MetaFun for generative art

Abstract
This article shows how MetaFun can be used to create generative art, by showing the construction of three projects, step by step.

Keywords
MetaFun, art, creation

Introduction

The idea of making generative art with MetaFun was for me a convolution of several elements. The first is a painting by Niele Toroni seen at the Museum of Modern Art of New-York (MOMA) showing imperfect red squares on a canvas: it was incredible. I recently redid a version with circles (not to copy the original version) shown in figure 1. The second element is that I used to make regular representations of random processes in Tikz, to illustrate lecture notes on stochastic processes. Not only these representations are useful, but sometimes beautiful too ! The third and last element is probably the beautiful covers of the ConTEXt manuals, combined with the discovery of the MetaFun manual.

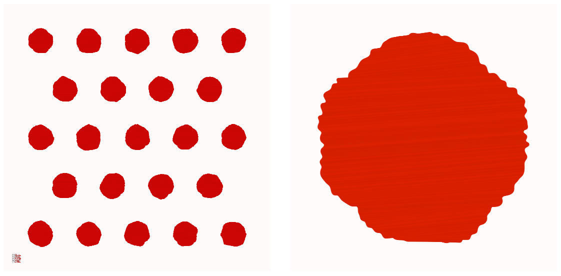


Figure 1. Left a version of Niele Toroni piece with circles instead of squares. Right a zoom on the first circle.

It seemed so easy and beautiful to represent random representations in MetaFun that I explored it. After making many drawings, simple at first, and then more advanced, I discovered that this activity had a name: this art form is called *generative art*. We can borrow the definition from Galenter (2013):

"*Generative art refers to any art practice where the artist uses a system, such as a set of natural language rules, a computer program, a machine, or other procedural invention, which is set into motion with some degree of autonomy contributing to or resulting in a completed work of art.*"

The key for generative art made by a drawing language, is to have random components. Usually, in my drawings, every part, even a tiny one is randomized: position, form, color, etc. An example of this is shown in the right part of figure 1, presenting a small part of the whole piece. As you can see, instead of filling the circle with a unique color, dozens of lines are used trying to give the feeling a real brush, using a randomized color. The accumulation of these details bring organicity to the piece.

So a such a drawing is nothing more than an algorithm with some random components. That means that the same algorithm will produce different results. In general, the more you leave room for randomness, the less predictable the result will be, the more surprising and interesting it will be, but the longer it will take to sort out the successful results among the ten or a hundred drawings made by the same algorithm. So more randomness equals more curation.

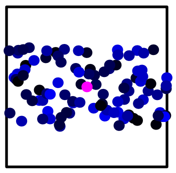
It is interesting to note that MetaFun can also be used to make movies (animations) with the help of ConTEXt : a loop in ConTEXt permits to generate easily several hundreds of pages, where some parameters of a drawing change from one page to the next. We can combine these frames, at a rate for example of 30 frames by second, to make a movie, where we can of course add music. Finally, before drawing, let's add that most generative artists use a javascript library named *p5.js*.

The *randomized* operator

The MetaFun manual is *the* document to read to learn properly the language, and I assume the reader has a familiarity with the basics of the language, although explanations will be as detailed as possible. I emphasize here a particular aspect of MetaFun, the randomized operator, heavily used for generative art. Briefly, this operator adds randomness to almost everything, and this randomness is the key for generative art.

Let's draw a square of size 60, it will be centered at position $(0,0)$, and a point at position $(0,0)$ in color magenta; then we will draw 50 randomized points around the $(0,0)$ point, in blue, randomized (60,30) :

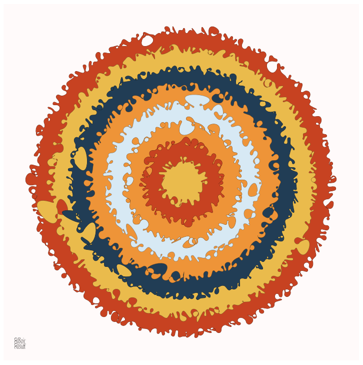
```
draw fullsquare scaled 60 ;
for i=1 upto 50:
       draw (0,0) randomized (60,30)
         withcolor blue randomized(0.1,2) ;
endfor;
draw (0,0) withcolor magenta ;
```



As you can see, the pair of numeric values following the randomized operator gives the amount of randomness in both directions, express in regular unit; moreover the color can be randomized too. In fact, as described in the MetaFun manual, “randomized can handle a numeric, pair, path and color, and its specification can be a numeric, pair or color, depending on what we’re dealing with.”

