QS EZFract 2015 Help

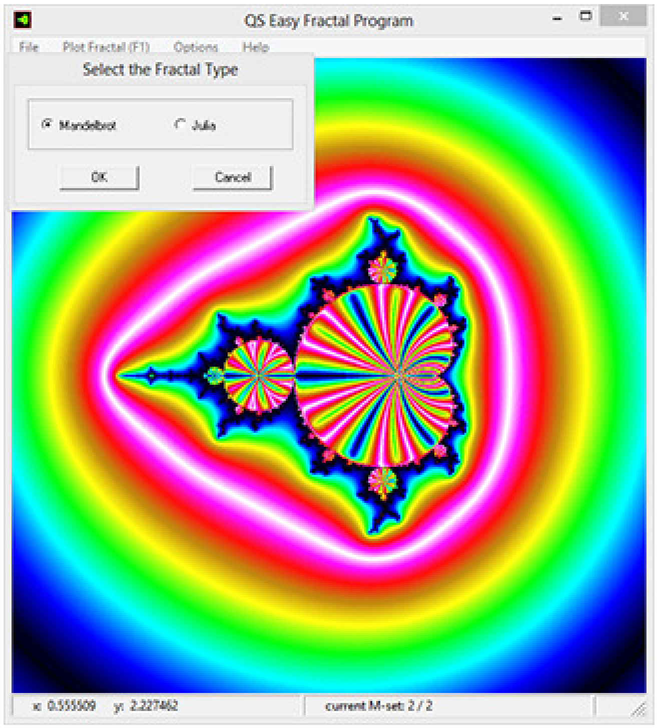
“The computer is looked upon as a diabolical instrument by many, scientists no less than artists and worried parents. Some, after a brief glance at the machine, find themselves completely addicted.”
H.-O. Peitgen *et al.* - *The Beauty of Fractals*, p. 3

“Chaos Wipes Out Every Computer”
H.-O. Peitgen *et al.* - *Chaos and Fractals*, Chapter 1.5 title

Contents

1. Introduction
2. Using the Program
3. Special Keys
4. Outside Coloring Methods
5. Inside Coloring Methods
6. Triangle Inequality Average
7. Periods and Buds
8. External Rays
9. Conclusion
10. Disclaimer

1. Introduction



QS EZFract 2015 builds upon my early programs *QSWFract* from 1996 and *EZFract* from 1997, which started as a simple template upon which to add code for various fractals and algorithms. I was motivated to undertake this project while I was learning to apply coloring methods in *Mathematica* to various objects, including Mandelbrot sets. This led to the resurrection of some of the original *EZFract* coloring methods, and to revisiting Jeff Field's program *EZ33*. It struck me that it could be entertaining and even educational to revise *EZFract* for the new millennium.

Unlike *QSWFract*, *EZFract* only draws Mandelbrot and Julia sets, and is limited to true color graphics modes. Although its scope is more focused, it introduces some new and hopefully useful features. Up to 50 parameter changes for each fractal type are saved, so that backing out of a zoom or returning to a previous image is possible. You can click anywhere on a Mandelbrot image to see a thumbnail of the associated Julia set, and then render it if you wish. You can bring up a Mandelbrot set that will animate the orbits of individual starting points that you click on. You can trace external rays over any image. And at least some inefficient code has been revised. There is no scrolling, because coding it is a tedious hassle, and it's really not necessary. You can work on an image that fits inside your window, and when it's ready, render it in any size you wish. Since this is done to a virtual screen, the size is limited only by your computer's memory. The TARGA saving routine works from this virtual screen. So WYSINWYG.

This program is dedicated to my fellow Fractaliers - Fausto Barbuto, Dave Dobbs and Jeff Field - as we forge ahead into our third decade.

I also would like to acknowledge:

Gaston Julia, Pierre Fatou, Benoit Mandelbrot, Adrien Douady and John H. Hubbard: giants whose shoulders we stand on.

the Stone Soup Group, creators of *Fractint*, especially Tim Wegner, who corresponded with me at length and welcomed my input to the program.

Heinz-Otto Peitgen, P.H. Richter, Hartmut Jurgens and Dietmar Saupe from Bremen, Germany, who gave mathematical legitimacy to the art of fractals long before the emergence of fractal *artistes*. The titles of their books say it all: *The Beauty of Fractals, The Science of Fractal Images, Chaos and Fractals.*

Wolf Jung, a mathematics teacher in Aachen, Germany, who created the *Mandel* program, which is dedicated to exploring the mathematical intricacies of Mandelbrot and Julia sets. The source code was instrumental in my eventual success in coding external angles and external rays. His Web site is:
http://www.mndynamics.com

Adam Majewski, whose *Maxima* code helped me to understand various aspects of fractals including application of the inverse iteration method for tracing external rays. Unfortunately, this method only works in the dynamic plane (Julia sets) and not in the parameter plane (Mandelbrot sets), so I didn't end up using it. His Web site is:
http://fraktal.republika.pl

Robert Devaney, whose book *Chaos, Fractals and Dynamics - Computer Experiments in Mathematics* describes in simple terms, on pages 111 and 123, how to use the boundary

scanning technique to draw the borders of M- and J-sets (and thus to approximate equipotential lines). He also describes how to use the polar coordinate system to calculate the square root of a complex number in *The Science of Fractal Images*, pages 152-3.

Cliff Pickover, whose multiple and often fanciful books have added significantly to the realm of fractal art and science, and who created the **"biomorph"** and **"epsilon cross"** coloring schemes that appear in this program.

Damien Jones and Kerry Mitchell, who were early participants in both the mathematics and the art of fractals, and who eventually contributed to *UltraFractal*. Some of their *UF* scripts were helpful with such topics as **smoothing** routines and **triangle inequality averaging**.

2. Using the Program

The menu commands should be largely intuitive, except for **Options → Coloring → Method** (see below) and **Options → External Rays** (**Section 8**). Here is a summary of various features. Additional information on program use will be found in **Section 3 (Special Keys)**.

Status bar: The status bar at the bottom of the window is divided into two parts. The left part shows the coordinates of the mouse cursor over the image, in *actual image* values (not pixels). Just like a standard graph (and not like computer screen coordinates), the y values are low at the bottom and high at the top.

The right part shows various types of information. By default it shows the position of the current image in the history list, *e.g.* "Current M-set: 7 / 50". When a zoom box is being dragged, it shows the aspect ratio, which is necessary to avoid distorting the image (see **Zooming** below). When a Julia thumbnail is being displayed, it shows the value of the *cx* and *cy* points.

Palettes consist of 256 RGB colors each, to allow for the use of *Fractint* MAP files, easily-editable text files, which can be loaded from the **Options → Palette** menu. The program starts with a default palette, derived from *QSEZF.map*, which can be reset from the menu. Additionally, a linear greyscale palette can be loaded from the menu.

Zooming: In typical fashion, the zoom box is formed by depressing the left mouse button and dragging. Watch the right part of the status bar, which shows the aspect ratio of the zoom box, to make sure the zoomed image is not distorted. The aspect ratio should be maintained as *xres / yres*. The program starts out with a default ratio of 1:1, but this can be changed by entering different values in the **Options → Parameters** dialog box.

Release the left button to complete the zoom box. If you're satisfied, press **Z** to render the image. You'll see that its position in the current history is updated in the right part of the status bar. If you're not satisfied, press **X** to erase the zoom box.

History: The program saves up to 50 parameter sets for both Mandelbrot and Julia images. But it does not save each coloring change within the same parameters. To go back to previous images, press **Backspace**. To go forward, press **Space**. You can follow the order of the images in the right part of the status bar. To reset the history of the active fractal type, deleting the currently-stored images, press **R**. As noted above, the right part of the status bar displays the position of the current image in the history list in the form: “Current [M | J]-set: X / 50”.

