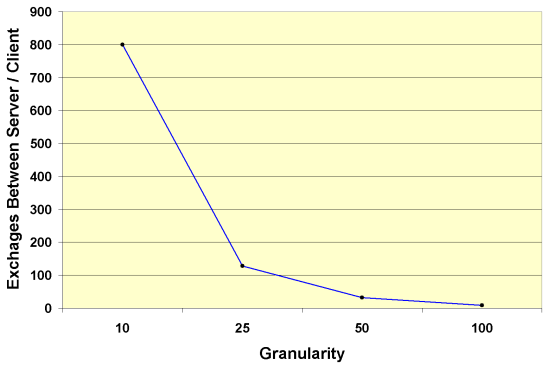

Figure 9: Number of Exchanges Required as Granularity Varies

3.3 Client / Server Operations

Overall the general methodology of the system is quite simple (Figure 10). The initial stages of the process are carried out on the Server. Firstly it is necessary to acquire the Input Settings for the Fractal Image, a Graphical User Interface (GUI) (Figure 4) is provided for this. When the user issues a request to generate a Fractal Image the parameters are gathered from the Fractal Image Settings GUI. The next stage is to calculate the parameters necessary for each client (this will depend on the number of clients currently connected). This yields a unique set of parameters for each client. Several other parameters are also passed which are the same for all clients (for example: formula type, number of iterations).

Once all the parameters have been finalised the operation of sending the Image Parameters to each connected client can commence. The parameter data is passed in the form of a string. A typical example of this string has the format of "width, height, xmin, ymin, xmax, ymax, iterations, equation type, cPower, zPower, invert, image segment number ". An example of the output string would be: "50, 200, -1.0, -2.0, 0.0, 2.0, 500, 0, 1, 2, 0, 1". The previous string would generate an image 50 x 200 pixels in size. The complex plane coordinates are "-1.0, -2.0, 0.0, 2.0". The client would carry out 500 iterations at each point. The generated Image would be the standard non inverted mandelbrot set $Z^2 + C$. The final parameter "image segment number" allows for the correct ordering of segments on the Server side.

The Client has in the meantime has been waiting for requests from the Server. Once a Client receives a request it must first parse the data to extract all of the required parameters necessary to generate the image. The next and most important stage is the actual generation of the fractal image. Each client will generate a small section of the image. The image section is then sent to the Server in the form of a sequence of integers using a DataOutputStream Object.

On the server side once it has issued its requests to all clients, it simply waits for incoming results. When a message is received from a client, the server examines the "image segment number" so the image will be placed in the correct order. Next it finds the length on the remaining incoming data, and initialises an array to be able to read all of the integer values representing the actual image. Once all the integer values have been read an Image object is created and positioned into is proper location based on the "image segment number". The process of waiting for client responses continues until all Image Sections are Retrieved. When the final image section is retrieved the Server displays the completed image on screen to the user.

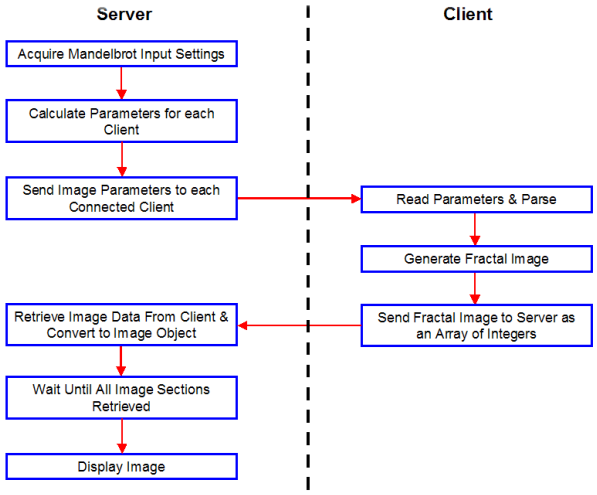

Figure 10: Client / Server Operations.

3.4 Bluetooth Networking

Typically the first step in a Networked Bluetooth application is to discover other Bluetooth Capable devices within the catchment area (10 meters for a class 3 Bluetooth Device, 100 meters for a class 1 device). For a Bluetooth device to advertise itself as being available it must be in "discoverable mode". There are two differing forms of this mode: General Unlimited Inquiry Access Code (GIAC) and Limited Dedicated Inquiry Access Code (LIAC). If the device is to be generally discoverable then it should be set to GIAC mode else it may be set as discoverable in a "limited" manner by using the LIAC mode. In this application the Client Devices are set to be generally discoverable, and as such when the Server Device is started it's first task is to discover all local devices available within it's catchment area.

