MATHEMATICA
& Wolfram Language

How to generate a random snowflake?

Asked 10 years, 9 months ago Modified 10 months ago Viewed 35k times

153

'Tis the season... And it's about time I posed my first question on Mathematica Stack Exchange. So, here's an holiday quest for you Graphics (and P-Chem?) gurus.

What is your best code for generating a (random) snowflake in Mathematica? By random I mean with different shapes that will mimic the diversity exhibited by *real* snowflakes. Here's a link to have an idea: http://www.its.caltech.edu/~atomic/snowcrystals/ , more specifically here are the different types of snowflakes: http://www.its.caltech.edu/~atomic/snowcrystals/class/class.htm .

Physics-based answers are to be preferred, but graphics only solutions are also welcome. There already is a thread on generating a snowfall, here: How to create animated snowfall? and one of the posts addresses the problem of generating snowflake-like elements. In the snowfall post, though, emphasis is on efficient generation of 'snowlike' *ensembles*. The purpose of this question (apart from having some 'seasonal' fun) is to create graphics that details the structure of a single snowflake. Efficiency is not the primary issue here: beauty is. A very detailed snowflake rendering could even take several minutes of computer power, thus making it unsuitable to incorporate into a snowfall simulation.

Here we are trying to generate a single snowflake (possibly with different parameters to tune its shape), the more *realistic*, the better. Three dimensional renderings, for adding translucency and colors are also welcome. Unleash your fantasy, go beyond the usual fractals!

And if your fantasy is momentarily faltering, as Silvia pointed out in a comment below, on this website http://psoup.math.wisc.edu/Snowfakes.htm you can find a lot of information - and even a C program for the *Gravner-Griffeath 2D Snowfake Simulator* - on how to generate 'virtual snowflakes', even in 3D (have a look at the pdf files: "*Modeling Snow Crystal Growth*" I, II and III).

graphics **random** **simulation** **generative-art**

Share Improve this question Follow

edited Dec 15, 2023 at 17:38

user64494
28.1k 4 28 56

asked Dec 24, 2013 at 13:59

Peltio
5,536 3 28 27

1 I don't want to kill the fun but.., did you see this? – Öskå Dec 24, 2013 at 14:28

@Öskå, yes, I was aware of this method of generating snowflake-like fractals. And it is one welcome method, of course, but I wish to find more 'physically' oriented answers. Also, the more 'real-life', the better. (This is an holiday post, let's have some fun). – Peltio Dec 24, 2013 at 15:05

Nice fractal art here deviantart.com/morelikethis/317568361?offset=25#skins – Dr. belisarius Dec 24, 2013 at 15:05

1 Have you seen this *Gravner-Griffeath Snowfakes*? I think they are kind of what you're looking for. – Silvia Dec 25, 2013 at 11:11

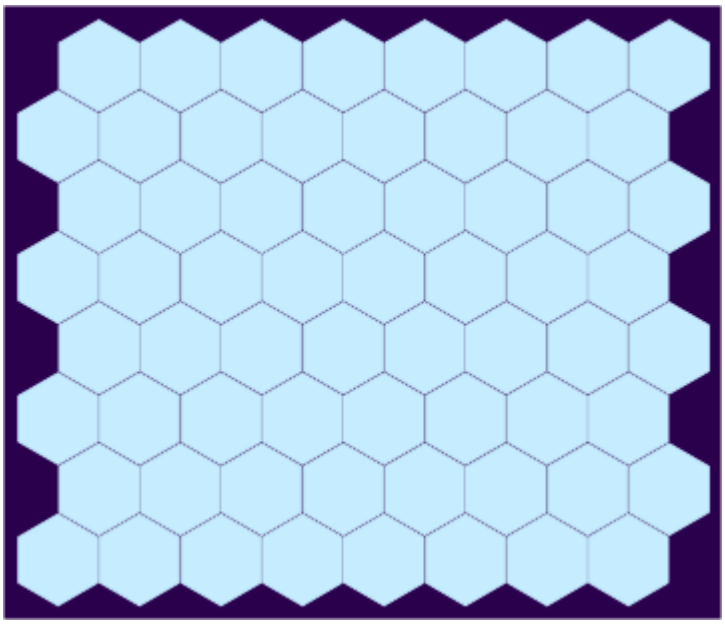
1 On Christmas day, this W `Integer` Wonderland post has 5 answers, 50 votes and 5 thousands views. It's fivelous! :-) I believe I will wait until New Year's day to accept an answer. – Peltio Dec 25, 2013 at 16:45

8 Answers

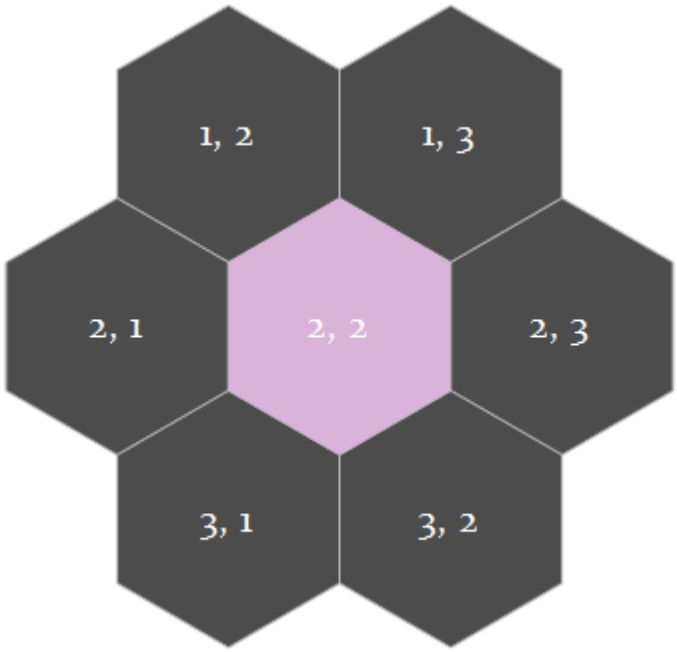
Sorted by: Highest score (default)

190

I did a very simple (in fact over-simple) snowflake simulator with `CellularAutomaton` years before. It's based on the hexagonal grid:



and range-1 rules:



Initial code

First we'll need some functions to display our snowflakes:

```
Clear[vertexFunc]
vertexFunc = Compile[{{para, _Real, 1}},
   Module[{center, ratio},
    center = para[[1 ;; 2]];
    ratio = para[[3]];
    {Re[#], Im[#]} + {{1, -(1/2)},{0, Sqrt[3]/2}}.Reverse[{-1, 1} center + {3, 0}] & /@
                (ratio 1/Sqrt[3] E^(I π/6) E^(I Range[6] π/3))
    ],
   RuntimeAttributes -> {Listable}, Parallelization -> True,
   RuntimeOptions -> "Speed"
   (*, CompilationTarget->"C"*)];

Clear[displayfunc]
displayfunc[array_, ratio_] := Graphics[{
   FaceForm[{ColorData["DeepSeaColors"][3]}],
   EdgeForm[{ColorData["DeepSeaColors"][4]}],
   Polygon[vertexFunc[Append[#, ratio]] & /@ Position[array, 1]]
   }, Background -> ColorData["DeepSeaColors"][0]]
```

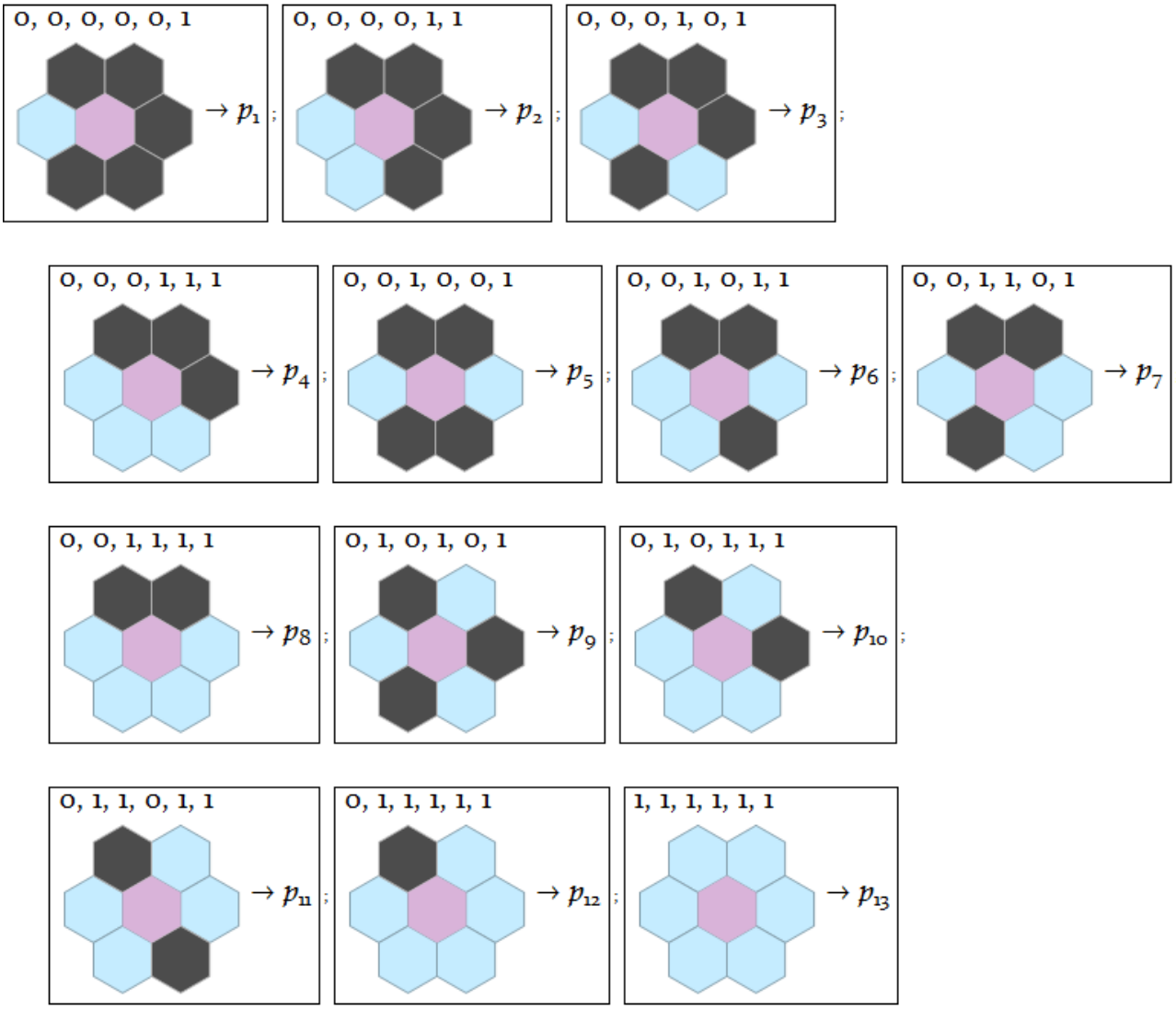
Main body

Consider 0/1 bistable states for every node on the hexagonal grid, where 0 stands for empty nodes and 1 stands for frozen nodes. Then, excluding all-zero case, there are 13 possible arrangements on the 6 vertices of a hexagon(those who are identical under rotation and reflection are considered as the same arrangement):

```
stateSet = Tuples[{0, 1}, 6] // Rest;
gatherTestFunc = Function[lst, Sort[RotateLeft[lst, # - 1] & /@ Flatten[Position[lst,
1]]]];
```

```
stateClsSet = Sort /@ Gather[stateSet, gatherTestFunc[#1] == gatherTestFunc[#2] &];
stateClsSetHomogeneous = ArrayPad[#, {{0, 6 - Length@#}, {0, 0}}] & /@ stateClsSet;
```

And the simplest physical rule might be linking different arrangement to different probability of freezing(from 0 to 1) or melting(from 1 to 0).



