Fractal Variations

Izabella Foltynowicz

Adam Mickiewicz University, Theoretical Chemistry Department

Ul. Grunwaldzka 6, PL 60-780, Poznań, Poland

iza@rovib.amu.edu.pl

Andrzej Walat

OEIIZK

Ul. Nowogrodzka 73, PL 02-006, Warsaw, Poland

andrzej@oeiizk.waw.pl

Abstract

"It seems so easy for nature to produce forms of great beauty. Yet in the past art has mostly just had to be content to imitate such forms. But now with the discovery that simple programs can capture the essential mechanism for all sorts of complex behaviour in nature, one can imagine just sampling such programs to explore generalizations of the forms we see in nature" (S. Wolfram, 2002). Our main aim is to experiment with simple procedures, written in Imagine Logo, which "generate pictures that have striking aesthetic qualities – sometimes reminiscent of nature, but often unlike anything ever seen before" (S. Wolfram, 2002).

Keywords

Fractals, Sierpiński Triangle, Logo, Imagine Logo, Sierpiński Gasket, Relatives of the Sierpiński Gasket

1. Introduction

We were brought up with the paradigm that a good and professional programmer always starts from the exact specification of the task, i.e. a clear and unambiguous description of what the program should do, and only after that he starts programming. Therefore we are a bit ashamed to confess that we both have a bad habit of looking at various simple programs and trying to modify them, very often without any clear goals. We have discovered, however, that such attitude to the programming activity may be extremely rewarding. We are happy that we found not so long ago a strong support to our way of thinking in the famous Wolfram's book *A New Kind of Science*.

"In our everyday experience with computers, the programs that we encounter are normally set up to perform very definite tasks. But the key idea that I had nearly twenty years ago – and that eventually led to the whole new kind of science in this book – was to ask what happens if one instead just looks at simple arbitrarily chosen programs, created without any specific task in mind. How do such programs typically behave?"

During the last EUROLOGO 2003 conference one of the two authors (I. F.) presented a paper that represents just that kind of attitude. She started from a relatively simple and common shape – the Sierpiński triangle – and generated a rich variety of interesting objects by small modifications of the rules. Now, once again, we have started from the same point but followed other paths and by this we have discovered a new rich variety of interesting relatives of the Sierpiński triangle (Peitgen *et al*. 1992a).

2. Starting point

Every child with an interest in mathematics knows the Sierpiński triangle. It seems impossible that such a simple object could hide anything unexpected. But probably everybody was surprised while learning the rules for the first time and seeing the results of the chaos game. We, like everyone, were very impressed while reading interesting pages devoted to the Sierpiński triangle and chaos game in the renowned book by Peitgen et al. (1992b). At the same time we were rather disappointed that the simple rules of the chaos game were translated into such a long, sophisticated and unclear code, i.e. - the computer program written in Basic in Peitgen et al. (1992c) and the about forty lines long code for a graphical calculator in Peitgen et al. (1998). They are in stark contrast to a very short, clear, and at the same time, very general, procedure we can write in Imagine Logo:

```
to go :apoint :listofpoints
  setpos :apoint dot
  go (:apoint + pick :listofpoints) / 2 :listofpoints
end
```

This simple procedure gives us an opportunity to explore a variety of interesting cases. Using the `go` procedure with the input parameters `[0 0][[-100 -100] [100 -100] [50 100]]` we get the kind of Sierpiński triangle that is shown in *Figure 1a*, however with just a slight change of the input parameters, `go [0 0][[-80 -100] [150 -100] [30 150] [-120 100]]`, we get a decorated quadrilateral as shown in *Figure 1b*.

