International Journal of Computers, Communications & Control
Vol. I (2006), No. 1, pp. 9-15

Visualising Infinity on a Mobile Device

Daniel C. Doolan, Sabin Tabirca

Abstract: This paper discusses how a Java 2 Micro Edition (J2ME) application may be developed to visualise a wide variety of differing fractal types on a mobile phone. A limited number of J2ME applications are available that are capable of generating the Mandelbrot Set. At present there are no J2ME applications capable of generating a multitude of fractal image types.
Keywords: J2ME, Mobile Phone, Fractals

1 Introduction

It has been shown that mobile devices are capable of generating high quality images of infinite detail [2]. The generated images have generally been limited to the Mandelbrot Set. Since the late 19^{th} century fractals have been a favourite topic for mathematicians. Since the dawn of modern day microprocessor based computing the study of fractals has taken a radical leap, as it is within the computing domain that all the nuances of fractal type images can be visualised.

Benoit Mandelbrot made a huge contribution in the late 1970's with the discovery of the Mandelbrot Set (an index for all the possible Julia Sets). The dawn of the 21^{st} century has seen a radical changed in what we consider a computer to be. It has seen the widespread uptake of mobile phones throughout the world. Devices that started life in the latter years of the 20^{th} century as a mobile communications medium have evolved and mutated into mobile computing devices of considerable processing power. No longer are mobile phones used for just telecommunications but for just about any type of application that a standard desktop machine is capable of.

The 2D and 3D visualisation of fractal images is one interesting topic that is coming into the realm of reality within the mobile computing domain. Current high end phones have processing speeds in the region of 100 to 200Mhz [7] [1], typically running ARM9 type processors. The next evolution in processing power will see such devices fitted with the ARM11 processor cores with speeds as high as 500Mhz. Mobile devices clearly have a huge processing potential especially if the combined processing power of the many millions of phones around the world were put to task on a singular problem. Examples of such computation are already in existence for example "Distributed Fractal Generation Across a Piconet" [3] demonstrates that the combined processing power of several mobile devices may be used to distribute the processing load between several devices that are connected together over a Bluetooth network.

1.1 Mobile Phone Market Penetration

The uptake of mobile devices around the world is staggering. In September 2004 the market penetration stood at 89% in Ireland [10], by March 2005 it stood at 94% [11]. This is a huge increase when in 2001 penetration stood at only 67% [8]. Ireland achieved 100% penetration in September 2005, showing an increase of over 11% in a twelve month period. This allows Ireland to join Spain, Finland the Netherlands and Austria in having a 100% penetration rage. Luxembourg is currently on top with a rate of 156% [12]. It is expected that Western Europe will exceed 100% usage by 2007 [15]. The year 2015 should see half the world's population (Four billion people) using mobile phones as a communications medium [9].

1.2 Primary Aims

The primary purpose of this paper is the development of an application capable of running on a single mobile device that has the ability to generate a variety of two dimensional fractal images. One of the chief aims is that the application should be easy to use. This would allow it to be used as a teaching tool. To achieve this each section of the application has a easy to use Graphical User Interface (GUI) to allow the user to specify the parameters for the image generation process. Once the user is happy with the image parameters they can then press a button to begin the image generation process. The resultant output will be a fractal image based on the input parameters.

2 Mandelbrot & Julia Set Generation

The discovery of the Julia Set in 1918 by Gaston Julia described in his paper "Mémoire sur l'itération des fonctions rationnelles" proved to be a most important work at the time. It became almost forgotten until Benoit Mandelbrot brought it back to the forefront with his discovery of the Mandelbrot Set. This ensued a new field of research that became known as fractal geometry. Both the Julia and Mandelbrot Set images can be generated by the repeated iteration of a simple function (see Figure 2). The Mandelbrot Set is an Index into the Julia Set, in other words every possible Julia Set can be represented within the Mandelbrot Set (see Figure 1).