Project one : delirious circles

MetaFun (MetaPost) is a vectorial language; it is then really easy and natural to draw smooth lines and curves, but it is a little more work to draw *agitated* (i.e. not smooth, chaotic, erratic...) paths. Of course, this is possible, because so many things are possible with MetaFun ! So we begin by building a simple piece illustrating *agitated* paths. Once this first piece is built, we will see how we can use this work to create a more complex piece. Our first objective is to build a piece similar to this :



The first thing to realize this piece is to be able to draw these *agitated* paths. This algorithm is now described and illustrated in figure 2, using a simple circle. In order to *agitate* a path P, the first step is to take a number of points n_1 along P (here $n_1 = 8$) and randomize these points by a quantity t_1 (t for turbulence); then build a path with these n_1 points, and we obtain a new path illustrated as “Step 1” in the illustration. We then repeat this strategy recursively a number of times S (for Steps), using the result of the previous step as the path to be randomized in the current step, taking more points and less noise at each step; this way, the first steps give the global form of the resulting path, and the last steps add some little noise along the path. Varying these parameters, we will obtain different results. Please note that the following MetaFun code is made to be comprehensive more than computationally optimal; let's do it:

```
numeric n[] , t[]; path P;
% Initial values ----------------- ;
n[1] := 8 ; t[1] := 10 ;
% Calculations of t_s and n_s ---- ;
for i=2 upto 5:
  n[i] := n[i-1] * 2 ;
  t[i] := t[i-1] * .8 ;
endfor;
% Initial shape ------------------ ;
P := fullcircle scaled 40;
% Let's add turbulence in S=5 steps ---- ;
for s=1 upto 5 :
 P := for i=1 upto n[s]:
  point (i/(n[s])) along P randomized t[s] ..
    endfor cycle ;
  draw P  ;
  drawpoints P withpen pencircle scaled 2
    withcolor red;
endfor;
```

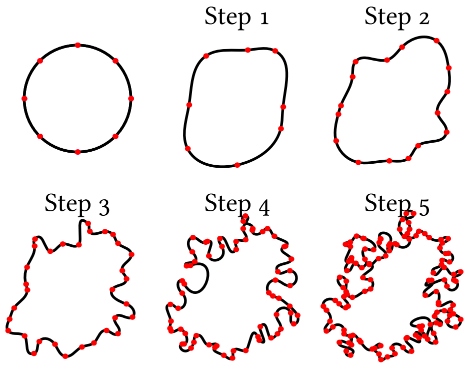


Figure 2. Illustration of the algorithm to *agitate* a path.

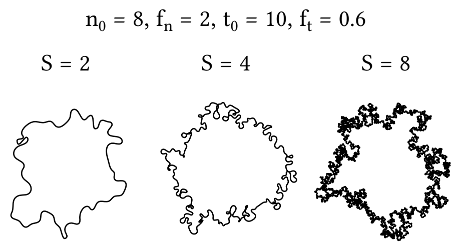
We can wrap this in a macro; to keep this simple here, we will assume that the path is a cycle (i.e a closed path), and that the number of points and the noise level at each step are given respectively by

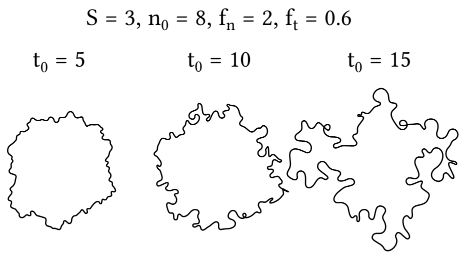
$$n_s = n_0 \times f_n^s, \quad t_s = t_0 \times f_t^s, \quad \text{for } s \geq 1, n_0\ t_0 \text{ known.}$$

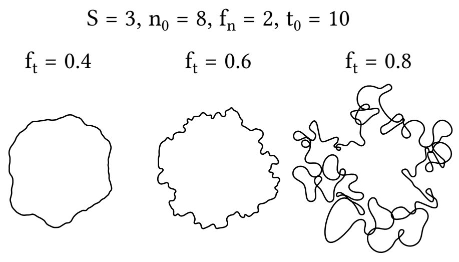
but of course, the macro can be modified for non cycled paths, and others expressions for n_s are t_s are possible. Here is our macro taking a path and others parameters as input and returning an *agitated* path (R):

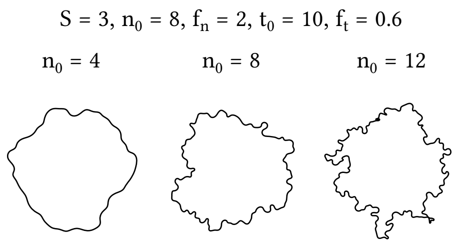
```
vardef agitate(expr thepath, S, n, fn, t, ft) =
 save R , nbpoints , noiselevel ;
 path R ; nbpoints := n ; noiselevel := t ;
 R := thepath ;
  for s=0 upto S :
   nbpoints := nbpoints * fn ;
   noiselevel := noiselevel * ft ;
   R := for i=1 upto nbpoints:
       point (i/nbpoints) along R
           randomized noiselevel ..
       endfor cycle ;
  endfor ;
  R
enddef ;
```