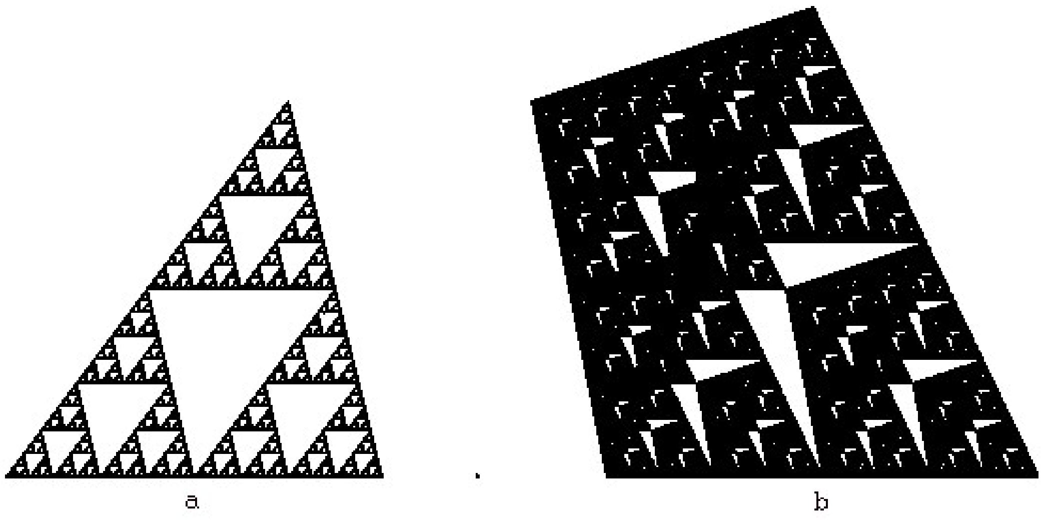

Figure 1. Sierpinski polygons

3. The first step – Addition of a scale factor

Instead of using a constant scale factor in the `go` procedure, as it was done in the example above, we can add a scale parameter:

```
to go :ap :lp :sc
  setpos :ap dot
  go :sc * :ap + (1 - :sc) * pick :lp :lp :sc
end
```

Using the `go` procedure with the input parameters `[0 0][[-100 -100] [100 -100] [50 100] [-150 150]] 0.45` we get another variant of the Sierpiński quadrilateral. To get more regular shapes we define the auxiliary operation `ngon` which returns the list of the all vertices of the regular polygon:

```
to ngon :n :r
  let "lp []
  let "a 360 / :n
  repeat :n [make "lp fput :r * se sin repc * :a cos repc * :a :lp]
  op :lp
end
```

After `cs go [0 0] ngon 5 150 (3 - sqrt 5) / 2` we get a pentagonal shape as shown in *Figure 2a*, whereas after `cs go [0 0] ngon 8 150 1 / (2 + sqrt 2)` – we get an octagonal shape as in *Figure 2b*. These two figures are examples of interesting distant relatives of the Sierpiński triangle since they are built according to similar rules but on the basis of another regular polygon.

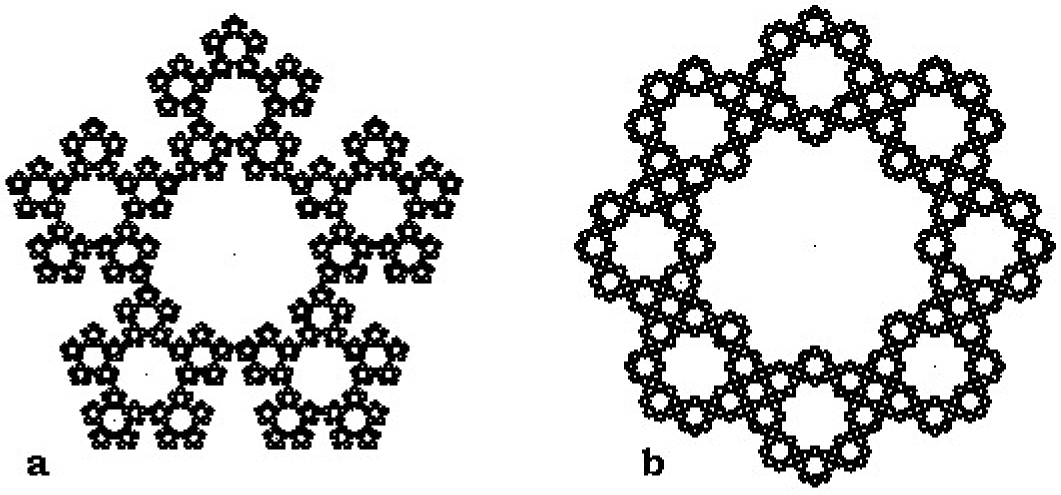

Figure 2. Two examples of ST relatives

Of course, it is also possible to achieve *Figure 2a* through the use of deterministic procedures:

```
to sierpent :n :side :scale
  if :n = 0 [pentagon :side stop]
  repeat 5 [sierpent :n – 1 :scale * :side :scale jfd :side rt 72]
end
to pentagon :side
  repeat 5 [fd :side rt 72]
end
to jfd :d
  pu fd :d pd
end
```