Those 26 probabilities, `pFreeze` and `pMelt`, can be determined by some serious physical models, or can be chosen randomly just for fun. Then they can be used to establish rule function for `CellularAutomaton`:

```
Clear[ruleFunc2Comp]
ruleFunc2Comp = With[{
                     stateClsSetHomogeneous = stateClsSetHomogeneous,
                     seedSet = RandomInteger[{0, 1000}, 1000],
                     pFreeze = {1, 0.2, 0.1, 0, 0.2, 0.1, 0.1, 0, 0.1, 0.1, 1, 1, 0},
                     pMelt = {0, 0.7, 0.5, 0.5, 0, 0, 0, 0.3, 0.5, 0, 0.2, 0.1, 0}
                    },
   Compile[{{neighborarry, _Integer, 2}, {step, _Integer}},
           Module[{cv, neighborlst, cls, rand},
                  cv = neighborarry[[2, 2]];
                  neighborlst = {#[[1, 2]], #[[1, 3]], #[[2, 3]], #[[3, 2]], #[[3, 1]],
#[[2, 1]]}&[neighborarry];
```

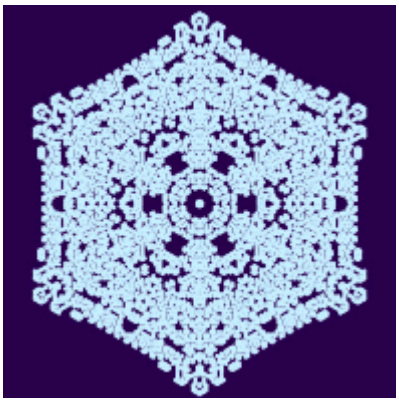
```
                    If[Total[neighborlst] == 0, cv,
                       cls = Position[stateClsSetHomogeneous, neighborlst][[1, 1]];
                       SeedRandom[seedSet[[step + 1]]];
                       rand = RandomReal[];
                       Boole@If[cv== 0, rand < pFreeze[[cls]], rand > pMelt[[cls]]]
                      ]
                    ],
             RuntimeAttributes -> {Listable}, Parallelization -> True, RuntimeOptions ->
"Speed"(*,CompilationTarget -> "C"*)
            ]
   ];
```

Apply `ruleFunc2Comp` on some initial state for some steps:

```
dataSet = Module[{rule,
                  initM = {{{0, 0, 0}, {0, 1, 0}, {0, 0, 0}}, 0},
                  rspec = {1, 1},
                  tmin = 0, tmax = 50, dt = 1
                 },
                 rule = {ruleFunc2Comp, {}, rspec};
                 CellularAutomaton[rule, initM, {{tmin, tmax, dt}}]
                ];
```

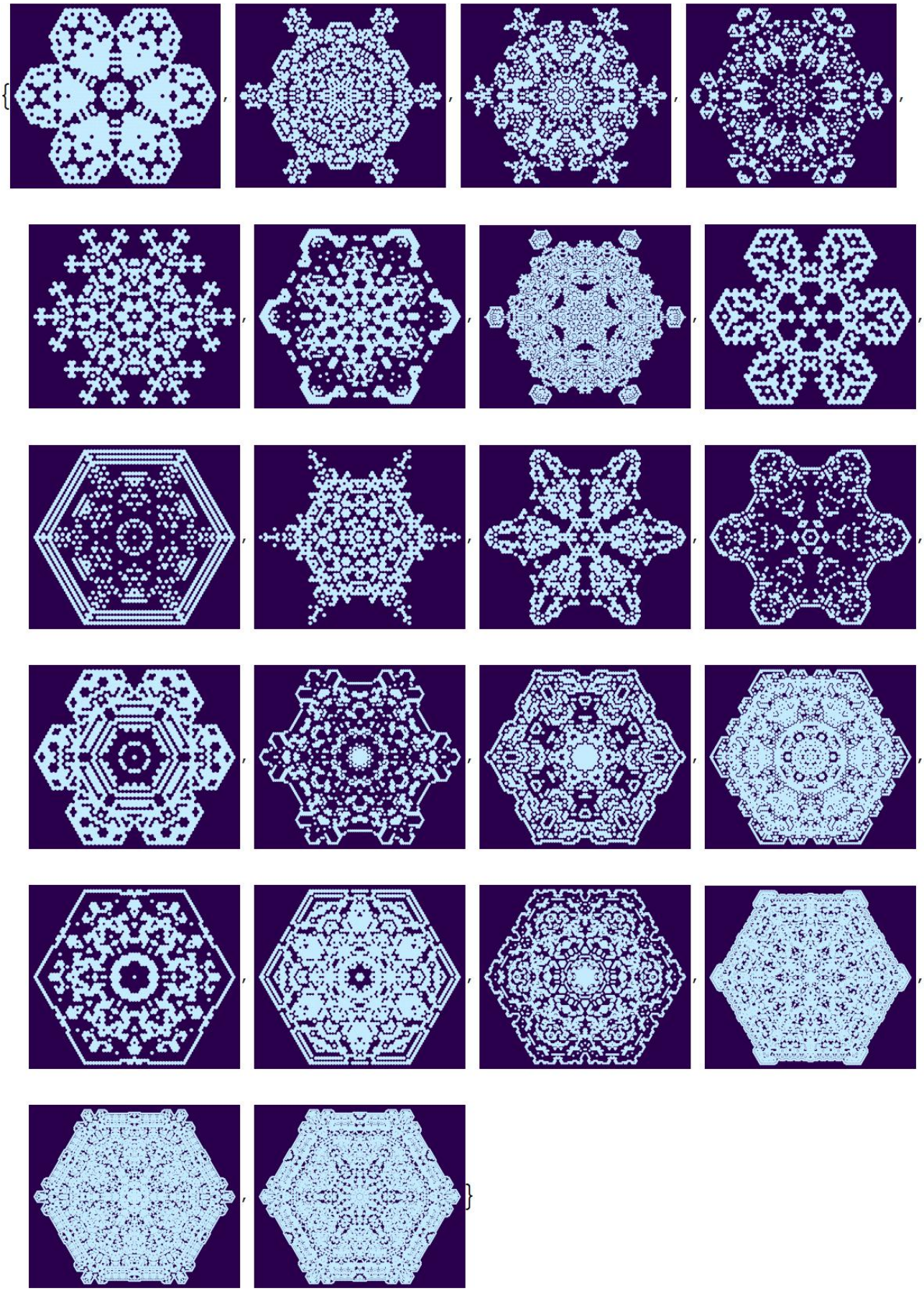
You can see how the snowflake grows:

```
Manipulate[
    Rotate[displayfunc[dataSet[[k]], .99], 90°],
    {k, 1, Length[dataSet], 1}]
```



More snowflakes

Some other examples generated with different `pFreeze`, `pMelt` and `tmax`:



Share Improve this answer Follow

edited Dec 24, 2013 at 22:04

answered Dec 24, 2013 at 16:11

Silvia

27.6k 3 85 165

3 This is fantastic! Also, this is the fastest I've seen an answer get to +10! Well done. – rm -rf ♦ Dec 24, 2013 at 16:32

2 @Peltio I only picked those parameters randomly. But there can be some really complex physical model. Like *The physics of snow crystals* by Kenneth G Libbrecht, Rep. Prog. Phys. 68 (2005) 855–895, etc. – Silvia Dec 24, 2013 at 16:46

4 @rm-rf Time for reddit? – Leonid Shifrin Dec 24, 2013 at 20:12

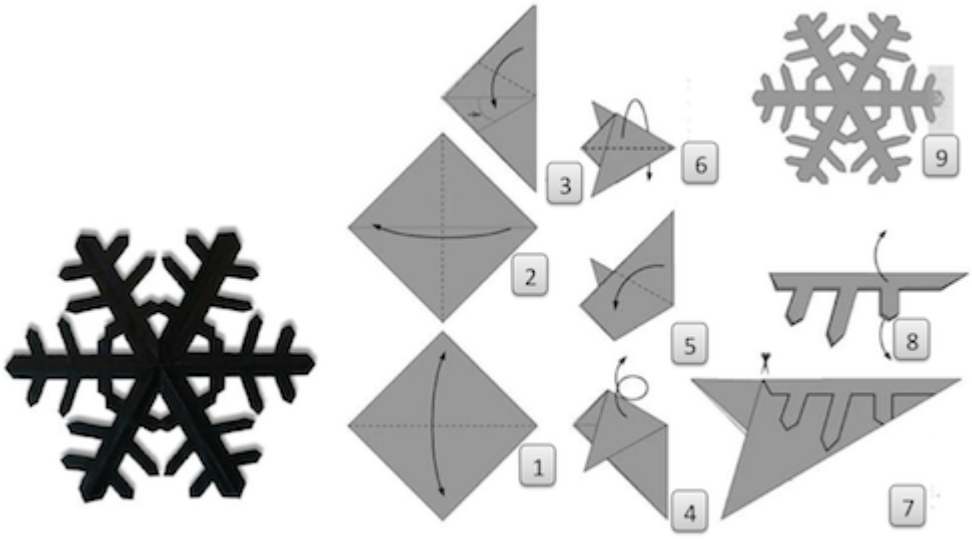
14 Just made an account to say that you guys are crazy. – Saturn Dec 24, 2013 at 23:16

2 Prepend `inHotArgentinaQ[] := Apply[And, {-40, -70} < FindGeoLocation[] < {-20, -55} // Thread] . If[inHotArgentinaQ[], WolframAlpha[ "image sun", {{"Image:AstronomicalData", 1}, "Content"}], #]&` – Rojo Dec 25, 2013 at 17:43

81

========== **update** ==========

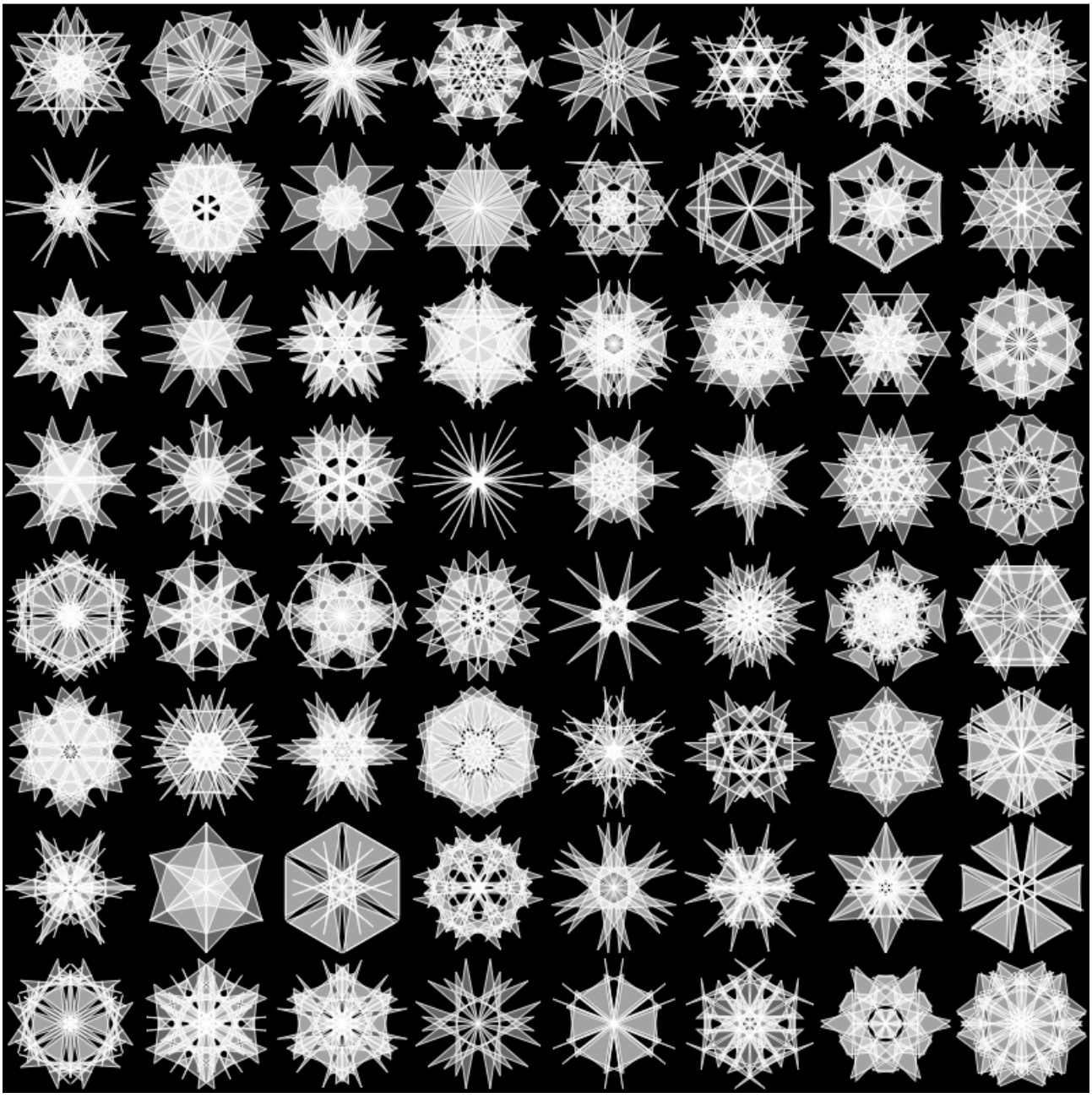
Remember guys how we can cut out a snowflake from a sheet of paper carving 12th folded part? Like the image below.