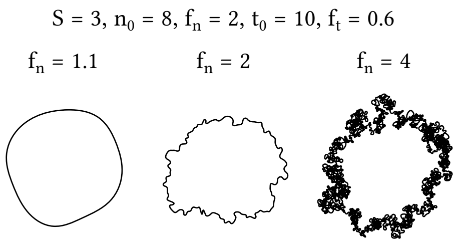
Please note that the variables `S` and `f_n` need to be reasonable... The processing time is exponential regarding these values, so caution is necessary in experimenting ! Some examples of realizations (starting with a circle again) :











Now we can go back to our objective. Of course we want random colors, but not completely random (it would not be pretty in general). It is useful to use palettes, so you can change easily from one set of colors to another set. Let's build a palette :

```
color MyPalette[] ; Ncolors := 5 ;
MyPalette[1] := (215/255,233/255,244/255);
MyPalette[2] := (234/255,187/255,076/255);
MyPalette[3] := (238/255,148/255,056/255);
MyPalette[4] := (199/255,066/255,033/255);
MyPalette[5] := (033/255,061/255,085/255);
```

To choose a random color, we need to choose an integer between 1 and `Ncolors`; a simple, useful and more general macro is made to choose a random integer in the interval $[mini, maxi]$:

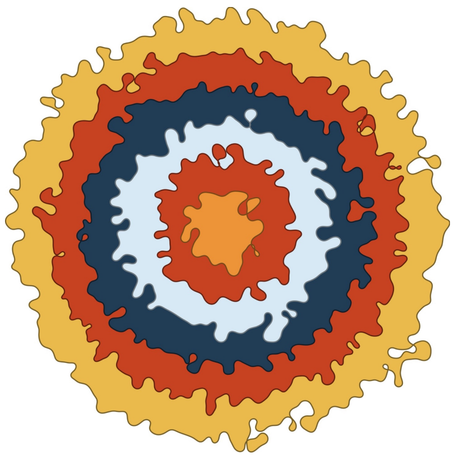
```
vardef ranint (expr mini , maxi ) =
  floor(uniformdeviate (maxi - mini +1 ) + mini)
enddef ;
```

For illustration, let's fill squares of random colors in our palette:

```
for i=1 upto 2:
 for j=1 upto 8:
  fill fullsquare randomized 0.1 scaled 15
    shifted (j*20,i*20)
    withcolor MyPalette[ranint(1,Ncolors)] ;
 endfor;
endfor;
```

We have now all the elements for our project. We just need to draw *agitated* circles decreasing in size, and fill them with different color. There is just one detail we need to pay attention to : the initial number of points used to agitate our circles should depend on the length of each path; so we add to the code in the loop `nzero := floor(arclength(P)/4.5);` the factor 4.5 is found by trial and error to obtain what we are looking for, and the function `floor` is too assure that `nzero` is an integer. In such code, we usually try to parametrize as much as possible, so it will be easier later to search optimal parameters.

```
randomseed := 1241 ; color AColor ;
path P, Q ;
NbCircles := 6 ; S := 8 ; f_n := 1.1 ;
tzero := 6 ; f_t := 0.80 ;
for c=NbCircles downto 1 :
 P := fullcircle scaled (c*30) ;
 AColor := MyPalette[ranint(1,Ncolors)] ;
 nzero := floor(arclength(P)*0.30);
 Q := agitate(P , S , nzero , f_n , tzero, f_t);
 eofill Q withcolor AColor ;
 draw Q withcolor .5[black,AColor];
endfor;
```



