turbo Mandelbrot sets

By Dietmar Soupe

The Mandelbrot set (M-Set for short) has recently attracted considerable attention from amateur scientists and home computer programmers, in large part due to an August 85 article in Scientific American and its associated cover picture (Dewdney, 85). Since then it has become clear that the M-Set and its cousins, the Julia sets, are capable of generating images with fascinating self-similarities and curious yet natural-looking shapes.

Probably more important, though, is that the complexity of these pictures is measured by the length of the programs capable of generating them, is quite small. Consequently, since the Scientific American article was published, such simple do-loops as the following have surfaced:

```c
#define LARGE 1000.0
int GetIterations (cx, cy, maxiter)
double cx, cy;
int maxiter;
{
    int iter = 0;
    double x, y, x2, y2;
    x = y = x2 = y2 = 0.0;
    while ((x2 + y2) < LARGE && (iter < maxiter)) {
        x = 2.0 * x * y + cx;
        y = y2 - x2 + cy;
        x2 = x * x;
        y2 = y * y;
        iter++;
    }
    return (iter);
}
```

**universe**
These do loops have easily devoted thousands of CPU hours on computers of all sizes. Also, some public domain M-set software packages have appeared since that time. With these, people have been able to compute, store, and recover images—thus opening up at a substantial cost. For instance, I recently tried one such program (MandelColor 3.0) from Robert Woodhead Inc., selecting a small section of the M-Set before starting the computation. First, the screen was erased, and then the following message appeared:

Now computing the Mandelbrot set. This will take a while for large values. Please be (very) patient.

And, indeed, the picture was generated pixel-by-pixel, line-by-line. The message was definite; it had to be taken seriously.

Here we describe a new approach to test M-Set computation, based on the exposition of Y. Fisher in The Science of Fractal Images (Fisher86). A beefed-up Macintosh II version of this same approach, containing several algorithms in addition to a color palette editor, is featured in The Gene of Fractal Images (Parmentier86).

The two images at the beginning of this article offer an example. The blow-up of the Mandelbrot section shown on the left was enlarged by a factor of 760,000 and rotated 90 degrees. The lower right corner of the image is at (1.773 0.000 0.000 0.01). Thanks to the new M-Set computation approach, only about 13 minutes of total CPU time was required on an IRIS 3130 to produce the image, thus three to four minutes to render the sphere's touch point was computed separately. The image on the right is a blow-up of the pixel taken from the center of the image on the left, enlarged by a total factor of 1,760,000,000. Thus, too, has been rotated 90 degrees. The lower right corner of the image is at (1.773 0.009 0.01 0.01), where the width is 0.000 0.000 0.01 0.768. Total CPU time on the 3130 came to about 15 minutes.
ALGORITHM TurboMSet()

Title Faster computation of Mandelbrot set

Globals
- rec
  - parameter of the algorithm (see MDisk)
- xmin, xlen
  - start and length of window in x
- ymin, ylen
  - start and length of window in y
- data
  - pointer to doubly indexed array of shorts
- xlen, ylen
  - integer dimension of viewport
- pixel
  - real, size of a pixel
- maxiter
  - maximum number of iterations allowed

Variables
- ix, iy
  - integers

Functions
- push (ix, iy, size) checks if (ix, iy) is not already marked
  in the data array and (if not) puts the coordinates (ix, iy)
  on one of the stacks (large values of size will select
  higher priority stacks)
- pop (ix, iy) pops the top entry of the high priority stack

BEGIN PROGRAM
... initialize the graphics routines
... define the world window coordinates
... allocate memory for the data array (size xlen by ylen)
... initialize stacks of candidate pixels

ix = xlen / (xlen - 1)^2 size of pixel

push (0, ylen-1, 100)  /* upper left corner on stack */
push (xlen-1, 0, 100)  /* lower right corner on stack */
push (xlen-1, ylen-1, 100)  /* upper right corner on stack */

FOR iy=0 TO ylen-1 DO
  FOR ix=0 TO xlen-1 DO
    IF data[ix][iy] = 0 THEN
      MDisk (ix, iy)
    END IF
    IF window queue not empty THEN
      process user interaction
    END IF
    WHILE queue empty and stacks not empty DO
      pop (ix, iy)  /* pop stack */
      IF data[ix][iy] = 0 THEN
        MDisk (ix, iy)
      END IF
    END WHILE
  END FOR
END FOR
END PROGRAM
For a fast algorithm, it would be extremely valuable to be able to compute the distance \( d(c,M) \) of a point \((x,y)\) on a pixel from the M-set \(M\). Let

\[
d(c,M) = \inf \{ \sqrt{(x-a)^2 + (y-b)^2} : |x-a| \leq M, \quad |y-b| \leq M \}
\]

Knowing the distance then, we could exclude a disk centered at the pixel \((x,y)\) with the radius \(d(c,M)\) from further computation.

