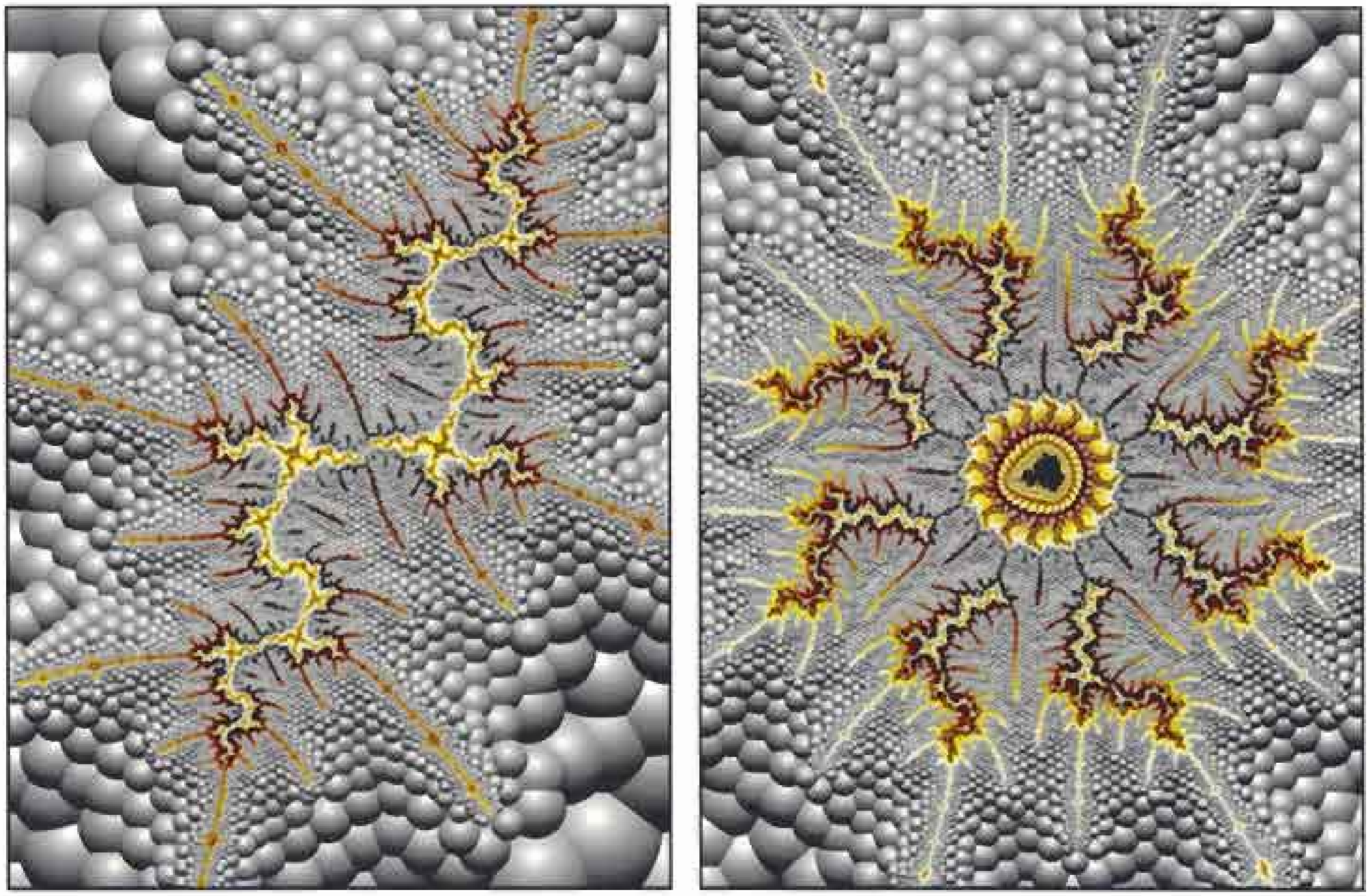


turbo Mandelbrot sets

By Dietmar Saupe

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 *[Dewdney85]*. 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, as 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:

```
#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)) {
          y = 2 * x * y + cy;
          x = x2 - y2 + cx;
          x2 = x * x;
          y2 = y * y;
          iter++;
     }
     return (iter);
}
```

These do loops have surely devoured 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 retrieve images--but often only 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 cleared, and then the following message appeared :

> *Now computing the Mandlebrot (sic!) set. This will take a while for large windows. Please be (very) patient.*

And, indeed, the picture was generated pixel-by-pixel, line-by-line. The message was definitely to be taken seriously.