Now that we have succeeded, we can try to explore the possibilities of the algorithm. We can randomly generate some drawings selecting some parameters in a certain range. One could draw several pieces on the same page, but an easier way is to generate several pieces one piece per page. The complete code is below, and explanations follows.

```
\starttext
% Inclusions ------------------------------- ;
\startMPinclusions
 vardef agitate(expr apath, S, n, fn, t, ft) =
  save R , nbpoints , noiselevel ;
  path R ; nbpoints := n ; noiselevel := t ;
  R := apath ;
   for s=0 upto S :
    nbpoints := nbpoints * fn ;
    noiselevel := noiselevel * ft ;
    R := for i=1 upto nbpoints:
       point (i/nbpoints) along R randomized
        noiselevel .. endfor cycle ;
   endfor ;
   R
enddef ;

color MyPalette[] ; Ncolors := 5 ;
MyPalette[1] := (215/255,233/255,244/255);
MyPalette[2] := (234/255,187/255,076/255);
MyPalette[3] := (238/255,148/255,056/255);
MyPalette[4] := (199/255,066/255,033/255);
MyPalette[5] := (033/255,061/255,085/255);

vardef ranint (expr mini , maxi ) =
  floor(uniformdeviate (maxi - mini +1) + mini)
enddef ;
vardef ranuni (expr mini , maxi ) =
  uniformdeviate (maxi - mini) + mini
enddef ;
\stopMPinclusions

\dorecurse{16}{ %
% MP page -- ------------------------------- ;
 \startMPpage

 randomseed := 100*#1 ;

 path O, P , Q ;
 O := fullcircle scaled 200 ;
 color AColor ;
 NbCircles := ranint(3,15) ;
 S := ranint(2,3) ; fn := ranuni(1.1,1.5) ;
 tzero := ranuni(3,8) ; ft := ranuni(0.5,0.9) ;
 CurrentColor := ranint(1,Ncolors) ;
 for c=NbCircles downto 1 :
    P := O scaled (c/NbCircles) ;
    AColor := MyPalette[CurrentColor] ;
    nzero := floor(arclength(P)*ranuni(0.2,0.8));
    Q := agitate(P , S , nzero , fn , tzero, ft);
    eofill Q withcolor AColor ;
    draw Q withcolor .5[black,AColor];

    forever:
       AnotherColor := ranint(1,Ncolors) ;
       exitif CurrentColor <> AnotherColor ;
    endfor;
    CurrentColor := AnotherColor ;
endfor;

\stopMPpage
}
\stoptext
```

Some remarks about this code:

a. Lines 2-32: functions already defined are placed between:

```
\startMPinclusions
```

Definitions of functions here..

```
\stopMPinclusions
```

b. Line 37: this instruction is a way to keep track of what random seed is giving which piece. For exemple, if you do 100 drawings and you want to reproduce only the 90th page, the same code with `randomseed := 90*100;` will suffice. This is useful to debug sometimes too.

c. Lines 35-61: the code which produce each page is between the two braces inside the `\dorecurse{16}{}`; here 16 pages will be created:

```
\dorecurse{16}{
    \startMPpage
```

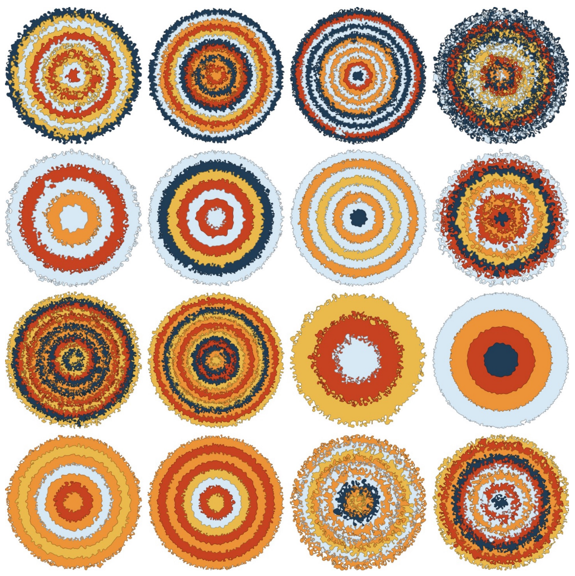
The code for the drawing itself

```
    \stopMPpage }
```

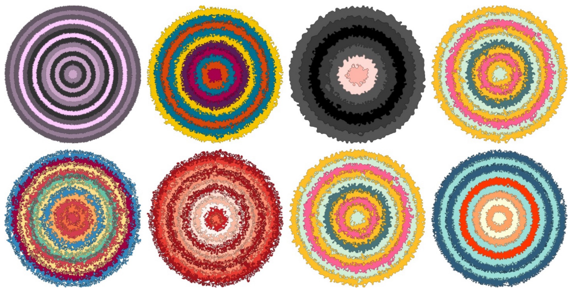
d. Lines 40,47: we have modified the previous code in order to have an algorithm able to manage a randomized path. At line 40, an original path `O` is created, here a circle, and this path is scaled down at line 47 at each step.

e. Lines 55-59: we have improved our previous algorithm by changing the color for each new circle, so two consecutive circles have not the same color; this is done in this loop by selecting a number in `(1,Ncolors)` until this result is different than the number of the current color.