4. The second step – Variations of Sierpiński pentagons

The `sierpent` procedure is a good starting point for creating a variety of variants which, in spite of their deterministic character, generate pictures that look chaotic. Let us demonstrate one of the countless examples.

```
to spent :n :sd :sc :a
  if :n = 0 [stop]
  repeat 5
  [
  rt :a pd spent :n - 1 :sc * :sd :sc :a lt :a
  setpc (se 45 + 30 * :n 255 - 30 * :n 0) setpw :n + 2 dot
  jfd :sd rt 90
  ]
end
```

Using `spent 6 200 0.5 0`, we get a regular geometrical shape shown in *Figure 3a*.

Using `spent 6 200 0.5 45`, on the other hand , we get a more organic shape shown in *Figure 3b*:

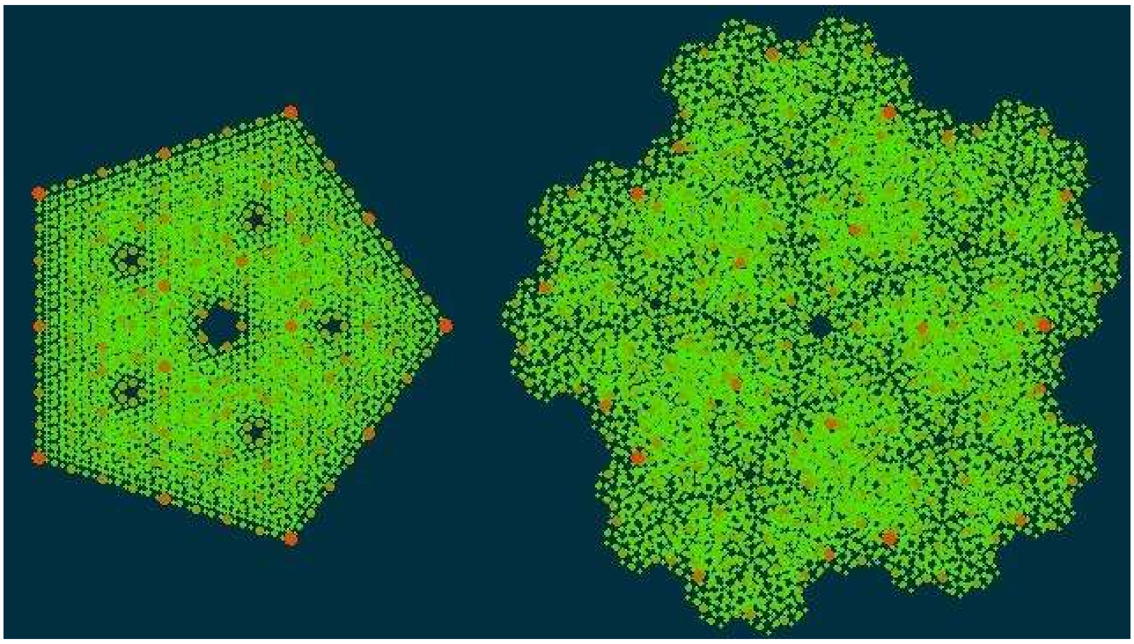

Figure 3. Pentagonal organic shapes

5. The third step – Why not add a bit of randomness?

Let us add a bit of randomness to the spent procedure:

```
to spent :n :sd :sc :a
  if :n = 0 [stop]
  repeat 5
  [
  if random 7 > 0 [rt :a pd spent :n - 1 :sc * :sd :sc :a lt :a]
  setpc (se 45 + 30 * :n 255 - 30 * :n 0) setpw :n + 2 dot
  jfd :sd rt 90
  ]
end
```

With that modified procedure we can generate more interesting and organic-like random shapes - see *Figure 4*.