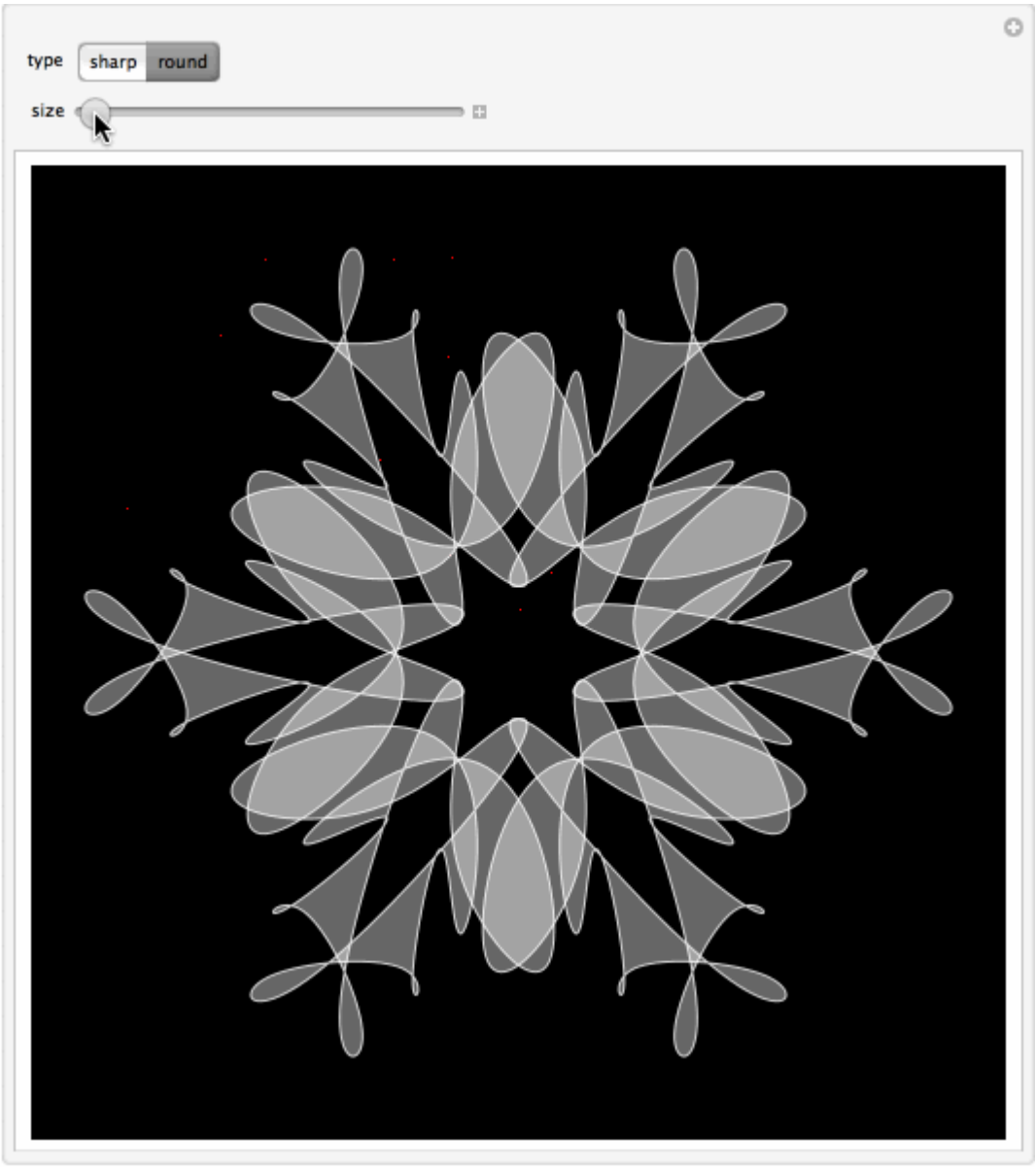
So I decided to **write an app** to imitate the process. It also can be used to make random snowflakes (similar to to @bill s' but with reflection to imitate real cutting paper process and reflective symmetry of snowflakes). App and random collection are below.

```
snow[pt_] := Graphics[

  {EdgeForm[Directive[White, Opacity[.8]]],
   FaceForm[Directive[White, Opacity[.4]]],
   Polygon[
    Outer[#1.#2 &,
     Table[RotationMatrix[a], {a, 0, 2 Pi - Pi/3, Pi/3}],
     Join[Map[ReflectionMatrix[{1, 0}].# &, #], #] &@
      Join[{{0, 0}}, pt],
```

```
        1]]}

   , Background -> Black, ImageSize -> 100 {1, 1}]

Grid[Partition[ParallelTable[
   snow[RandomReal[{-1, 1}, {RandomInteger[{3, 9}], 2}]],
   {64}], 8], Spacings -> {0, 0}]
```



This is **preview of the app**:

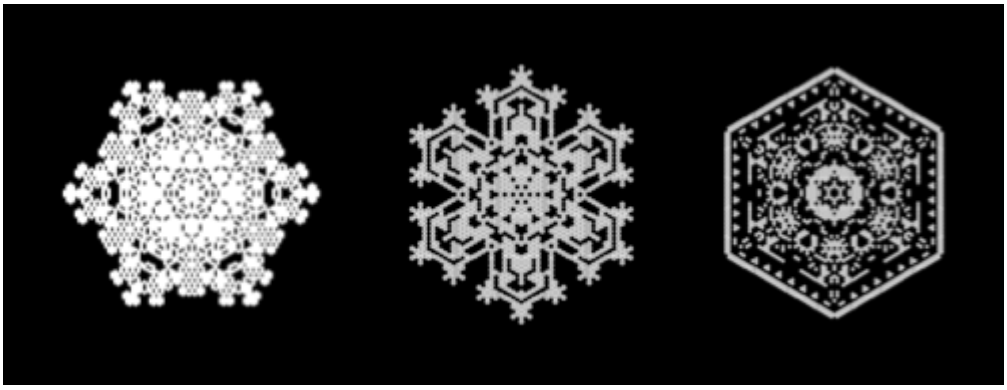


========== older versions ==========

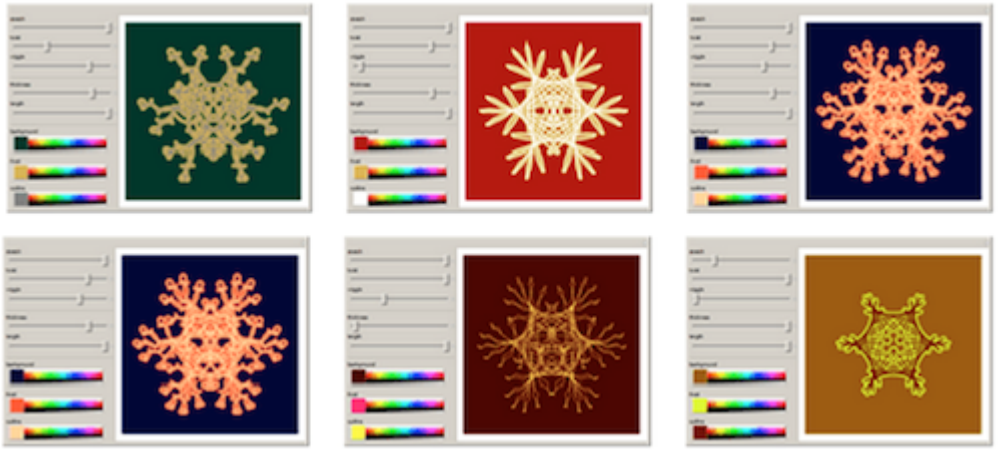
@Silvia did a beautiful job, especially at design and explanation. I still want to point out similar things with Cellular Automata (for the sake of a bit alternative implementation) and a bit different things in general.

1) Ed Pegg's Demo and related competition:

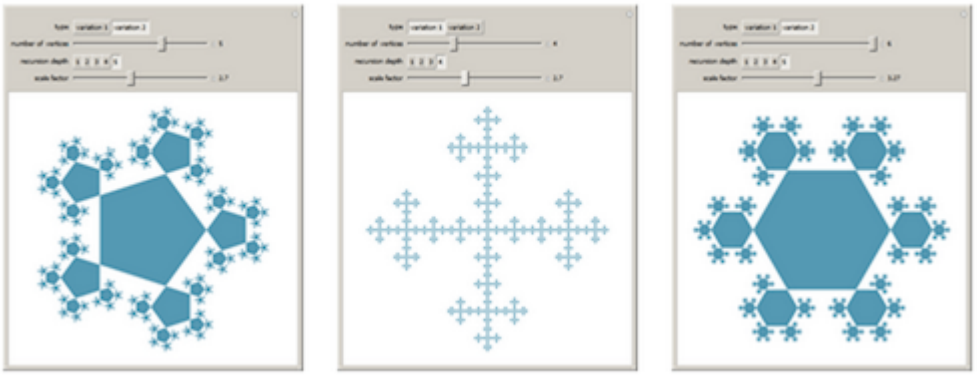
- Snowflake-Like Patterns
- Two Hundred Thousand Snowflake Greetings to You and Yours



2) Spinoff of Herbert W. Franke: Parametric Snowflake Design

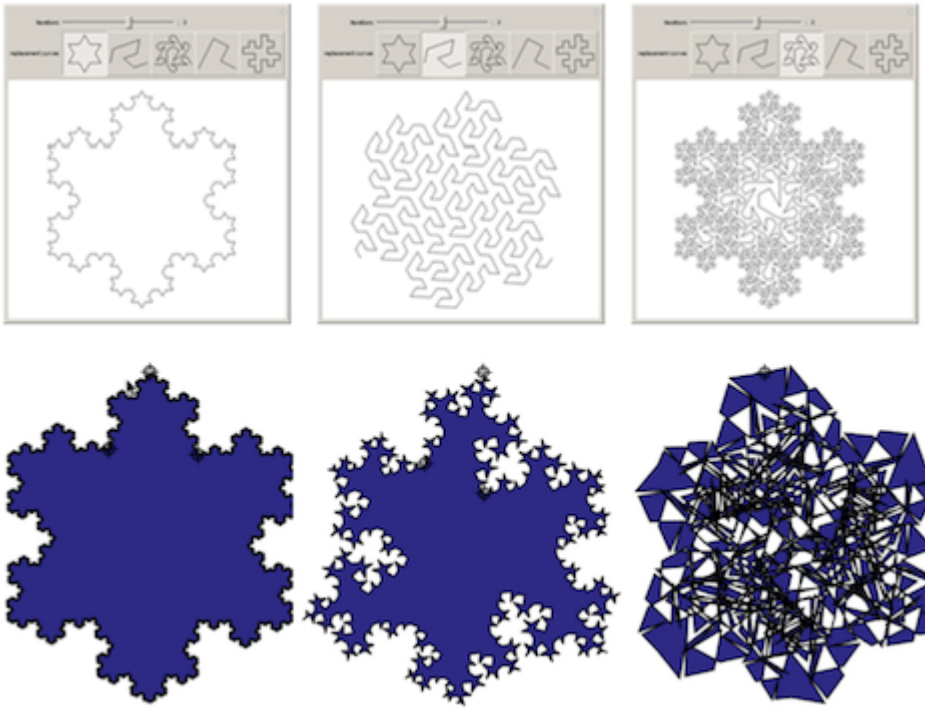


3) Croucher & Weisstein's n-Flakes



4) Some awesome Koch's:

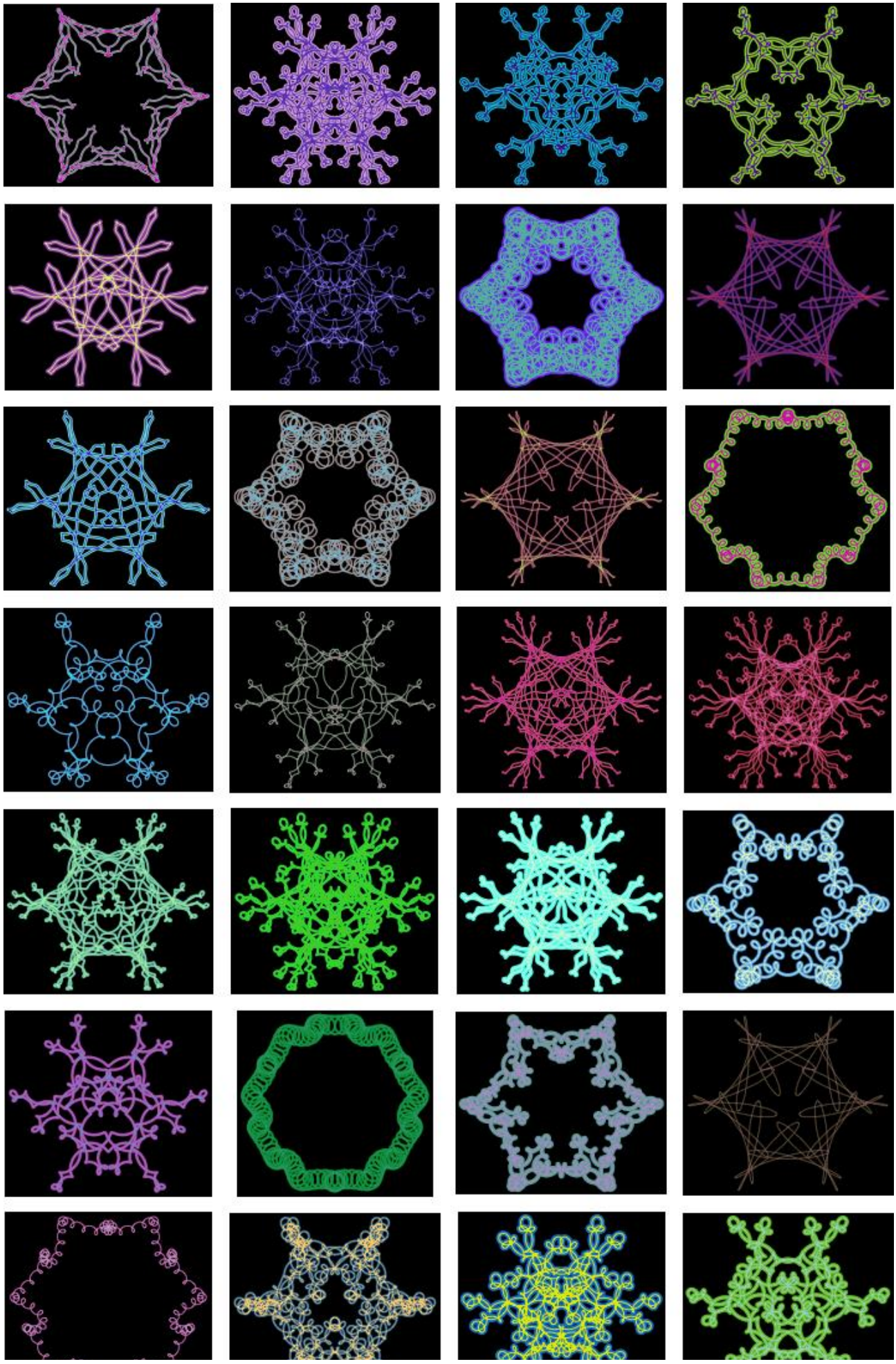
- Create Alternative Koch Snowflakes
- Fractal Curves

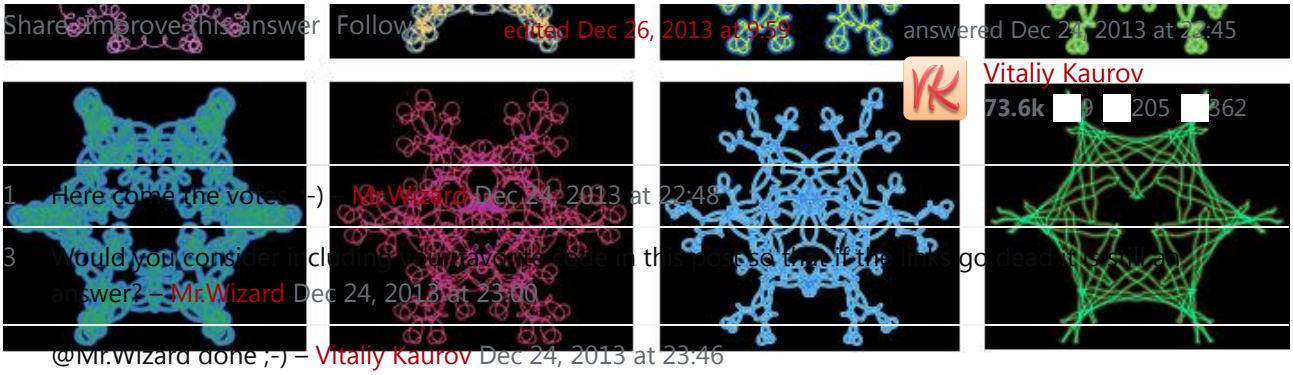


5) Some more

As Mr. Wizzard asked in the answer I am including the (modified) code for 2) due to its simplicity and beauty:

```
x1[a_, b_, c_, t_] :=  Sin[.5 t] - a Sin[b t]*Cos[t] - .1 c Sin[10 b t];
y1[a_, b_, c_, t_] :=  Cos[.5 t] - a Sin[b t]*Sin[t] - .1 c Cos[10 b t];

GraphicsGrid[Partition[ParallelTable[
   With[{
     a = RandomReal[{-1.5, 1.5}],
     b = RandomInteger[{3, 15}],
     c = RandomReal[{0, 1.5}],
     clr1 = Black,
     clr2 = RGBColor @@ RandomReal[1, 3],
     clr3 = RGBColor @@ RandomReal[1, 3],
     thick = RandomReal[{.04, .5}],
     tm = 1},
    ParametricPlot[
     Evaluate[{{x1[a, b, c, t], y1[a, b, c, t]}, {x1[a, b, c, t],
        y1[a, b, c, t]}}], {t, 0, tm 4 \[Pi]},
     PlotStyle -> {{clr2, Thickness[0.001` + 0.05` thick]}, {clr3,
        Thickness[0.001` + 0.01` thick]}}, Axes -> False,
     PlotPoints -> 200, PlotRange -> All, Background -> clr1]]
   , {n, 32}], 4], ImageSize -> 600]
```





Share Improve this answer Follow edited Dec 26, 2013 at 9:59 answered Dec 24, 2013 at 22:45

Vitaliy Kaurov
73.6k 9 205 362

1 Here come the votes ;-) – Mr.Wizard Dec 24, 2013 at 22:48

3 Would you consider including your favorite code in this post so that if the links go dead this is still an answer? – Mr.Wizard Dec 24, 2013 at 23:00

@Mr.Wizard done ;-) – Vitaliy Kaurov Dec 24, 2013 at 23:46

1 Wow, the parametric method is GREAT! – Silvia Dec 24, 2013 at 23:49

4 some of those remind me of snowflakes i've seen on other planets. ah the memories – amr Dec 26, 2013 at 7:01

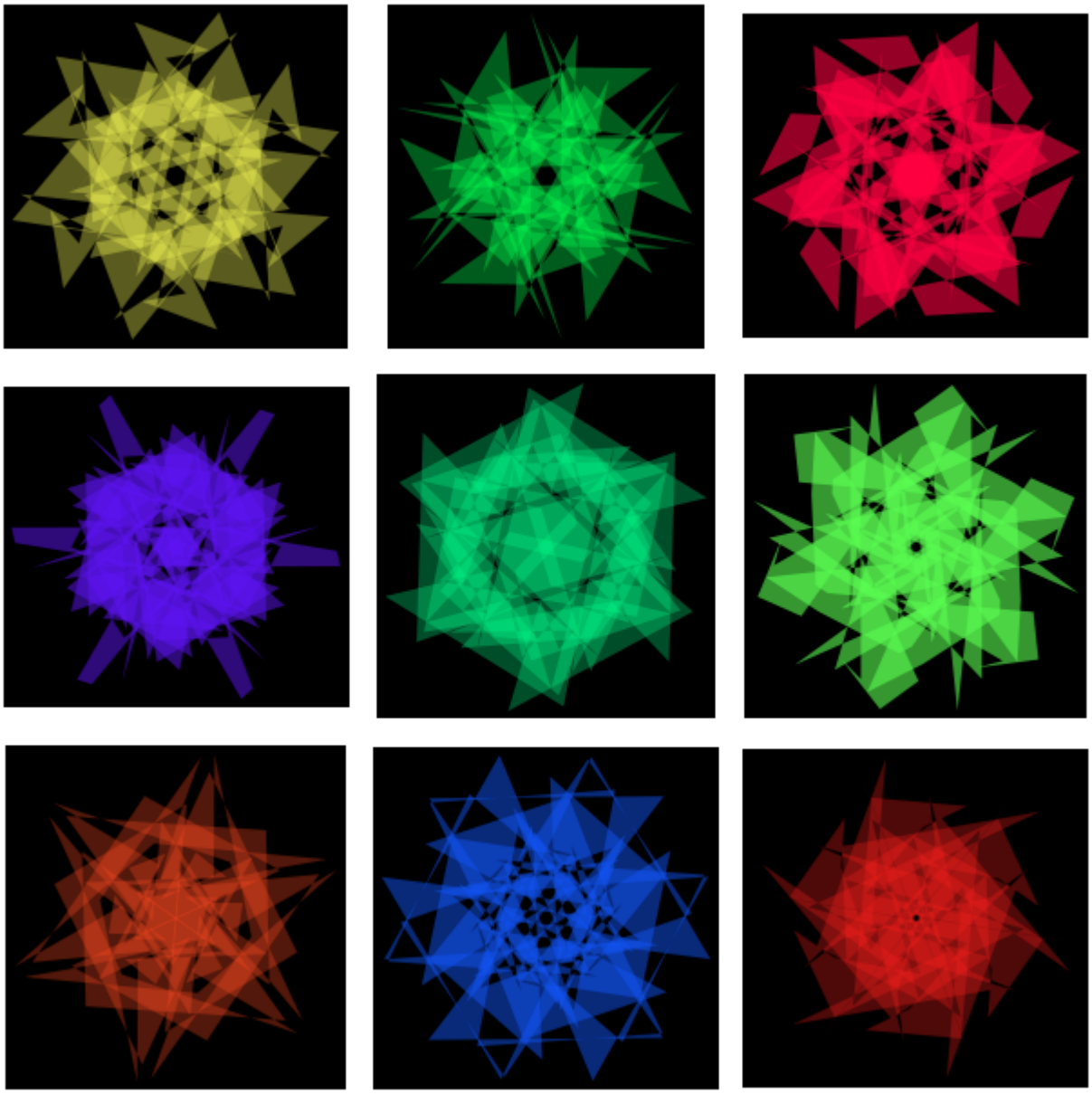
32

Here is a simple method that begins with an n-sided polygon (defined by the n points in `tab`), then rotates the polygon and superimposes it six times to achieve the six-fold symmetry. The `makeFlake` function is:

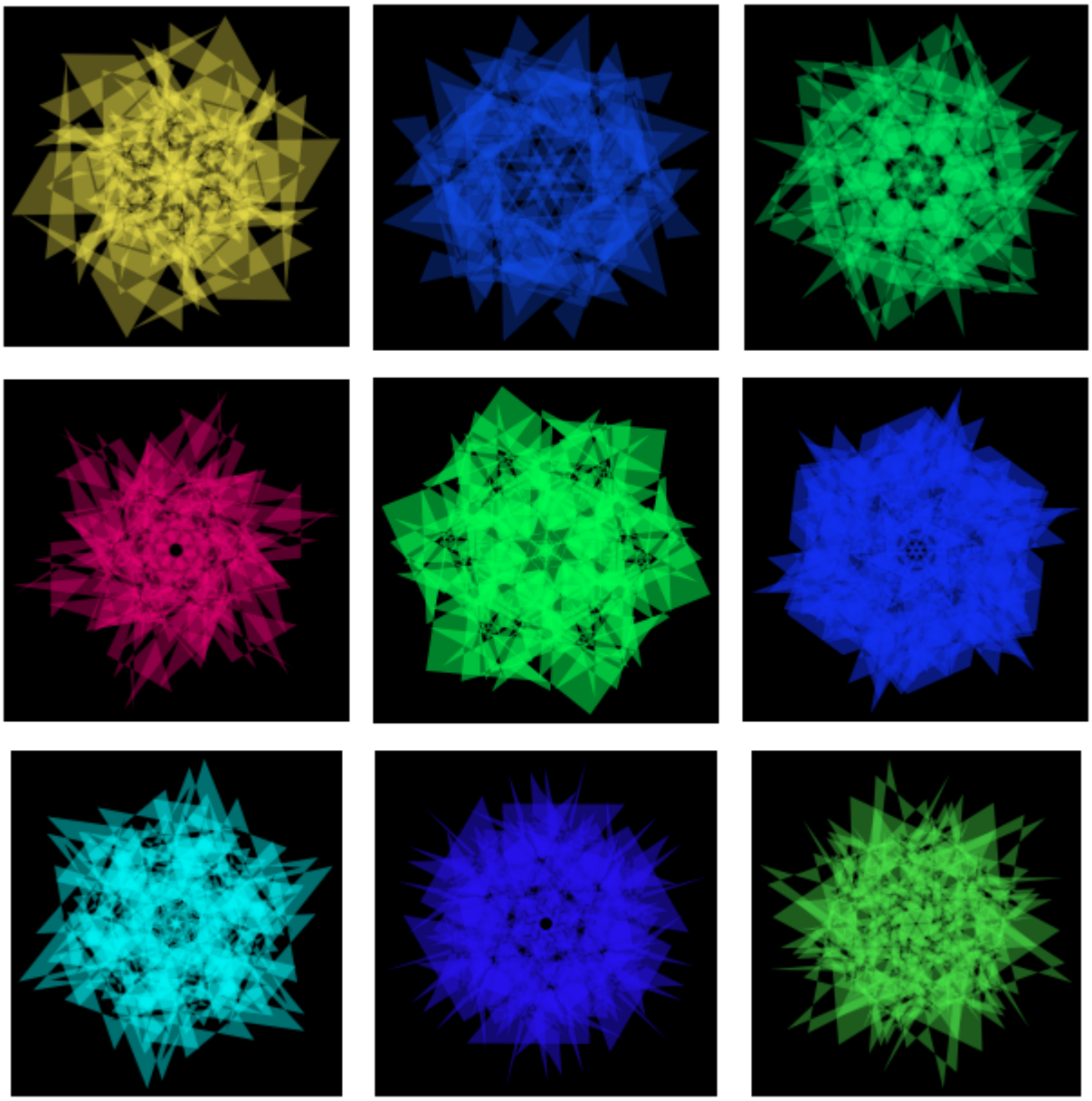
```
makeFlake[n_] := Module[{tab, rot},
  tab = RandomReal[{-1/2, 1/2}, {n, 2}];
  rot = RotationMatrix[Pi/3];
  Graphics[{Hue[RandomReal[]], Opacity[RandomReal[{0.3, 0.6}]],
    Polygon[tab.MatrixPower[rot, #]] & /@ Range[6]}, Background -> Black]]
```

Some sample output:

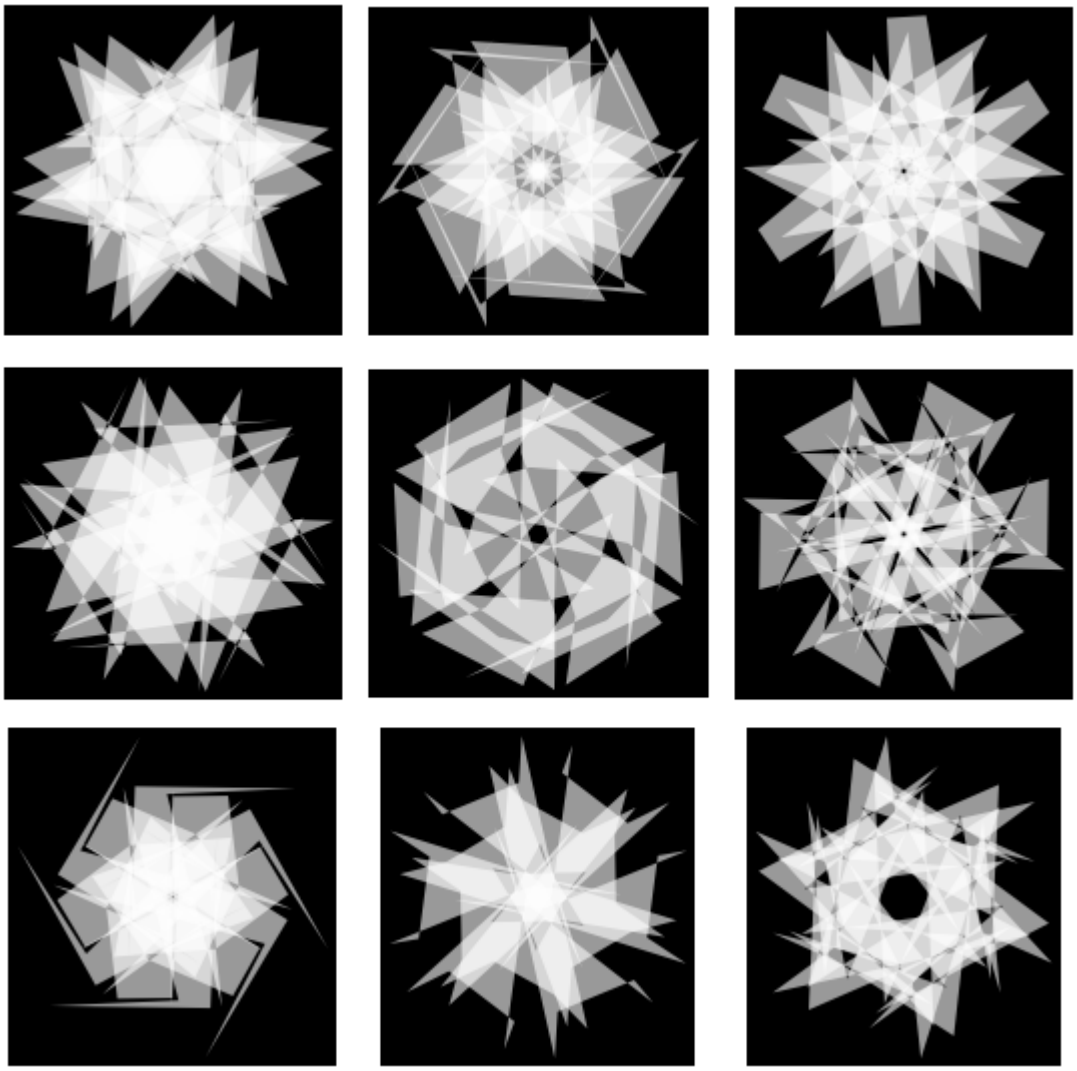
```
GraphicsGrid[Table[makeFlake[16], {i, 3}, {j, 3}]]
```



For more complex shapes, increase the value of `n`, which is the number of sides of each polygon. Here are some examples with `n=32`.

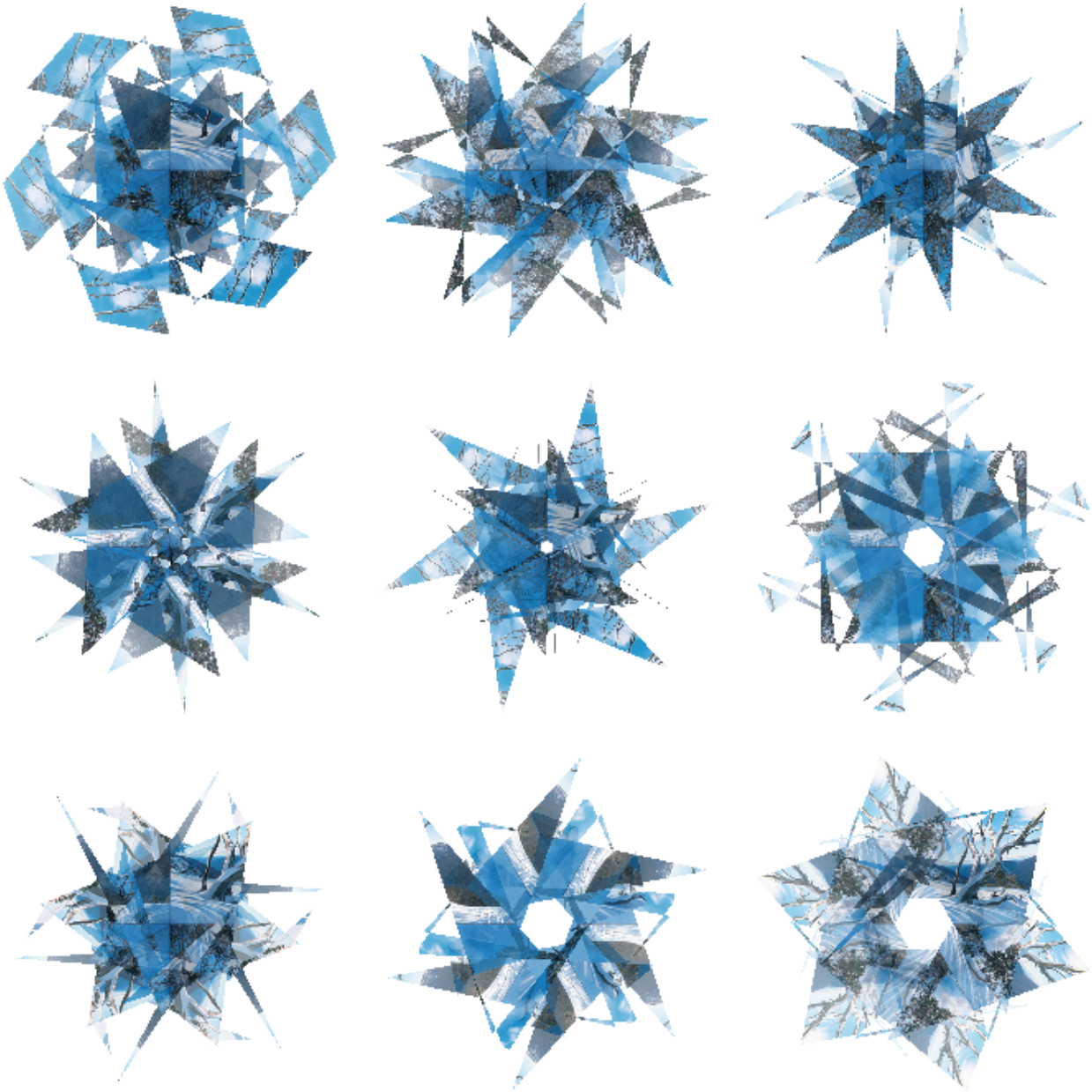


If, for some reason you think that snowflakes ought to be white, change `Hue[RandomReal[]]` to `White`:



Update: It is also simple to "color" the polygon using an image. For example, using a snow-filled winter scene as a texture in the polygons:

```
thisImg = Import["https://i.sstatic.net/kMCN1.jpg"];
poly[img_, x_] := {Opacity[RandomReal[{0.5, 0.8}]], EdgeForm[],
   Texture[img], Polygon[x, VertexTextureCoordinates -> x]};
makeFlakeImage[img_, n_] := Module[{},
   tab = RandomReal[{-1/2, 1/2}, {n, 2}];
   rot = RotationMatrix[Pi/3, {0, 0, 1}];
   Graphics[Rotate[poly[img, tab], #, {0, 0}] & /@ Range[0, 2 Pi, Pi/3]]];

GraphicsGrid[Table[makeFlakeImage[thisImg, 10], {i, 3}, {j, 3}]]
```



Share Improve this answer Follow

edited Dec 26, 2013 at 3:02

answered Dec 25, 2013 at 16:00

bill s
69.7k 4 103 197

1 The frosty feel from images is so wintery! Nice! – Vitaliy Kaurov Dec 26, 2013 at 9:54

Beautiful fragile gems! – Silvia Dec 27, 2013 at 1:28

27

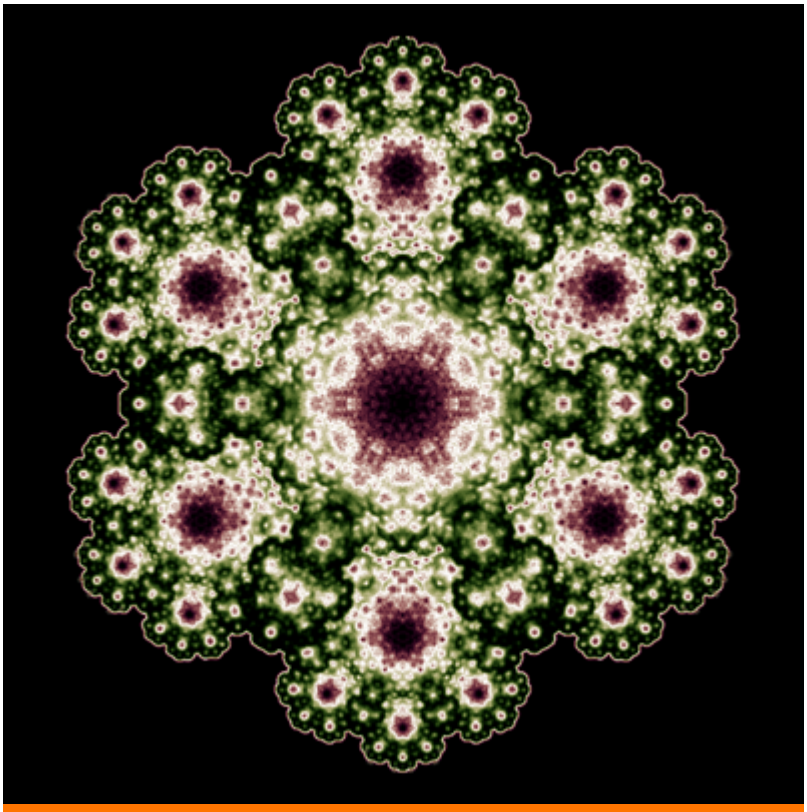
Not so much snowflakes as random artworks with the same symmetry as snowflakes, but I wanted to join in the festive fun! These are generated with a "randomart" package I wrote a while ago (code at the bottom of the answer). It uses a kind of non-linear iterated function system to generate random images.