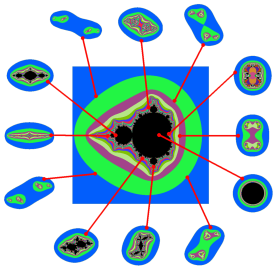

Figure 1: Index of the Julia Set (the Mandelbrot Set)

$$J_c = \left\{ Z_0 \in \mathbb{C} | \lim_{n\to\infty} Z_n \neq \infty \right\} where:$$

$$Z_0 = C, Z_{n+1} = f(Z_n), n \geq 0$$

$$M = \left\{ c \in \mathbb{C} | \lim_{n\to\infty} Z_n \neq \infty \right\} where:$$

$$Z_0 = 0, Z_{n+1} = f(Z_n), n \geq 0$$

Figure 2: Julia & Mandelbrot Set Definitions

Fractal images are usually obtained when the generating function $f(z)$ is non linear. The Mandelbrot Set is obtained by iterating the function $f(z) = z^2 + c$. When the generating function has the form of $f(z) = z^u + c^v$ many other Mandel-like Sets may be produced. Algorithms 1 and 2 show how both the Julia and Mandelbrot Sets can be generated.

Algorithm 1 The Julia Set Algorithm

for each (x,y) in $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$
  construct $z_0 = x + j \times y$;
  find the orbit of z_0 [first N_{iter} elements]
  if(all the orbit points are under the threshold)
    draw (x,y)

Algorithm 2 The Mandelbrot Set Algorithm

for each (x,y) in $[x_{min}, x_{max}] \times [y_{min}, y_{max}]$
  $c = x + i \times y$;
  find the orbit of z_0 while under the threshold R
  if(all the orbit points are not under the threshold)
    draw (x,y)

2.1 Implementation

A simple to use Graphical User Interface (GUI) is provided within the application to allow the user to enter various parameters detailing the type of image to be generated (see Figure 3). The parameters dealing with the fractal image itself include: the image size, number of iterations, radius, cPower, zPower, formula type and image inversion. The other options are for the rate of fractal zoom, and the accuracy of the crosshair (seen in the Image output screen (Canvas)).

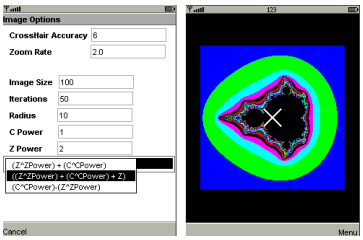

Figure 3: Mandelbrot Settings GUI and Output Image

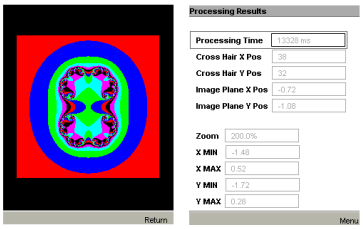

Figure 4: Julia Set Image & Results Output Screen

A thread is used for the generation of the fractal image to allow for user interaction with the image as it is being generated. The outputted image is redrawn at regular intervals so the user can see the progress of the image generation process. The fractal image itself is generated as an array of Integers (Listing 1). This array is then passed

to the createRGBImage(...) method of the Image class to generate the Image object that is ultimately displayed within the onscreen Canvas. The Canvas has a crosshair present to allow the user to navigate around the image and select an area to zoom on. The crosshair is controlled by the directional keys of the mobile device. The user may also view the corresponding Julia Set (Figure 4) for any point that the crosshair is currently indicating, by selecting the "View Julia Set" option from the Mandelbrot Set Canvas Menu.

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

Listing 1: Code listing of Mandelbrot Set Function

The application allows for a variety of Mandel-like images to be generated of the form $f(z) = z^u + c^v$, $f(z) = z^u - c^v$ and $f(z) = z^u + c^v + z$. The cPower and zPower parameters of the GUI dictate the values of u and v. The inverted representation of each form may also be generated by selecting the appropriate option from the GUI, producing images of the form $f(z) = z^u + inv(c^v)$, $f(z) = z^u - inv(c^v)$ and $f(z) = z^u + inv(c^v) + z$.