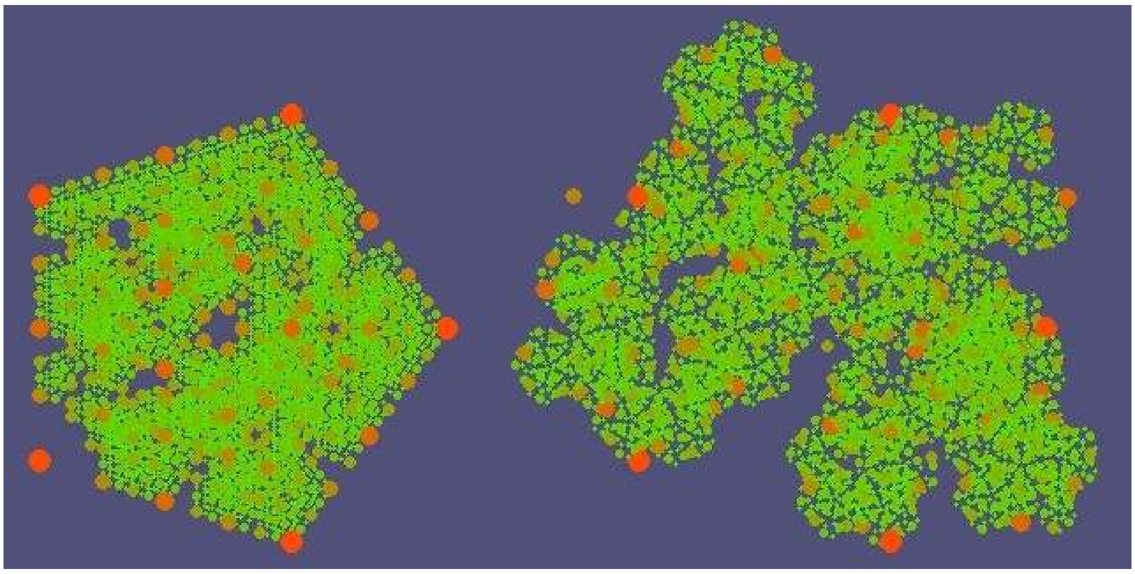

Figure 4. Pentagonal random shapes

6. The fourth step – Exploiting rotations of the square

In this section we will exploit the rotations of the square to generate a whole family of shapes. Let us denote this family by *RS* – rotated squares. The simplest example of the rich family of shapes is an ordinary square. The more complex shape is built of the three simpler ones, which are rotated according to the rules illustrated in figure 5.

