

QS Attractor



Ikeda attractor

About the Program

This program is a collection of various chaotic attractors and related mathematical images. It originally was written for DOS in 1994. This revision is a 32-bit Windows program which should continue to work in current 64-bit Windows versions.

The final 12 formulae in the program are from Ramiro Perez, who sent a DOS program script to the FRAC-L listserv on 12-23-94. This is part of his message to the group:

Here's a program that I've made, which displays 40 strange attractors in SVGA....
I offer this program as a Christmas gift to FRAC-L people.

Ramiro's Web site, *Incendia*, features his 3D fractal engine based on IFS:

<https://www.incendia.net/index.php>

Running the Program

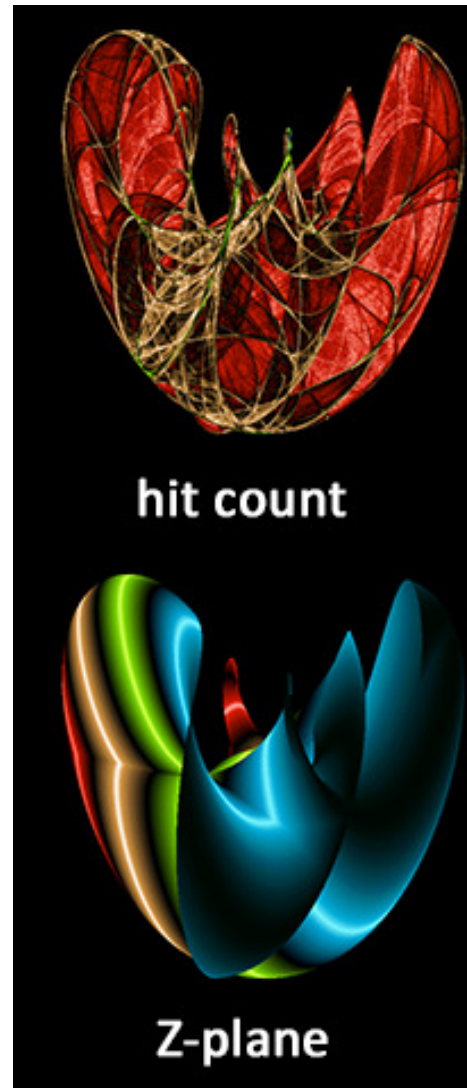
Images can be rendered in resolutions of up to 4000 x 4000 pixels. Rendering, pausing and resuming are controlled by the **R** key, or by left or right mouse button clicks. Rendering must be paused to access the menu options. There are two default palettes. You also can select any valid Fractint MAP file. The background can be specified as a triplet of RGB values. The image window can be adjusted for zooming in or out. Several image types can be rotated around any or all of the X, Y and Z axes.

Images can be saved as TARGA files. Parameter sets can be saved as “ATR” files and can be loaded into the program later. After associating these files with the program, you then can load them by double-clicking on their icons to start the program.

For several formulae, random sets of parameters can be generated. However, chaotic attractors are by nature unpredictable. Many parameter sets will converge to fixed points, lines or periodic closed curves, yielding uninteresting images. Some of the techniques that can be used to screen for stable and interesting images are described in Appendix 1 of this file.

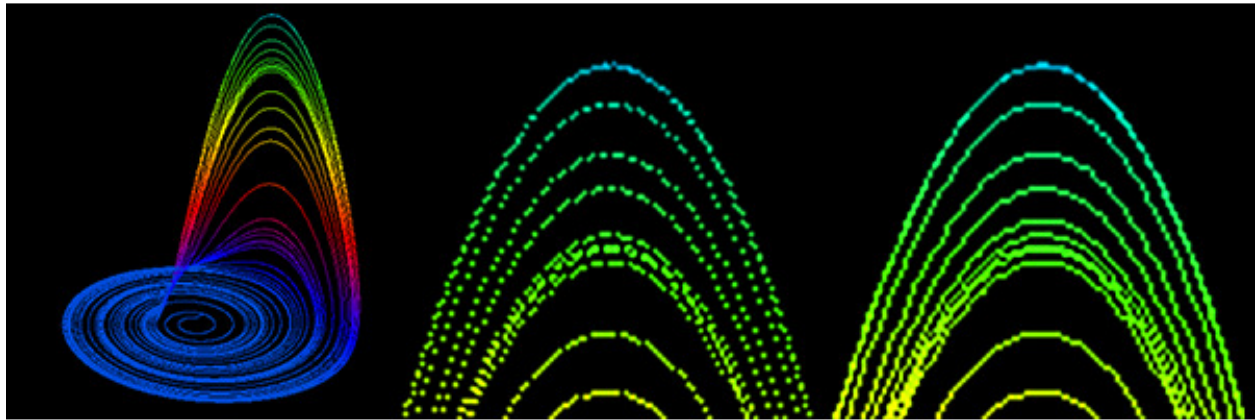
There are three coloring methods. The “hit count” algorithm advances the color of each pixel through a 256-color palette, each time it is hit during iteration of the formula. Therefore it takes time for the colors of the image to emerge, much like a photographic print in developing fluid. So be patient and give the image a chance. However, as iteration continues, progressively more points go beyond 255 hits and are colored with the last palette entry, “overdeveloping” the image. Ultimately, most of the image can assume that one color. To compensate for these trends, palette entries can be redistributed through the normalization and equalization options. Normalization compresses or expands the distribution into the range of 1 to 255. However, this significantly under-represents the upper range of the palette, though this still might produce appealing images. An alternative to linear normalization uses the logarithms of the hit counts, which tends to emphasize the middle range of the palette. Equalization is an attempt to spread the color values evenly throughout the image, making better use of the entire palette. Further discussion of these techniques can be found in Appendix 2 of this file.

The “Z-plane” algorithm colors pixels according to the position along the Z axis of their associated iterated points. This gives the images a more “solid” character.



The “speed” algorithm is used frequently for chaotic attractors. It uses the distance between the currently-calculated position and the previous one as an indicator of how “quickly” the point had

to move to get there, assuming a constant iteration rate, and assigns palette colors accordingly. For several formulae, such as the Rössler attractor, points can be connected by lines.



Several formulae have parameters that can be changed. These are indicated in the “Attractor Type” dialog box with a “+” symbol. Those formulae which can have random parameters selected by the computer are indicated with a “\$” symbol. These options are summarized in the following table, which also can be displayed within the program via the “Options” item in the “Help” menu.

attractor type	rendering speed	formula parameters	rotation	image boundaries	random parameters
Lorenz - 3D	√		√		
Ganymede		√		√	√
Pickover - 3D		√	√	√	√
Peter de Jong		√		√	√
Henon Phase Diagram		√		√	
IFS		√		√	
Duffing Oscillator	√				
Hopalong				√	
Volterra-Lotka		√		√	
Lace		√			
Popcorn	√	√			
Gingerbread Men		√			
Unity				√	
Chrysanthemum - 3D			√		
Rössler - 3D	√		√		
Butterfly - 3D			√		
KAM Torus - 3D		√	√		
Iris		√		√	√
Atom		√		√	√
Wings		√		√	√
Medusa				√	
Disk				√	
Ikeda IFS				√	
Storm Julia		√		√	√
Frost Julia		√		√	√
Star				√	
Spirals		√		√	√
Cosine Waves				√	

Attractor Formulae

1 - Hénon

$$x_{n+1} = y_n + 1.0 - (a * x_n * x_n);$$
$$y_{n+1} = b * x_n;$$

$$x_0 = 0.1; y_0 = 0.1$$

$$a = 1.4; b = 0.3$$

Many values for a and b diverge to infinity.

2 - Ikeda

$$x_{n+1} = \rho + c2 * (x_n * \cos(\text{temp}) - y_n * \sin(\text{temp}));$$
$$y_{n+1} = c2 * (x_n * \sin(\text{temp}) + y_n * \cos(\text{temp}));$$

$$x_0 = 0.1; y_0 = 0.1$$

$$c1 = 0.4; c2 = 0.9; c3 = 6.0; \rho = 1.0$$

$$\text{temp} = c1 - c3 / (1.0 + x_n * x_n + y_n * y_n);$$

3 - Bifurcation (Feigenbaum) Diagram

$$x_{n+1} = a * x_n * (1.0 - x_n) \quad (\text{"logistic equation"})$$

$$x_0 = 0.01$$

iterate for $2.9 < a \leq 4.0$ (diverges if > 4.0)

for each value of a, normalize $x_{200} - x_{300}$ between $-0.1 < y < 2.2$ and plot (a, y)

to plot "critical lines", set $x_0 = 0.5$ (critical point)

for each value of a, iterate just from 0 to k for a set of small values of k (e.g. 1 to 8)

For symmetrical diagrams, select an "odd" function, e.g. $x_{n+1} = a * x_n * (1.0 - x_n^2)$, and iterate both "halves" by choosing positive and negative initial values. Cf. Field, Michael and Golubitsky, Martin, *Symmetry in Chaos*, Oxford University Press, 1992, pp. 108-12.