Julia sets: When a Mandelbrot image is displayed, you can right-click anywhere on the image to display a thumbnail image of the associated Julia set at the upper left corner of the window. Its values for *cx* and *cy* will be displayed in the right part of the status bar. If you wish to render the image, press **J**. To remove the thumbnail, right-click again anywhere. You can display and erase as many thumbnails as you want. I saw this done first in *Fractint*, and it's also done in *Mandel*. I've always wanted to reproduce it.

Options → Coloring → Method: There are two choices for this menu item. **Image** yields a standard image. **Height Field** yields a greyscale image that can be used as a height field by a 3-dimensional rendering program such as *POV-Ray*. Dark areas are interpreted as low along the z-axis, and light colors are high. This option works especially well with the **Continuous Potential** outside coloring option.

Orbit-tracing mode: This is another idea from *Fractint* that I've expanded upon. To enter the mode, press **O**. A full Mandelbrot image will be drawn. By right-clicking anywhere on the image, either inside or outside the M-set, the orbit of the corresponding point will be animated as successive points connected by line segments as iteration progresses. Various interesting patterns emerge, especially for points within the M-set, including spiral arms that are formed periodically. Tracing orbits from within buds will reveal their periods. To exit from the mode, press **O** again. The default line-drawing mode can be toggled off and on by pressing **K**. Menu options are limited, but you can change *xres*, *yres* and *maxiter* via **Options → Parameters**.

Escher-Julia mode: This tangential and somewhat esoteric mode appears in Peitgen's chapter in *The Science of Fractal Images* on pages 185 and 187. A unit-circle Julia set ($z = z^2$) is iterated, and points are tested to see if they are contained in a second, “target” Julia set ($z = z^2 + c$). As Peitgen writes, “This method opens a simple [*sic*] and systematic approach to Escher-like tilings.” I've included it because it's interesting and unique and I was able to figure out how to code it. I was pleased to have it incorporated into *Fractint*.

To enter the mode, press **E**. Menu options are limited, but you can change *xres*, *yres* and *maxiter* via **Options → Parameters**. You also can change the target set with the *cx* and *cy* parameters. You also can use **Options → Palette** to change the color palette. To exit from the mode, press **E** again.

Inversion mode: Based on polar coordinates, this mode transforms an image by mapping each point to be iterated to a corresponding position along its argument (angle), on the opposite side of a given point designated the “radius”. Typically, the radius is 1.0; using a different radius has a scaling effect. So, a point at $(0.5, \pi / 4)$ will become $(2.0, \pi / 4)$ and a point at $(4.0, \pi / 3)$ will become $(0.25, \pi / 3)$.

The result is an image with the M-set or filled J-set on the outside, and the complement of the set on the inside, with level sets becoming progressively circular as they surround a tiny dot which represents infinity.

To enter the mode, press **I**. This option is disabled for certain coloring methods, such as equipotential lines, and the external ray option is disabled. To exit from the mode, press **I** again.

3. Special Keys

Accelerator keys are keyboard shortcuts to menu items and are indicated on those menu items. They include **F1 - F9, Esc, D, G, L, M, P** and **S**.

Further information on the following special keys can be found in **Using the Program**.

Pause, not surprisingly, pauses the rendering of an image. To resume, click **Plot Fractal** or press **F1**.

Backspace: Moves back to previously-rendered images.

Space: Moves ahead to previously-rendered images.

E: Enters **Escher-Julia mode**. To exit from this mode, press **E** again.

I: Enters **inversion mode**. To exit from this mode, press **I** again.

J: Renders a Julia set if the current image is a Mandelbrot set, and a thumbnail has been drawn by right-clicking on a point in the image.

K: In **orbit-tracing mode**, orbiting points are connected by line segments. To toggle this option off and on, press **K**.

O: Enters **orbit-tracing mode** and draws a full Mandelbrot image. By right-clicking any point in the image, the orbit for that point is animated as it is iterated. This can be repeated indefinitely. To exit from this mode, press **O** again.

R: Resets the history of the active fractal type, Mandelbrot or Julia. Currently-stored parameters are deleted.

T: In **triangle inequality average** coloring methods, cycles between the default, Damien Jones's and Kerry Mitchell's algorithms, explained in the **Triangle Inequality Average** section.

X: Cancels a completed zoom box.

Z: Renders the image in a completed zoom box.

4. Outside Coloring Methods

Choosing several of these coloring methods brings up secondary dialog boxes to set method-specific parameters. To re-open these dialog boxes, re-select the coloring method. Most of these methods are affected significantly by the choice of iteration count (*maxiter*) and bailout value (*maxsize*). The palette also can make a major difference in the appearance of the image.

Solid Color: The same secondary dialog box is used to set both the outside and inside solid colors. But only the selected option - outside or inside - will be applied.

Level Sets: The time-honored outside coloring method. Sequential colors from the palette are assigned to points based on how many iterations are required for them to escape.

Smoothing: Level sets are adjusted to form smooth gradients, substituting aesthetics for mathematical purity. The basic algorithm for the normalization of the iteration count *n* is:

$$nn = (n + 1) - [\log(\log(size)) - \log(\log(maxsize))] / \log(2.0).$$

size is the distance of the escaped point from the origin. Dividing by log(2.0) converts the log from base 10 to base 2.

Continuous Potential: This method is exploited by *Fractint* to produce smooth gradients that spread out the palette in a way that produces height field images that work well with 3-dimensional rendering programs such as *POV-Ray*. The potential of an escaped point is calculated as the log of its *size* value (its distance from the origin) divided by 2^n, where *n* is the iteration count. This method works best with very large values for *maxsize* (at least several hundred).

A necessary parameter called *slope* should be set in the **Options → Parameters** dialog box. On page 263 of *Image Lab*, his guide to several classic graphics programs of the 1990's, Tim Wegner writes:

"Slope ... affects how rapidly the colors change. If this value is too high, you will run out of colors and the 'bottom' will be a solid color. If it is too low, the whole range of colors will not show.... The best value varies ... - you will have to experiment with values from 200 and up."

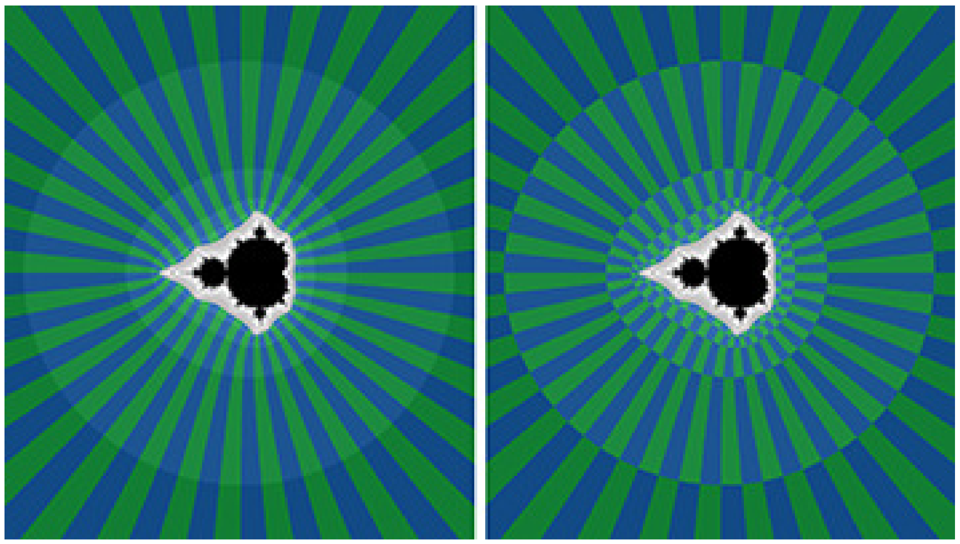
Equipotential Lines: Douady, Peitgen *et al.* draw analogies between the level set boundaries and **External Rays** of the "quadratic family" of fractals on the one hand, and electrostatic equipotential lines and "field lines" on the other. This option draws a set of approximate equipotential lines using a boundary scanning algorithm. These lines represent boundaries between level sets, and are always orthogonal to the external rays.