Here we describe a new approach to faster M-Set computation, based on the exposition of Y. Fisher in *The Science of Fractal Images* [*Fisher88*]. A beefed-up Macintosh II version of this same approach, containing several algorithms in addition to a color palette editor, is offered in *The Game of Fractal Images [Parmet88]*.

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 (-1.773 520 992, 0,006 835 510), the width is 0.000 003 931. 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, plus three to four minutes to render the spheres (each 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 1,700,000,000 (this, too, has been rotated 90 degrees). The lower right corner of the image is at (-1.7773 519 041 423, 0.006 836 860 241), where the width is 0.000 000 001 765. Total CPU time on the 3130 came to about 43 minutes.

suggested table of user interactions

Entries for the event queue :

LEFTMOUSE	place the pixel at the mouse position on the highest priority stack; the Turbo algorithm will process that pixel next, drawing a disk.
DKEY	toggle between single and double-precision calculation.
AKEY	toggle between Turbo method and usual iteration without disks.
ZKEY	output number of iterations accumulated, compare against the number that would be necessary with the standard algorithm (can be computed from the stored array of pixel values).
MIDDLEMOUSE	invokes pop-up menu.

Entries for pop-up menu :

Continue	proceed with the computation of the image (otherwise the pop-up menu is brought up again to allow more interaction).
Maxiter	brings up another pop-up menu of numerical values to be chosen for the number maxiter (the maximum number of iterations allowed per pixel).
Recur	brings up a menu for selection of the parameter recur (determines the smallest disks allowed to define subsequent candidate points around its boundary).
Zoom	lets the user define a section of the current image to be used as the next world-window in the algorithm; the present computation should be aborted.
Save	save the image and the world-window coordinates in one or two files.
Load	load an image which previously had been stored on disk, the coordinates are also retrieved such that further zooms are possible.
Show Disks	enters a mode in which disks at the current mouse position are computed and drawn in a highlighted color.
Clear	clears the screen and memory to start a fresh picture (maybe with different parameter settings).
Job	collect all information (window, parameters, etc) about the current image and store it in a file, which later can be processed by a batch version of the program.
Quit	exit the program.