4 - Lorenz

E.N. Lorenz's simplified atmospheric convection model (1962):

From *The Chaos Hypertextbook* - Glenn Elert:

<http://hypertextbook.com/chaos>

Imagine a rectangular slice of air heated from below and cooled from above by edges kept at constant temperatures. This is our atmosphere in its simplest description. The bottom is heated

by the earth and the top is cooled by the void of outer space. Within this slice, warm air rises and cool air sinks. In the model as in the atmosphere, convection cells develop, transferring heat from bottom to top.

The state of the atmosphere in this model can be completely described by three time-evolving variables:

x = the convective flow
y = the horizontal temperature distribution
z = the vertical temperature distribution

with three parameters describing the character of the model itself:

σ [sigma] = the ratio of viscosity to thermal conductivity
 ρ [rho] = the temperature difference between the top and bottom of the slice
 β [beta] = the width to height ratio of the slice

and three ordinary differential equations describing the appropriate laws of fluid dynamics

$$\begin{aligned} dx/dt &= \sigma (y - x) \\ dy/dt &= (\rho - z) * x - y \\ dz/dt &= x * y - \beta * z \end{aligned}$$

$$x_0 = y_0 = z_0 = 0.6$$

$$\sigma = 10; \rho = 28; \beta = 8/3$$

$$\sigma = 28; \rho = 46.92; \beta = 4 \quad (\text{alternate constants})$$

$$dx = x_{n+1} - x_n; dy = y_{n+1} - y_n; dz = z_{n+1} - z_n$$

To discretize the equations, a value of 0.005 yields acceptable results; thus:

$$x_{n+1} = x_n + 0.005 * \sigma (y_n - x_n)$$

$$y_{n+1} = y_n + 0.005 * (\rho - z_n) * x_n - y_n$$

$$z_{n+1} = z_n + 0.005 * x_n * y_n - \beta * z_n$$

5 - Ganymede

Cliff Pickover's "standard" Ganymede formula

Cf. Chaos in Wonderland, St. Martin's Press, 1994

To add a third dimension, $z_{(n+1)} = \sin(x_n)$ also is calculated.

$$\begin{aligned} x_{n+1} &= \sin(y_n * B) + C * \sin(x_n * B) \\ y_{n+1} &= \sin(x_n * A) + D * \sin(y_n * A) \end{aligned}$$

$$x_0 = 0.1; y_0 = 0.1$$

$$A \text{ and } B: -3.0 \text{ to } 3.0; C \text{ and } D: -1.5 - 1.5$$

The book also utilizes 3 "mutations" of the standard formula. They can be explored in the program *QS Ganymede*.

Alpha

$$x_{n+1} = \sin(y_n * B) + \sin^2(x_n * B) + \sin^3(x_n * B)$$

$$y_{n+1} = \sin(x_n * A) + \sin^2(y_n * A) + \sin^3(y_n * C)$$

Beta

$$x_{n+1} = \sin(y_n * B) + \sin^2(x_n * B)$$

$$y_{n+1} = \sin(x_n * A) + \sin^2(y_n * A)$$

Gamma

$$x_{n+1} = |\sin(y_n * B)| + \sin^2(x_n * B)$$

$$y_{n+1} = |\sin(x_n * A)| + \sin^2(y_n * A)$$

Latööcarfian Dreams: Sample Parameter Sets

Formula	A	B	C	D
standard	-0.966918	2.879879	0.765145	0.744728
standard	-2.905148	-2.030427	1.44055	0.70307
standard	-2.951292	1.187750	0.517396	1.090625
standard	2.668752	1.225105	0.709998	0.637272
standard	1.380932	2.656301	1.157857	1.272576
standard	2.733940	1.369945	1.471923	0.869182
standard	1.008118	2.65392	0.599124	0.6507
standard	-2.767266	-0.633839	1.352107	0.705481
standard	-0.299661	-2.315714	1.071856	1.404386
standard	-2.164647	-0.641713	1.277032	1.003342
standard	1.966155	-1.005921	1.091876	1.466765
standard	-2.570031	-1.395348	1.039303	1.214590
standard	-2.759713	1.710489	1.483123	1.263417
standard	2.848162	1.503699	0.898574	1.112473
standard	-2.905148	-2.030427	1.44055	0.70307
alpha	0.992187	-1.111942	-1.713095	
alpha	-1.804285	-2.513108	0.957945	
alpha	2.570322	1.125386	0.810594	
alpha	1.811489	1.318427	1.282961	
alpha	0.956008	-2.994230	1.387227	
beta	2.755364	2.791253		
beta	2.039949	1.322977		
beta	-1.894918	1.579553		
gamma	0.805414	2.132054		
gamma	-1.918189	2.382361		

6 - Pickover 3D

A similar 3-dimensional attractor

$$\begin{aligned}x_{n+1} &= \sin(A * y_n) - z_n * \cos(B * x_n) \\y_{n+1} &= z_n * \sin(C * x_n) - \cos(D * y_n) \\z_{(n+1)} &= \sin(x_n)\end{aligned}$$

$$x_0 = y_0 = z_0 = 0.1$$

A, B, C and D: -3.0 to 3.0

7- Peter de Jong

Another similar 3-dimensional attractor

$$\begin{aligned}x_{n+1} &= \sin(A * y_n) - \cos(B * x_n); \\y_{n+1} &= \sin(C * x_n) - \cos(D * y_n) \\z_{(n+1)} &= \sin(x_n)\end{aligned}$$

$$x_0 = y_0 = 0.1$$

A, B, C and D: -3.0 to 3.0

A	B	C	D
1.641	1.902	0.316	1.525
0.970	-1.899	1.381	-1.506
1.4	-2.3	2.4	-2.1
2.01	-2.53	1.61	-0.33
-2.7	-0.09	-0.86	-2.2
-0.827	-1.637	1.659	-0.943
-2.24	0.43	-0.65	-2.43
-2.0	-2.0	-1.2	2.0
-0.709	1.638	0.452	1.74

8 - Hénon Phase Diagram

$$\begin{aligned}x_{n+1} &= x_n * \cos(\theta) - (y_n - x_n^2) * \sin(\theta) \\y_{n+1} &= x_n * \sin(\theta) + (y_n - x_n^2) * \cos(\theta)\end{aligned}$$

$$x_0 = y_0 = 0.1$$

Every 1000 iterations, reverse the signs of x_n and y_n

If the size of (x_n, y_n) is diverging, reset them randomly to values between -1 and 1

Sample values of θ (in degrees):

65 (default) 1 90 92 109 115 126 147 181

Examples at: <http://paulbourke.net/fractals/henonphase>

9 - Sierpinski Polygons as a Chaos Game

The chaos game: plot next point at $r * [\text{distance between random vertex and current point}]$.

$$\begin{aligned} v &= \text{random}(nv) \\ x_{n+1} &= \text{vertex}[v].x + (x_n - \text{vertex}[v].x) * r \\ y_{n+1} &= \text{vertex}[v].y + (y_n - \text{vertex}[v].y) * r \end{aligned}$$

nv = number of vertices

$vh = [\text{pixel resolution}] * (\sqrt{3} / 2)$: can be used as a “virtual height” to maintain the aspect ratio for equilateral triangles and hexagons

$$r = \frac{1}{2 * \left(1 + \sum_{k=1}^{[n/4]} \cos(2\pi k / n) \right)}$$

$nv = 3, r = 1/2$

$nv = 4, r = 1/2, 1/3$

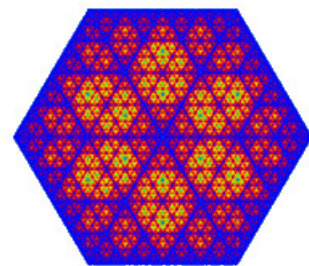
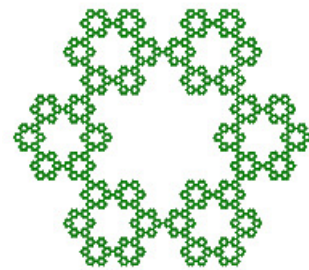
$nv = 5, r = 0.38196601$

$nv = 6, r = 1/3$

For pentagons and hexagons, if $r = 1/2$, the sub-polygons will overlap; in this situation, hit count coloring can produce interesting images; this option is available in *QS Attractor*.