2.2 Processing Results

The application was tested using a number of image sizes as well as a varying the number of iterations (see Table 1). The 6680 was unable to generate an image of 600 pixels square. Testing with a Nokia 3220 it was unable to generate an image of 200^2 pixels. However it was capable of generating images of 100^2 and 150^2 pixels, but with considerable processing times, 56,503ms and 298,365ms for 100^2 at 50 and 500 iterations respectively.

Device	**Iter**	100×100	200×200	300×300	400×400	500×500	600×600
Nokia 6630	50	3,000ms	9,812ms	19,484ms	33,812ms	53,359ms	76,859ms
Nokia 6680	50	3,141ms	12,516ms	21,859ms	36,234ms	60,859ms	N/A
Nokia 6630	500	13,281ms	52,344ms	111,484ms	196,125ms	301,344ms	429,047ms
Nokia 6680	500	12,359ms	48,000ms	103,219ms	187,797ms	290,062ms	N/A

Table 1: Image Generation Times for the Mandelbrot Set (xmin,ymin -2.0, xmax,ymax 2.0).

3 Prime Number Fractal Generation

Mathematicians have been studying prime numbers for thousands of years, with it origins going all the way back to ancient Greece and the period of Euclid. Prime numbers have remained a mystery throughout the intervening centuries. Their is currently ongoing work to find a prime number with ten million digits or more [4]. The largest prime found to date was discovered 15^{th} December 2005 by Dr. Curtis Cooper and Dr. Steven Boone, the 43^{rd} Mersenne Prime $2^{30,402,457} - 1$ being $9,152,052$ digits in length. The second largest was discovered by Dr. Martin Nowak 18^{th} February 2005, the 42^{nd} Mersenne Prime, $2^{25,964,951} - 1$ containing 7,816,230 digits. The next largest was discovered 15^{th} May 2004 by Josh Findley (41^{st} Mersenne Prime, $2^{24,036,583} - 1$ containing 7,235,733 digits. These discoveries were part of the Great Internet Mersenne Prime Search (GIMPS) project. It is likely that within the next year or so a ten million digit prime number will be found.

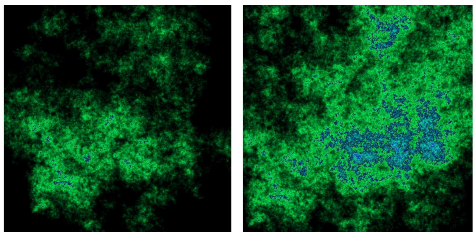

Figure 5: PNF Examples with 5 & 20 Million Primes

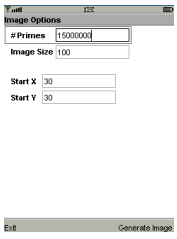

Figure 6: Prime Number Fractal User Interface

The visualisation of prime numbers is again of interest to many in the mathematical community. Probably the most well know visualisation was discovered by Stanislaw Ulam (the Ulam Spiral) in 1963 while doodling during

a scientific meeting [14]. An alternate visualisation is the prime number fractal (Figure 5) where the resultant image has a central area of brightness and typically resembles a gaseous nebula or some other cosmic object. It is generally recognised that Adrian Leatherland from Monash University in Australia constructed the first prime number fractal [6].

The application has a very simple User Interface with only a few components (Figure 6). The options are: the Sieve Size, Image Size, and initial coordinates for x and y.

3.1 Theoretical Background

The most important and evident feature of a prime number fractal image is the central area of brightness. This results in the pixels within the vicinity of the central area are visited more often than pixels around the periphery of the image. The movements in the Up, Down, Left, Right directions occur randomly but each direction produces approximately the same number of moves (see Table 3). Hence the trajectory of moves is random, generally staying around the central area of brightness.

