Mandelbrot Fractals

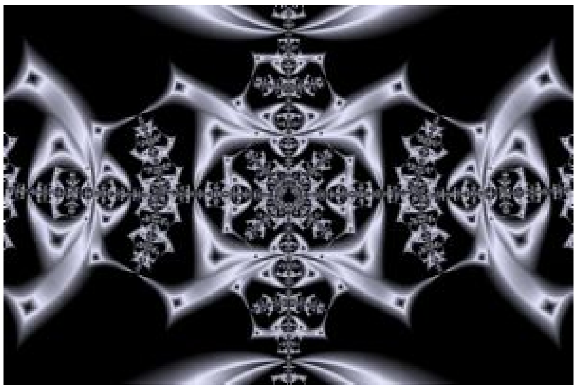
Mandelbrot Fractal Overview

The Fractal Science Kit fractal generator **Mandelbrot** fractals encompass several related fractal types including **Mandelbrot** fractals, **Julia** fractals, **Convergent** fractals, **Newton** fractals, and **Orbit Traps**.

Mandelbrot Fractals

Mandelbrot fractals are the result of iterating a fractal formula. A fractal formula is a statement like:

```
z = z^2 + c
```



This statement takes 2 complex values found in the variables `z` and `c`, and combines them based on the expression to the right of the equal sign; in this case, by squaring `z` and adding `c` to the result. The resulting complex value is assigned to the variable `z`, replacing the previous value of `z`. This completes the 1st iteration of the formula. A 2nd iteration would evaluate the expression again, this time starting with the new value of `z` computed in the 1st iteration. This process continues with each step producing a new value for `z`. The process terminates when the **magnitude** of `z` exceeds some threshold value or the specified maximum number of iterations is reached. The **magnitude** of `z` is the distance of the point `z` from the origin of the complex plane (`0+0i`). The set of all the `z` values over the entire iteration of the fractal formula is called the **orbit** and the different `z` values are called the **orbit points**.

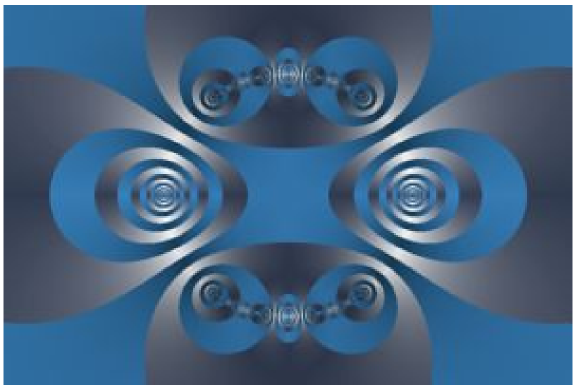
To produce a fractal image from this process, a window is mapped onto the complex plane composed of a grid of points called **pixels**. The number of rows and columns of the grid is determined by the size and resolution of the window. For each **pixel** in the window, we set `z` to an **initial z** value, substitute the complex value associated with the pixel into the formula for `c`, and iterate the formula. If the iteration terminates because the magnitude of `z` exceeds the threshold, the pixel is said to have escaped and is outside of the **Mandelbrot set**. The pixel is colored based on the number of iterations it took to escape and/or other characteristics of `z`. If the magnitude does not exceed the threshold when the maximum number of iterations is reached, the pixel is assumed to be inside the **Mandelbrot set** and is usually colored black but other colorings are possible based on the value of `z` when the iteration is terminated. Of course, the pixel may have escaped had we only executed additional iterations of the formula. Fractals generated using this algorithm, are called **Mandelbrot** fractals.

Julia Fractals

A 2nd algorithm begins each orbit by setting z to the complex value associated with the pixel, and c to a fixed complex value, called the **Julia Constant**. The rest of the algorithm is identical to the algorithm described above. Fractals generated in this way, are called **Julia** fractals.

The choice of the **Julia Constant** in large part controls the character of the resulting Julia fractal. Surprisingly, the Mandelbrot fractal for the same fractal formula provides the best interface for choosing the **Julia Constant**. It turns out that the best choice for a **Julia Constant** is a point on the complex plane near the Mandelbrot set boundary. The resulting Julia fractal will have many of the characteristics of the Mandelbrot fractal in the neighborhood of the **Julia Constant**. The Mandelbrot fractal can be used as a map for choosing the **Julia Constant**. To this end, the Fractal Science Kit allows you to click on any Mandelbrot fractal to produce a Julia fractal in a small window called the Preview Window. Clicking on the image displayed in the Preview Window causes the Julia fractal to be displayed in a full size window for further exploration.