Cf. Heinz-Otto Peitgen et al., Chaos and Fractals
(Springer Verlag, 1992), Chapter 2

Cf. Barnsley, Michael, in Peitgen and Dietmar Saupe, eds.,
The Science of Fractal Images, Springer Verlag,
1988, Chapter 5



In *Chaos and Fractals*, this code, termed “Sierpinski Gasket by Binary Addresses” is shown to be a complete function for plotting a Sierpinski triangle. The base of the triangle will continue to expand with y_res , so to cut the image off at a solid line, “ y_res ” must be a power of 2.

```
for (j = 0; j < y_res; j++)
  for (i = 0; i < j; i++)
    if ((i & (j - i)) == 0) PlotPixel(i + ((y_res >> 1) + 30) - (j >> 1), (j) + 30, color)
```

10 - Barnsley Fern

Cf. Barnsley, Chapter 5 in *The Science of Fractal Images*

Cf. Barnsley, *Fractals Everywhere*, Academic Press, 1988

Cf. *Chaos and Fractals*, Chapter 5

Barnsley's fern uses four affine transformations. The formula for one transformation is:

$$f(x,y) = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \quad \text{or} \quad \begin{aligned} x_{n+1} &= a * x_n + b * y_n + e \\ y_{n+1} &= c * x_n + d * y_n + f \end{aligned}$$

Barnsley shows the IFS code for his Black Spleenwort *Asplenium adiantum* fern fractal as the transformations shown in the table. "p" is the probability factor for each transformation.

a	b	c	d	e	f	p	Portion generated
0.0	0.0	0.0	0.16	0.0	0.0	0.01	Stem
0.85	0.04	-0.04	0.85	0.0	1.60	0.85	Successively smaller leaflets
0.20	-0.26	0.23	0.22	0.0	1.60	0.07	Largest left-hand leaflet
-0.15	0.28	0.26	0.24	0.0	0.44	0.07	Largest right-hand leaflet

Variations by David Nicholls - Canberra

<http://www.home.aone.net.au/~byzantium/ferns/fractal.html>

a	b	c	d	e	f	p	
0.0	0.0	0.0	0.2	0.0	-0.12	0.01	- Barnsley variation
0.845	0.035	-0.035	0.82	0.0	1.6	0.85	
0.2	-0.31	0.255	0.245	0.0	0.29	0.07	
-0.15	0.24	0.25	0.2	0.0	0.68	0.07	
0.0	0.0	0.0	0.25	0.0	-0.14	0.02	- <i>Calochlaenia</i> / <i>Culcita</i>
0.85	0.02	-0.02	0.83	0.0	1.0	0.84	
0.09	-0.28	0.3	0.11	0.0	0.6	0.07	
-0.09	0.28	0.3	0.09	0.0	0.7	0.07	
0.0	0.0	0.0	0.25	0.0	-0.4	0.02	- "fishbone"
0.95	0.002	-0.002	0.93	-0.002	0.5	0.84	
0.035	-0.11	0.27	0.01	-0.05	0.005	0.07	
-0.04	0.11	0.27	0.01	0.047	0.06	0.07	
0.0	0.0	0.0	0.25	0.0	-0.4	0.02	- <i>Cyclosorus</i>
0.95	0.005	-0.005	0.93	-0.002	0.5	0.84	
0.035	-0.2	0.16	0.04	-0.09	0.02	0.07	
-0.04	0.2	0.16	0.04	0.083	0.12	0.07	

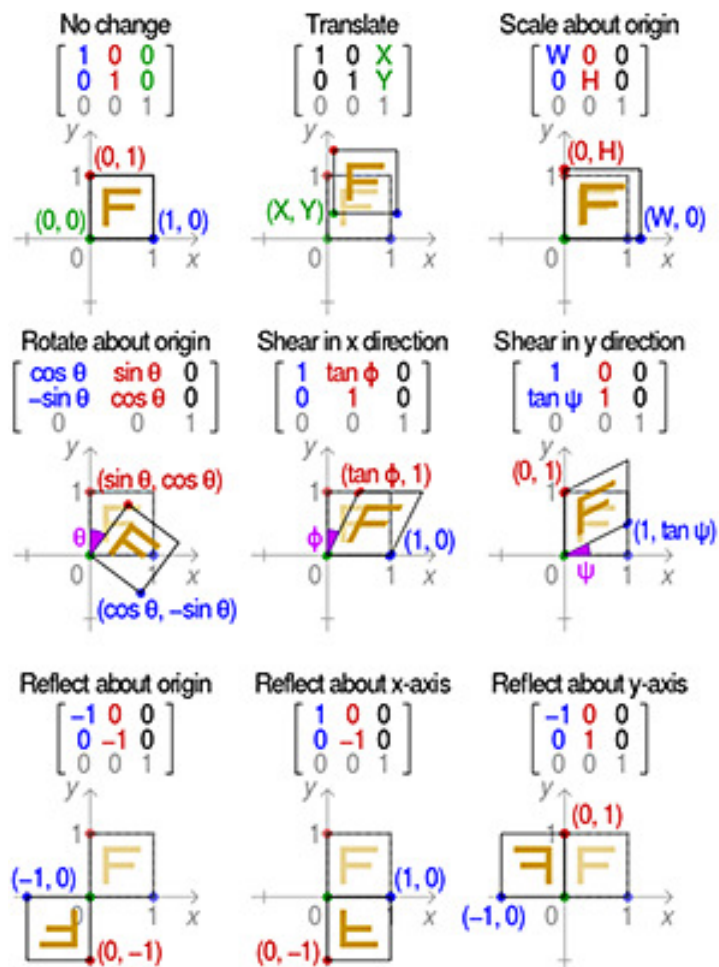
Sample code for these transformations (multiply p by 100):

```
rp = random(100);
if (rp < p1)
{
    xn+1 = a1 * xn + b1 * yn + e1
    yn+1 = c1 * xn + d1 * yn + f1 }
else if (rp >= p1 && rp < p1 + p2)
{
    xn+1 = a2 * xn + b2 * yn + e2
    yn+1 = c2 * xn + d2 * yn + f2 }
else if (rp >= p1 + p2 && rp < p1 + p2 + p3)
{
    xn+1 = a3 * xn + b3 * yn + e3
    yn+1 = c3 * xn + d3 * yn + f3 }
else
{
    xn+1 = a4 * xn + b4 * yn + e4
    yn+1 = c4 * xn + d4 * yn + f4 }
```

$x_0 = y_0 = 0.1$



These are typical transformations. When multiple transformations are applied, the order usually is important; this is reflected by the fact that multiplication of matrices is not commutative.



These are additional sets, many of them from *Chaos and Fractals*, Chapter 5. The probability factors should be roughly proportional to the areas of the rectangles.

Also cf. Larry Riddle's IFS site at: <http://ecademy.agnesscott.edu/~lriddle/ifs/ifs.htm>

a	b	c	d	e	f	
+/- 0.5	0.0	0.0	+/- 0.5	0.0	0.0	Sierpinski triangle
+/- 0.5	0.0	0.0	+/- 0.5	0.5	0.0	and relatives
+/- 0.5	0.0	0.0	+/- 0.5	0.0 / 0.25	0.5 / 0.433	0.25 / 0.433 for equilateral
0.46	0.0	0.0	0.46	0.0	0.0	Sierpinski square
0.46	0.0	0.0	0.46	0.5	0.0	
0.46	0.0	0.0	0.46	0.0	0.5	
0.46	0.0	0.0	0.46	0.5	0.5	
0.382	0.0	0.0	0.382	0.3072	0.619	Sierpinski pentagon
0.382	0.0	0.0	0.382	0.6033	0.4044	
0.382	0.0	0.0	0.382	0.0139	0.4044	
0.382	0.0	0.0	0.382	0.1253	0.0595	
0.382	0.0	0.0	0.382	0.492	0.0595	
0.0	-0.5	0.5	0.0	0.5	0.0	twin Christmas tree
0.0	0.5	-0.5	0.0	0.5	0.5	
0.5	0.0	0.0	0.5	0.25	0.5	
0.0	0.577	-0.577	0.0	0.0951	0.5893	dragon
0.0	0.577	-0.577	0.0	0.4413	0.7893	
0.0	0.577	-0.577	0.0	0.0952	0.9893	
0.333	0.0	0.0	0.333	0.333	0.666	Cantor maze
0.0	0.333	1.0	0.0	0.666	0.0	
0.0	-0.333	1.0	0.0	0.333	0.0	
$r \cos \theta$	$-r \sin \theta$	$r \sin \theta$	$r \cos \theta$	0	1	binary / Pythagorean tree
$r \cos \theta$	$r \sin \theta$	$-r \sin \theta$	$r \cos \theta$	0	1	r = scaling factor
1.0	0.0	0.0	1.0	0.0	0.0	If $a == 0.0$, trunk is line
0.33	0.0	0.0	0.33	0.33	0.0	Vicsek fractal
0.33	0.0	0.0	0.33	0.0	0.33	
0.33	0.0	0.0	0.33	0.33	0.33	
0.33	0.0	0.0	0.33	0.67	0.33	
0.33	0.0	0.0	0.33	0.33	0.67	

a	b	c	d	e	f	
0.255	0.0	0.0	0.255	0.3726	0.6714	crystal
0.255	0.0	0.0	0.255	0.1146	0.2232	
0.255	0.0	0.0	0.255	0.6306	0.2232	
0.37	-0.642	0.642	0.37	0.6356	-0.0061	
0.341	-0.071	0.071	0.341	0.0	0.0	pentadendrite
0.341	-0.071	0.071	0.341	0.649	0.136	composite image
0.341	-0.071	0.071	0.341	0.071	0.659	
0.341	-0.071	0.071	0.341	-0.604	0.291	
0.341	-0.071	0.071	0.341	-0.445	-0.491	
0.341	-0.071	0.071	0.341	0.33	-0.575	
0.195	-0.488	0.344	0.443	0.4431	0.2452	Tree
0.462	0.414	-0.252	0.361	0.2511	0.5692	
-0.058	-0.07	0.453	-0.111	0.5976	0.0969	
-0.035	0.07	-0.469	-0.022	0.4884	0.5069	
-0.637	0.0	0.0	0.501	0.8562	0.2513	