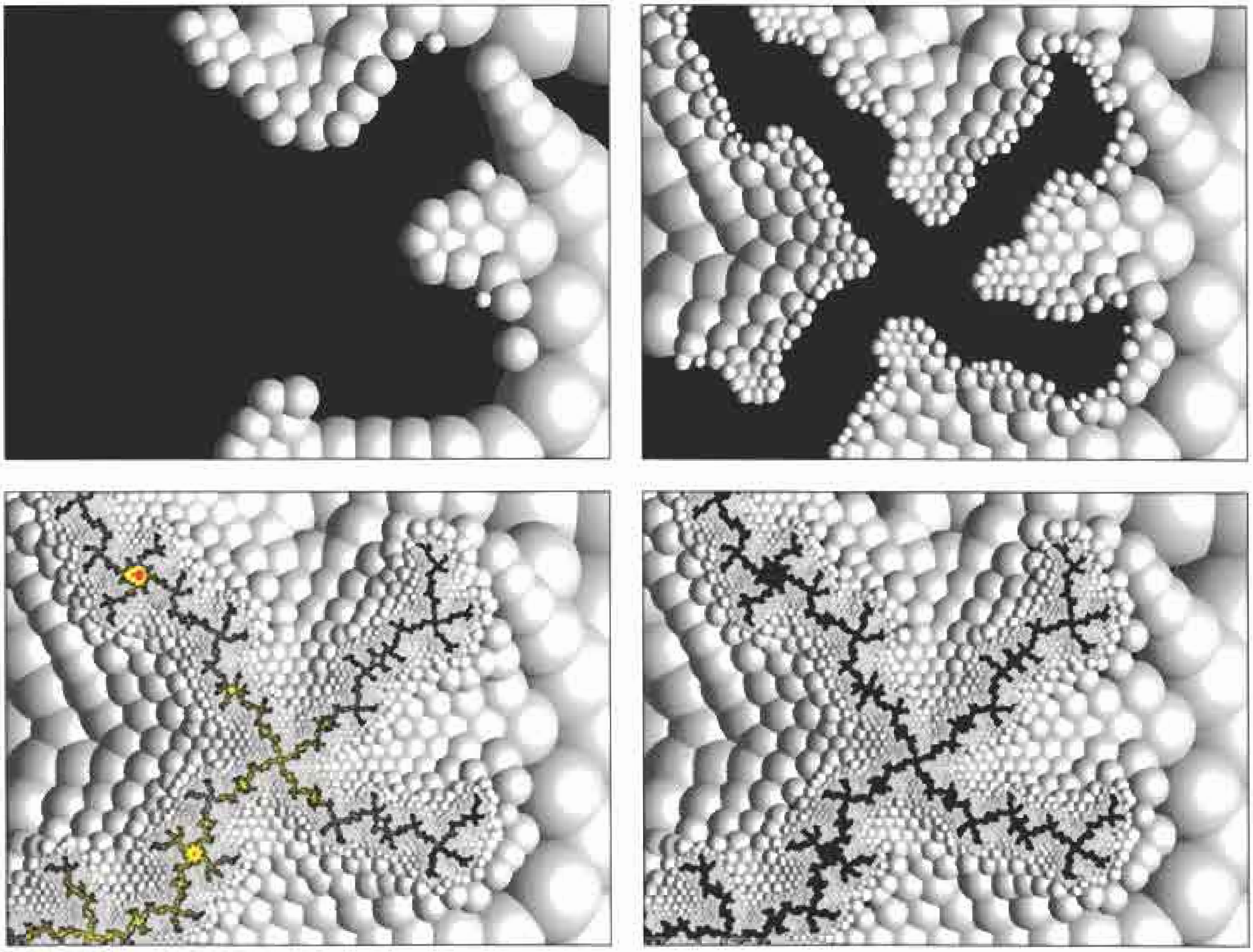
```
ALGORITHM
          TurboMSet()
Title
          Fast computation of Mandelbrot set
Globals
          recur                parameter of the algorithm (see MDisk)
          xmin, xlen           start and length of window in x
          ymin, ylen           start and length of window in y
          data                 pointer to doubly indexed array of shorts
          xlen, iylen          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 ixlen by iylen)
          .... initialize stacks of candidate pixels
          pixel := xlen / (ixlen - 1)/* size of pixel */
          push (0, iylen-1, 100)              /* upper left corner on stack */
          push (ixlen-1, 0, 100)              /* lower right corner on stack */
          push (ixlen-1, iylen-1, 100)        /* upper right corner on stack */
          FOR iy=0 TO iylen-1 DO
                    FOR ix=0 TO ixlen-1 DO
                              IF data[ix][iy] = 0 THEN
                                        MDisk (ix, iy)
                                        IF event 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 IF
                    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 (pixel) c from the M-Set M, like so:

$$d(c,M) = \inf \{ \sqrt{(x-\mathrm{Re}\,c)^2 + (y-\mathrm{Im}\,c)^2} \mid x + iy \in M \}.$$

Knowing the distance, then, we could exclude a disk centered at the pixel c with the radius $d(c,M)$ from further computations, since none of the pixels inside the disk belong to the M-Set. Given that the disk may contain dozens or even thousands of pixels, you can imagine the potential savings such an exclusion represents. Also, although no formula exists for calculating the distance, there is an estimate for it, namely:

$$R = \frac{\sinh G(c)}{2 \exp(G(c))\, |G'(c)|} < d(c,M).$$



*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 (0.350 319, 0.658 221), the width is 0.043 271, maxiter = 200, recur = 1, and resolution = 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) 950, (in the upper right image) 5400, 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 on the lower right. Thus the overall cost is roughly 2 * 280 K + 870 K = 1430 K standard iterations. This compares to 7000 K 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 fine 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 IRIS 3130 to produce the final 1,150,000-iteration image shown here is about 70 seconds (using disks, not spheres).*