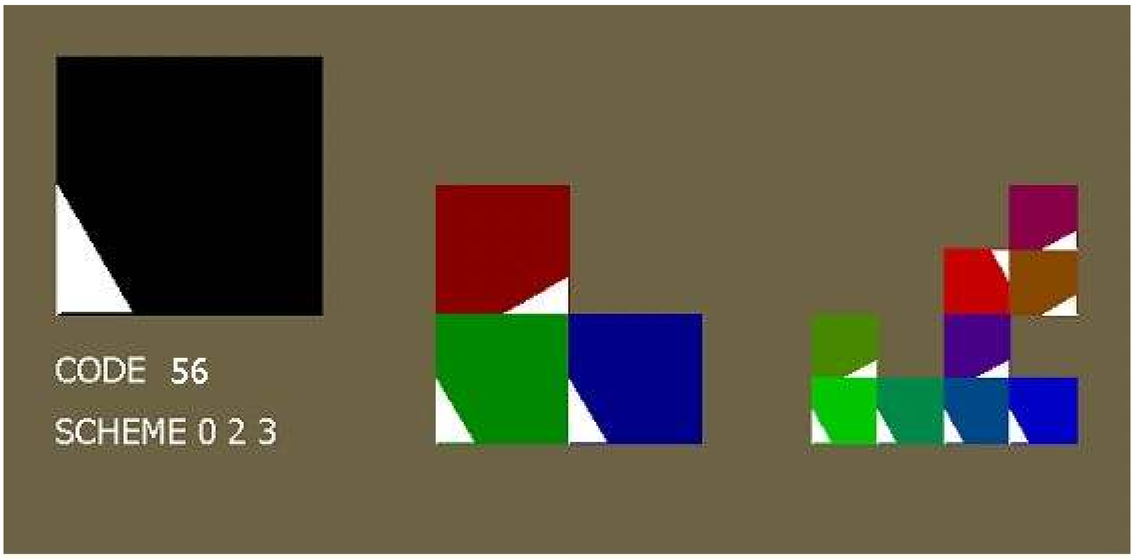

Figure 5. Rotated squares

To see more complex examples we have to run the following `drs` procedure:

```
to drs :code :n :s
  let "n1 mod :code 4 let "code div :code 4
  let "n2 mod :code 4 let "n3 div :code 4
  ars :n1 :n2 :n3 :n :s
end
```

```
  to ars :n1 :n2 :n3 :n :s
    rs :n :s [0 0 0] 128
  end
  to rs :n :s :rgb :d
    if :n = 0 [setpc :rgb polygon list 4 list :s 90 stop]
    lt 90 repeat :n1 [jfd :s / 2 rt 90]
    rs :n – 1 :s / 2 :rgb + (list :d 0 0) :d / 2
    if :n1 > 0 [repeat 4 - :n1 [jfd :s / 2 rt 90]]
    lt 90 repeat :n2 [jfd :s / 2 rt 90]
    rs :n – 1 :s / 2 :rgb + (list 0 :d  0) :d / 2
    if :n2 > 0 [repeat 4 - :n2 [jfd :s / 2 rt 90]]
    lt 90 repeat :n3 [jfd :s / 2 rt 90]
    rs :n – 1 :s / 2 :rgb + (list 0 0 :d) :d / 2
    if :n3 > 0 [repeat 4 - :n3 [jfd :s / 2 rt 90]]
    lt 90
  end
  to jfd :a
   pu fd :a pd
  end
```

The command `drs 26 8 240` gives the result shown in the left part of *Figure 6*. The command `drs 57 8 240` gives, again, the Sierpiński triangle, shown in the right part of *Figure 6*.

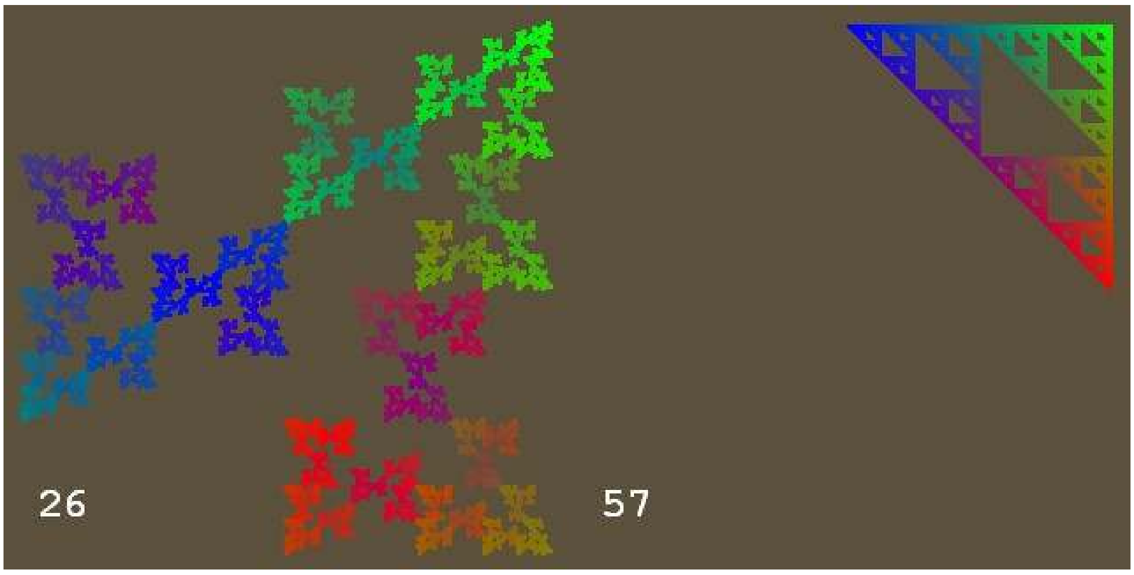

Figure 6. The two results of the drs procedure

7. The fifth step – Cumulated RS

By modifying slightly the `rs` procedure, which was used in the program above, we get a tool for generating cumulated *RS* shapes:

```
to rs :n :s :rgb :d
  setpc :rgb polygon list 4 list :s 90
  if :n = 0 [stop]
  lt 90 repeat :n1 [jfd :s / 2 rt 90]
  rs :n – 1 :s / 2 :rgb + (list :d 0 0) :d / 2
  if :n1 > 0 [repeat 4 - :n1 [jfd :s / 2 rt 90]]
  lt 90 repeat :n2 [jfd :s / 2 rt 90]
  rs :n – 1 :s / 2 :rgb + (list 0 :d  0) :d / 2
  if :n2 > 0 [repeat 4 - :n2 [jfd :s / 2 rt 90]]
  lt 90 repeat :n3 [jfd :s / 2 rt 90]
  rs :n – 1 :s / 2 :rgb + (list 0 0 :d) :d / 2
  if :n3 > 0 [repeat 4 - :n3 [jfd :s / 2 rt 90]]
  lt 90
end
```

Two examples of shapes generated by that procedure are shown in *Figure 7*.