Here are the 16 pages that the previous code produces:



Continuing to play with our project, it is now very easy to change the color palette. Imagine you have access to say 100 color palettes, we can choose randomly a palette (after the `randomseed` instruction on line 37), and here is a sample of what is possible (code not shown):



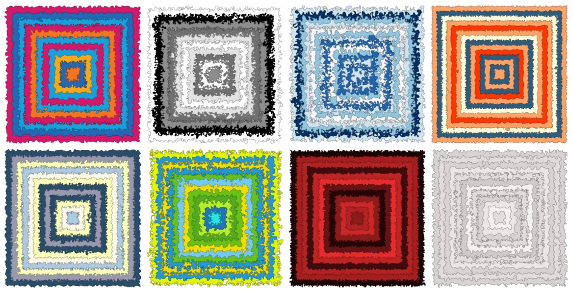
Now, continuing our experimentation, if we change

```
O := fullcircle scaled 200 ;
```

by

```
O := fullsquare scaled 200 ;
```

we could obtain something like this:

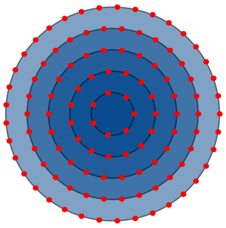


As you can see, once a drawing algorithm is made, it is quite easy to modify the parameters, the shapes, the colors... to explore the possibilities of the algorithm, and maybe discover an amazing creation resulting from the combination of a human idea and chance.

Project two : a sun

We can try to exploit our new function `agitate()` to create more lively pieces. We would like the piece to have the spirit of a cell, or a sun, something like this. So the strategy here is be to fill several agitated circles one above the other, like before, but this time, the border of the circles will be more chaotic, we will use more circles, and we will fill them with a transparent color.

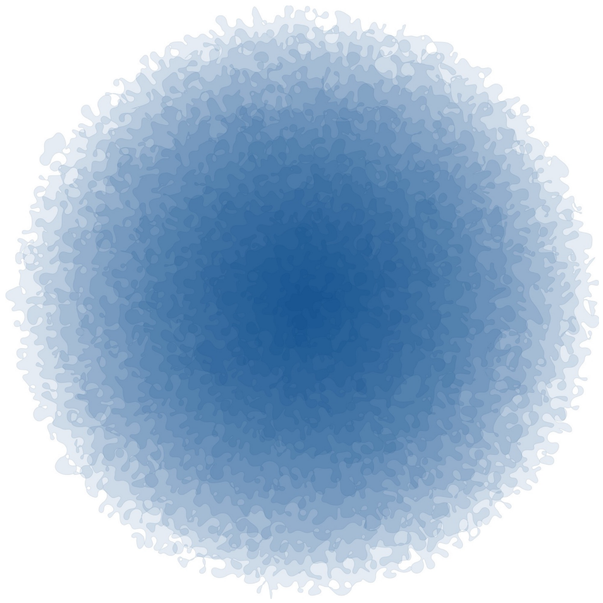
Here a sketch of the structure :



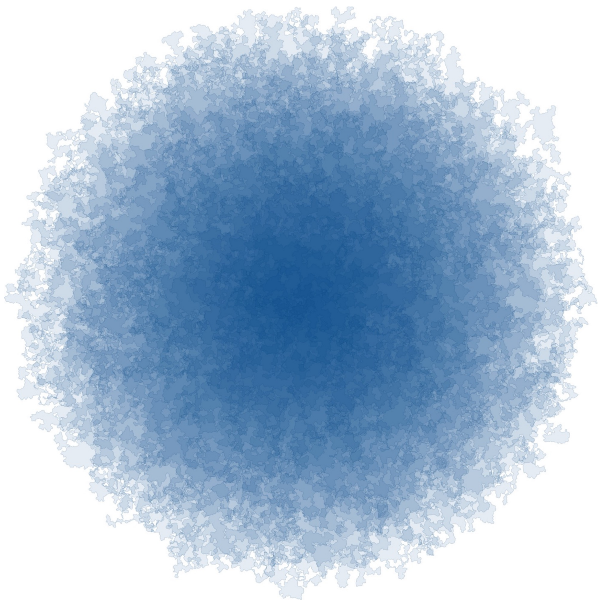
It is a matter of seconds to run this code:

```
vardef agitate(expr thepath, S, n, fn, t, ft) =
       (...)
enddef ;
path P , Q ;
color AColor ;
NbCircles := 20; S := 1; nzero:= 10; fn := 1.3;
tzero := 5; ft := 0.8;

% with \usecolors[crayola] ;
AColor := \MPcolor{MidnightBlue};

for c=NbCircles downto 1 :

 P := fullcircle scaled (c*10.5) ;
 nzero := floor(arclength(P)*0.5);
 Q := agitate( P , S , nzero , fn , tzero, ft);
 eofill Q
   withcolor transparent(1,2/NbCircles,AColor);
 draw Q  withpen pencircle scaled 0.1
   transparent(1,4/NbCircles,.90[black,AColor]);

endfor;
```