11 - Feather

$$\begin{aligned}
 x_{n+1} &= B * y_n + w \\
 u &= x_{n+1} * x_{n+1} \\
 w &= A * x_{n+1} + ((C * u) / (1.0 + u)) \\
 y_{n+1} &= w - x_n
 \end{aligned}$$

$$\begin{aligned}
 x_0 &= 3.0; y_0 = 0.0 \\
 w_0 &= A * x_0 + (C * x_0^2) / (1.0 + x_0^2) = 1.224 \\
 A &= -0.48; B = 0.93 \\
 C &= 2.0 - (2.0 * A) = 2.96
 \end{aligned}$$

12 - Duffing Oscillator

$$C = 0.293517; D = 0.472627$$

$$\begin{aligned}
 b &= 0.01 * a \\
 x_{n+1} &= x_n + y_n / 2\pi \\
 y_{n+1} &= y_n + (x_n - x_n^3 - C * y_n + D * \cos(b)) / 2\pi; \\
 a &= a + 1.0 \\
 \text{if } (a > 627.0) &a = 0.0
 \end{aligned}$$

$$x_0: 0.157; y_0: -0.045; a_0 = 0.0$$

13 - Hopalong

credited to Barry Martin in Fractint

a: 1.67; b: 0.556; c: 4.6

a: 0.4; b: 1.0; c: 0.0 (original values)

if ($x_n < 0.0$) sign = -1.0; else sign = 1.0

$x_{n+1} = y_n - \text{sign} * \sqrt{(|b * x_n - c|)}$

$y_{n+1} = a - x_n$

$x_0 = y_0 = 0.0$

14 - Sierpinski Web

for each pixel:

z = complex(pixel_x, pixel_y)

$zx_{n+1} = zx_n * (zx_n^2 - 3.0 * zy_n^2)$ /* cube of complex number */

$zy_{n+1} = zy_n * (3.0 * zx_n^2 - zy_n^2)$

$zx_{n+1} = 0.2 * |zx_{n+1}|$

$zy_{n+1} = 0.2 * |zy_{n+1}|$

size = 0.0; n = 0;

while (size <= 16.0 && n++ < 256)

{ $zx_{n+1} = zx_{n+1} * 2.0$;

$zy_{n+1} = zy_{n+1} * 2.0$;

if ($zy_{n+1} > 1.0$) $zy_{n+1} = zy_{n+1} - 1.0$

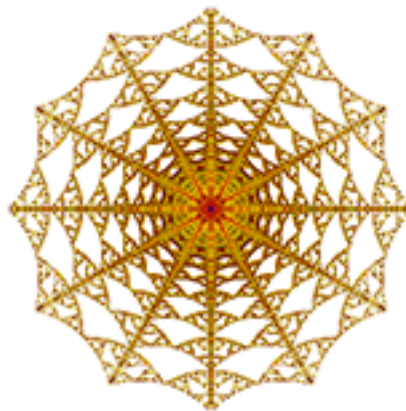
else if ($zx_{n+1} > 1.0$) $zx_{n+1} = zx_{n+1} - 1.0$;

size = $zx_{n+1}^2 + zy_{n+1}^2$ }

Plotting points when $n >$ a small value, e.g. 9, allows the background to show

A escape-time fractal, attributed to Javier Barrallo.

<http://members.tripod.com/vismath7/proceedings/barrallo.htm>



15 - Volterra-Lotka Predator-Prey Equations

$h = \rho = 0.739$ (default); $A = B = C = D = 1.0$

$$\begin{aligned} dx / x &= A - By & \text{or} & \quad dx = Ax - Bxy = f(x, y) \\ dy / y &= -C + Dx & \text{or} & \quad dy = -Cy + Dxy = g(x, y) \end{aligned}$$

These differential equations can be discretized to yield individual points with Euler's method:

$$\begin{aligned} x_{n+1} &= x_n + h * f(x_n, y_n) \\ y_{n+1} &= y_n + h * g(x_n, y_n) \end{aligned}$$

But unless h is very small, all points except for $(1.0, 1.0)$ diverge to infinity (see below). Euler's method can be improved upon with other methods. In *The Beauty of Fractals* (Springer Verlag 1986, Chapter 8) Peitgen and Peter H. Richter employ Huen's method as follows, with $h == \rho$:

$$\begin{aligned} x_{n+1} &= x_n + h/2 * \{ f(x_n, y_n) + f[x_n + \rho f(x_n, y_n), y_n + \rho g(x_n, y_n)] \} \\ y_{n+1} &= y_n + h/2 * \{ g(x_n, y_n) + g[x_n + \rho f(x_n, y_n), y_n + \rho g(x_n, y_n)] \} \end{aligned}$$

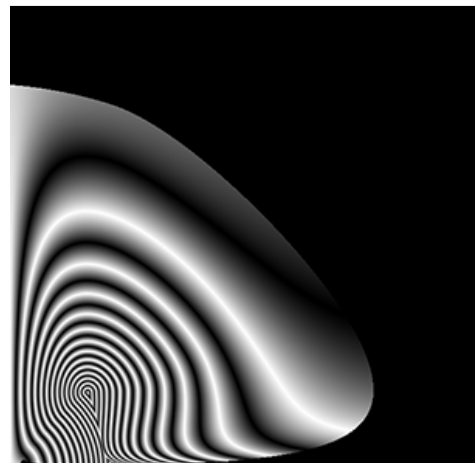
Then they allow h and ρ to differ. This results in a number of possibilities in which there are different types of finite attractors (see below).

This can be coded as an escape-time fractal as follows:

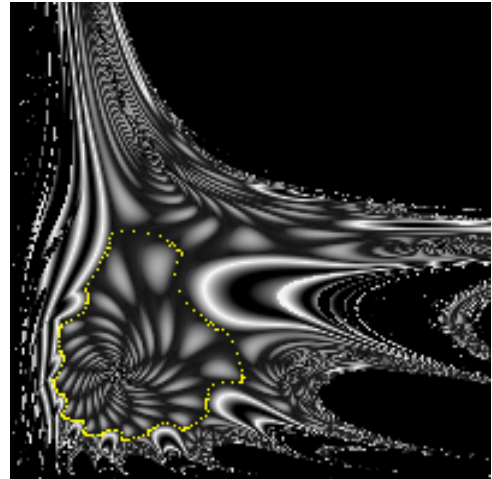
```
for each pixel, x = pixel_x; y = pixel_y; size = 0.0; n = 0
while (size <= maxsize && n++ < maxiterations)
{
    xy = x_n * y_n
    fxy = x_n - xy
    gxy = -y_n + xy
    xpfxy = x_n + rho * fxy
    ypgxy = y_n + rho * gxy
    x_{n+1} = x_n + h / 2 * (fxy + xpfxy - (xpfxy * ypgxy))
    y_{n+1} = y_n + h / 2 * (gxy - ypgxy + (xpfxy * ypgxy))
    size = x_{n+1}^2 + y_{n+1}^2
}
```

if (size <= maxsize) $\rightarrow$ plot these points (which do not escape) with a suitable color scheme. *QS Attractor* uses two options: colors assigned according to relative size of the iterated point, or by level sets of equal attraction to the attractor(s).

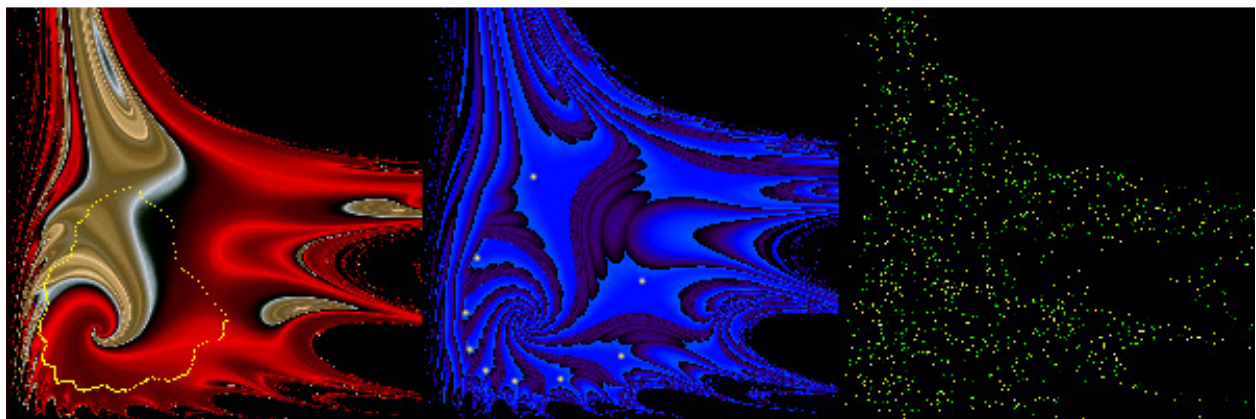
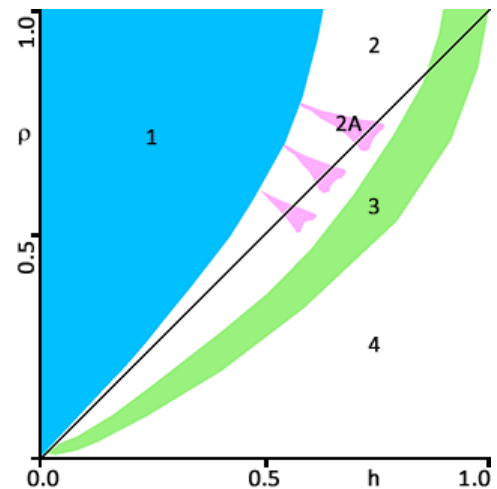
This image was rendered using Euler's method with $h = 0.015$ ($\rho = 0.0$). The range of x and y is 0.0 to 6.0 . As the value of h increases, the shaded portion of the image shrinks, until all that is left is a dot at $(1.0, 1.0)$.