An image showing equipotential lines can be plotted, after which one or more sets of external rays can be drawn over it, to produce images like the ones found in the Bremen books.

A secondary dialog box offers the option of *white* or *dark gradient* backgrounds. The latter are drawn with a variation of the **Field Line Gradient** method described below.

Binary Decomposition: This is the classic method showcased in *The Beauty of Fractals*. Level sets are decomposed based on whether the imaginary part of *z* is positive or negative, which is a convenient simplification of whether the angle of the point is <= or > π. Each level set of the *n*th iteration is divided into 2^n "cells". The pattern is influenced by the choice of the bailout value (*maxsize*). This method provides an easy demonstration of **External Rays**. To see the relationship, set *maxsize* to 256.0, render a Mandelbrot set and then draw the set of external rays with a denominator of 64.

Binary Decomposition: B / W: For purists, a black and white version.


Peitgen Band Decomposition: On pages 851-2 of *Chaos and Fractals*, Peitgen *et al.* show that bands following external rays can be made by altering the standard binary decomposition algorithm so that all escaping points are iterated the same number of times, even if they escape with fewer iterations. They call these bands, which cross all level sets, “cells”. This image reproduces Figure 14.7, which shows the cells and an alternating color mapping which simultaneously makes the level sets more visible.

Field Line Gradients: While it is challenging to draw individual external rays, simply calculating the external angle of a given point is easier. This method produces approximately radial gradients by coloring pixels according to their external angles expressed in “turns”, which are (radians / 2π), ranging from 0 (along the positive x-axis) to 1 (sweeping in a counter-clockwise direction back to the positive x-axis).

A necessary parameter called *period* controls how many times the gradient is repeated in the sweep around the image. This in effect creates “cells” (see **Peitgen Band Decomposition** above) with boundaries along a set of external rays whose denominator is equal to *period*. This value can be set in the **Options → Parameters** dialog box or in a secondary dialog box which is brought up when this coloring method is selected.

The secondary dialog box also offers the option of *radial* or *radial reflected* mapping. The first option applies the range of the palette from 0 to 255 across each period cell as it sweeps around. The second option applies the palette range in reverse below the x-axis, resulting in a symmetrical image.

For best results, choose an inside color that contrasts with the first color(s) of the palette. Otherwise, a “bleeding” effect from inside to outside will obscure the boundary region, resulting in a blurry image.

Triangle Inequality Average: This method is described in detail in the separate **Triangle Inequality Average** section. A necessary parameter called *bailout* should be set in the **Options → Parameters** dialog box. For the classic pattern, it should be the same as *maxsize*. Varying these two parameters can result in potentially interesting variations in the decomposition. While in any TIA mode, pressing **T** cycles between the default, Damien Jones's and Kerry Mitchell's algorithms, explained in the **Triangle Inequality Average** section.

Smoothing + TIA and **CP + TIA** combine these pairs of methods together in layers.

Integer TIA began as a mistake in coding, which, however, resulted in interesting images. I recoded it to make it seem like it was intentional.

"Kirlian" Decomposition: A method named by Fausto Barbuto, which sets the colors based on sin(*zx*) - cos(*zy*) + atan(*zy* / *zx*). The method is very sensitive to the bailout value (*maxsize*).

Pickover Biomorphs: A method created by Cliff Pickover, producing images which might appear to some as resembling primitive life forms that could be grown in a computer simulation. If either the real part *zx* or the imaginary part *zy* of the escaped point is less than the bailout value *maxsize*, the associated pixel is colored differently from the others in the level set.

Species of Origin: An orbit-trapping method that I adapted from an *UltraFractal* script by Mark Townsend.

Cardioid: Another orbit-trapping method that I adapted from an *UltraFractal* script by Mark Townsend.

N + Real: This method colors escaping points by adding their iteration counts to the floor of the real part *zx* of the points. One could substitute the imaginary part *zy*, or add both, or create other variations on this theme.

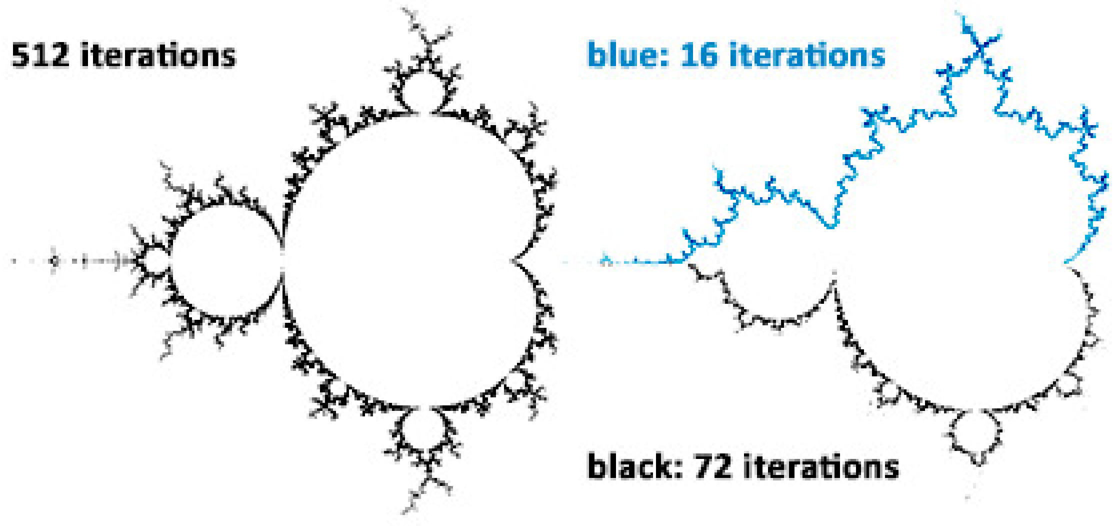
Distance Estimator: This method is described in *The Science of Fractal Images* as a means of computing the boundaries of Mandelbrot and Julia sets. It is capable of showing the thin filaments of the Mandelbrot set, which otherwise could be skipped over, in considerable detail by mapping them to pixels. The method is based on the work of John Milnor at the Institute for Advanced Study in Princeton. Peitgen approximates Milnor's equation for estimating the distance of a point *z*, which has escaped from the set, to the boundary of the set as:

$$2 \frac{|z_{n+1}|}{|z'_{n+1}|} \log(|z_{n+1}|), \quad \text{from} \quad \frac{2 \sinh G}{|G'|}, \quad \text{where} \quad G = \frac{\log |z_k|}{2^k}, \quad \text{the continuous potential as } (k \to \infty)$$

z'_{n+1} can be calculated as $2 * z_n * z'_n + 1$ during iteration, or afterward, having saved the iterated values of z_n in a buffer and re-iterating. The results of this method depend on setting a suitable threshold value for the distance, with points whose distance is less being considered a part of the boundary. Additionally, points which are very close to the boundary can cause an overflow while their distance is being calculated, so a suitable overflow limit also should be set. The extent of the outline can be affected significantly by adjusting these values. In particular, some Julia sets might require a lower threshold and a higher overflow limit than the Mandelbrot defaults.