In many Client to Server Applications the client requests services from the Server. In this case the Server usually starts running first and begins a cycle of waiting to accept new client requests. This application however the operation of the Client / Server system is slightly different. As with most Bluetooth applications the Master adds each client into the piconnet. The Master however issues requests to all client on the piconet (for example generate a fractal image) the client devices then carryout the actual processing work and return results to the Master (Server). This is akin to how SETI@Home [SETIatHome] works where the clients carry out the processing and return the results to the Server Application. An invaluable aid in the development of this system came in the form of a simple application from Nokia that demonstrated the use of a Point-multi-Point configuration [Nokia 2004]. Sony-Ericsson [Sony-Ericsson 2004] also have some very useful Developers Training Material dealing with Bluetooth Applications programming.

3.5 Execution Results

The experimental tests of this distributed fractal generating were carried out using 4 of Nokia 6630 phones. The scheduling partitions used in the experiment were *Uniform Block*, *Cyclic*, *Dynamic Scheduling with* $g = 50$ *and Dynamic Scheduling with* $g = 25$. The Mandelbrot fractal was generated into a 200 pixel square image to cover the plane region $[-2,2] \times [-2,2]$ with 500, 750 and 1000 iterations. Recall that the generation of the fractal on a single Nokia 6630 devices at 500 iterations requires 55,657ms to process, a figure far in excess of the execution times that will be presented below. Even when the Sony-Ericsson Wireless Toolkit (WTK) 2.2. emulator (Sony-Ericsson P900 Mobile Phone) was used the overall results showed a significant improvement in rendering time (see Tables 3 and 1).

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	13,675	4,453	11,456	7,455	2,235
750	14,047	4,609	11,838	7,422	2,557
1,000	14,907	4,437	12,101	7,469	2,610

Table 3: Image Generation Times for the Mandelbrot Set using the Sony-Ericsson WTK Emulator

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	25,878	8,246	20,457	13,379	4,217
750	26,051	8,230	21,064	13,277	4,542
1,000	26,442	8,311	21,418	12,945	4,547

Table 4: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Uniform Block Partition)

Table 4 presents the execution times for the *Uniform Block* scheduling. One can see that the second and third clients will receive the regions with more details so that they generate the highest computation times. While the forth client receives the region with lesser details so that it has the smallest execution time. As consequence those execution times show a huge load imbalance of the computation.

The use of the *Cyclic* scheduling corrects this load imbalance. The primary advantage with this scheme is that all areas of the image are distributed uniformly onto all the connected clients. The result of this is that areas where there is high computation cost are carried out by all processors. This means that the Server no longer has to wait for one or two nodes to finish the computation well after all other nodes have completed their assigned tasks. Table 5 shows the processing times for the cyclic scheduling scheme are very well balanced.

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	18,734	14,038	14,740	13.734	14,251
750	19,042	14,734	15,183	14.361	14,793
1000	19,623	14,829	15,053	14.853	15,391

Table 5: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Cyclic Scheduling Algorithm)

The *Dynamic Load Balancing* method was tested for two granularities. When $g = 50$ the fractal was splitted up into 16 grid blocks so that each client gets 4 blocks to compute. The computation times (see Table 6) show a very good load balance but they are bigger than the those of the *Cyclic* scheduling. In this case each client has at least 3 idle periods waiting for the communication with the Server to complete. Similarly, for granularity $g = 25$ each client has 16 grid blocks to compute so that it has 15 idle periods of waiting for communication. This however increases the computation times of each processor with around 8 seconds as Table Table 7 presents.

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	23,475	20.243,	20,048	21.482	21,729
750	23,739	21,473	21,762	21,906	21,845
1000	23,957	21,883	21,834	22.045	21,983

Table 6: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Dynamic Scheduling Algorithm with granularity $g = 50$ pixels)

Iter	Total Time	Node 0	Node 1	Node 2	Node 3
500	30,754	28.393,	28,851	28.703	28,929
750	31083	30,710	29,675	29,496	30,025
1000	31,672	30,031	29,871	29.582	29,832

Table 7: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones (Dynamic Scheduling Algorithm with granularity $g = 25$ pixels)

Certainly, these experimental tests showed that the distributed computation over a piconet with Bluetooth reduces the generating time. Table 8 shows the reduction of the execution times for 500 iterations when the number of clients is $nrClients = 1, 2, 3, 4$.