Theorem 1. *The number of Up, Down, Back, Forward movements are asymptotically equal.*

Proof. Dirichlet's theorem assures if a and b are relatively prime then there are an infinity of primes in the set $a \cdot k + b, k > 0$. This means that the random walk has an infinity of Up, Down, Left, Right moves. If $\pi_{a,b}(x)$ denotes the number of primes of the form $a \cdot k + b$ less than x then we know from a very recent result of Weisstein [16] that $\lim_{x\to\infty} \frac{\pi_{a,b}(x)}{li(x)} = \frac{1}{\varphi(a)}$ where $li(x)$ is the logarithmic integral function and $\varphi(a)$ the Euler totient function. The particular case $a = 5$ gives $\lim_{x\to\infty} \frac{\pi_{5,k}(x)}{li(x)} = \frac{1}{\varphi(5)} = \frac{1}{4}, \forall k \in 1,2,3,4$, which means that $\pi_{5,1} \approx \pi_{5,2} \approx \pi_{5,3} \approx \pi_{5,4}(x) \approx \frac{li(x)}{4}$ clearly the two dimensional prime number fractal algorithm has asymptotically the same number of Up, Down, Left and Right moves.

Even on Desktop systems the process of generating primes can take a significant amount of time for example 21,157ms on a AMD Athlon XP2600 system (Table 4). The generation of the primes examines all the numbers between 0 and n the input size. As the prime numbers become larger their distribution becomes far sparser (Table 2). Their should be at least one prime between n and $2(n)$.

# Primes	Sieve Size
1 Million	15,485,865
2 Million	32,452,850
5 Million	86,028,130
10 Million	179,424,680

Table 2: Sieve Size

#Primes	1 Million	5 Million	10 Million	20 Million
Left	249,934	1,249,832	2,499,755	4,999,690
Right	205,015	1,250,079	2,500,284	5,000,241
Up	250,110	1,250,195	2,500,209	5,000,270
Down	249,940	1,249,893	2,499,751	4,999,798

Table 3: Distribution of Moves for a two Dimensional Prime Number Fractal

#Primes	1 Million	5 Million	10 Million
Athlon XP2600	1,781ms	9,752ms	21,157ms
AMD 500Mhz System	4,084ms	23,946ms	51,552ms

Table 4: Processing Times to Generate Primes on Desktop Systems

3.2 The Fractal Algorithm

The generation of the fractal image requires the iteration through a loop (Algorithm 3) for all numbers from 1 to n. The first requirement of the algorithm is to determine if the number i is Prime or not.

The sieve of Eratosthenes is an time efficient method for determining primality of a sequence of numbers. The time complexity is $O(n \cdot log\, log\, n)$, however the space requirement is $O(n)$. The algorithm iterates through the sequence from 2 to n crossing off all numbers > 2 that are divisible by 2. It then moves on to the smallest remaining number, and removes all of its multiples. The process of moving on and removing multiples continues until all numbers up to $\sqrt{n}$ have been crossed off. In Java one may use an array of boolean values to indicate if the number at index i of the array is prime. A boolean in J2SE requires one byte for storage, however the BitSet class may be used. This reduces the storage requirements to $O(\frac{n}{8})$ bytes. As mobile devices have such a limited memory

this approach cannot be used, hence the need for an alternate method. Determination of the primality is achieved by calling the method *isPrime*(i) (Algorithm 4). It returns a boolean value indicating if the number is prime or not.

If the number is prime then the process of plotting the prime takes place. Firstly the direction is calculated by carrying out modular division by 5 yielding $p \bmod 5 \in \{1,2,3,4\}$ this mapping can be clearly seen in Table 5. Next depending of the direction the current pixel location is updated to reflect the direction indicated by the prime. Lastly the colour of the pixel at the new cell location is incremented.

Algorithm 3 2D Prime Number Fractal Algorithm

```
for i = 0 to n -1 do begin
    if(isPrime(i)) then
        dir = p[i] mod 5;
        if dir = 1 then x–;
        if dir = 2 then x++
        if dir = 3 then y–
        if dir = 4 then y++
        pixels[x,y] ++;
    end if;
end for;
```

Algorithm 4 isPrime( ) Algorithm

```
boolean isPrime(long p)
begin
    long d;
    if(p=1) return false;
    if(p=2 || p=3) return true;
    if(p % 2=0 || p % 3=0) return false;
    for(d=3;d <= sqrt(p); d=d+2)
        if(p % d=0) return false;
    return true;
end
```