Convergent Fractals

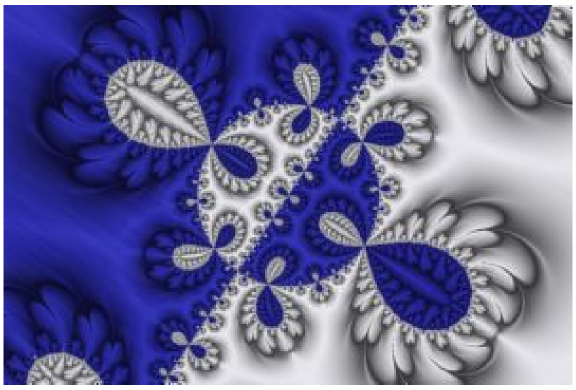


The algorithms above can be modified to watch for convergence rather than divergence. In the process described above we execute the formula over and over until the magnitude of z exceeds some threshold. Fractals with this type of **termination condition** are called **Divergent** fractals. If instead, we execute the formula until the distance between the point z and the orbit point just before z is less than some threshold, we generate what is called a **Convergent** fractal. This distance represents how far z moved in the last iteration of the orbit and the termination condition is basically checking that the orbit is converging to single point. Which termination condition to use, depends largely on the fractal formula. Like **Divergent** fractals, **Convergent** fractals can be generated using the Mandelbrot or Julia algorithms.

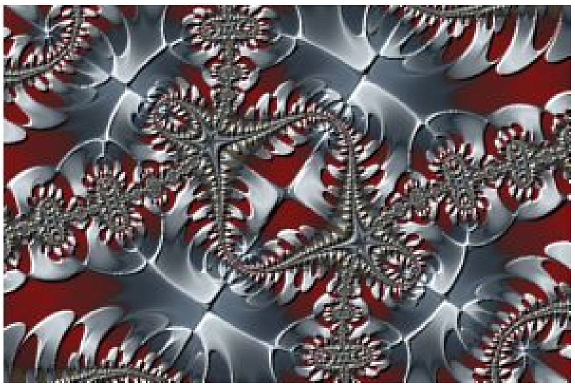
Newton Fractals

The most common example of a convergent fractal is a fractal based on **Newton's method** of finding the roots of an equation, called **Newton** fractals.

Newton fractals are a visual representation of a process called the **Newton-Raphson method** or simply **Newton's method**, named after Isaac Newton and Joseph Raphson, used to find the roots of an equation. Given an approximate value for one of the roots, a simple procedure is used to find a better approximation for the root. This procedure is applied over and over again, each time using the value obtained from the previous step, thereby moving closer and closer to the actual root. If this process converges to a value, you have found a root. The approximate value chosen at the beginning of the process, determines to which root you will eventually converge. If we assign each root a color, and we color each point on the complex plane based on the root to which that point converges, a Newton fractal



is the result.



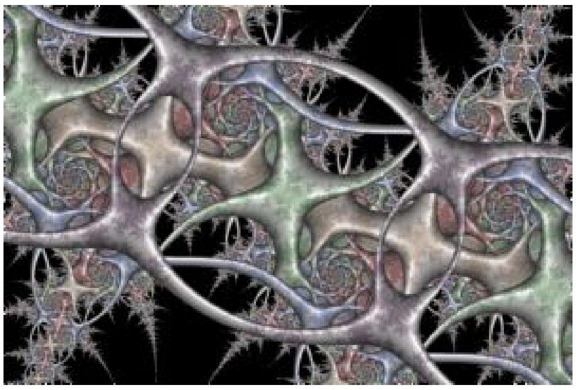
To create a Newton fractal for a given function `f(z)`, you set up a **Julia** fractal with a convergent termination condition, based on **Newton's method** for finding the roots of an equation; i.e., `z = z-f(z)/f'(z)`, where `f'(z)` is the derivative of `f(z)`. If you plug in a value for `z` into the expression on the right side of the assignment, the value of the expression is closer to the root than the original value `z`. To generate the fractal, you start with a complex value associated with a point on the complex plane and perform this assignment until the value converges to a root, and then color the initial point based on that root.

Of course, there are many other root-finding methods including: **Schroder's** method, **Halley's** method, **Whittaker's** method, **Cauchy's** method, the **Super-Newton** method (also called **Householder's** method), the **Super-Halley** method, the **Chebyshev-Halley** family of methods, and the **C-Iterative** family of methods. Fractals associated with each of these methods are supported by the Fractal Science Kit. Developers can set up a program to examine the fractals associated with these methods with just a few lines of code. See Solver Functions for details and example programs.

Built-in Fractal Equations

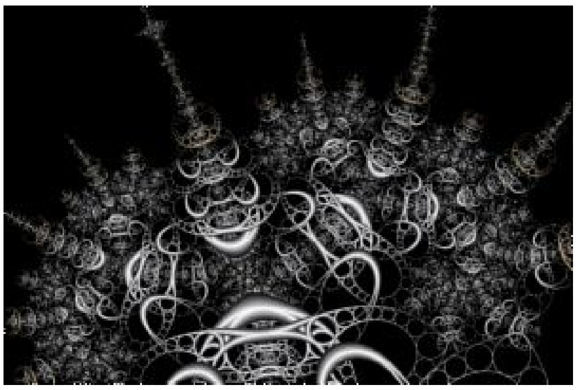
The Fractal Science Kit fractal generator has over 180 built-in Fractal Equations including **Mandelbrot**, **Mandelbar**, **Cubic**, **Lambda**, **Phoenix**, **Tetrate**, **Newton**, **Nova**, **Barnsley**, **Magnet**, as well as studies in convergent fractals, polynomial fractals, root-finding method based fractals, Julia maps, fractals based on trigonometric and hyperbolic functions, and fractals based on exponential formulas. Many of these programs define properties that can be used to produce countless different variations. See Built-in Fractal Equations for a complete list.