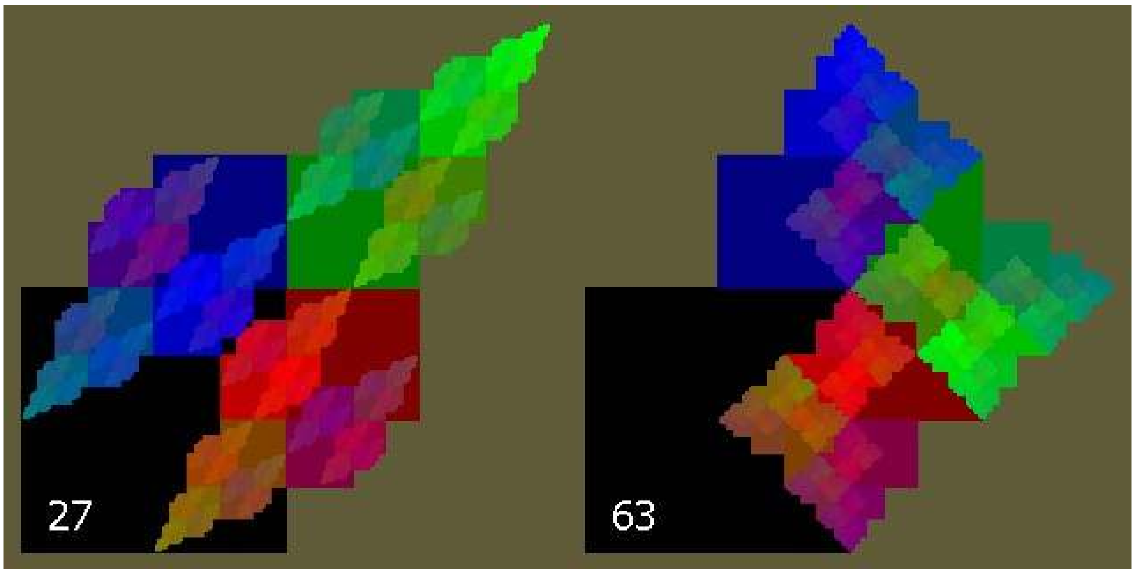

Figure 7. Two examples of cumulated *RS* shapes

8. The sixth step – Seeing the simple and cumulated RS shapes together

If we modify the auxiliary `ars` procedure and add a new `rs0` procedure, we will get a tool that allows us to see simple and cumulated *RS* shapes together:

```
to ars :n1 :n2 :n3 :n :s
  rs :n :s [0 0 0] 128
  setpc "white setpw 2
  rs0 :n :s
end
to rs0 :n :s
  if :n = 0 [dot stop]
  lt 90 repeat :n1 [jfd :s / 2 rt 90]
```

```
    rs0 :n - 1 :s / 2
    if :n1 > 0 [repeat 4 - :n1 [jfd :s / 2 rt 90]]
    lt 90 repeat :n2 [jfd :s / 2 rt 90]
    rs0 :n - 1 :s / 2
    if :n2 > 0 [repeat 4 - :n2 [jfd :s / 2 rt 90]]
    lt 90 repeat :n3 [jfd :s / 2 rt 90]
    rs0 :n - 1 :s / 2
    if :n3 > 0 [repeat 4 - :n3 [jfd :s / 2 rt 90]]
    lt 90
  end
```

The `drs` command modified in that way will now give results as shown in *Figure 8*.