	1 Client	2 Clients	3 Clients	4 Clients
Block	55,657	43,864	32,704	25,878
Cyclic	55,493	36,586	23,193	18,734
Dynamic	55,723	38,741	29,704	23,475

Table 8: Image Generation Times for the Mandelbrot Set using a set of Nokia 6630 Phones for $nrClients = 1, 2, 3, 4$.

3.6 Further Work

Many alternative possibilities exist in terms of load balancing, examination and implementation of other scheduling schemes could provide noticeable improvement in the overall processing time. The implementation of alternate schemes would result in higher communication costs.

Another possibility is to make use of Scatternets. From the results so far it is clear that sections of the fractal image that contain a high level of detail require much more processing than areas of far less detail. So for the areas that require significant processing those sections could be distributed to a client that also acts as a server for another piconet.

4 Conclusion

A method of carrying out distributed fractal image generation across a piconet has been developed. This method of distributed fractal image generation has shown that it is capable of increasing the rate at which fractal images can be rendered on a mobile device.

Several methods of load balancing were implemented: Uniform Block, Cyclic Load Balancing and Dynamic Load Balancing. The Cyclic and Dynamic forms produce a far more even distribution of the work among the connected clients. It is clear from Tables 4, 5, 6 that the processing times reduce dramatically compared with the processing times with a single mobile device, for example 55,657 with a single Nokia 6630 (500 Iterations).

The method outlined for distributing the processing of a processor intensive task could be used in many other areas besides fractal image generation. The continually increasing number of mobile devices may prove to be a useful processing resource in the future. In time mobile devices may contribute to projects such as Seti@Home, DNA Analysis, Prime Number Search's to name but a few.

References

BOOK, M. Parallel fractal image generation. `http://www.matthiasbook.de/papers/parallelfractals/`.

DOOLAN, D. C., AND TABIRCA, S. 2005. Interactive teaching tool to visualize fractals on mobile devices. In *Proceedings of Eurographics Ireland Chapter Workshop*, Eurographics Ireland Chapter, 7–12.

KLINGSHEIM, A. N. 2004. *J2ME Bluetooth Programming*. Master's thesis, University of Bergen.

LONG, B. 2004. *A Study of Java Games in Bluetooth Wireless Netowrks*. Master's thesis, University College Cork.

LU, N. 1997. *Fractal Imaging*. Academic Press, San Diego, London, Boston.

LUCAS, G., AND MARQUAND, R., 1983. Star wars, return of the jedi. DVD Release Sept 2004.

MEYER, N., 1982. Star trek: The wrath of khan. DVD Release May 2003.

MPI. Message passing interface. `http://www-unix.mcs.anl.gov/mpi/`.

MURTY, R. 2004. Juliet: A distributed fault tolerant load balancer for .net web services. In *Proceedings of International Conference on Web Services (ICWS'04)*, IEEE.

NOKIA. Nokia platforms. `http://www.forum.nokia.com/main/0,6566,010,00.html`.

NOKIA, 2004. Introduction to developing networked midlets using bluetooth. `http://www.forum.nokia.com/info/sw.nokia.com/id/c0d95e6e-ccb7-4793-b3fc-2e88c9871bf5/Introduction_To_Developing_Networked_MIDlets_Using_Bluetooth_v1_0.zip.html`.

O'MAHONY, C. 2004. *Distributed Multimedia Processing*. Master's thesis, University College Cork.

SETIATHOME. The search for extra terrestial intelligence at home. `http://setiathome.ssl.berkeley.edu/`.

SONY-ERICSSON, 2004. Developing applications with the java api's for bluetooth (jsr-82). `http://developer.sonyericsson.com/getDocument.do?docId=65246`.

SONY-ERICSSON, 2005. Java platforms for sony-ericsson. `http://www.sonyericsson.com/developerimages/javaplatformversionsandscreensizesjune2005.xls`, June.

SUN-MICROSYSTEMS. Sun java studio mobility 6 2004q3. `http://www.sun.com/software/products/jsmobility/`.

SUN-MICROSYSTEMS. Windows wireless toolkit. `http://java.sun.com/products/sjwtoolkit/download-2_2.html`.