Demo: New View on Plasma Fractals

From the High Point of Array Languages

Oleg Kiselyov
Tohoku University
Japan
oleg@okmij.org

Toshihiro Nakayama
Tohoku University
Japan
nakayama.toshihiro.t6@dc.tohoku.ac.jp

Abstract

Plasma fractals is a technique to generate random and realistic clouds, textures and terrains – traditionally using recursive subdivision. We demonstrate a new approach, based on iterative expansion. It gives a family of algorithms that includes the standard square-diamond algorithm and offers various interesting ways of extending it, and hence generating nicer pictures. The approach came about from exploring plasma fractals from the point of view of an array language (which we implemented as an embedded DSL in OCaml) – that is, from the perspective of declaring whole image transformations rather than fiddling with individual pixels.

1 Summary

Plasma fractals [Fournier et al. 1982] are widely used to simulate clouds, generate textures and build height-maps for realistic terrains.

We present a new approach to generating plasma fractals, generalizing the standard square-diamond algorithm. The approach leads to a family of algorithms which we have just started to explore. It came about as we were contemplating realizations of plasma fractals in an array language – the kind of language which discourages fiddling with individual pixels and forces us to think of transformations on images as a whole.

The bird-eye view afforded by array languages made us realize that plasma fractals emerge from repeated noisy image expansion, with appropriately scaled noise – in marked contrast with the conventional view as recursive subdivision. Our algorithms are not recursive but iterative.

Along with plasma fractals we demonstrate our array programming language – in the spirit of APL but implemented as an embedded DSL in OCaml and typed. Thanks to the rich host language, we enjoy modularity, expressivity, standard library and powerful abstractions while avoiding puzzling and error-prone behavior due to implicit conversions and pervasive overloading typical of array languages.

The complete, self-contained code is available at https://okmij.org/ftp/image/ArrayL/.

2 Plasma Fractals as Recursive Subdivision

Conventionally, plasma fractals are presented as recursive subdivision, following [Miller 1986]. In the simplest midpoint displacement algorithm, we start with the initially empty image of the target size and seed its four corners: black circles in Fig. 1(a). Midpoints along the sides (yellow circles in Fig. 1(b)) are filled with the averages of the two closest corners – plus a random displacement. The center point (Fig. 1(c)) is the average of the four corners, plus a random displacement. As the result, the four sub-squares (Fig. 1(d)) have their corners filled, and the procedure is recursively applied to them

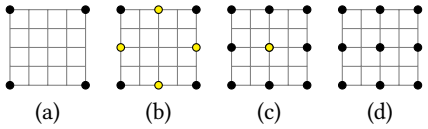

Figure 1. Midpoint displacement algorithm

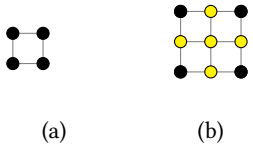

Figure 2. Expansion Algorithm

(with a scaled down random displacement) – until the entire image is filled.

The square-diamond algorithm is an elaboration to avoid line artifacts of the simple algorithm.

3 Plasma Fractal = Expansion + Noise

As we were pondering how to implement plasma fractals in an array language (see §4) and avoid concerning ourselves with filling-in of individual pixels, we have come across a new presentation. Whereas the convention talks about recursive subdivision, we think of progressive (iterative) enlargement.

We start with a 2×2 seed picture, Fig. 2(a), and expand it: Fig. 2(b). The yellow circles represent 'new pixels', whose values are determined by resampling the original image, with the added random noise. With bilinear interpolation as resampling, the side pixels are the averages of their two side neighbors, and the middle is the average of its four neighbors. The result is hence equivalent to (a scaled down) Fig. 1(d). Repeating the expansion gives the image of the target size.

Thinking in terms of images rather than pixels (as array languages encourage us to do [McIntyre 1991; Smillie 2005]), the noisy expansion step is the standard image enlargement to which a noise image is added. The enlargement (upsampling) can be represented as a 2D-convolution. For bilinear interpolation upsampling, the convolution kernel is the 2×2 array: is the outer product of $[1; 1]/2$ with itself. Interestingly, the square-diamond algorithm can also be represented as the repeated scaling/convolution+noise, using a particular convolution kernel. Fig. 3 shows sample plasma fractals generated with bicubic interpolation upsampling – the new algorithm that improves on square-diamond and produces images with fewer artifacts.

4 Array Language

What made the exploration of plasma fractals easier is an array language – in our case, one embedded in OCaml. It gave a playground to quickly try various algorithms and see their results.