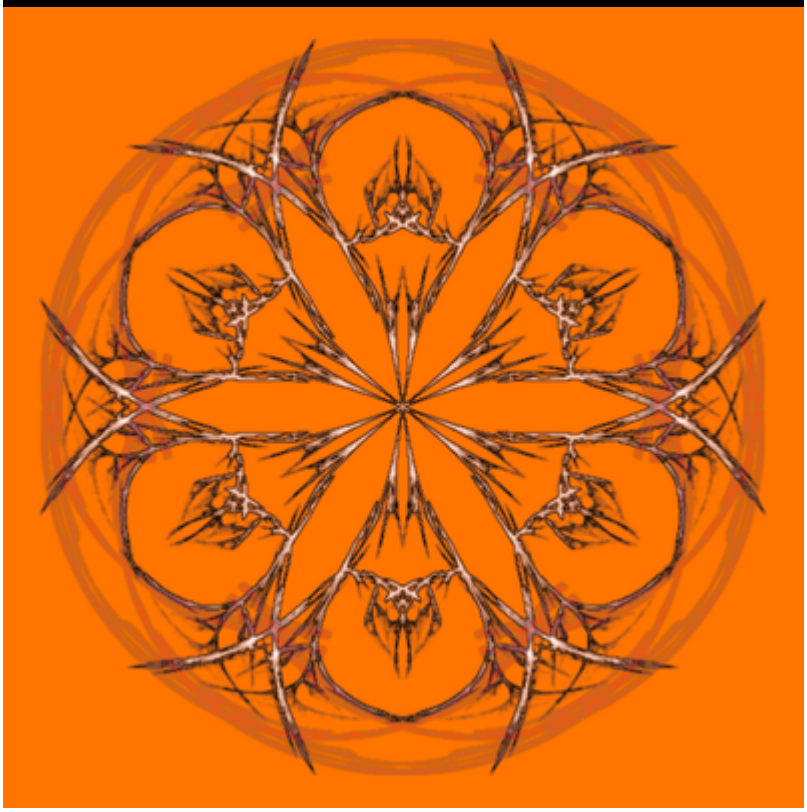
Here's a grid of random images with snowflake symmetry:

```
Table[randomart[100, RandomInteger[{1, 4}], Conjugate, 6], {5}, {5}] // Grid
```

If you specify a larger image size the code will do more iterations to give more detail. Here are a couple at 400 x 400 pixels:

```
randomart[400, RandomInteger[{1, 4}], Conjugate, 6]
```





Here's the package code:

```
BeginPackage["randomart`"];
randomart::usage =
  "randomart[size, n, sym, m] produces a random image. size is the \
image size. n is the number of individual patterns to compose \
together. sym is a symmetry function to apply (the points making up \
the image are represented as complex numbers, so for example use \
Conjugate to obtain left-right symmetry). To apply no symmetry \
```

```
function use {}. If supplied, m will cause the image to be created \
with m seed points evenly distributed around the unit circle, leading \
to m-fold rotational symmetry in the final image. For more 'organic' \
results use no symmetry function and omit m (or set to zero).
  Large sizes will be expensive in time and memory, 500 is a good size.
  randomart[size] will produce an image with random parameters.
  Results are a bit hit & miss - be prepared to generate several \
images to find a nice one. ";
Begin["`Private`"];
gradients = {x, x^2, Sqrt[x], 1 - x, (1 - x)^2, 1 - x^2, Sqrt[1 - x],
   1 - Sqrt[1 - x], x (2 - x), Abs[-1 + 2 x], 1 - Abs[-1 + 2 x],
   4 x (1 - x), Sqrt[Abs[-1 + 2 x]], 1 - Sqrt[Abs[-1 + 2 x]]};
hsbfunc :=
  Module[{h1, h2, ff}, h1 = RandomReal[];
   h2 = h1 + 0.5 RandomReal[{-1, 1}];
   ff = {h1 (1 - x) + (h2) x}~Join~RandomChoice[gradients, 2];
   Function @@ Hold[x, Evaluate[ff]]];
rc := Sqrt[#1] Exp[2 Pi I #2] & @@ RandomReal[{0, 1}, 2];
parameters[m_] :=
  Module[{n, points, r1, r2, z1, z2, z3},
   If[m == 0, n = RandomInteger[{3, 12}]; points = Table[rc, {n}],
    n = m; points = RandomReal[{0, 1}] Exp[2. I Pi Range[n]/n]];
   r1 = RandomReal[{0, 1}];
   r2 = RandomChoice[{1, 2} -> {r1, Abs[rc]}];
   z1 = RandomChoice[{1, rc, Abs[rc]}];
   z2 = RandomChoice[{2, 1} -> {z1, rc}];
   z3 = Exp[I RandomReal[{0, 2 Pi}]];
   {n, points, r1, r2, z1, z2, z3}];
binlog = Compile[{{q, _Real, 2}, {size, _Integer}},
   Block[{x, qr, qi}, x = ConstantArray[1, {size, size}];
    {qr, qi} = 1 + Floor[(size - 1) q];
    Do[x[[qr[[j]], qi[[j]]]] += 1, {j, Length[qr]}];
    N[Log[x]]], CompilationTarget -> "C", RuntimeOptions -> "Speed"];
rawimage[{n_, points_, r1_, r2_, z1_, z2_, z3_}, size_, sym_] :=
  Module[{c, data, q},
   c = Compile[{{x, _Complex, 1}},
     Evaluate[Abs[z1 - z3 x]^r1 Exp[I r2 Arg[z2 - z3 x]] points],
     RuntimeAttributes -> {Listable}];
   data = Flatten@Nest[c, points, Floor[Log[12 size^2]/Log[n]] - 1];
   q = process[data, sym];
   GaussianFilter[Clip[1.5 Rescale[binlog[q, size]], {0, 1}], 2]];
process[data_, sym_] :=
  Module[{dat, q}, dat = If[sym === {}, data, data~Join~sym[data]];
   q = Rescale[{Re[dat], Im[dat]}];
   If[sym === {}, q, centre[q]]];
centre[q_] := q + (0.5 - 0.5 (Max[#] + Min[#]) & /@ q);
prettify[pic_] :=
  Module[{i},
   i = Image[Re[hsbfunc[pic]], Interleaving -> False,
     ColorSpace -> "HSB"];
   ImagePad[ColorConvert[Image[i, Interleaving -> True], "RGB"],
    Round[Length[pic]/23], Automatic]];
oneframe[size_, sym_, m_] :=
  prettify@rawimage[parameters[m], size, sym];
alpha[image_] := Module[{a}, a = ColorConvert[image, "Grayscale"];
   If[PixelValue[a, {1, 1}] > 0.5, a = ColorNegate@a];
   SetAlphaChannel[image, a]];
compose[n_, size_, sym_, m_] :=
  Fold[ImageCompose, oneframe[size, sym, m],
   Table[alpha@oneframe[size, sym, m], {n - 1}]];
randomart[size_, n_, sym_: {}, m_: 0] :=
  ImageResize[compose[n, Round[69/50 size], sym, m], Scaled[2/3]];
```

```
randomart[size_] :=
  randomart[size, RandomInteger[{1, 4}],
   RandomChoice[{2, 1} -> {{}, Conjugate}], 0];
End[];
EndPackage[];
```

Share Improve this answer Follow

answered Dec 26, 2013 at 9:34

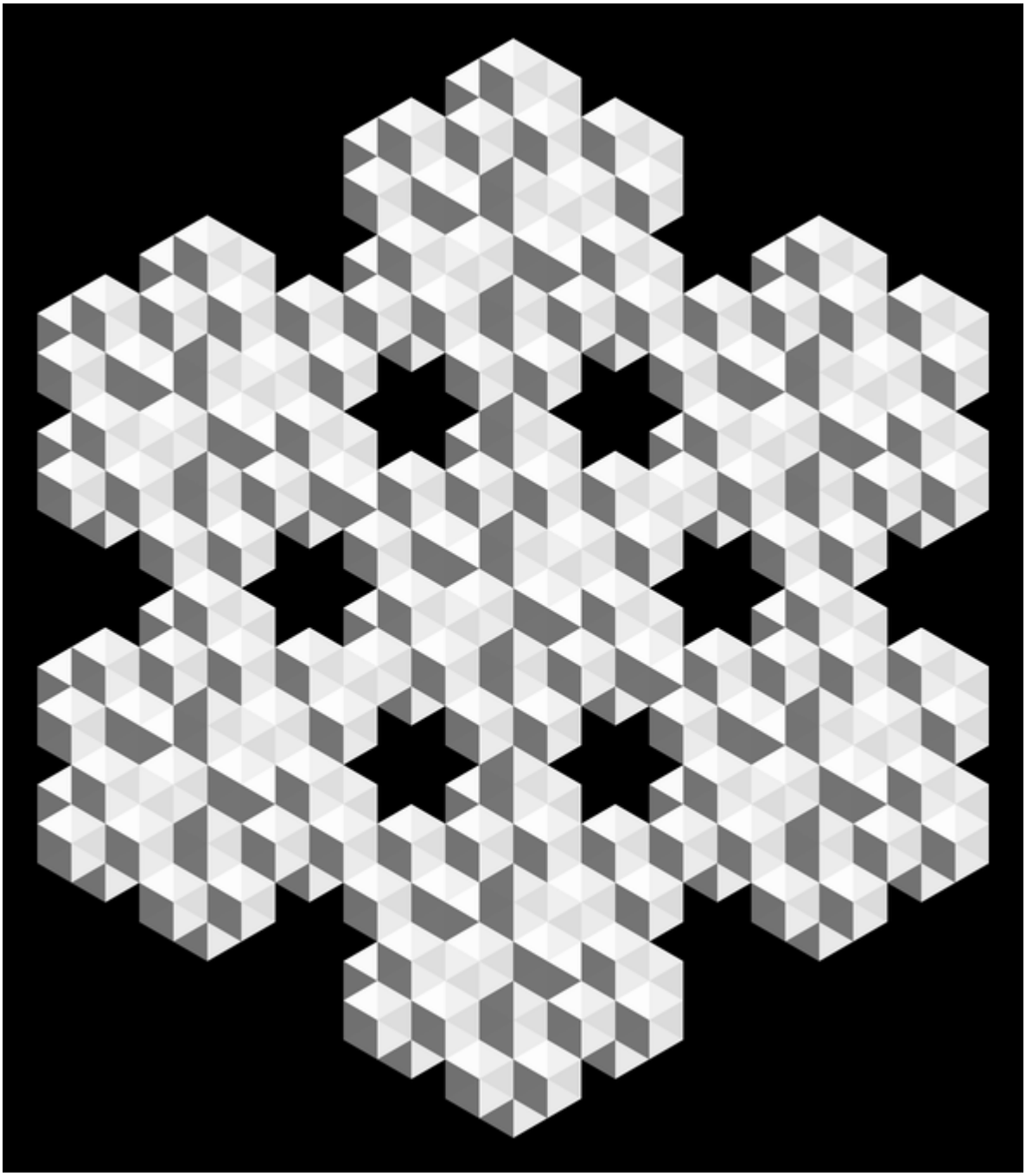
Simon Woods
85.3k 8 179 325

2 +1 Love the idea of iteration, great as usual! – Vitaliy Kaurov Dec 26, 2013 at 9:53

24

Well I guess one more couldn't hurt. Using an iterated matrix-replacement scheme and some fancy opacity:

```
powzerz = 2;
width = 550;
primitive = Scale[Cuboid[], 0.99999];
matrix0 = {{{1}}};
matrixT = CrossMatrix[{1, 1, 1}];
rules = {0 -> (0 #1 &), 1 -> (#1 &)};

iterate[matrix0_, matrixT_, rules_, power_] :=
    Nest[Function[prev,
        ArrayFlatten[Map[#[prev] &,
            Replace[matrixT, rules, {3}], {3}], 3]],
      matrix0, power];

g = With[{objects = Translate[primitive, Position[iterate[matrix0, matrixT, rules,
powzerz], 1]]},
   Graphics3D[{White, Opacity[.9], EdgeForm[None], objects},
    Lighting -> "Neutral", Method -> {"ShrinkWrap" -> True}, ImageSize -> 4 width,
    Boxed -> False, ViewPoint -> 2000 {1, 1, 1}, ViewVertical -> {0, 0, 1}, Background
-> Black]];

ImageResize[Rasterize[g], Scaled[1/4]]~ImagePad~20
```