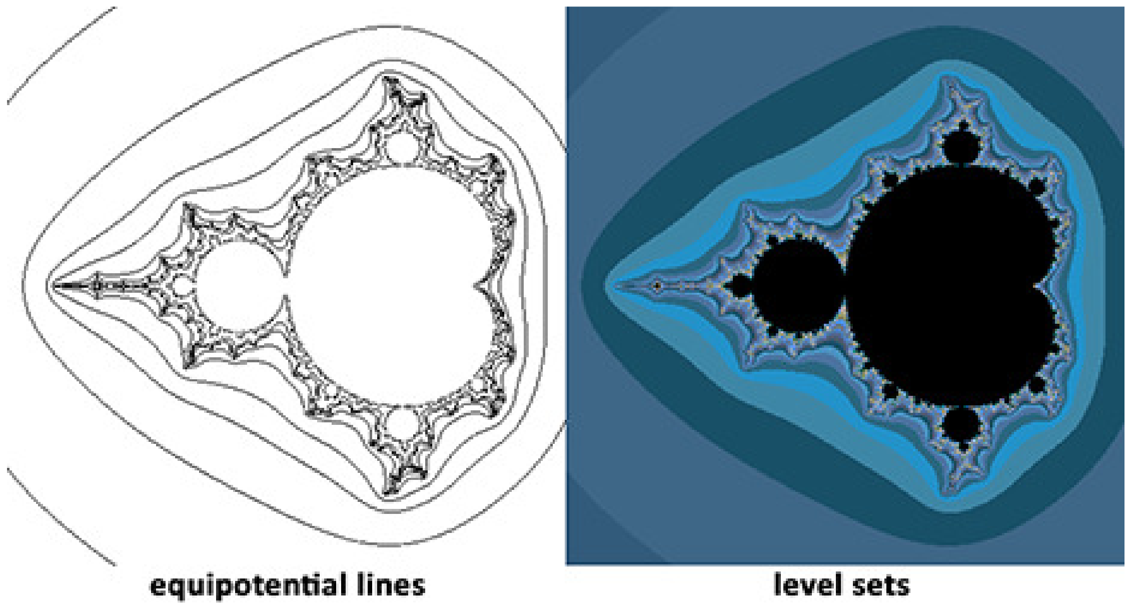
This method also works using the points which do not escape. With similar calculations, the set boundaries can be approximated, though with less precision than if the escaped points are used.

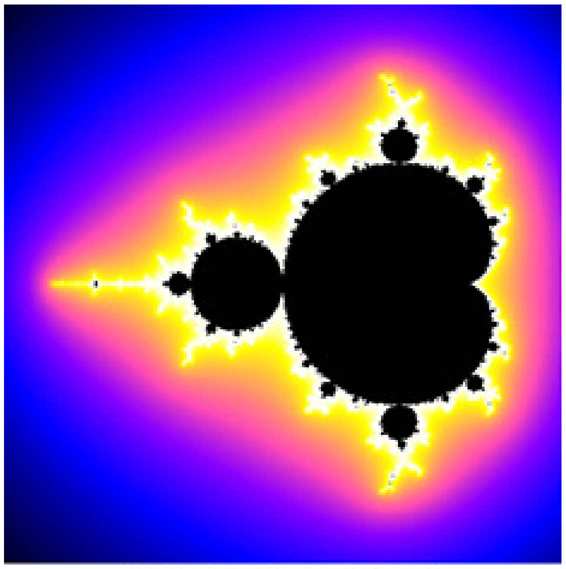
This illustration shows boundaries calculated from outside (left) and inside (right) points. With inside points, there is more detail with fewer iterations, but the boundary is less accurate.

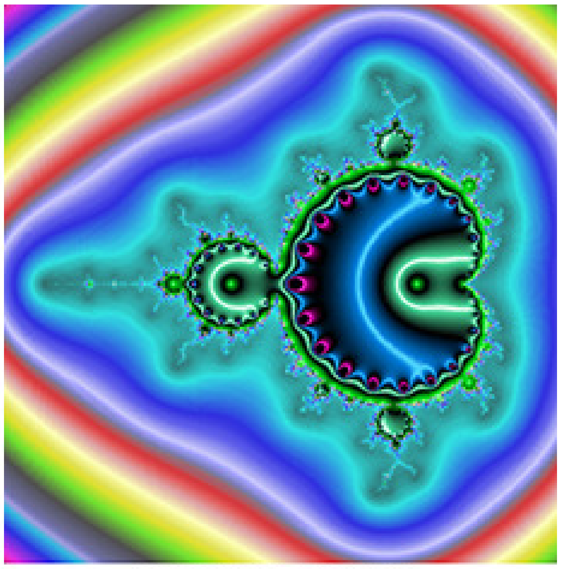


For black and white boundary images, Peitgen assigns each point “c” a value “l(c)”: 0) point inside the set; -1) point causing overflow; 1) point's distance is less than the threshold; 2) escaped point. The boundary can be traced by coloring pixels black if | l(c) | == 1 and white otherwise. However, the outside points can be colored by assigning them a value proportionate to the ratio of the point's distance to the maximum distance of any point within the image window. This of course requires pre-iterating the set to obtain the value of the maximum distance. The inside points also can be colored in this way, but a scaling factor is needed to spread the colors evenly, and relatively low maximum iteration values should be used.

Outside points colored by the distance estimator method do not follow the typical distributions of level sets or continuous potential. Here is a comparison of these methods:




continuous potential


distance estimator

5. Inside Coloring Methods

In this section, “point” typically refers to the position of a point in its orbit at the end of the selected iteration cycle, when *n* has reached *maxiter* while *size* remains less than the bailout value *maxsize*.

Solid Color: The same secondary dialog box is used to set both the outside and inside solid colors. But only the selected option - outside or inside - will be applied.

ZMag: Points are colored according to the value of *size* - the distance of the iterated point from the origin - at *maxiter*.

BOF 33/34: These methods were named in *Fractint* after figures 33 and 34 in *The Beauty of Fractals*, although later versions have changed the names to BOF 60/61, after the pages they appear on. BOF 33 colors points according to the closest distance of their orbits to the origin until *maxiter* is reached. BOF 34 colors points according to their iteration numbers when their orbits are closest to the origin. See **Periods and Buds** for a more detailed discussion of this method and related concepts.

There are 7 methods based on **functions**, primarily trigonometric, of *zx*, *zy* and *size*. There is nothing specific to say about them except that their names explain how they are applied.

Triangle Inequality Average: This uses the same algorithms as the **Outside Coloring Method**, but is applied to the regions *inside* the respective sets. See **Triangle Inequality Average.**

Epsilon Cross: This is another method created by Cliff Pickover. Zooming in reveals why this method also is referred to as “Stalks”. Points are evaluated on whether they are closer than a minimum value (in this program, 0.01) to either the real or the imaginary axes (hence “cross”). They then are colored based on their relative distances to the closer axis.

Angular Decomposition: This method forms a smooth, asymmetrical radial gradient which is somewhat analogous to the **Field Line Gradients** option in **Outside Coloring Methods**. However, the angle that the point makes with the origin, rather than the external argument, is used.

Distance Estimator: This method uses the same type of algorithm as the outside coloring method. See the description in **Outside Coloring Methods** for details.