```
ALGORITHM       MDisk (ix, iy)
Title           Compute and draw a disk around given pixel
Arguments       ix, iy                           integer coordinates of pixel
Globals         as above in TurboMSet
Variables       R                                estimated radius of disk
                rad                              radius in pixels (integer)
                idx, idy                         integers
                iter                             number of iterations
                cx, cy                           real numbers, pixel coordinates
                ColorIndex                       integer index for color lookup table
                r                                integer
Functions       MSetRad (cx, cy, maxiter, R, iter)   computes radius R of
                disk around (cx,cy) completely outside of M-Set and number
                of iterations iter (see [3])
                FillDisk (ix, iy, rad, index) fills a disk centered at
                (ix,iy) with radius rad on the screen with color given by
                index, fills corresponding disk in data array data[][]
                DrawPoint (ix, iy, index) same as FillDisk for individual
                pixel (ix,iy)
                push (ix, iy, size) see TurboMSet
BEGIN PROGRAM
        cx := xmin + ix * pixel              /* world x coordinate of point */
        cy := ymin + iy * pixel              /* world y coordinate of point */
        MSetRad (cx, cy, maxiter, R, iter)   /* compute R, iter */
        ColorIndex := iter
        rad := (int) (R / pixel)             /* radius in integer pixel units */
        IF rad > 0 THEN
                FillDisk (ix, iy, rad, ColorIndex)
        ELSE
                DrawPoint (ix, iy, ColorIndex)
        END IF
        /* if the radius is still bigger than recur pixels, then push 6
          new pixels around the boundary of disk on the stacks */
        IF R > recur*pixel THEN
                r := rad + 1
                idx := (int) 0.500 + 0.500 * r
                idy := (int) 0.500 + 0.866 * r
                push (ix + r, iy, rad)
                push (ix - idx, iy + idy, rad)
                push (ix - idx, iy - idy, rad)
                push (ix + idx, iy + idy, rad)
                push (ix - r, iy, rad)
                push (ix + idx, iy - idy, rad)
        END IF
END PROGRAM
```

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

$$G(c)=\lim_{k \to \infty} \frac{\log |z_k|}{2^k}$$

where

$$z_{k+1} = (z_k)^2 + c, \; z_0 = c$$

In computing *R*, further simplifications are possible. The result is a scheme about twice 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 "Turbo-fast" scheme to create M-Sets: 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 scanline 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 predefined tolerance, a parameter of the algorithm (delta). 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 pixels 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 care about), 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 3000 system, they'll have been drawn the first way (for example, the Bresenham circle drawing algorithm, as detailed in *Computer Graphics [Hearn86]*, is a good option). 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 **circfi** routine.

For aesthetic reasons all the images presented in this article were rendered using shaded and Z-buffered spheres 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) estimation formula that can be used to calculate for 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 *[Fisher88]* and *[Parmet88]*).

references

[Dewdney88] A. K. Dewdney, "Computer Recreations: Exploring the Mandelbrot Set", *Scientific American* (August 1985).
[Fisher88] Y. Fisher, "Exploring the Mandelbrot Set", *The Science of Fractal Images*, H.O. Peitgen, D. Saupe (editors), pp. 287-296, Springer-Verlag, New York (1988).
[Hearn86] D. Hearn, M.P. Baker, *Computer Graphics*, Prentice-Hall, Englewood Cliffs (1986).
[Parmet88] M. Parmet, H.O. Peitgen, H. Jurgens, D. Saupe, "The Game of Fractal Images", available on computer disk through Springer-Verlag, New York (1988).

Dietmar Saupe is an Assistant Professor in the Department of Mathematics at the University of Bremen (2880 Bremen 33, West Germany).