Orbit Traps

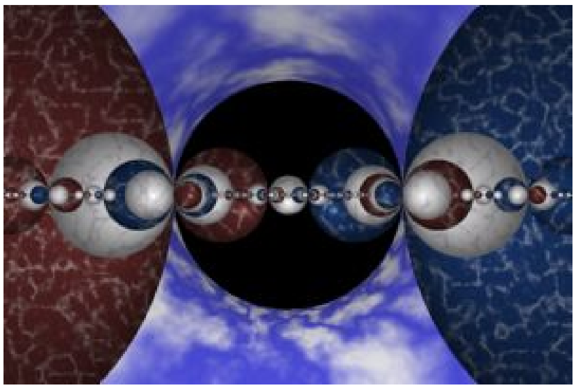


The **Classic** display of the different Mandelbrot and Julia fractals (both divergent and convergent) iterates the fractal formula as described previously and maps the resulting orbit to a color. Another popular method of generating fractals is to define a set of geometric objects located on the complex plane called **Orbit Traps** and during the fractal iteration, keep statistics related to how close the orbit points come to the **Orbit Traps**. **Orbit Traps** provide a fertile ground for new and unusual fractals. You can combine the built-in traps with your own Orbit Trap instructions to produce unique fractal designs. The Fractal Science Kit fractal generator supports most of the common traps, including: **Circle**, **Cross**, **Cycloid of Ceva**, **Epicycloid**, **Flower**, **Folium**, **Hypocycloid**, **Lemniscate**, **Limacon**, **Line**, **Oscillator**, **Polygon**, **Rectangle**, **Rose**, **Sectors**, **Shape**, **Spiral**, **Star of David**, **Super Ellipse**, **Swirl**, and more. Many unusual traps are supported as well, including: **Apollonian Gasket**, **Apollonius Grid**, **Borromean**

Rings, **Circle Inversion**, **Circle Web**, **Circular Vine**, **Elliptic Circles**, **Daisy**, **Epitrochoid Net**, **Epitrochoid Rose**, **Faceted Polygon**, **Farris Wheels**, **Farris Wheels Net**, **Ford Circles**, **Fractal Gasket**, **Harmonograph**, **Hyperbolic Circles**, **Isogonal Polygon**, **Kleinian Group**, **Koch Triangle**, **L-System**, **Hypotrochoid Net**, **Lemniscate Net**, **Limacon Net**, **Maurer Rose**, **Nephroid Net**, **Parabolic Circles**, **Parabolic Grid**, **Penrose Kite**, **Polygon Net**, **Polygon Whirl**, **Schottky Group**, **Sierpinski Triangle**, **Sound Ornament**, **Spirolateral**, **Star Polygon**, **Steiner Chain, Tangent Circles**, **Unit Circle Group**, and many more. See Orbit Trap Types and Built-in Orbit Traps for a complete list.



The **Orbit Trap** Properties Pages allow you to define a list of traps and specify how to process them. You can blend the traps together using various blending techniques (e.g., harmonic mean) or process them separately. You can modify the trap's position/angle per iteration, based on values of the current/previous orbit points. You can control the trap envelope. You can transform input points prior to passing them to the trap. These are but a few of the many different **Orbit Trap** options available.



Some of the orbit traps are stand-alone fractals in their own right. For example, the **Kleinian Group** trap allows you to produce Quasifuchsian, Single Cusp, and Double Cusp, Two-Generator Group fractals described in the book **Indra's Pearls - The Vision of Felix Klein** by David Mumford, Caroline Series, and David Wright. The **Schottky Group** trap explores the world of nesting Schottky disks and groups of Mobius maps that form the basis for the Two-Generator Group fractals above. **Schottky Group** fractals are also described in the book **Indra's Pearls**. The **L-System** trap allows you to create an orbit trap using a set of statements that define an **L-System** or **Lindenmayer System**. **Lindenmayer System** fractals were developed in 1968 by Aristid Lindenmayer. The **Apollonian Gasket**, **Circle Inversion**, **Ford Circles**, and **Unit Circle Group** traps also produce stand-alone fractals. Other traps (e.g., **Epitrochoid Net**, **Epitrochoid Rose**, **Farris Wheels**, **Farris Wheels Net**, **Harmonograph**, **Hypotrochoid Net**, **Isogonal Polygon**, **Maurer Rose**, **Sound Ornament**, **Spirolateral**, **Star Polygon**) can produce beautiful mathematical art when combined with complex transformations and other supported features.

Copyright © 2004-2016 Hilbert, LLC
All rights reserved