1. Left: $p \bmod 5 = 1 \Rightarrow (x,y)$ goes in $(x-inc,y)$
2. Right: $p \bmod 5 = 2 \Rightarrow (x,y)$ goes in $(x+inc,y)$
3. Up: $p \bmod 5 = 3 \Rightarrow (x,y)$ goes in $(x,y-inc)$
4. Down: $p \bmod 5 = 4 \Rightarrow (x,y)$ goes in $(x,y+inc)$

Table 5: Mapping of Direction Values to Pixel Movement

3.3 Processing Results

As is the case with Desktop systems the generation of the primes is the most computationally expensive operation. The results (Table 6) show that a significant amount of time is required to generate the prime numbers. This limits the computation of the image to just a few hundred thousand primes so the image may be generated and displayed to the user in a reasonable amount of time.

#Primes	20,000	40,000	60,000	80,000	100,000
Nokia 6630	44,797ms	129,235ms	240,313ms	373,437ms	524,703ms
Nokia 6680	44,984ms	129,000ms	240,078ms	372,875ms	524,046ms

Table 6: Processing Times to Generate PNF on Mobile Devices

4 Plasma Fractal Generation

Plasma Fractals are often referred to as “Fractal Clouds” and the resultant image typically has a cloud like appearance (Figure 7). The generation of this type of fractal uses a recursive algorithm known as Random Midpoint Displacement. Applying the exact same algorithm in the 3D universe to height values will result in the generation of fractal terrain. An example of this method being used for the generation of terrain in the film industry is Star Trek II “The Wrath of Kahn” where Random Midpoint Displacement was used to generate the terrain of a moon, the scene being called the “Geneses Sequence”.

The procedure for generating a “Fractal Cloud” (Algorithm 5) begins by assigning a colour to each of the four corners of a blank image. Executing the “divide(...)” algorithm will firstly find the colour for the central point of the image by calculating the average value of the four corners. The colour value at the central point is then randomly displaced. The image area is then divided into four smaller sections by recursively calling the “divide(...)” again for each of the four quadrants. This division process will continue until the image cannot be

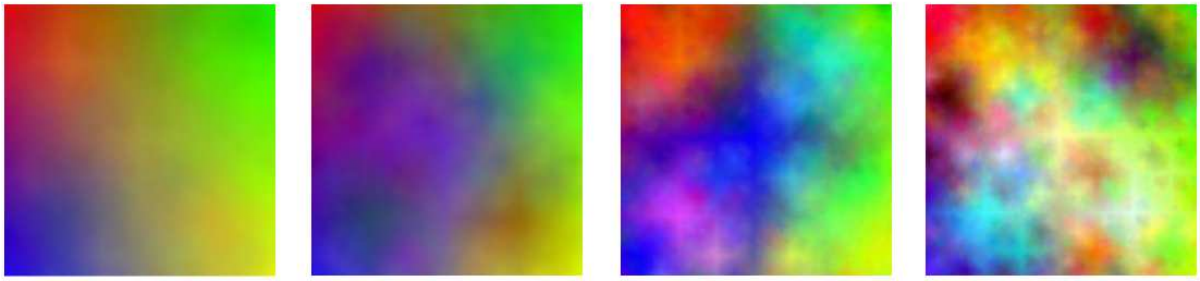

Figure 7: Plasma Fractal Examples (Grain 1.2 , 2.2, 5.8, 9.6)

further broken down. By this time the sub quadrants have reached the pixel level. Several example Applets that use this procedure may be found on the Internet [13] [5].

Algorithm 5 Plasma Fractal Algorithm

```
divide(x,y,w,h,tLC,tRC,bRC,bLC)
float nW = w /2, nH = h /2;
if(w > 1 || h > 1)
    int displace = displace(nW,nH);
    Color top = avgColors() + displace;
    Color right = avgColors() + displace;
    Color bottom = avgColors() + displace;
    Color left = avgColors() + displace;
    Color centre = avgColors() + displace;
    divide(x,y,nW,nH,tLC,top,centre,left);
    divide(x+nW,y,w,h,top,tRC,right,centre);
    divide(x+nW,y+nH,w,h,centre,right,bRC,bottom);
    divide(x,y+nH,w,h,left,centre,bottom,bLC);
else
    drawPixel(x,y)
```

4.1 Processing Results

The results show that mobile devices are capable of generating respectably sized Plasma Fractal Images in a reasonable amount of time (Table 7).