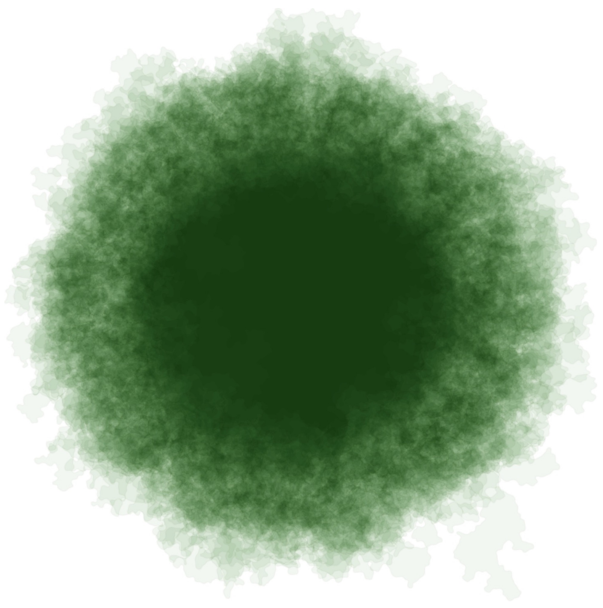
and to obtain this result :



But the borders are too smooth. So increasing the number of steps of the agitate() function so 15, after approximately one hour, we have this result :



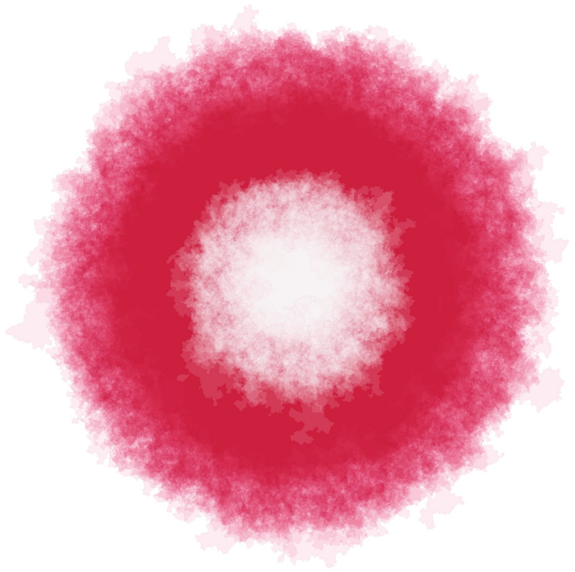
Increasing the number of circles to 40, and changing the color (to explore), but this time every circle will have the same size, we obtain this nice blurry effect :



It is tempting to try a donut by simply adding in the center a series `nbrep := 25;` of white circles. Caution : this code is very computationally intensive. if you try it, reduce the value of S and nbrep, and increase it slowly.

The code look like this :

```
path P , Q ;
color AColor ;
S := 18; nzero:= 10; fn := 1.3; tzero := 5;
ft := 0.8;

AColor := \MPcolor{Razzmatazz} ;
P := fullcircle scaled 60 ;
nzero := floor(arclength(P)*0.5);
nbrep := 25;
for rep = 1 upto nbrep:
   Q := agitate(P, S, nzero, fn, tzero, ft);
   eofill Q
     withcolor transparent(2,2/nbrep,AColor);
   draw Q  withpen pencircle scaled 0.1
     transparent(9,1/nbrep,.90[black,AColor]);
endfor;

P := fullcircle scaled 22 ;
nzero := floor(arclength(P)*0.5);
for rep = 1 upto nbrep:
   Q := agitate(P, S, nzero, fn, tzero, ft);
   eofill Q
     withcolor transparent(3,3/nbrep,white);
   draw Q  withpen pencircle scaled 0.1
     transparent(9,1/nbrep,.90[black,AColor]);
endfor;
```

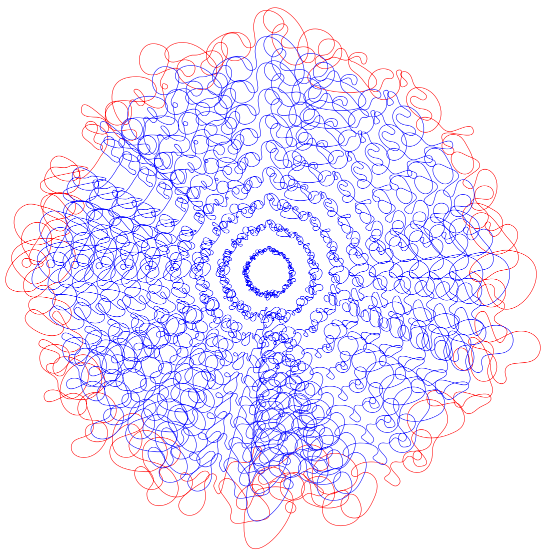


This last piece is very satisfying. One has an impression of bubbling, of life, like a gaseous planet. Satisfied, let us stop the exploration here!