6. Triangle Inequality Average

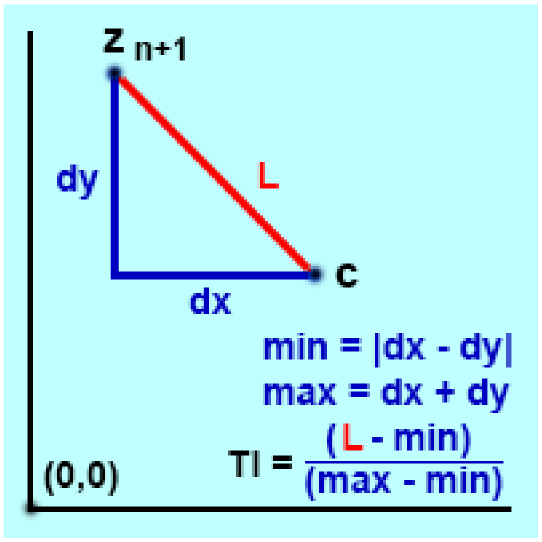
TIA is based on the concept that for any triangle, the length of one side must be greater than the difference and less than the sum of the other two sides. The triangle in a typical fractal calculation can be formed by several different combinations of points. Representative code includes the following:

```
//----- Initialize before entering nested loops -----
//----- Adapted from Damien Jones's UF routine -----
il = 1.0 / log(2.0);
lp = log(log(bailout) / 2.0);
```

bailout usually is set to the same value as *maxsize*, though varying it can yield interesting images.

```
//----- Typical iteration loop for each point/pixel -----
n = 0; zx = zy = size = sum = sum2 = 0.0;
while ((size <= maxsize) && (n < maxiter))
{
        x = zx; y = zy;
         zx = (x + y) * (x - y) + cx; zy = 2.0 * x * y + cy;
         size = (sqrt(zx * zx + zy * zy)); n++;

         [Insert TIA algorithm before terminating loop]
}
```



My algorithm uses the right triangle formed by the points *c* and *z(n+1)*. The value of *min* is abs(*dx* - *dy*) and *max* is *dx* + *dy*. The side to be evaluated (*L*), is sqrt(*dx* * *dx* + *dy* * *dy*). The triangle inequality is expressed as the ratio *(L - min) / (max - min)*. When the iteration loop is terminated, the ratios are averaged, and then this average is normalized to yield a color index for the pixel. In C code, we have, within the iteration loop:

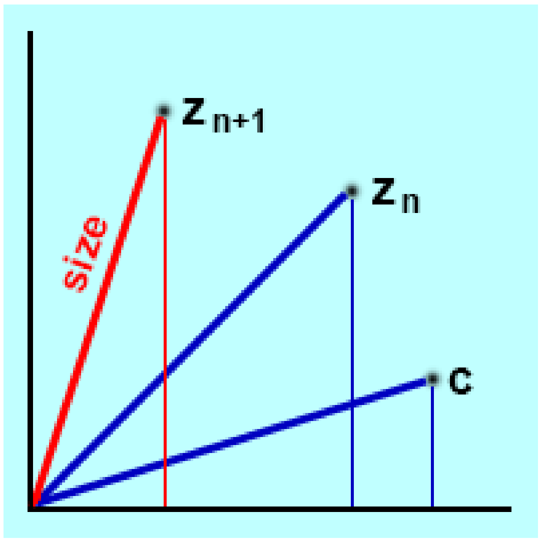
```
sum2 = sum;
z_c = sqrt((zx - cx) * (zx - cx) + (zy - cy) * (zy - cy));
 min = fabs(fabs((zx - cx)) - fabs((zy - cy)));
 max = fabs((zx - cx)) + fabs((zy - cy));
 if (min == max) sum += 0.5;
 else sum = sum + (z_c - min) / (max - min);
```

and then when the loop is terminated:

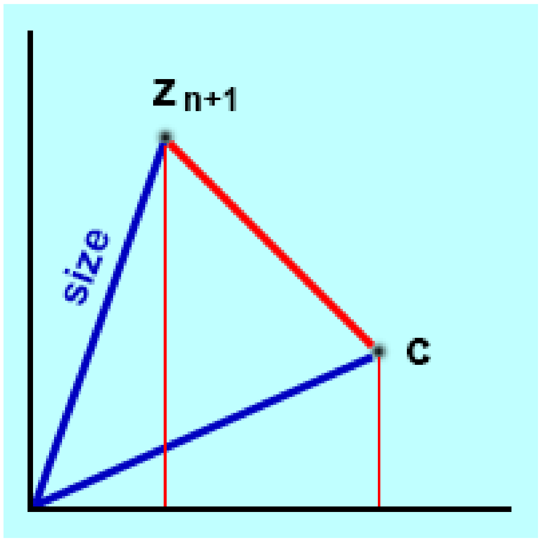
```
if (size > maxsize)
{
        sum = sum / (double)(n);
        sum2 = sum2 / (double(n - 1);
        fx = (double)(1.0 + il * lp – il * log(log(sqrt(size))));
         if (fx < 0.0) fx = 0.0;
        index = sum2 + (sum - sum2) * fx;
        color = (int)(index * 256.0);
}
```

In the (originally inadvertent) integer form of this algorithm, the hypotenuse of the triangle and the values of *min* and *max* are calculated from the integer values of the base and side. The color is calculated without using *fx* or *index*:

```
 color = ((int)((double)(n + 255) *
(sum2 + (sum - sum2)))) % 255 + 1;
```



Kerry Mitchell's algorithm from *UltraFractal* evaluates the size of *z(n+1)* (which is represented by the *size* variable in the code). *min* and *max* are calculated from the sizes of *z(n)* (which is represented by *x* and *y* in the code) and *c*.



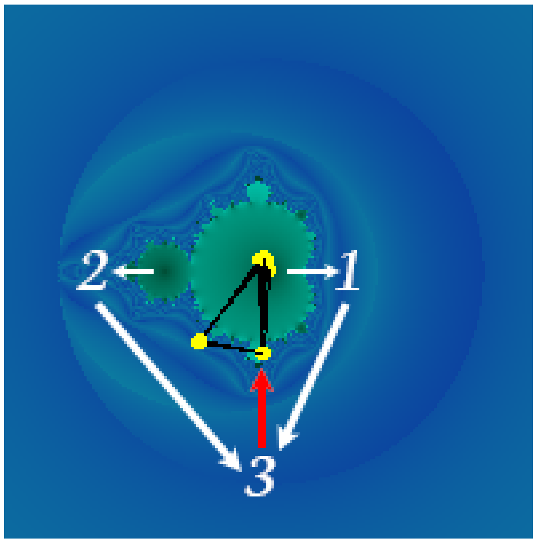
Damien Jones's algorithm from *UltraFractal* uses the triangle formed by *c*, *z(n+1)* and the origin. The side to be evaluated is the length from *c* to *z(n+1).* *min* and *max* are calculated from the sizes of *z(n+1)* (*size*) and *c*.

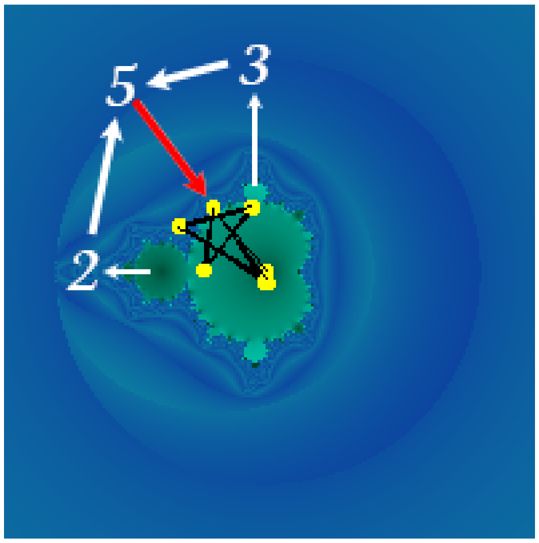
One feature of the last 2 algorithms is that they require large values for *maxsize* and *bailout*, in the order of 1 million or more, to yield the characteristic image patterns. Smaller values yield pleasing “precursor” images. Using different values for each parameter causes different scaling within each level set, yielding a form of binary decomposition image.