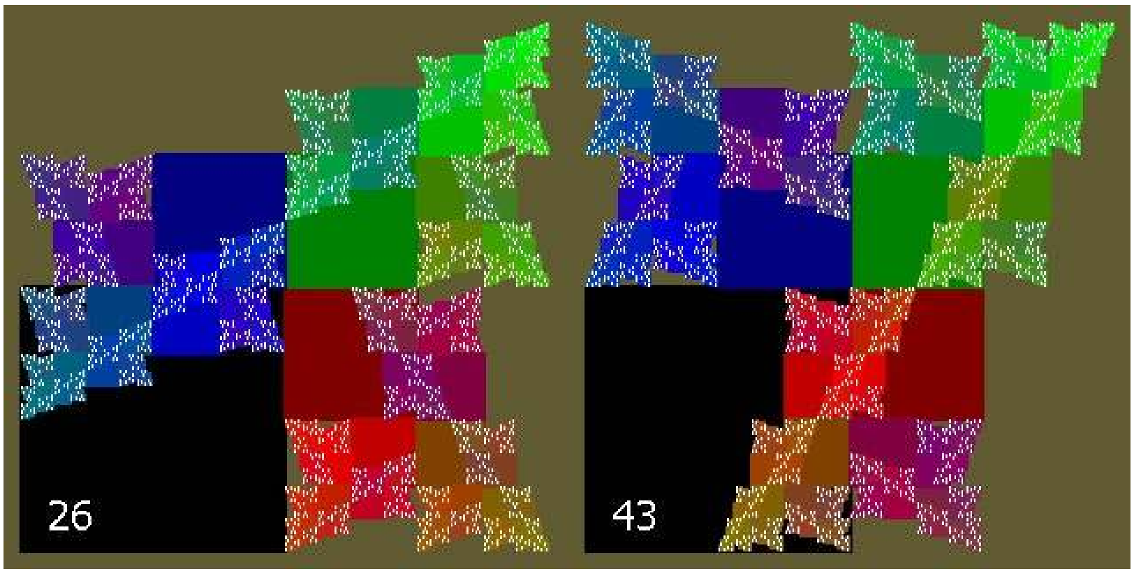

Figure 8. Simple and cumulated RS shapes joined together

9. The seventh step – Adding reflection to rotations

The *RS* family consist of only 64 members (the four rotations, when applied to three figures will give $4^3 = 64$ different attractors), but if we include the reflections (additionally to rotations), we will get a much richer and interesting family comprising 512 members (8 symmetry transformations of a square gives $8^3 = 512$ possibilities). Of course, we must modify our procedures appropriately once again:

```
  to shortdemo
    repeat 10 [demors random 512 wait 5000]
  end
  to demors :code
    cs setbgcolour [155 165 205] rt 180
    drs :code 7 240
    pu setpos [-210 -190] pd
    setheading 0 setpc "black tt :code
  end
```

```
to drs :code :n :s
  let "c div :code 8 let "a mod :code 8
  let "n1 mod :c 4 let "c div :c 4
  let "n2 mod :c 4 let "n3 div :c 4
  let "a1 180 * (mod :a 2) - 90 let "a div :a 2
  let "a2 180 * (mod :a 2) - 90 let "a3 180 * (div :a 2) - 90
  ars :n1 :n2 :n3 :a1 :a2 :a3 :n :s
end
to ars :n1 :n2 :n3 :a1 :a2 :a3 :n :s
  rs :n 90 :s [255 255 255] 128
  setpc "green3 setpw 2
  rs0 :n 90 :s
end
to rs :n :a :s :rgb :d
  setpc :rgb polygon list 4 list :s :a
  if :n = 0 [stop]
  lt :a repeat :n1 [jfd :s / 2 rt :a]
  rs :n – 1 :a1 :s / 2 :rgb - (list :d 0 0) :d / 2
  if :n1 > 0 [repeat 4 - :n1 [jfd :s / 2 rt :a]]
  lt :a repeat :n2 [jfd :s / 2 rt :a]
  rs :n – 1 :a2 :s / 2 :rgb - (list 0 :d  0) :d / 2
  if :n2 > 0 [repeat 4 - :n2 [jfd :s / 2 rt :a]]
  lt :a repeat :n3 [jfd :s / 2 rt :a]
  rs :n – 1 :a3 :s / 2 :rgb - (list 0 0 :d) :d / 2
  if :n3 > 0 [repeat 4 - :n3 [jfd :s / 2 rt :a]]
  lt :a
end
to rs0 :n :a :s
  if :n = 0 [dot stop]
  lt :a repeat :n1 [jfd :s / 2 rt :a]
  rs0 :n – 1 :a1 :s / 2
  if :n1 > 0 [repeat 4 - :n1 [jfd :s / 2 rt :a]]
  lt :a repeat :n2 [jfd :s / 2 rt :a]
  rs0 :n – 1 :a2 :s / 2
  if :n2 > 0 [repeat 4 - :n2 [jfd :s / 2 rt :a]]
  lt :a repeat :n3 [jfd :s / 2 rt :a]
  rs0 :n – 1 :a3 :s / 2
  if :n3 > 0 [repeat 4 - :n3 [jfd :s / 2 rt :a]]
  lt :a
end
```