Image Size	100×100	200×200	300×300	400×400	500×500	600×600
Nokia 6630	672ms	2,219ms	6,672ms	6,328ms	6,156ms	23,453ms
Nokia 6680	671ms	2,328ms	7,468ms	7,344ms	6,828ms	25,329ms

Table 7: Processing Times to Generate Plasma Fractals

5 Further Work

The primary focus of this paper was on the generation of two dimensional fractal on Mobile Devices. The obvious progression from this application is to expand the generating functions into the third dimension. In the case of the Plasma Fractal this will yield a randomly generated terrain if the height values of a plane are randomly displaced instead of the colour values of a two dimensional image.

6 Conclusion

An integrated tool has been developed to generate a variety of two dimensional fractals. The fractals in question being the Mandelbrot Set, Julia Set, Prime Number Fractal and the Plasma Fractal. It has been shown that the Mandelbrot and Plasma Fractal Image can be generated in reasonable time. The generation of the PNF image however does involve significant computation resources, especially when the number of primes is greater than 100 thousand.

References

[1] A. Baker, "Mini Review - Enhancements in Nokia 6680," http://www.i-symbian.com/forum/images/articles/43/Mini_Review-Nokia_6680_Enhancements.pdf, 2005.

[2] D. Doolan, S. Tabirca, "Interactive Teaching Tool to Visualize Fractals on Mobile Devices," *Proceedings of Eurographics Ireland Chapter Workshop*,Dublin, Ireland pp. 7–12, 2005.

[3] D. Doolan, S. Tabirca, "Distributed Fractal Generation Across a Piconet," *Proceedings of SIGRAD05 - Mobile Computer Graphics Conference*,Lund, Sweden pp. 63–68, 2005.

[4] GIMPS, "Mersenne Prime Search," http://mersenne.org/.

[5] J. Lawlor, "Plasma Fractal Notes," http://charm.cs.uiuc.edu/users/olawlor/projects/2000/plasma/, 2001.

[6] A. Leatherland, "Pulchritudinous Primes: Visualizing the Distribution of Prime Numbers," http://yoyo.cc.monash.edu.au/~bunyip/primes.

[7] E. Murtazine, "Review GSM/UMTS Smartphone Nokia N90," http://www.mobile-review.com/review/nokia-n90-en.shtml, 2005.

[8] RTE, "Internet and Mobile Penetration Still Rising," http://www.rte.ie/business/2001/0308/odtr.html, 2001.

[9] RTE, "Half the world to have mobile phones by 2015," http://www.rte.ie/business/2004/0225/phones.html, 2004.

[10] RTE, "Mobile Penetration Now Stands at 89%," http://www.rte.ie/business/2004/0907/comreg.html, 2004.

[11] RTE, "Mobile Penetration Now Stands at 94%," http://www.rte.ie/business/2005/0318/comreg.html, 2005.

[12] RTE, "Mobile Penetration Now Stands at 100%," http://www.rte.ie/news/2005/1220/mobilephones.html, 2005.

[13] J. Seyster, "Plasma Fractals," http://www.ic.sunysb.edu/Stu/jseyster/plasma/, 2002.

[14] H. Systems, "Plasma Number Spiral," http://www.hermetic.ch/pns/pns.htm.

[15] C. Taylor, "Mobile penetration to hit 100% in Europe," http://www.enn.ie/news.html?code=9604990, 2005.

[16] E. Weisstein, "Arbitrarily Long Progressions of Primes," http://mathworld.wolfram.com/news/2004-04-12/primeprogressions/, 2004.

Daniel C. Doolan, Sabin Tabirca
University College Cork
Department of Computer Science
Cork, Ireland
E-mail: {d.doolan, tabirca }@cs.ucc.ie