In the program, you can cycle between the default, Damien Jones's and Kerry Mitchell's algorithms by pressing **T** when you are using one of the TIA methods. With Damien's and Kerry's algorithms, you can raise the values of *maxsize* and *bailout* as high as 1e+16 before artifacts begin to appear. When applied as an **Inside Coloring Method**, this method is incompatible with the **Species of Origin** and **Cardioid Outside Coloring Methods**.

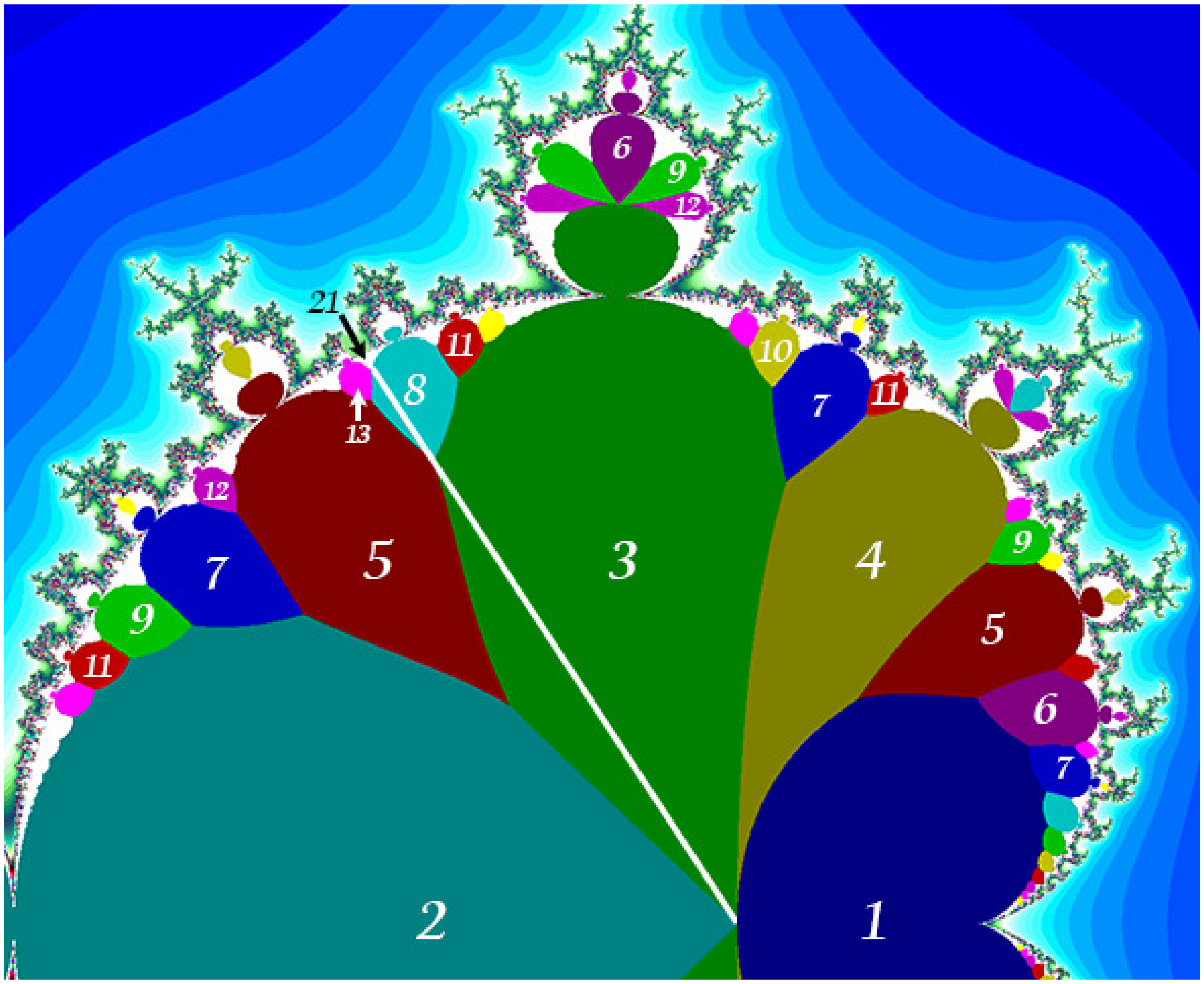
7. Periods and Buds

As stated in **Inside Coloring Methods**,}, the **BOF 34** method colors points in the Mandelbrot set and filled Julia sets according to their iteration numbers when their orbits are closest to the origin. In the Mandelbrot set, the result is a group of regions, or “domains”, each of which is associated with a bud that has the same period as the region's iteration index. (See the second image below.)





The term “periodic” refers to a cycle of alternating states, which eventually returns to the original state; the “period” is the number of such states per cycle. If a number of states are visited before the periodic cycle is entered, the system is called “pre-periodic”. In fractals, an iterating point can continue in its orbit cycle or escape to infinity. Each of the infinite number of buds in the Mandelbrot set has a characteristic period. The orbit-tracing mode, described in **Using the Program**, can reveal the period of individual buds. The above image shows orbits of points with periods of 3 and 5, starting at their respective buds, which are indicated by the red arrows. (See the next paragraph for the summation rule.)

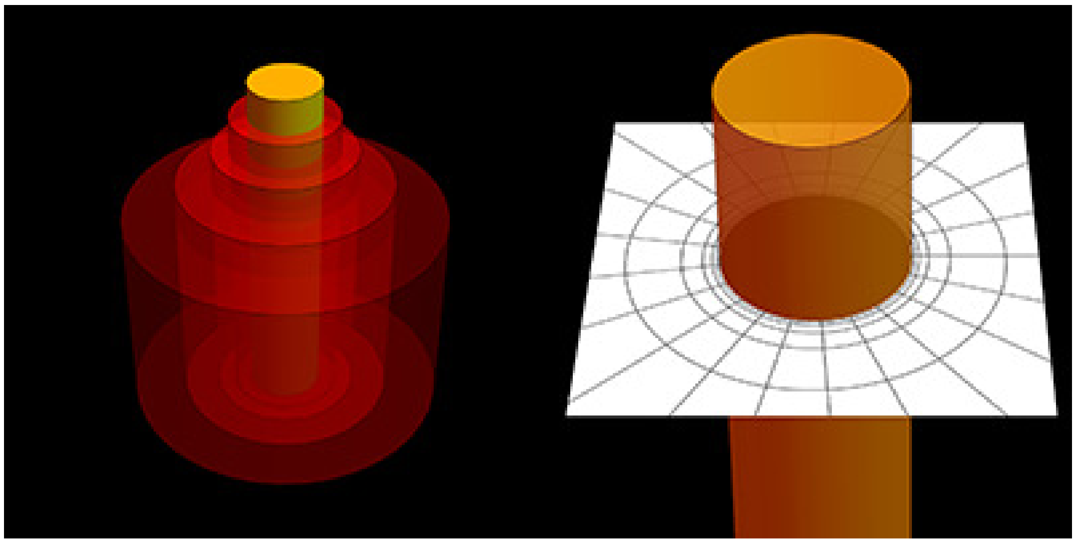


In the above image, note that the index of each region (domain) is the sum of the indices of its two flanking regions. Thus, Peitgen and Richter's observation follows that the series of indices "introduces a Fibonacci partition", which I've indicated on the image by the white line starting at the origin and proceeding through the regions for indices 1 through 21. The basic pattern of regions and indices observed in the main cardioid is repeated in each bud, but the intervals between matching regions are multiplied by the value of the period. In more general terms, describing buds rather than the regions of a specific coloring scheme, Peitgen *et al.* state that "Two given buds of periods p and q at the cardioid determine the period of the largest bud in between them as $p + q$. Similar rules are true for the buds on buds." (*Chaos and Fractals*, p. 866)