This image was rendered using Huen's method with $h = \rho = 0.739$. The range of x and y is 0.0 to 4.5. The points which do not escape are attracted to an irregularly-shaped invariant circle, shown in yellow, which surrounds the point (1.0, 1.0). These parameters are located in Region 2, as described below.



Peitgen and Richter present a plot of h versus ρ , showing the system characteristics for different pairs of these parameters. In Region 1, the fixed point (1.0, 1.0) is attractive. In Region 2, the attractor becomes an invariant circle. In Region 2A, for which only three of the infinitely many components are shown, the attractor becomes a periodic orbit. In Region 3, the system becomes progressively chaotic, and in Region 4, ultimately all the orbits diverge to infinity, except for (1.0, 1.0) and cases of Euler's method where $\rho = 0.0$ and h is small.



Region 2
 $h = 0.675$ $\rho = 0.655$
invariant circle attractor

Region 2A
 $h = 0.7$ $\rho = 0.7$
orbital attractor, period 9

Region 3
 $h = 0.65$ $\rho = 0.5$
chaotic attractor

16 - Lace

```
r0 =  $\sqrt{x_n^2 + y_n^2}$ 
switch (random(4))
case 0:
    w = atan2(yn, xn - 1)
    yn+1 = -r0 * cos(w) / r + 1.0
    xn+1 = -r0 * sin(w) / r
case 1:
    w = atan2(yn -  $\sqrt{3} / 2.0$ , xn + 0.5)
    yn+1 = -r0 * cos(w) / r - 0.5
    xn+1 = -r0 * sin(w) / r +  $\sqrt{3} / 2.0$ 
case 2:
    w = atan2(yn +  $\sqrt{3} / 2.0$ , xn + 0.5)
    yn+1 = -r0 * cos(w) / r - 0.5
    xn+1 = -r0 * sin(w) / r -  $\sqrt{3} / 2.0$ 
case 3:
    w = atan2(yn, xn)
    yn+1 = -r0 * cos(w) / r
    xn+1 = -r0 * sin(w) / r
```

$x_0 = 0.5$; $y_0 = 0.75$; $r = 2.0$ (range 1.5 - 4.0)

based on code by Roger Bagula and Robert Rodgers at <http://paulbourke.net/fractals/lace>

17 - Popcorn

credited to Richard Sherry in Fractint

for each pixel, $x = \text{pixel_x}$; $y = \text{pixel_y}$
h: 0.1 - 0.5, default 0.15

```
for (n = 0; n < 1; n++)
{
    xn+1 = xn - h * sin(yn + tan(yn * 3.0))
    yn+1 = yn - h * sin(xn + tan(xn * 3.0)) }
}
```

iterating more than once can overwhelm the screen

18 - Perturbated Gingerbread Men

A modification of the classic gingerbread men attractor, by Fausto Barbuto and Victor Ielo.

$$x_{n+1} = 1.0 - y_n + |x_n| + F$$

$$y_{n+1} = x_n$$

x_0, y_0 = seed values between -10 and +10

F = choice of perturbation functions, with transcendental element(s):

$$F = 0.0 \quad \text{classic formula}$$

$$F = -0.000002 * \tan(y_n)$$

$$F = 0.0001 * \cos(y_n) * |x_n|$$

$$F = 0.0001 * \cos(|y_n|) * \sin(|x_n|)$$

$$F = 0.00001 * \sin(|x_n|) / \cos(|y_n|)$$

$$F = 0.000005 * \cos(|y_n|) / \sin(|y_n|)$$

$$F = 0.00005 * \cos(|y_n|) / \sin(|x_n|)$$

$$F = 0.00001 * \cos(|y_n|) / \cos(|x_n|)$$

19 - Frothy Basin

Original escape time version by James C. Alexander, University of Maryland via Fractint.
Cf. Timothy Wegner and Bert Tyler, *Fractal Creations*, Waite Group Press, 1993, p.324

Also can be iterated from one starting point as an attractor.

```
for each pixel, z = complex(pixel_x + pixel_y * i)
    z = conjugate of z = complex(pixel_x - pixel_y * i);
    c = complex(1.0, 1.02871376822 * i)
    size = 0.0; n = 0
```

```
while (size <= maxsize && n++ < maxiterations)
```

```
{
    zn+1 = zn2 - c *  $\overline{z}$ 
    size = | z | }
}
```

if (size <= maxsize) → plot these points (which do not escape) with a suitable color scheme.

20 - Unity

An escape time fractal credited to Mark Peterson in *Fractint* - *cf. Fractal Creations*, p. 325
Optionally can be rendered with inversion

```
for each pixel, x = pixel_x; y = pixel_y; size = 0.0; n = 0
while ((size <= maxsize) && (n < maxiter))
{
    one =  $x_n^2 + y_n^2$ 
    if (one > 2.0) break
     $y_{n+1} = (2.0 - one) * x_n$ ;
     $x_{n+1} = (2.0 - one) * y_{n+1}$ ;
    size =  $x_{n+1}^2 + y_{n+1}^2$ 
}
```

if (size <= maxsize) → plot these points (which do not escape) with a suitable inside coloring scheme.

Otherwise, plot the points with a suitable outside coloring scheme.

21 - Chrysanthemum

3D chrysanthemum attributed to Temple Fay at <http://paulbourke.net/geometry/chrysanthemum/>

Arbitrarily iterate in sets of (maxcount = 10,000); initialize count to 0

```
 $\theta = \text{count} * 21.0 * \pi / \text{maxcount}$ 
 $p = \sin(17.0 * \theta / 3.0)$ 
 $q = \sin(2.0 * \cos(3.0 * \theta) - 28.0 * \theta)$ ;
 $r = 5.0 * (1.0 + \sin(11.0 * \theta / 5.0)) - 4.0 * p^4 * q^8$ 
 $x_{\text{count}} = r * \cos\theta$  [ * scaling factor]
 $y_{\text{count}} = r * \sin\theta$  [ * scaling factor]
 $z_{\text{count}} = ((r / 3.0) * \sin(r * 2\pi / 7))$  [ * scaling factor]
count++
```

22 - Rössler

Otto Rössler's "equation for continuous chaos" (1976).
Cf. discussion in *Chaos and Fractals*, Chapter 12.3

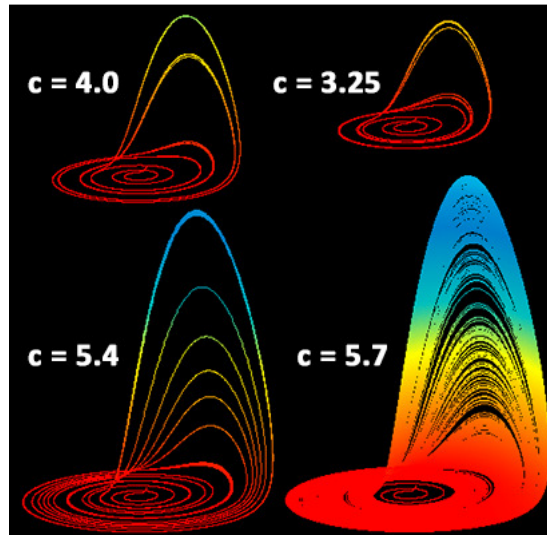
$$\begin{aligned} dx/dt &= -(y + z) \\ dy/dt &= x + a * y \\ dz/dt &= b + x * z - c * z \end{aligned}$$

$x_0 = y_0 = z_0 = 0.6$
 $a = b = 2.0$; $2.5 < c < 10.0$

$dx = x_{n+1} - x_n$; $dy = y_{n+1} - y_n$; $dz = z_{n+1} - z_n$
 dt approaches 0; for this program a value of 0.005 yields acceptable results; thus:

$$\begin{aligned}x_{n+1} &= x_n + 0.005 * (-y_n - z_n) \\y_{n+1} &= y_n + 0.005 * (x_n + a * y_n); \\z_{n+1} &= z_n + 0.005 * (b + x_n * z_n - c * z_n)\end{aligned}$$

Peitgen uses a Feigenbaum diagram to show that the system undergoes period-doubling as the value of c increases. Values like 3.25, 4.0 and 5.4 yield loops that are periodic, while 5.7 yields a chaotic system.