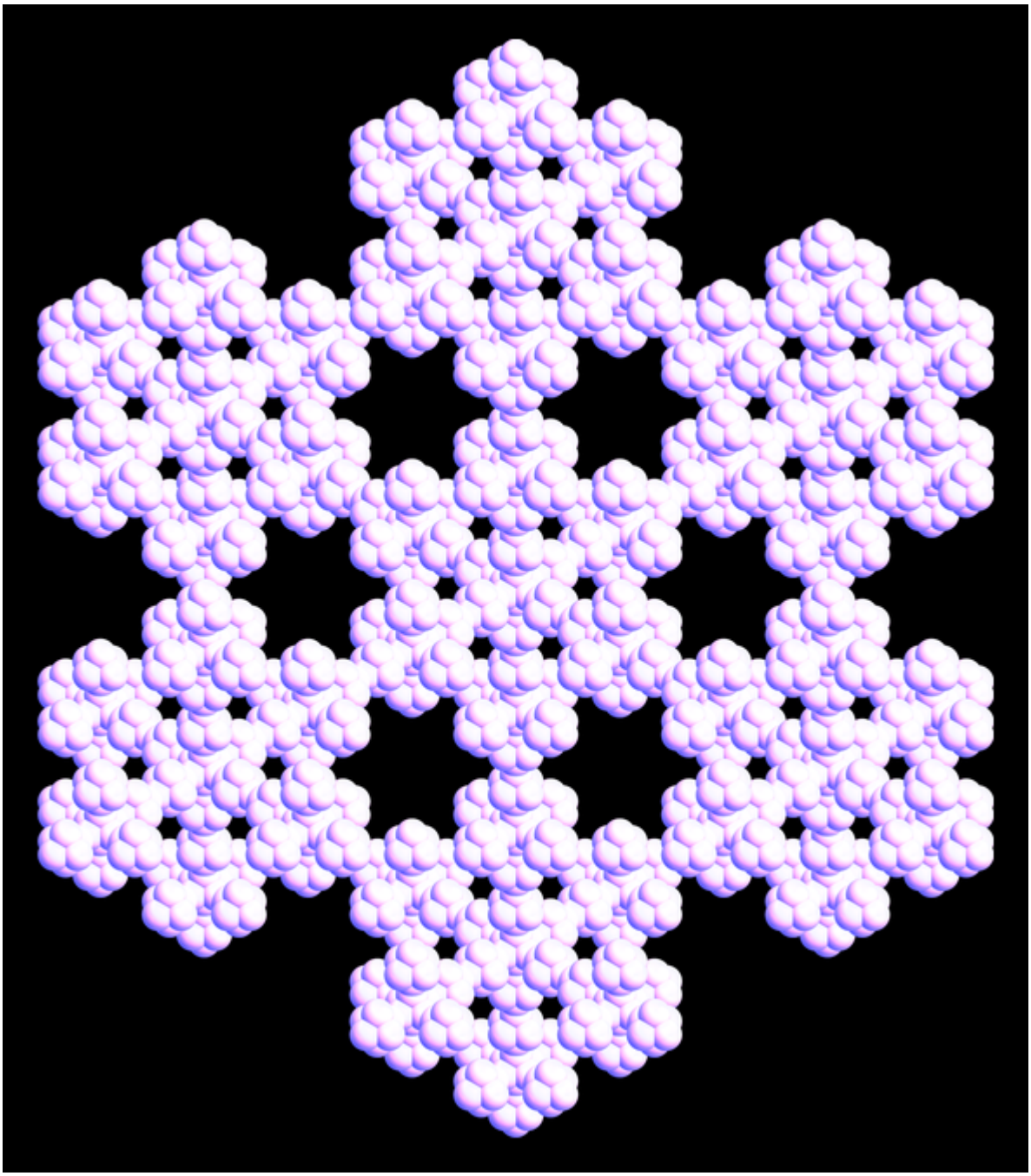


It's a simple 3D cross fractal (this code is a reduced version of this monster). Although it's 3D, you get 2D figures. In this case Koch outlines. I wonder what kinds of 2D hex systems you could describe in terms of 3D, and vice-versa (e.g. an automaton rule on a 3D grid which is perspectivally equivalent to an automaton rule on a hex lattice).

For no reason, a bright cotton candy version:

```
powzerz = 4;
width = 550;
primitive = Sphere[.5 {1, 1, 1}];
matrix0 = {{{1}}};
matrixT = CrossMatrix[{1, 1, 1}];
rules = {0 -> (0 #1 &), 1 -> (#1 &)};

iterate[matrix0_, matrixT_, rules_, power_] :=
```

```
    Nest[Function[prev,
        ArrayFlatten[Map[#[prev] &,
            Replace[matrixT, rules, {3}], {3}], 3]],
      matrix0, power];

g = With[{objects = Translate[primitive, Position[iterate[matrix0, matrixT, rules,
powzerz], 1]]},
   Graphics3D[{White, Opacity[.95], Glow[Blue], Specularity[Darker@Red],
EdgeForm[None], objects},
    Lighting -> "Neutral", Method -> {"ShrinkWrap" -> True}, ImageSize -> 4 width,
    Boxed -> False, ViewPoint -> 2000 {1, 1, 1}, ViewVertical -> {0, 0, 1}, Background
-> Black]];

ImageResize[Rasterize[g], Scaled[1/4]]~ImagePad~20
```



Share Improve this answer Follow edited Dec 26, 2013 at 7:01 answered Dec 26, 2013 at 6:40

amr
5,497 1 24 32

2 i just noticed this isn't particularly in line with what OP was looking for. i don't think it'll hurt anyone though and it's in the Christmas spirit – amr Dec 26, 2013 at 7:10

+1 at least it looks a bit like a snowflake - half the images on this page are just wannabe hexagons... :) – cormullion Dec 26, 2013 at 11:20

2 The OP said in the comments: "I hoped to collect here some code to produce visual marvels... to embellish season's greetings cards.. even if virtual or just something that looks like a snowfake." Almost all the answers fit this description! – bill s Dec 26, 2013 at 13:57

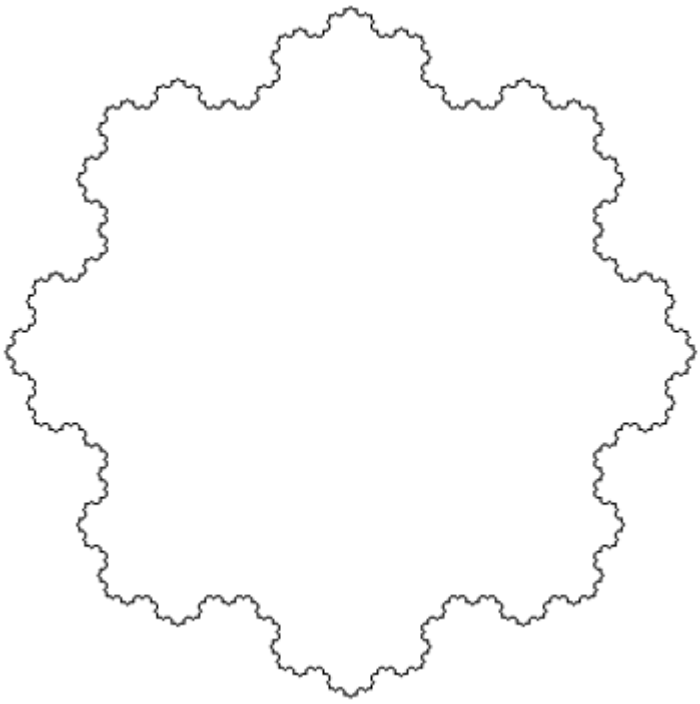
21

Here is an un-golfed and simplified version of an L-System production based on a previous answer of mine:

```
f1[initState_, rotAngle_, prodRules_, iters_] :=
 Module[{currAngle = 0, currPos = {0, 0}, res = {}},
  (res = {res, Line@{currPos, currPos += {Cos@currAngle, Sin@currAngle}}};
          If[NumericQ@#, currAngle += I^# rotAngle]) & /@
                                              Nest[Flatten[# /. prodRules] &,
initState, iters];
  Graphics@Flatten@res
  ]
```

Used to produce a Koch Snowflake (not random, just fractal):

```
f1[{C[1], 2, 2, C[1], 2, 2, C[1], 2, 2, C[1], 2, 2, C[1]},
    Pi/4, {C[1] -> {C[1], 4, C[1], 2, 2, C[1], 4, C[1]}}, 4]
```



Usage instructions and original golfed version here.

Share Improve this answer Follow

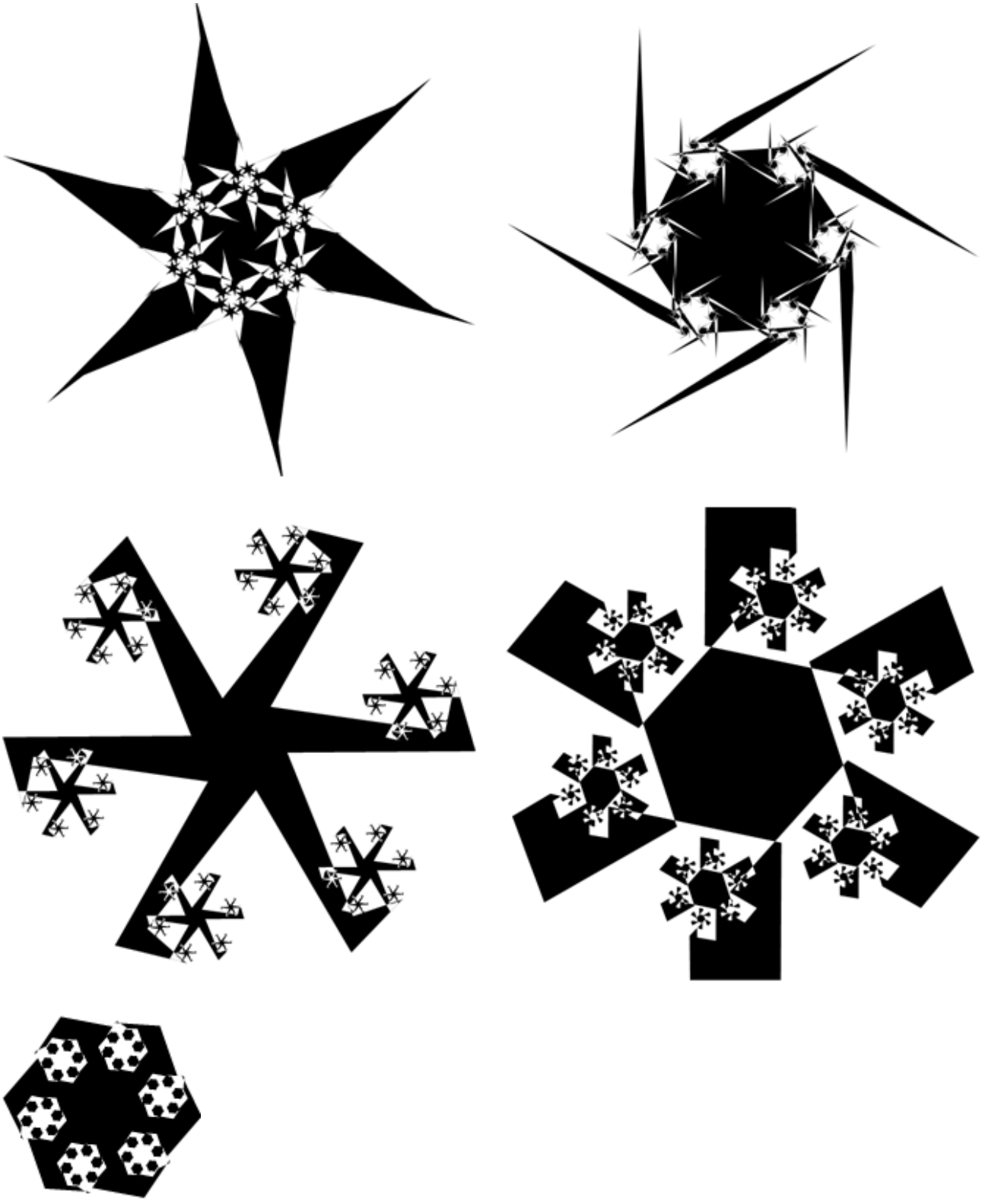
edited Apr 13, 2017 at 12:39
Community Bot
1

answered Dec 24, 2013 at 15:29
Dr. belisarius
116k 13 205 455

20

A smooth changing fractal snowflake:

```
{s, d, t} = {0, 1, 3};
Dynamic@Graphics@
  Polygon@Reap[
     If[# != 0, t += 8.^-5;
        Do[#0[# - 1];
         Sow[d = Sign@d #; {Re[s += d], Im@s}] & /@ (# E^(I t #) &@
           Range@6/(5^(4 - #))); d *= E^((\[Pi] - 63 t)/3 I), {6}]] &@
      3][[2, 1]]
```



Share Improve this answer Follow

edited Dec 26, 2013 at 7:47

answered Dec 26, 2013 at 4:31

kptnw
1,416 8 16

1 You might want to fix the image size... I almost got a headache from watching it! :) – rm -rf ♦ Dec 26, 2013 at 6:16

4

For couple of years I was thinking that I should post an answer here using the `RandomMandala` framework.