8. External Rays and Related Concepts

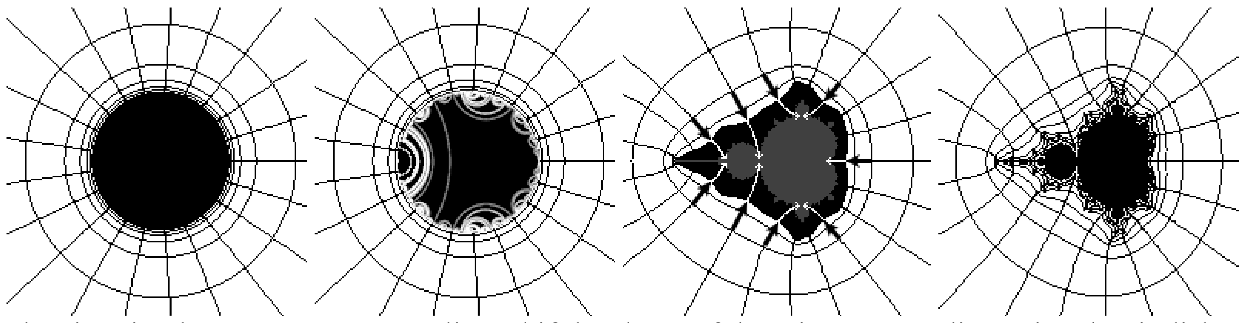
"Irritating ... or motivating; after all mathematicians live on problems more than on answers."
Adrien Douady - *The Beauty of Fractals*, p.167

Douady and his former student John H. Hubbard were early pioneers in the study of fractal mathematics in the 1980's, after Mandelbrot produced the first computer-generated images of the set which was named after him by Douady. One of their many contributions was to demonstrate the significance of equipotential lines, external angles and external rays by using an analogy from electrostatics. Extensive elaboration is found in the books of the Bremen group. This discussion will attempt to present my limited understanding of these concepts in a form which does not require a degree in mathematics to understand.



Imagine a straight, infinitely long wire, with a charge applied to its surface. There will be a potential difference between a point on the surface and another point at a given distance. The sets of all points with the same potential differences will consist of concentric cylinders. A cross section of this system will yield a plane that contains a two-dimensional representation of this system, a "unit disk" which corresponds to the Julia set of c = (0 + 0i). *Field lines* from the center of the disk connect surface points on the boundary of the disk (*i.e.* the actual Julia set) to outside points and will be orthogonal to the *equipotential lines* formed by the cross sections of the concentric cylinders. The position of an outside point is given by the angle, or *external argument*, formed by the field line, or *external ray*, at its intersection with the boundary at the surface point. Since the equipotential lines are circles and the external rays are straight, the external argument is simply the angle of the ray relative to the origin, or center of the wire. It is convenient to express these angles in "turns" from 0 to 1. A turn is simply (radians / 2π). Most of the angles of interest are rational and typically are referred to as (numerator / denominator).

In Mandelbrot and Julia sets, the equipotential lines of interest correspond to the boundaries between level sets, which are determined by *n*, the iteration counts of escaping points. The potential of a point can be approximated by the expression: $\log(size) / 2^n$, which is used in the **Continuous Potential** outside coloring algorithm, described in the **Outside Coloring Methods** section.



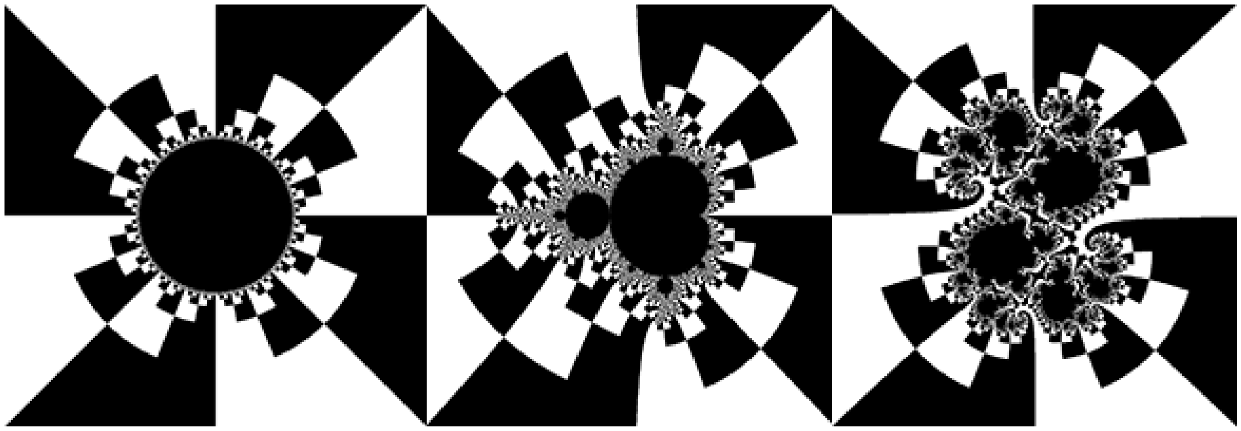
The situation becomes more complicated if the shape of the wire, or two-dimensional unit disk, changes. Imagine pinching some points of the disk together to form the *roots* of buds, and stretching other points out to form branching filaments. The ultimate result is the Mandelbrot set. Equipotential lines close to the set follow its contour closely. However, the closer these lines approach "infinity", or a suitably-determined bailout value, the closer $z^2 + c$ approaches z^2, and the lines become more circular. The external rays must remain orthogonal to the equipotential lines, so they are no longer straight, and thus determining the external arguments and tracing the rays becomes more difficult.

However, Douady and Hubbard showed, via a form of Boettcher conjugation, that the unit disk can be mapped to the points of Julia and Mandelbrot images. A point can be assigned polar coordinates which correspond to its potential and external argument respectively.

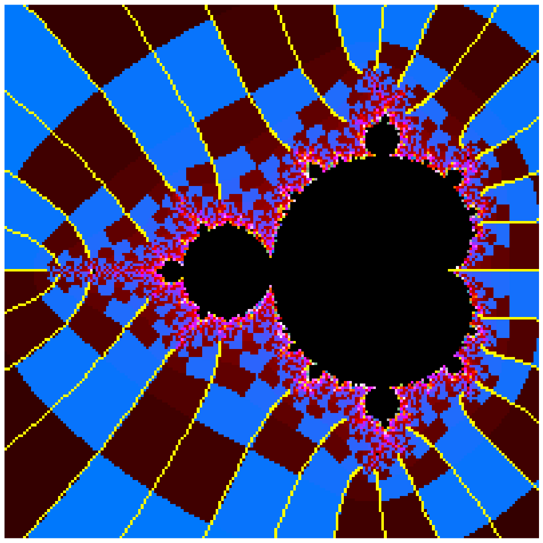
Period doubling can be used to express an argument as a binary expansion. When a starting point is iterated, it is squared. In polar coordinates, this means squaring the *modulus*, (its length) and doubling the *argument* (its angle). So, the square of (3.0, 1/6) is (9.0, 1/3). With each iteration, the resulting argument can be observed to be greater than 1/2, or not. The expansion is amended progressively with a 1 or 0 accordingly. For example, iterating 1/3 leads to 2/3, 4/3, 8/3 &c. which when normalized becomes a repeating series 1/3, 2/3, 1/3, 2/3 .... The binary expansion becomes 0.0, 0.01, 0.010, 0.0101, 0.0101 ... 01. It consists of a repeating period of 01, which starts immediately, so the argument is considered *periodic* with a period of 2. Iterating 2/7 leads to 2/7, 4/7, (8/7 == 1/7), 2/7, 4/7, 1/7.... The binary expansion is 0.0, 0.01. 0.010, 0.0100, 0.01001, 0.010010... 010. It is periodic with a period of 3. Iterating 1/12 leads to 2/12, 4/12, 8/12, (16/12 == 4/12), (32/12 == 8/12), (64/12 == 4/12) &c., so the expansion becomes 0.0, 0.00, 0.000, 0.0001, 0.00010, 0.000101 ... 01. It has a period of 2, but this period is not reached until the third iteration, so the argument is considered *pre-periodic*. It can be seen by this process that arguments with odd denominators are periodic and those with even denominators are pre-periodic.