23 - Butterfly

Cf. Chaos in Wonderland, p. 262

Arbitrarily iterate in sets of (maxcount = 10,000); initialize count to 0

```
 $\theta = \text{count} * 24.0 * \pi / \text{maxcount}$ 
 $r = e^{\cos \theta} - 2.0 * \cos(4.0 * \theta) + \sin(\theta / 12.0)^5$ 
 $x_{\text{count}} = r * \sin \theta$  [ * scaling factor]
 $y_{\text{count}} = r * \cos \theta$  [ * scaling factor]
 $z_{\text{count}} = |y_{\text{count}}| / 2.0$ 
count++
```

24 - KAMTorus

The byproduct of efforts to answer to the question of whether or not the solar system is stable. It is named for Russian mathematician A.N. Kolmogorov, who developed a theory predicting the form and stability of the orbits of the planets. The theory was independently confirmed by Kolmogorov's student, V.I. Arnold and the German mathematician J. Moser (hence K, A, M).

From Noel Giffin: This type is created by superimposing orbits generated by a set of equations, with a variable incremented each time.

$$\begin{aligned}x_0 &= y_0 = \text{orbit}/3; \\x_{n+1} &= x_n * \cos\theta + (x_n^2 - y_n) * \sin\theta \\y_{n+1} &= x_n * \sin\theta - (x_n^2 - y_n) * \cos\theta\end{aligned}$$

After each orbit, “orbit” is incremented by a step size. The parameters are angle θ , step size for incrementing “orbit”, limit value for “orbit”, and points per orbit. For 3D rendering, set the z coordinate to “orbit” (plus shift factor).

http://www.nahee.com/spanky/www/fractint/kamtorus_type.html

This code uses a typical “speed” algorithm to obtain color indices for the selected palette. Points may be rotated before calculating xplot and yplot.

```

θ = 1.3 radians;  orbit0 = 0.0;  step = 0.005;  limit = 6.0;  ppo = 110

while (orbit < limit)
{
    x0 = y0 = orbit / 3.0
    for (i = 0; i < ppo; i++)                // points per orbit
    {
        x1 = x0 * cosθ + (x02 - y0) * sinθ
        y1 = y0 * sinθ - (x02 - y0) * cosθ
        z1 = orbit - 1.25                    // -1.25 to center the image
        dx = x1 - x0                        // for palette adjustment
        dy = y1 - y0
        colorindex = 255 * √(dx2 + dy2) / 1.5
        plotpixel(xplot, yplot, colorindex)  // xplot, yplot from x1, y1, z1
    }
    orbit += step
}

```

25 - Samardzija Oscillator

$$\begin{aligned}x_0 &= 0.06; y_0 = 0.09 \\a &= -0.6, b = 1.2, c = 0.2, d = -0.6, e_0 = -0.6 \\x_{n+1} &= x_n + 0.05 * (a * x_n + b * y_n - e_n) \\y_{n+1} &= y_n + 0.05 * (x_n + c * y_n - x_n^3) \\e_{n+1} &= e + x_n * d * 0.01\end{aligned}$$

The oscillator is animated using a stack that stores 1000 points at a time, to erase the point that was plotted 1000 iterations before the current point.

26 - Lissajous oscillator

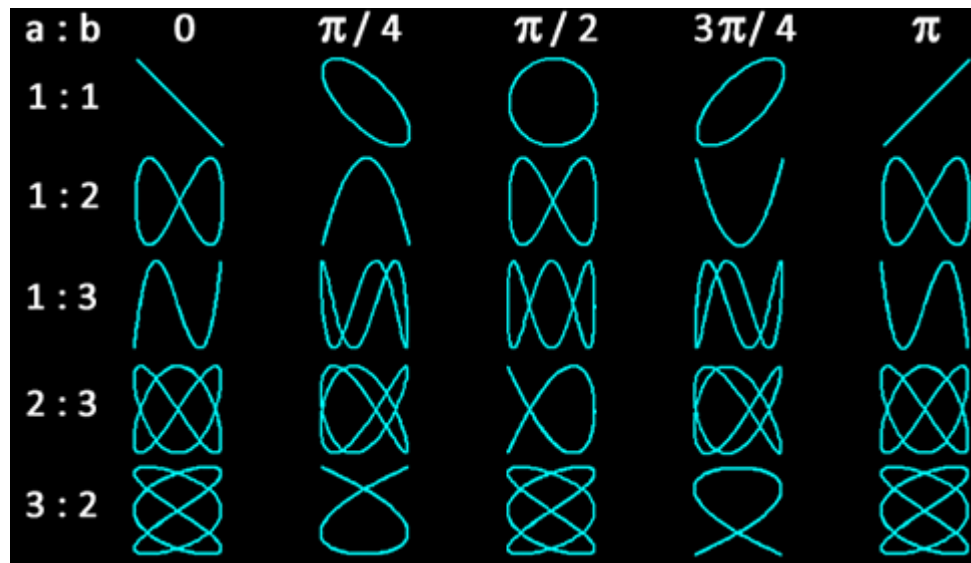
Lissajous curves (or Bowditch curves), first were investigated by Nathaniel Bowditch in 1815, and later by Jules Antoine Lissajous in 1857. They represent the combination of 2 sine waves of frequencies a and b , related by a phase angle δ . A and B control the aspect ratio of the curve, whose shape and complexity are determined by the ratio of a to b , and the phase angle.

$$x_t = A * \sin(a * t + \delta)$$
$$y_t = B * \sin(b * t)$$

A third-dimensional component can be obtained by setting $z_t = \sin(t)$.

The program uses a set of sprites of graduated size to increase the illusion of 3 dimensions.

These are some representative curves:



Attractors from Ramiro's Collection

27 - Iris

Sample parameters:

$$x_0 = 0.042587; \quad y_0 = -0.140620; \quad a = -1.748367; \quad b = 1.530674$$
$$x_0 = -0.024491; \quad y_0 = 0.207343; \quad a = -1.739309; \quad b = -1.271214$$

$$x_{n+1} = a * \sin(x_n) + b * \cos(y_n)$$

$$y_{n+1} = a * \cos(x_n) + b * \sin(y_n)$$

28 - Atom

$x_0 = 0.01$ $y_0 = 0.1$ $a = 0.584639$

$x_{n+1} = x * \sin(y_n) + 4.0 * a * \cos(y_n)$
 $y_{n+1} = y * \cos(x_n) + 4.0 * a * \sin(y_n)$

29 - Ramiro's KAMTorus - randomized variations

```
mult1 = 0.999; mult2 = 1.001; c = 1
rand1 = random(); rand2 = random() // random floats between 0.0 and 1.0
an = 10.0 * (rand1 - rand2)
can = mult1 * cos(an); san = mult1 * sin(an)
can1 = mult2 * cos(an); san1 = mult2 * sin(an)
x3 = 0.01; y3 = 0.01;
do {
    xa = x3 * x3 - y3
    x2 = x3 * can1 + xa * san1
    y2 = x3 * san1 - xa * can1
    x3 = x2; y3 = y2
    x = x2; y = y2
    a = 0;
    do {
        xa = x * x - y
        x1 = x * can + xa * san
        y1 = x * san - xa * can
        x = x1
        y = y1
        a++
    } while ( | x1 | <= 2.0e3 && | y1 | <= 2.0e3 && a <= 100)
} while ( | x2 | <= 2.0e3 && | y2 | <= 2.0e3 && ++count < 20000)
```

30 - Wings

$x_0 = 0.056$; $y_0 = -0.074$; $a = -0.622$; $b = 0.342$

if ($x_0 > 0.0$) coeff = 1.0 else coeff = -1.0
 $x_{n+1} = x_n - \text{coeff} * \sqrt{|(y_n - b * x_n)|}$
 $y_{n+1} = x_n * a + y_n$

31 - Medusa

$x_0 = 0.009$; $y_0 = 0.011$; $p1 = -0.438016$; $p2 = 0.819984$; $p3 = -0.381065$

$$\begin{aligned}x_{n+1} &= p1 * x_n + y_n - p2 * x_n^2 \\ y_{n+1} &= p3 * y_n - x_n + p2 * y_n^2\end{aligned}$$

32 - Disk

$x_0 = 0.016$; $y_0 = -0.016$; $a = -1.496$; $b = -0.617$

$$\begin{aligned}x_{n+1} &= a + b * (x_n * \cos(x_n * y_n) - y_n * \sin(x_n * y_n)) \\ y_{n+1} &= b * (x_n * \sin(x_n * y_n) + y_n * \cos(x_n * y_n))\end{aligned}$$