Our language is greatly inspired by APL. Unlike APL, J, K, etc., however, it has the conventional programming language syntax (actually that of OCaml): expressions may span as many lines as needed to nicely display them, highlight their structure and attach comments; one may locally name (sub)expressions for readability; parsing/understanding requires less look-ahead.[1] There are further, less noticeable but just as important differences from APL: the flow of data is left-to-right (as common in electrical engineering and signal processing) rather than right-to-left. Not everything is an array: there are also numbers, pairs, strings, booleans, and all other OCaml data types. There are types. Implicit conversions, padding, overloading, slicing are eliminated to minimize surprises and subtle errors, especially due to typos. For example, a conversion from int to float has to be notated explicitly, using OCaml's standard float_of_int. That may become cumbersome – but OCaml's local module open, local definitions and other abstractions help. The whole OCaml and its libraries are available for syntax sweetening, in the manner advocated in [Kiselyov 2019].

Still our language is an array language, distinguishing array shape and contents, relying on compositions and array arithmetic; tacit programming (a.k.a. point-free style) is also available, with the set of common combinators. Our arrays are defined as

```
type (ι,α) arr = Arr of ι * (ι → α)
```

literally as the combination of shape, or index domain (represented by the type ι) and content: the function from an index to an element. For images, the index domain type is the pair of integers

```
type d2 = int * int
```

and the shape is the pair of the largest row and column indices. In this representation, operations on arrays are automatically fused and copying is avoided. In fact, no arrays are allocated unless explicitly requested, by calling materialize2.

As an example, the noisy enlargement step of the plasma fractal algorithm from §3 is implemented as

```
let noise (Arr (d,_)) = Arr (d, fun (i,j) →
    if i land 1 = 0 && j land 1 = 0 then 0 else rand ())

let expander (scaler: ((d2,int) arr → (d2,int) arr)) (nsf: float)
 : (d2,int) arr → (d2,int) arr =
  map (fimul nsf) ▷ scaler ▷ fun m2 →
  zip_with (+) m2 (noise m2) ► materialize2 0
```

where ► is left-to-right application and ▷ is left-to-right composition. The type annotations are optional and given for clarity. The argument scaler is an enlargement function, such as bilinear, bicubic or square-diamond convolutional upsampling. The argument nsf is noise scaling related to the fractal dimension. For plasma fractals, it should be around 1.1–2.2.

Fig. 3 (top) is the result of

```
let m0 = of_array [|4;4;4;4|] ► rho2 (2,2) in
m0 ► ntimes 8 (expander scale_twice_bc 1.2)
```

[1] APL parsing requires unlimited look-ahead (or, more precisely, look-behind, since APL is parsed right-to-left): http://dfns.dyalog.com/n_parse.htm.

The bottom image on the figure is obtained with the nsf parameter set to 2.0.

5 From Now On

The new family of plasma fractal algorithms is vast: any image expansion algorithm[2] instantly gives a plasma fractal algorithm. We have just begun to explore this landscape.

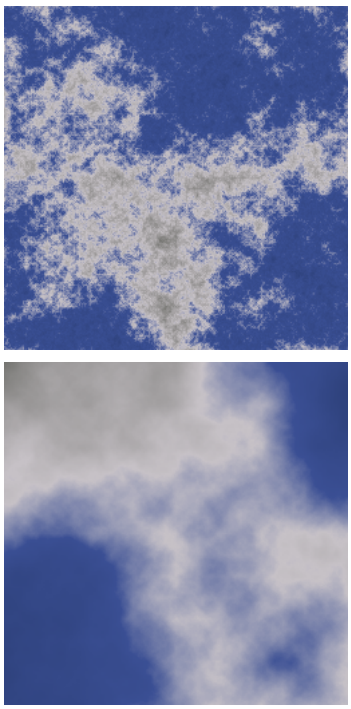

Figure 3. Generated plasma fractals (resampling by bicubic interpolation)

References

Alain Fournier, Donald S. Fussell, and Loren C. Carpenter. Computer rendering of stochastic models. *Communications of the ACM*, 25(6):371–384, June 1982. doi: 10.1145/358523.358553.

Oleg Kiselyov. Effects without monads: Non-determinism – back to the Meta Language. *EPTCS*, 294:15–40, 2019. doi: 10.4204/EPTCS.294.2.

Donald B. McIntyre. Language as an intellectual tool: From hieroglyphics to APL. *IBM Systems Journal*, 30(4):554–581, 1991. doi: 10.1147/sj.304.0554.

Gavin S. P. Miller. The definition and rendering of terrain maps. In David C. Evans and Rusell J. Athay, editors, *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1986, Dallas, Texas, USA, August 18-22, 1986*, volume 20, pages 39–48, August 1986. doi: 10.1145/15922.15890.

Keith Smillie. My life with array languages. October 2005.

[2] see https://en.wikipedia.org/wiki/Resampling_(bitmap) for a good list