Figure 9 shows examples of randomly chosen pictures drawn with the `shortdemo` procedure:

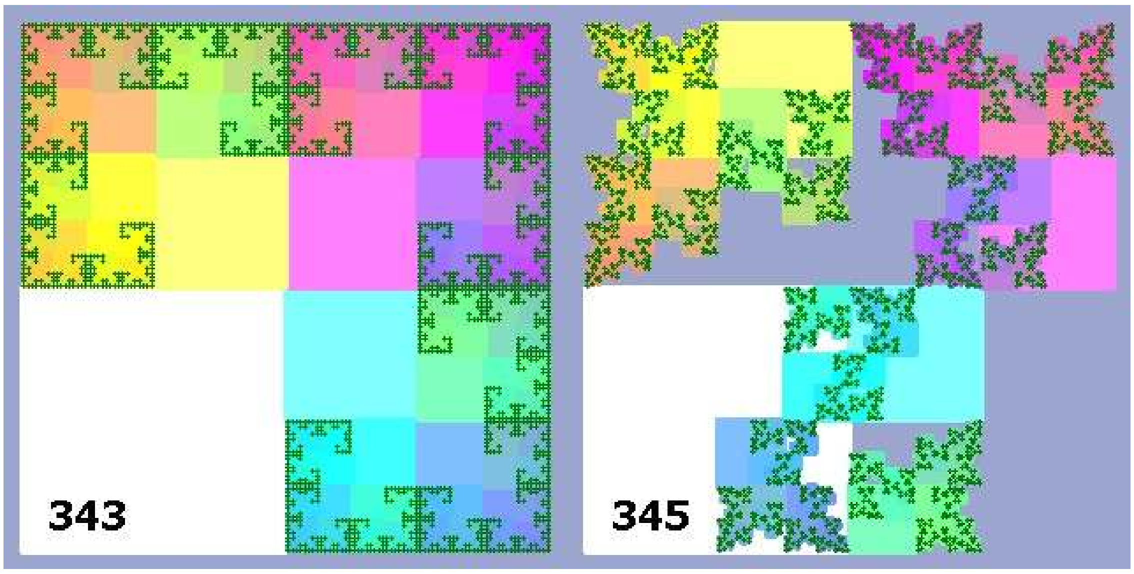

Figure 9. Two examples of extended *RS* family

After finishing this paper we have discovered an excellent book Frame & Mandelbrot ed. (2002), whose authors exploit similar ideas but in a different way and with different results.

References

Foltynowicz Izabella (2003). *The algorithmic Beauty of Recursive Structures*, EUROLOGO 2003, 91-101.

Frame M.,Mandelbrot B.B. ed. (2002). *Fractals, Graphics and Mathematics Education.*, The Mathematical Association of America.

Heinz-Otto Peitgen, Hartmut Jurgens, (1992a) Dietmar Saupe, *Chaos and Fractals. New Frontiers of Science*, Springer-Verlag, 244-251.

Peitgen H.O., RichterP. H (1986). *The Beauty of Fractals.* Springer-Verlag Berlin, Heidelberg.

Peitgen H.O., Jürgens H., Saupe D. (1992b). *Fractals for the Classroom*. Part 1: *Introduction to Fractals and Chaos*. Springer-Verlag New York.

Peitgen H.O., Jürgens H., Saupe D. (1992c). *Fractals for the Classroom*. Part 2: *Complex Systems and Mandelbrot Set*. Springer-Verlag New York.

Peitgen H.O., Jürgens H., Saupe D., Maletsky E., Perciante T, Yunker L. (1998). *Fractals for the Classroom*. *Strategic Activities Volume One*. NCTM, Springer-Verlag New York.

Wolfram S. (2002). *A New Kind of Science*. Wolfram Media, Inc.