33 - Ikeda IFS

$x_0 = 0.049$; $y_0 = -0.074$; $p1 = 0.242695$; $p2 = 0.538835$; $p9 = 0.292347$
 $p10 = -0.715191$; $p11 = 0.404434$; $p12 = 0.178096$

```
if ( random(11) > 7 ) { // random integers between 0 and 10
    p4 = p1 - 6 / (1 + x_n^2 + y_n^2)
    x_{n+1} = 1 + p2 * (x_n * cos(p4) - y_n * sin(p4))
    y_{n+1} = p2 * (x_n * sin(p4) + y_n * cos(p4)) }
else {
    x_{n+1} = p10 * x_n + p9 * y_n + p11
    y_{n+1} = -p9 * x_n + p10 * y_n + p12 }
```

34 - Storm Julia

Sample parameters:

$x_0 = 0.054631$; $y_0 = 0.003381$; $a = -1.924232$; $b = -0.439203$; $c = -0.6711$; $d = -0.286204$
 $x_0 = 0.116414$; $y_0 = -0.166867$; $a = -0.399915$; $b = -0.591196$; $c = -0.009328$; $d = 0.338827$

```
if ( rand() > rand() ) // random integers between 0 and 7FFFh (32767)
{ x_n = x_n - a; y_n = y_n - b } else { x_n = x_n - c; y_n = y_n - d }
k = sqrt(x_n^2 + y_n^2)
x_{n+1} = sqrt((x_n + k) / 2)
y_{n+1} = sqrt((-x_n + k) / 2)
if (y_n < 0) y_{n+1} = -y_{n+1}
if ( rand() > rand() ) {
    x_{n+1} = -x_{n+1}
    y_{n+1} = -y_{n+1} }
```

35 - Frost Julia

$x_0 = 0.088$; $y_0 = 0.043$; $a = 0.306$; $b = 0.047$

```
p1 = xn - a
p2 = yn - b
if ( p1 > 0.0 ) p3 = atan(p2 / p1)
if ( p1 < 0.0 ) p3 = π + atan(p2 / p1)
if ( p1 == 0 ) p3 = π / 2
p3 = 0.5 * p3
p4 = p12 + p22
p4 = √(√(√(p43)))
if ( rand() > rand() ) p4 = -p4
xn+1 = p4 * cos(p3)
yn+1 = p4 * sin(p3)
```

36 -Star

Sample parameters:

$x_0 = -0.117826$; $y_0 = -0.139416$; $a = 0.777824$; $b = -0.388353$; $c = 0.205505$
 $x_0 = -0.15713$; $y_0 = 0.004947$ $a = -0.15713$; $b = 1.04713$; $c = 0.842761$

```
p4 = | xn |
p5 = | yn |
xn+1 = 4.0 * (a * xn + b * xn * (p4 + p5) + c * (xn * p4 - xn * p5))
yn+1 = 4.0 * (a * yn + b * yn * (p4 + p5) + c * (yn * -p4 + yn * p5))
```

37 - Spirals

$x_0 = -0.094$; $y_0 = 0.1$; $a = 0.057$ or -1.265 ; $b = 1.0125$;
 $p3 = \sin(a * 2 \pi) * b$; $p4 = \cos(a * 2 \pi) * b$

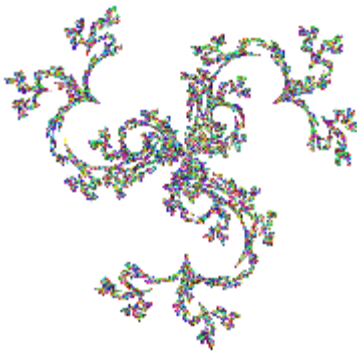
```
p9 = xn2 - yn
p1 = xn * p3 + p9 * p4
p2 = xn * p4 - p9 * p3
p7 = √(p12 + p22)
if (p7 > 2.0) {
    p1 = -p1 * 0.005
    p2 = -p2 * 0.005 }
xn+1 = p1
yn+1 = p2
```

38 - Cosine Waves

$x_0 = 0.069954$; $y_0 = -0.100327$; $a = -1.251683$; $b = -0.840133$

$$x_{n+1} = y_n - a * \cos(x_n * \pi)$$

$$y_{n+1} = x_n * b$$



lace attractor

Programming and help file authoring by:
Michael Sargent
South Burlington, Vermont
March 2018

msargent@uvm.edu
<http://www.uvm.edu/~msargent>

Disclaimer

This is unsupported, non-standard software. It is provided “as is” and any express or implied warranties, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose are disclaimed. In no event shall the author be liable for any direct, indirect, incidental, special, exemplary, or consequential damages (including, but not limited to, procurement of substitute goods or services; loss of use, data or profits; or business interruption) however caused and on any theory of liability, whether in contract, strict liability, or tort (including negligence or otherwise) arising in any way out of the use of this software, even if advised of the possibility of such damage.

Appendix 1: Evaluating Parameter Sets

After a formula is iterated many times, four results are possible:

- 1) It will converge to a fixed attracting point.
- 2) It will enter a periodic attracting cycle.
- 3) It will diverge to infinity.
- 4) It will exhibit the complex chaotic behavior characteristic of a *strange attractor*.

Option 3) will cause an overflow condition in which the size of the numbers eventually can exceed the computer's capacity. The tendency toward the other options can be characterized by the *Lyapunov exponent*. Calculation of this value is described by Julien Clinton Sprott in various publications, including

<http://sprott.physics.wisc.edu/chaos/lyapexp.htm>

and by Peitgen *et al.* in *Chaos and Fractals*, Chapter 12.5. The process can be summarized as follows: Pairs of close points which are further apart after iteration tend to show chaotic behavior, but if they are closer, they tend to converge to an attracting system. Assume that an attractor formula yields a sequence of points (x, y) . After a hundred or more iterations to establish that subsequent points are on the attractor, the next point (x_n, y_n) is perturbed by a small error “ e ” at an arbitrary angle. The value of the angle does not matter. In one of the simplest instances, if the angle is 0.0, the perturbed point (x_{e_n}, y_{e_n}) will be $(x_n + e, y_n)$. After one iteration, the resulting points (x_{n+1}, y_{n+1}) and $(x_{e_{n+1}}, y_{e_{n+1}})$ will be a distance “ d ” apart. The logarithm of the ratio d/e will be negative, zero or positive, depending on whether $d < e$, $d = e$ or $d > e$. The average of this logarithm over many iterations yields the Lyapunov exponent. (To be rigorous, the value should be calculated as e approaches 0.0, which requires using the derivative of the attractor's transformation. This involves much more math than is necessary for a recreational computer program. See *Chaos and Fractals*, pp. 709-13, for details.) After each iteration, the point (x_{e_n}, y_{e_n}) should be “renormalized” so that it is the original distance e from (x_n, y_n) , and in the same direction.

Ultimately, if its Lyapunov exponent is positive, a system will exhibit chaotic behavior, as the distances between the iterations of nearby points will, on the average, diverge. Thus, images of such systems should produce aesthetic, or at least interesting, patterns. In contrast, if the exponent is 0.0 or less, the system will converge. In this program, random parameter sets are accepted when the exponent arbitrarily is greater than 0.3 (base 2).



Atom: Lyapunov exponents 0.9023, -0.1252

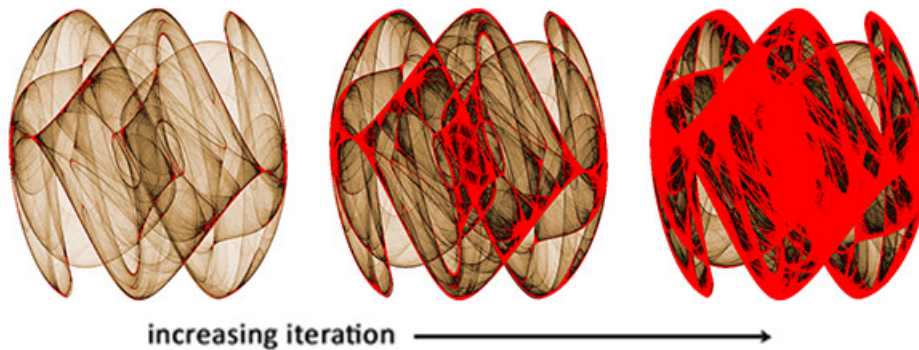
Appendix 2: Image Enhancement

For images of chaotic attractors, an effective coloring method assigns an entry from a color palette according to the number of times the pixel has been “hit” by an iterating point. It should be understood that as iteration continues, an infinite number of different points can land within the boundaries of a single pixel. However, the likelihood of this happening will diminish gradually as the pixel resolution of the image increases.

This is a representative palette of 256 colors, with indices of 0 to 255, which is derived from a text file called a MAP file.