Project three : a fabric

Finally we will use the same function, in a very different way, and use a simple technique to create a form that looks like a fabric. The first step is to create an *agitated* circle, a simple one this time, and scale it down a few times (here 10 times), to obtain this sketch :

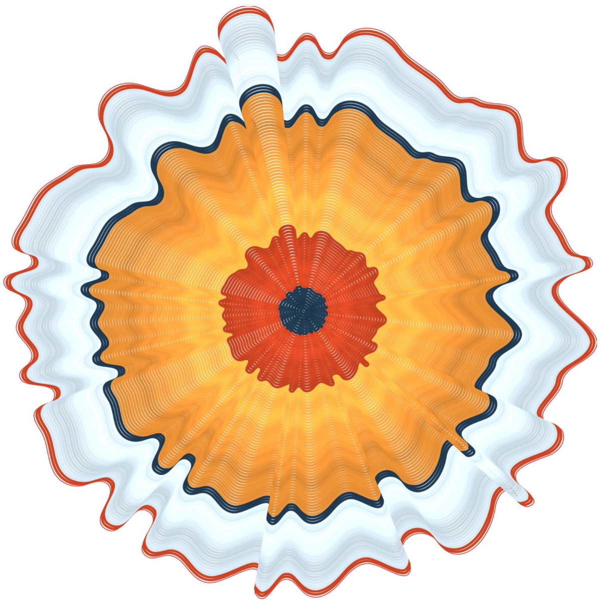
```
path P, Q;
S := 1; nzero:= 10; fn := 1.2; tzero := 15;
ft :=0.8;
P := fullcircle scaled 170 ;
nzero := floor(arclength(P)*0.5);
Q := agitate( P , S , nzero , fn , tzero, ft);
draw Q  withpen pencircle scaled 0.2
  withcolor red;
path R ; nblines := 10 ;
for i=1 upto nblines:
  R := Q scaled ((nblines-i)/nblines) ;
  draw R withpen pencircle scaled 0.2
    withcolor blue;
endfor;
```



Now we will do the same strategy, increasing the number of lines, and changing color over time, randomly of course :

```
randomseed := 3354 ;
S :=1; nzero :=10; fn :=1.1; tzero :=15; ft :=0.8;

P := fullcircle scaled 180 ;
nzero := floor(arclength(P)*0.12);
Q := agitate( P , S , nzero , fn , tzero, ft) ;
draw Q withpen pencircle scaled 0.2 withcolor red;
path R ;
color CurrentColor , RealColor;
CurrentColor := MyPalette[ranint(1,Ncolors)];
nblines := 100 ;
```

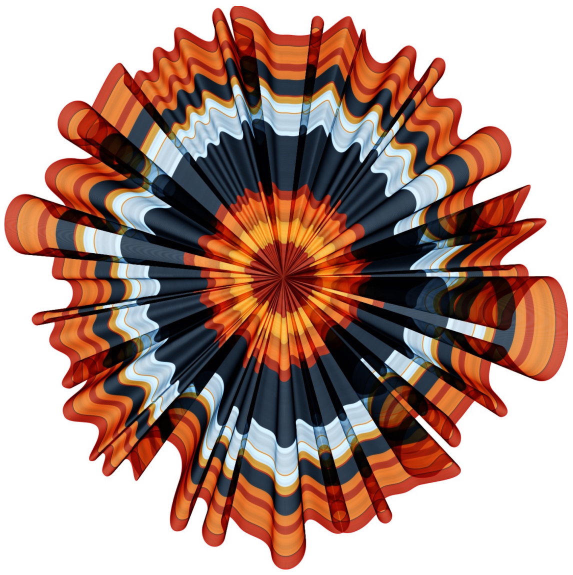
```
for i=1 upto nblines:
  CurrentColor := CurrentColor
    randomized(0.95,1.05);
  R := Q scaled ((nblines-i)/nblines) ;

  if uniformdeviate(1) < 0.08:
    CurrentColor := MyPalette[ranint(1,Ncolors)];
  fi;
  RealColor := CurrentColor ;
  draw R withpen pencircle scaled .8
    withcolor RealColor;
endfor;
```



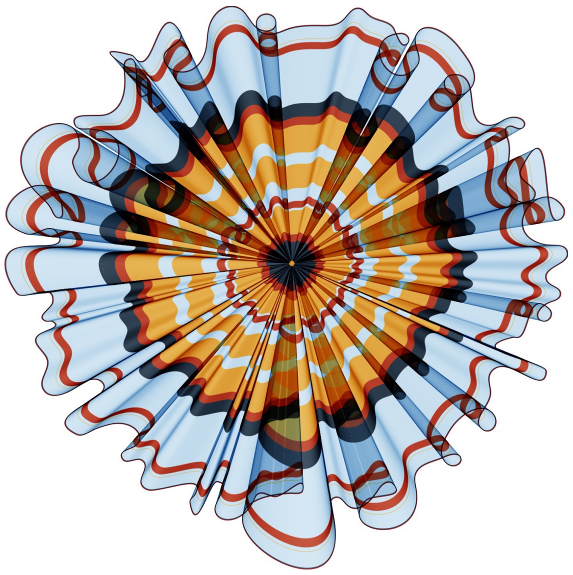
And finally, we increase dramatically the number of lines, randomized the same color en epsilon at each line, and add transparency to create relief :