So, finally (after some experimentation) I got *reasonable enough* snowflakes.

Remark: The code of this answer produces snowflake images that resemble the snowflakes of both Silvia's answer and Vitaliy's answer.

Remark: The "main idea" is also described in more detail my answer of "How to add doodles and sketches in a mandala".

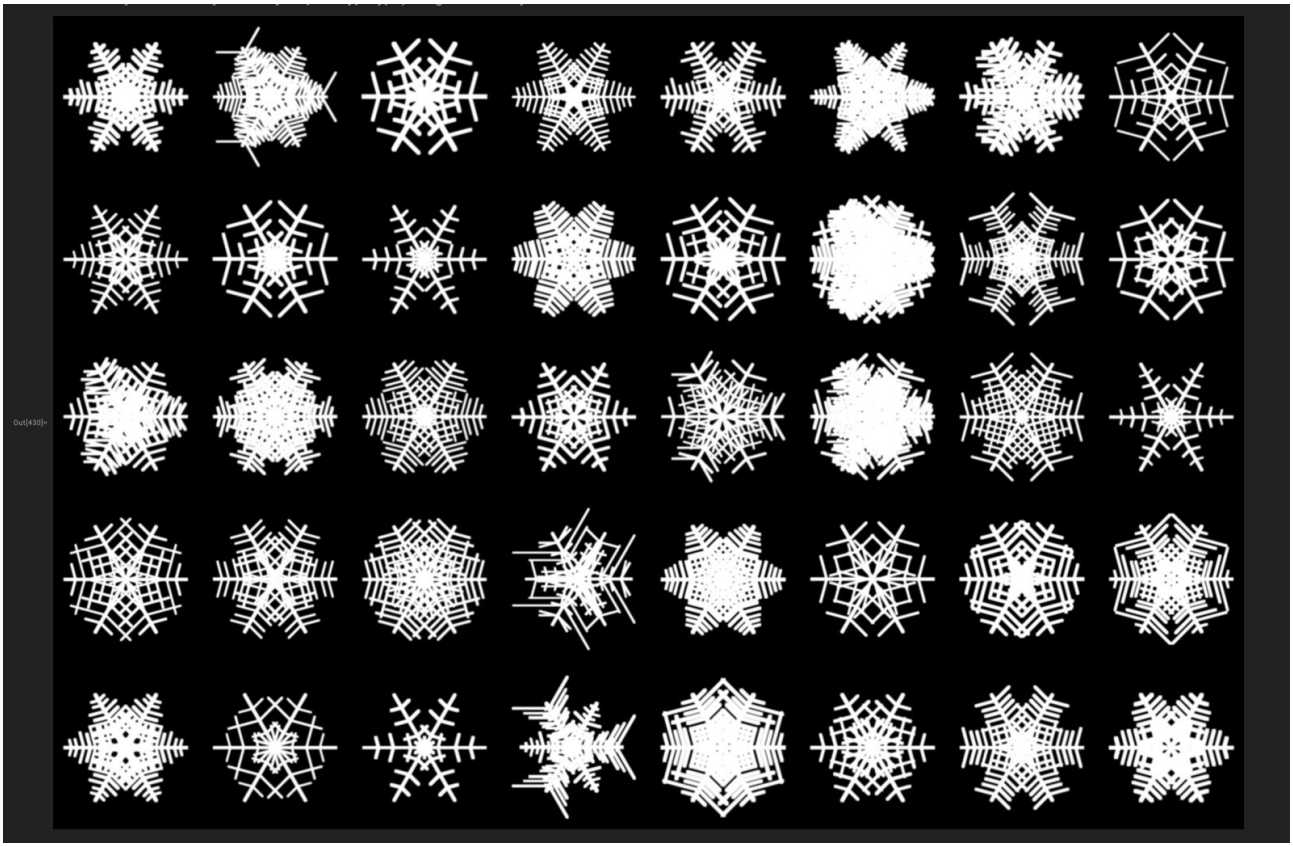
Code

```
Clear[TwigSeed];
Options[TwigSeed] = {"ShiftOffset" -> 0.3, "ShiftFactor" -> 1/3,
   "ShiftRange" -> {0.5, 2}, "Inversions" -> 2};
TwigSeed[radius_, angle_, n_Integer : 5, args___,
   opts : OptionsPattern[]] :=
  Block[{shiftOffset, shiftFactor, shiftRange, inversions, t, p, ps,
    ps2, fs, inds},

   shiftOffset = OptionValue[TwigSeed, "ShiftOffset"];
   shiftFactor = OptionValue[TwigSeed, "ShiftFactor"];
   shiftRange = OptionValue[TwigSeed, "ShiftRange"];
   inversions = OptionValue[TwigSeed, "Inversions"];

   ps = {#, 0} & /@ Rest[Range[0, 1, 1/n]];
   ps = Map[RandomReal[{0.95, 1.05}]*# &, ps];
   ps2 = Map[#[[1]]*{Cos[angle], Sin[angle]} &, ps];

   fs = Sort@
     MapThread[
      Norm[(#2 - #1)]*shiftOffset +
        shiftFactor*RandomReal[shiftRange] &, {ps, ps2}];
   Which[
    IntegerQ[inversions],
    Do[
     inds = RandomChoice[Range@Length@fs, 2];
     fs = ReplacePart[fs, Thread[inds -> fs[[Reverse[inds]]]]],
     {i, inversions}
     ],

    MemberQ[{Random, RandomSample}, inversions],
    fs = RandomSample[fs],

    MemberQ[{Reverse, ReverseSort}, inversions],
    fs = ReverseSort@fs
    ];

   t = MapThread[{#1, #1 + (#2 - #1)/
          Norm[(#2 - #1)]*#3} &, {{1 - #[[1]], #[[2]]} & /@
       ps, {1 - #[[1]], #[[2]]} & /@ ps2, fs}];
   t = Map[{AbsoluteThickness[RandomReal[{2, 5}]], Line[#]} &, t];
   t = Prepend[
     t, {AbsoluteThickness[RandomReal[{4, 7}]],
      Line[{{0, 0}, {1, 0}}]}];
```

```
   (*GeometricTransformation[t,ScalingTransform[{radius,radius}]]
   *)t
   ];


Clear[Snowflake];
Snowflake[blurQ_ : True, effectQ_ : True] := Block[{gr, img},
   gr = ResourceFunction["RandomMandala"][
     "RotationalSymmetryOrder" -> RandomChoice[{8, 2} -> {6, {6, 3}}],
      "SymmetricSeed" -> True,
     "SeedFunction" -> (TwigSeed[##,
         "ShiftOffset" -> RandomReal[{0.2, 0.6}],
         "ShiftFactor" -> RandomReal[{0.2, 0.6}],
         "ShiftRange" -> {0.2, 0.9},
         "Inversions" ->
          RandomChoice[{6, 2, 2} -> {2, None, Random}]] &),
     "NumberOfSeedElements" -> RandomInteger[{4, 12}],
     ImageSize -> 400, ImageMargins -> {30, 30},
     ColorFunction -> (Black &)];

   img = ImagePad[ColorNegate@Image[gr], 30];

   If[TrueQ[blurQ],
    img = Blur[Dilation[img, RandomReal[{2, 4}]], 4]
    ];

   If[TrueQ[effectQ],
    img =
     ImageEffect[
      ImageEffect[
       img, {"Jitter",
        Max[RandomVariate[NormalDistribution[6, 3]],
         1]}], {"OilPainting", RandomInteger[{2, 6}]}]
    ];

   img
   ];
```

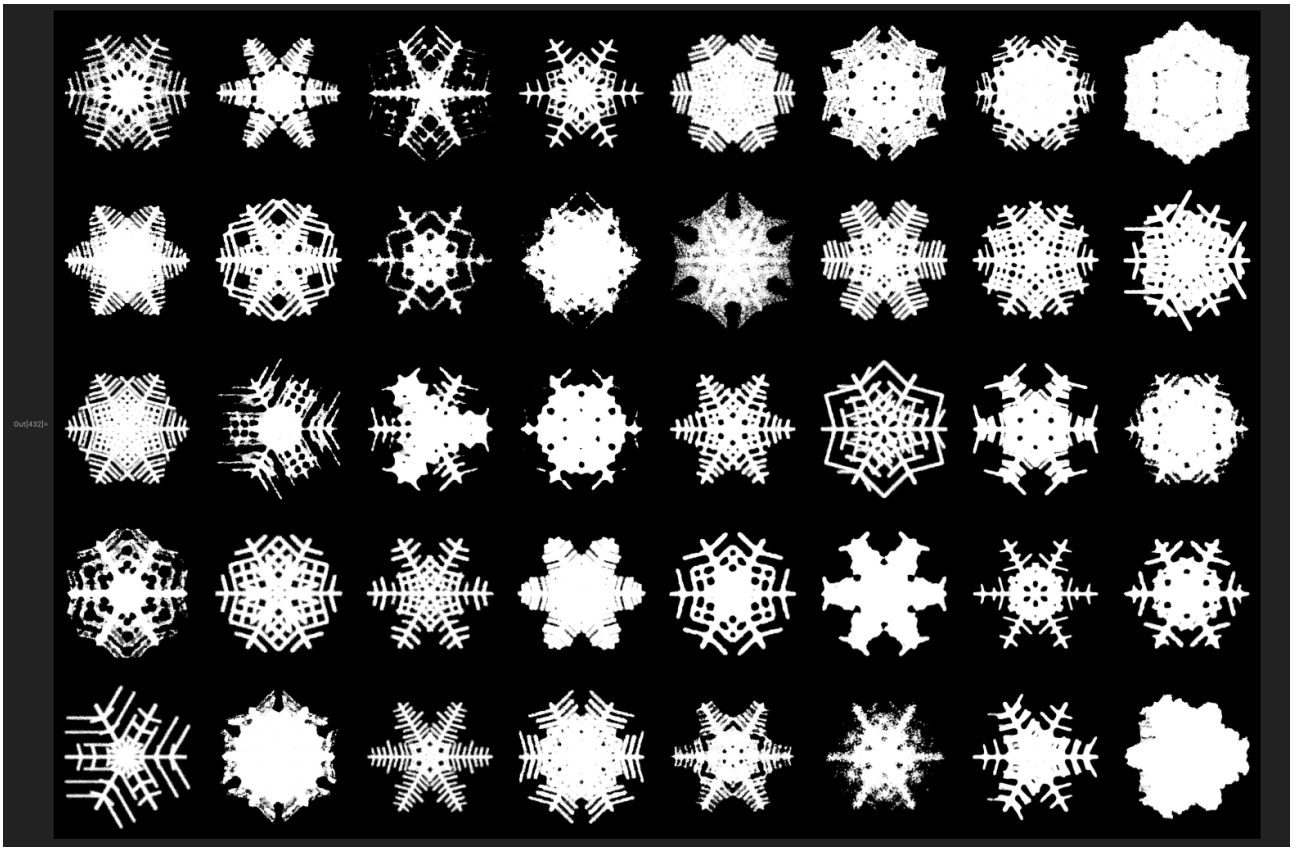
Without additional image effects

```
SeedRandom[11];
Multicolumn[ParallelTable[Snowflake[True, False], 40], 8, Background -> Black]
```



With additional image effects

```
SeedRandom[11];
Multicolumn[ParallelTable[Snowflake[True, True], 40], 8, Background -> Black]
```



Share Improve this answer Follow

answered Dec 15, 2023 at 17:25

Anton Antonov

38k 3 101 178

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.