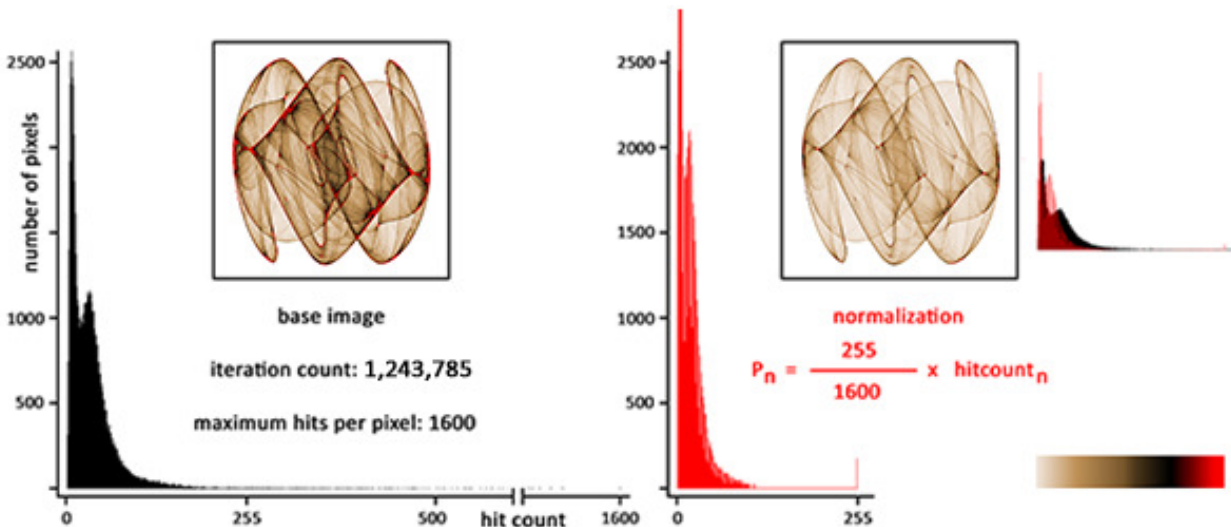
These files were developed for the pioneering fractal rendering software *FractInt* as an easy way of creating, using and exchanging palettes. The first color of the palette is pure black. This is commonly done so that `palette[0]` can be used for the background by a program. Therefore, *QS Attractor*, like many other programs, starts coloring images with `palette[1]` and ignores all background pixels, i.e. those that are not hit at all and therefore have a hit count of zero. Using this palette with the “Ganymede” formula, we can generate images like these. The image on the right demonstrates a liability of this method. As iteration continues, more and more pixels will be hit in excess of 255 times, and so the number of pixels colored with `palette[255]` will increase until potentially all of the image could be engulfed. Alternatively, the palette index for the 256th hit could be “wrapped” back to 1, but arguably this results in less appealing images.



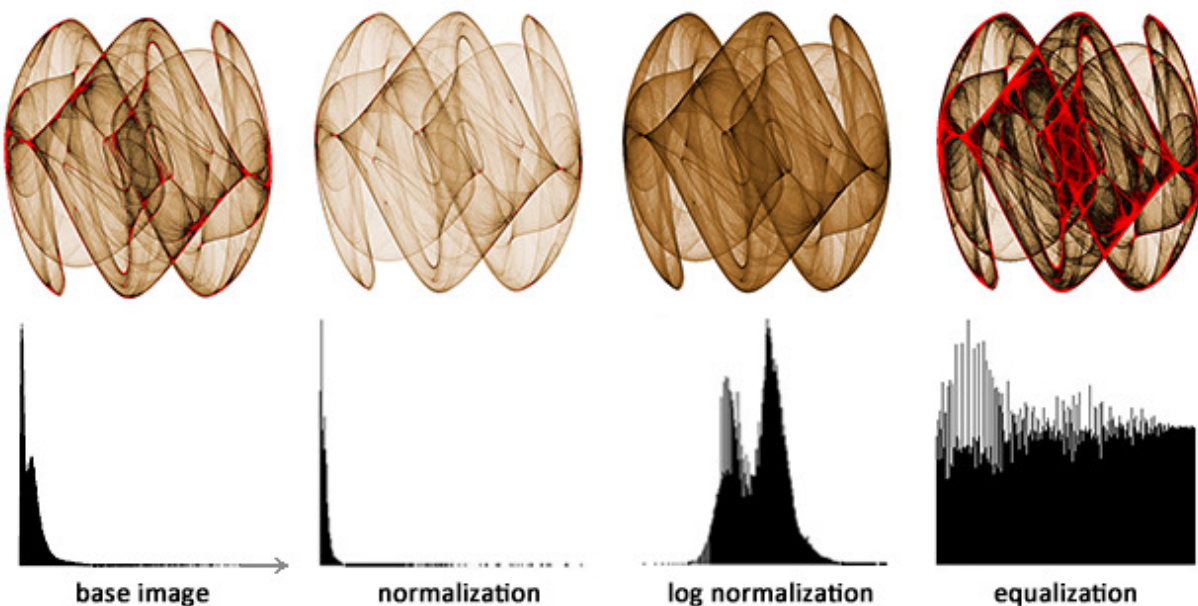
One method of compensating for this situation is *normalization* of the hit counts, in which the count for each pixel is multiplied by a factor consisting of 255 divided by the maximum hit count of all the pixels. Thus, all hit counts will be normalized to a range of 1 to 255. But this method also has potential liabilities. The following example shows an image which has been iterated 1,243,785 times, resulting in a maximum hit count of 1600. This means that at least one pixel has been hit 1600 times, and it turns out that there is only one. The next highest hit count is 1569, again only for one pixel. The highest hit count shared by 2 pixels is 1131.

On the left side, a histogram shows the number of pixels for each hit count. Clearly, the majority of pixels have a relatively low count, and therefore colors from the lower portion of the palette predominate. On the right side, the hit counts have been normalized. The histogram, now also corresponding directly to palette indices, is the same shape, but it has been “squeezed” even

further to the left, and since the number of pixels has not changed, the height of the histogram is about twice that of the original. The resulting image has even less representation of colors from the higher portion of the palette. Although this sometimes results in appealing images, it would be desirable to see a more even balance of colors.



This example shows how the base image can be adjusted. We already have seen how *normalization* affects the range of the palette, which again is confirmed by the histogram. In *logarithmic normalization*, 255 is multiplied by the logarithm of the hit count divided by the logarithm of the maximum hit count. This results in emphasis on the middle range of the palette. If the two logarithms are squared, a similar histogram shape is seen, but it is shifted more toward the lower end of the palette. Finally, we see an *equalized* image with an even distribution of colors across the palette. (Note that these histograms represent the same number of total pixels, so to fit them into the same-sized spaces, their Y-axis scales have been adjusted).



In the process of histogram equalization, an array of the number of pixels associated with each hit count is converted to a “cumulative distribution function” (cdf) or “cumulative histogram”, in which each hit count is associated with its own number of pixels added to the sum of all the preceding numbers. Thus, in the following formula, $\text{cdf}_{\min}$ is the lowest non-zero value in the histogram, and $\text{cdf}_{\max}$ is the last value, corresponding to the total number of pixels of the image. P is the number of palette entries (256). An adjustment is made because background pixels with a hit count of 0 are ignored, so that the desired range of palette indices is 1 to 255.

$$\text{index}_n = \frac{\text{cdf}_n - \text{cdf}_{\min}}{\text{cdf}_{\max} - 1} \times (P - 2) + 1$$

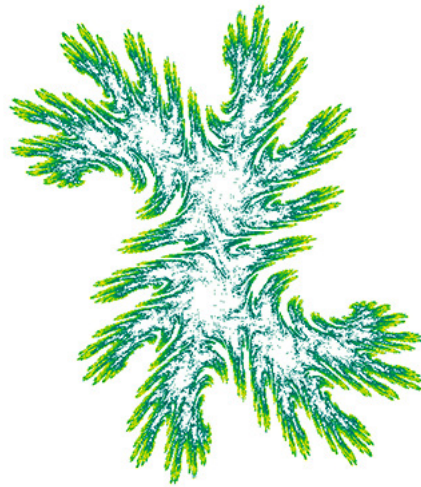
The following table shows representative data from various hit counts for another image in which rendering was stopped at a maximum hit count of 1600.

For hit count 4,
$$\text{index}_4 = \frac{4047 - 295}{67415 - 1} \times (256 - 2) + 1 = 15$$

hit count	number of pixels	cdf	palette index
1	295	295	1
2	690	985	3
3	1294	2279	8
4	1768	4047	15
5	2187	6234	23
6	2475	8709	32
7	2530	11239	42
8	2386	13625	51
-	-	-	-
17	921	26167	98
18	872	27039	101
19	931	27970	105
-	-	-	-
65	261	59814	225
66	224	60038	226
67	214	60252	226
68	223	60475	227
-	-	-	-
1599	0	67414	253
1600	1	67415	253

Notice that as the hit counts increase, the algorithm assigns the same palette index to pixels with different hit counts if the numbers of pixels are small. Index 226 is assigned to pixels with hit counts of 66 and 67, and index 253 is assigned to all pixels with hit counts from 382 to 1600. The calculations are independent of the actual palette that is used.

From an esthetic viewpoint, equalization is not necessarily “better” than normalization, logarithmic normalization, equalization of a normalized image, or any enhancement at all. Ultimately, any of these approaches has the potential to produce subjectively pleasing images, depending on the number of iterations and the selected palette.



Frost Julia