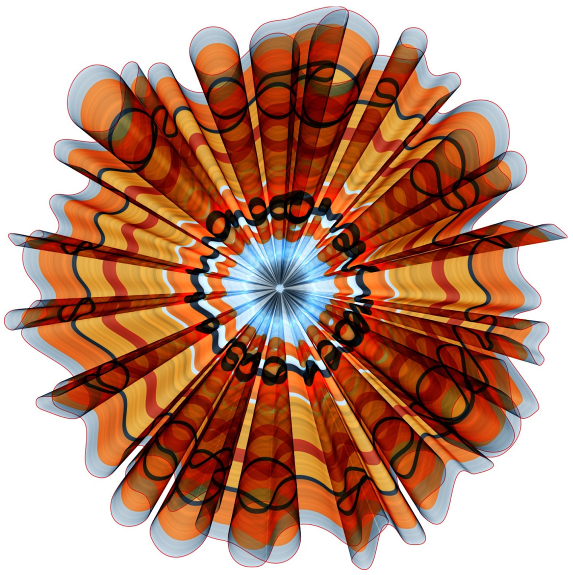
```
randomseed := 2562 ;
S := 1 ; fn := 1.05 ; tzero := 20 ; ft := 0.8 ;

P := fullcircle scaled 180 ;
nzero := floor(arclength(P)*0.18);
Q := agitate( P , S , nzero , fn , tzero, ft) ;
draw Q  withpen pencircle scaled 0.2
  withcolor red;
path R ;
color CurrentColor , RealColor;
CurrentColor := MyPalette[ranint(1,Ncolors)];
nblines := 750 ;

for i=1 upto nblines:
  CurrentColor :=
    CurrentColor randomized(0.98,1.02);
  R := Q scaled ((nblines-i)/nblines) ;
```

```
  if uniformdeviate(1) < 0.025:
   CurrentColor :=
     MyPalette[ranint(1,Ncolors)];
  fi;
  RealColor := CurrentColor ;
  draw R withpen pencircle scaled .2
   withcolor transparent(2,.8,RealColor);
endfor;
```



Here are other examples changing the random seed :





And a last example decreasing the number of points `nzero`, and increasing a bit the turbulence `tzero` :

It just seems extraordinary to me that a few simple lines of code, using such elementary functions, can give such rich and varied results.

Technical addendum

Hans Hagen made a remark that the agitate code could be improved; in the `agitate()` macro we can find this instruction :

```
R := for i=1 upto nbpoints:
  point (i/nbpoints) along R
    randomized noiselevel ..
  endfor cycle ;
```

At each step of the loop `for i=1 upto nbpoints`, the instruction `along` is used, which means that the function `arclength` is called `nbpoints` times, and this function takes time. As the length of the path `R` does not change, a better way to do is to calculate this length only one time outside the loop :

```
rlength := (arclength R) / nbpoints;
R := for i=1 upto nbpoints:
  (point (arctime (i * rlength) of R) of R)
    randomized noiselevel ..
  endfor cycle ;
```

This improvement speed up the processing time significantly. A second improvement is possible, using the “double” mode improve the processing time. So all the code of this paper can be adapted like this:

```
\startMPinclusions{doublefun}
       (...)
\stopMPinclusions
\startMPpage[instance=doublefun]
       (...)
\stopMPage
```

These two modifications together speedup processing time by a factor 5 on project 2.

Conclusion

We have presented in this article how MetaFun can be used to do generative art. The `randomized` operator is simple but powerful, and can be used on several types of objects. Moreover, macros are easy to define in order to introduce new creation tools, as we did for the `agitate()` function. The ability to draw one result per page is very useful when producing a large number of results from a given algorithm, and the vector nature of the PDF output makes each drawing easily scalable and printable on a large scale. All of this makes MetaFun definitely a powerful tool, allowing to create drawings or artworks with few limitations.

Bibiography

Galenter, Phillip. 2013. *What is Generative Art? Complexity Theory as a Context for Art Theory*. In *GA2003 – 6th Generative Art Conference.*

I would like to thank Frans Goddijn for his helpful comments on the article, as well as Hans Hagen and Taco Hoekwater for their enlightenment on technical aspects of MetaFun.

Fabrice Larribe