Visualizing the Mandelbrot set with METAPOST

Max Günther

Abstract

With the advance of modern programming languages allowing for parallelized and optimized computation, visualizing the Mandelbrot set has become easier than ever before. METAPOST was not designed for such time-consuming tasks, nevertheless it has surprisingly acceptable performance.

1 Introduction

The Mandelbrot set is a set of points in the complex plane. A point c is part of this set if the sequence z_n defined as $z_{n+1} = z_n^2 + c$ with $z_n, c \in \mathbb{C}$ and $z_0 = 0$ does not diverge to infinity [1, 2]. In practical applications, it is impossible to perform an infinite number of iterations for z_n. For this reason we define a maximum number of iterations $n_{\max}$. In addition, we can stop the iteration as soon as the norm of the position vector $\vec{z_n}$ exceeds a radius of 2, since it can be shown that in this case z_n will eventually diverge to infinity [4].

Herbert Voß has already implemented an algorithm for visualizing the Mandelbrot set in the German journal DTK [3]. My goal was to port his code from Lua to METAPOST, examining how it will perform and what difficulties will arise in this process.

2 Where are the complex numbers?

The first obstacle I encountered was the lack of complex numbers in METAPOST. To be fair, it is pretty unlikely that you will need complex numbers when creating graphics, so nobody bothered to include them. Luckily, the calculation can be performed using basic algebra!

Recall that a complex number $a = x+yi$ consists of both a real part x and an imaginary part y, which can be used to represent the complex number as a point in the complex plane, as shown in figure 1. It is worth noting that this is a two-dimensional coordinate system, which is (of course) supported by METAPOST.

We also need to be able to add and square complex numbers. By looking at the real and imaginary part of a complex number separately, we can calculate $a + b$ and a^2 with ease:

$$(x + yi) + (u + vi) = (x + u) + (y + v)i$$
$$(x + yi)^2 = x^2 - y^2 + 2xyi$$

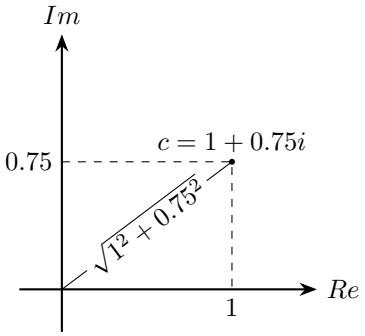

Figure 1: The complex number $c = 1 + 0.75i$ in the complex plane. *Re* is the real axis, *Im* the imaginary axis. The equation notated diagonally uses Pythagoras' theorem for calculating the distance between point c and the origin of the coordinate system.

3 Several arithmetic overflows later...

After successfully setting up the innermost loop and checking that the sequence z_n has been calculated correctly, I tried to integrate the two outer loops. During this stage I encountered multiple arithmetic overflows. The mistake was that I carelessly mixed the operators `=` and `:=` in the loop body. An equal sign (without the colon) is the instruction for solving linear equations, *not* the assignment operator. This resulted in unnecessarily constructing a gigantic equation, too huge to be handled by METAPOST.

4 Let it be colorful!

At this point the Mandelbrot set was easy to recognize, but rather dull: each pixel belonging to the set was colored black, the rest was white. To uncover more detail of the Mandelbrot set — especially in the aura — we can use the escape time algorithm. The color of a pixel depends on the number of iterations n completed before the norm of the position vector $\vec{z}$ exceeds the radius of 2. In Voß' implementation, this results in a value between 0 and 255; however, METAPOST expects RGB values between 0 and 1. For that, we can use the following equation:

$$\frac{(n_{\max} - n)}{n_{\max}}$$

5 Optimizing the code

To speed up the computation, I implemented the following optimizations:

1. Extract constants (like `dx` and `dy`) from the loop body to prevent unnecessary computation.
2. Square Pythagoras' theorem, so `sqrt(re**2 + im**2) > 2` becomes `re**2 + im**2 > 4`.
3. Fill the whole image with black and only draw pixels not belonging to the Mandelbrot set.