If none of the pixels inside the disk belong to the M-set, we can save the potential saving such an exclusion represents. Also, although no formula exists for calculating the distance, there is an estimate for it, namely:

\[
R = \frac{\sin G(c)}{2 \exp(G(c)) |G(c)|} < d(c,M).
\]

Here is a clockwise sequence illustrating the progression of the Turbo algorithm in four snapshots. The lower left corner of the upper left image is at \((0.00019, 0.00019)\), the width is 0.00021, height is 0.00021, and resolution is 768 by 576. The number of iterations used in the Turbo algorithm, moving clockwise from the upper left image are: (in the upper left image) 150, (in the upper right image) 540, and (in the lower right image) 280,000. In the lower left image, another 870,000 standard iterations were used to finish the picture shown in the lower right. Thus the overall cost is roughly 2.280 K + 870 K = 1,150 K standard iterations. This compares to 7,050 K standard iterations used by the standard algorithm. The improvement by a factor of approximately five may not seem all that great at first blush but one has to consider that most of CPU time under the Turbo approach is spent filling in very thin detail whereas the overall shape is generated quite quickly. In interactive experimentation, therefore, one should never have to wait for the conclusion of the program. The total time on an IBM 3138 to produce the final 1,280,000 iteration image shown here is about 70 seconds (using disks, not spheres).
ALGORITHM

Title

Arguments

Globals

Variables

Functions

BEGIN PROGRAM

END IF

END PROGRAM
Therefore, a disk drawn around \( z \) with radius \( R \) also does not intersect the Mandelbrot set. In this formula, \( G(z) \) denotes the potential of \( M \) at the point \( z \), the negative base 2 logarithm of which is essentially the number of iterations computed by the iteration:

\[
G(z) = \lim_{k \to \infty} \log_2 |z_k|
\]

where

\[
z_{k+1} = z_k^2 + c.
\]

In computing \( R \), further simplifications are possible. The result is a scheme about two or as costly as the usual iteration above, plus the cost for one evaluation of a logarithm and a square root (see The Science of Fractal Images for details and pseudo-code).

Given a routine to compute \( R \), one can use the following "Tube-Sizing" scheme to create M-Set: start with the first pixel in first scan line, compute radius \( R \), and draw a disk of that radius. Then, proceed to the next pixel on the first scan line not covered by the disk. Again, compute and draw a disk. Continue in this fashion for the remaining pixels in the first and, later, in all other scanlines.

A more interesting approach results in the following recursive method: for each disk drawn, select some pixels next to the disk (we have used six pixels in the images presented here). For each point not already covered by a disk, recursively draw disks and determine more candidate points along their boundaries. Recursion should stop when the disks become smaller than some predetermined tolerance, a parameter of the algorithm itself. When no more candidate pixels are to be processed, the next step is to scan through the image to see which pixels have not yet been covered. For those, recursion can be resumed until the picture is complete (see the pseudo-code in Figure 1 for more details). A few more points may also prove useful.

1. The color of the disks used in the approach just described should be uniform for example. You can use the number of iterations as an index for a color lookup table.
2. While the computation is proceeding, you can input seed points by clicking a mouse button, thus instructing the algorithm to go to those locations first.
3. It is somewhat better to work with several stacks as compared to using pure recursion. Depending on the size of the currently drawn disk, you may enter candidate pixels around the boundary into one of the stacks. The algorithm always checks first for those candidates that originate from the largest disks. That way larger disks are drawn first, and a rough global picture can be quickly obtained.
4. Enter pixels into one of the stacks only if the pixel is in the window and not already tested.
5. The algorithm should always be prepared to accept user input for defining the subsection that is to be computed next.
6. At some point (when all further disks seem to be too small to be seen), you may want to disable the computation of disk radii in order to achieve a speed-up of nearly two-fold. This can be done automatically when all stacks are empty. From then on, however, one has to check periodically for possible disks.
7. There are two ways to check a pixel to see if it already is set, either disks have been drawn within an internal array in CPU memory or they have been drawn only in the image memory (framebuffer). Typically, on an IRIS 8000 system, they'll have been drawn the first way, for example, the Béchdel circle drawing algorithm, as detailed in Computer Graphics (Hearn), as a good example. But whenever disks have been drawn in CPU memory, notice that the result in the image will not be identical to the outcome of the IRIS-8000 routine.

For aesthetic reasons all the images presented in this article were rendered using shaded and X-buffered sphere in place of disks. The projection of the spheres onto the plane \( z = 0 \) corresponds exactly to the disks described above. Finally, there is also a more complicated optimization formula that can be used to calculate the distance between a point inside the M-Set and the M-Set's boundary. Thus, besides filling the outside of the M-Set with disks, one can also quickly fill the interior. Notice also that very similar ideas can be applied to the computation of connected Julia sets (see Fath and Tanenbaum).