doi.org/10.47397/tb/45-1/tb139guenther-mandelbrot

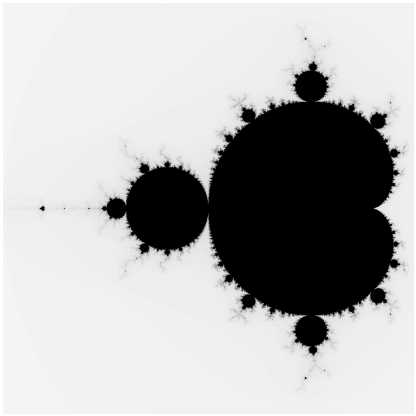

Figure 2: The output of the METAPOST program. The generation of this image with a resolution of 1000 by 1000 pixels took about three minutes on an 11th Gen Intel i7. The following values were used: $x_{\min} = -2$, $x_{\max} = 0.5$, $y_{\min} = -1.25$, $x_{\max} = 1.25$, $n_{\max} = 200$ and $res = 1000$.

The last optimization has the positive side effect of reducing the number of conditional statements needed for drawing a pixel in the correct color.

6 Conclusion

Trying to implement Herbert Voß' algorithm for visualizing the Mandelbrot set was a great experience. After about 3.5 hours of tinkering I finally achieved convincing results. I enjoyed the process and now feel more confident about working with METAPOST in the future.

Figure 2 shows the final image of the "Apfelmännchen", as this detail of the Mandelbrot set is called in Germany, because it is reminiscent of a man rotated by 90 degrees. The source code is displayed in section 7. Feel free to try it out by yourself and play around with the values to explore different parts of the Mandelbrot set.

7 Final code

Save the following code as `mandelbrot.mp` and run it using `mpost mandelbrot.mp`. The resulting image is called `mandelbrot.png`.

```
outputtemplate := "%j.png";
outputformat := "png";

beginfig(1)
  numeric x_min, x_max, y_min, y_max, res;
  x_min := -2; x_max := 0.5;
  y_min := -1.25; y_max := 1.25;
  res := 1000;

  numeric n_max, dx, dy; n_max := 200;
  dx := (x_max - x_min) / res;
  dy := (y_max - y_min) / res;

  fill (0,0)--(res-1,0)--(res-1,res-1)
    --(0,res-1)--cycle;

  for x=0 upto res - 1:
    for y=0 upto res - 1:
      numeric re, im, old_re, old_im, a, b;
      re := 0; im := 0;
      re_old := 0; im_old := 0;

      a := x * dx + x_min;
      b := y * dy + y_min;

      for n=0 upto n_max:
        numeric squared_re, squared_im;
        squared_re := re**2;
        squared_im := im**2;

        re_old := squared_re - squared_im;
        im_old := 2.0 * re * im;
        re := a + re_old;
        im := b + im_old;

        if squared_re + squared_im > 4:
          numeric c; c := (n_max - n) / n_max;
          numeric o; o := 0.95;
          fill (x,y)--(x+o,y)--(x+o,y+o)
            --(x,y+o)--cycle withpen pencircle
            scaled .1pt withcolor (c, c, c);
        fi;
        exitif squared_re + squared_im > 4;
      endfor;
    endfor;
  endfor;
endfig
end
```

References

[1] B. Fredriksson. An introduction to the Mandelbrot set, Jan. 2015. www.kth.se/social/files/5504b42ff276543e4aa5f5a1/An_introduction_to_the_Mandelbrot_Set.pdf

[2] J. Montelius. Generating a Mandelbrot Image, 2018. people.kth.se/~johanmon/courses/id1019/seminars/mandel/mandel.pdf

[3] H. Voß. Chaotische Symmetrien mit Lua berechnet. *Die TEXnische Komödie*, 32(3):51–57, Aug. 2020. archiv.dante.de/DTK/PDF/komoedie_2020_3.pdf

[4] E.W. Weisstein. The Mandelbrot set — from Wolfram MathWorld. mathworld.wolfram.com/MandelbrotSet.html

◇ Max Günther
code-mg (at) mailbox dot org
www.guemax.de