To express the external argument of a point *z* in a form that can be used in computer programs, we can return to the Boettcher conjugation. The argument arg(*z*) of the point is the angle it forms with the real axis at the origin. Its external argument $\arg_c(z)$ is calculated for a suitable number of iterations *n* as follows:

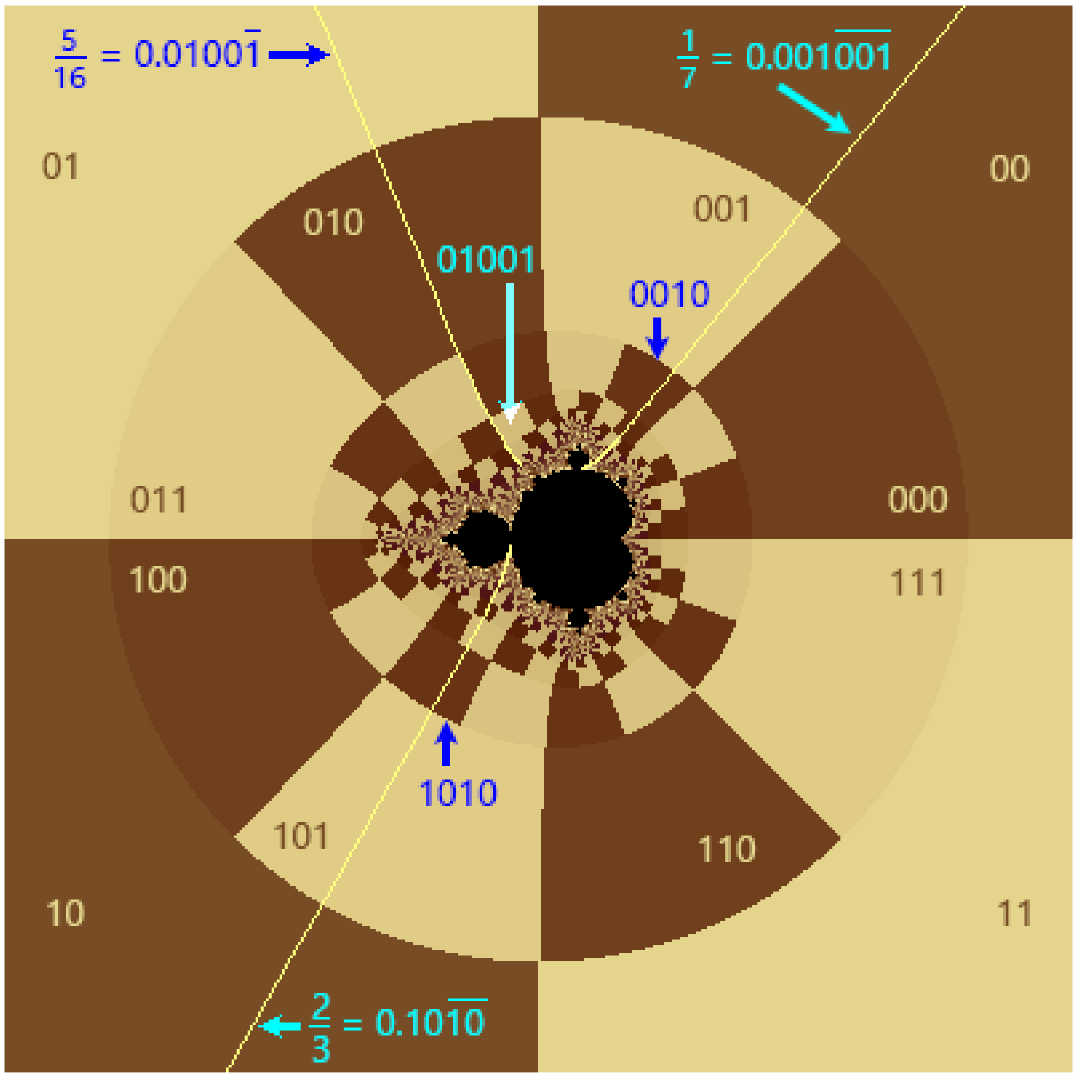
$$arg_c(z) = arg(z) + \sum_{n=1}^{\infty} \left(\frac{1}{2^n} * arg\left(\frac{f_c^n(z)}{f_c^n(z) - c} \right) \right)$$



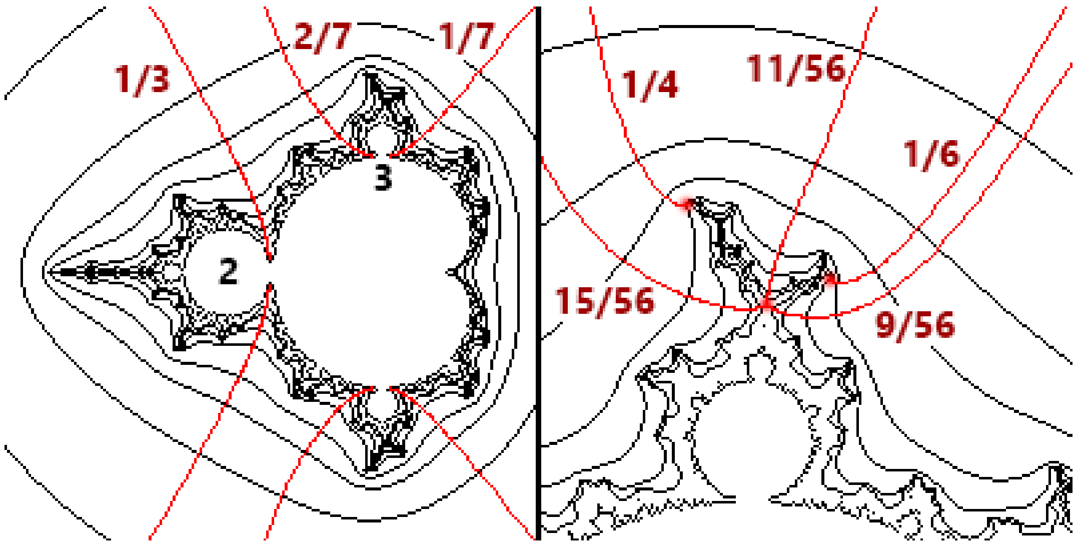
External rays are the curve segments which consist of points with the same external arguments. They can be demonstrated as boundaries between cells in binary decomposition images (in which points are colored according to whether the values of their arguments are greater than 1/2 or not).



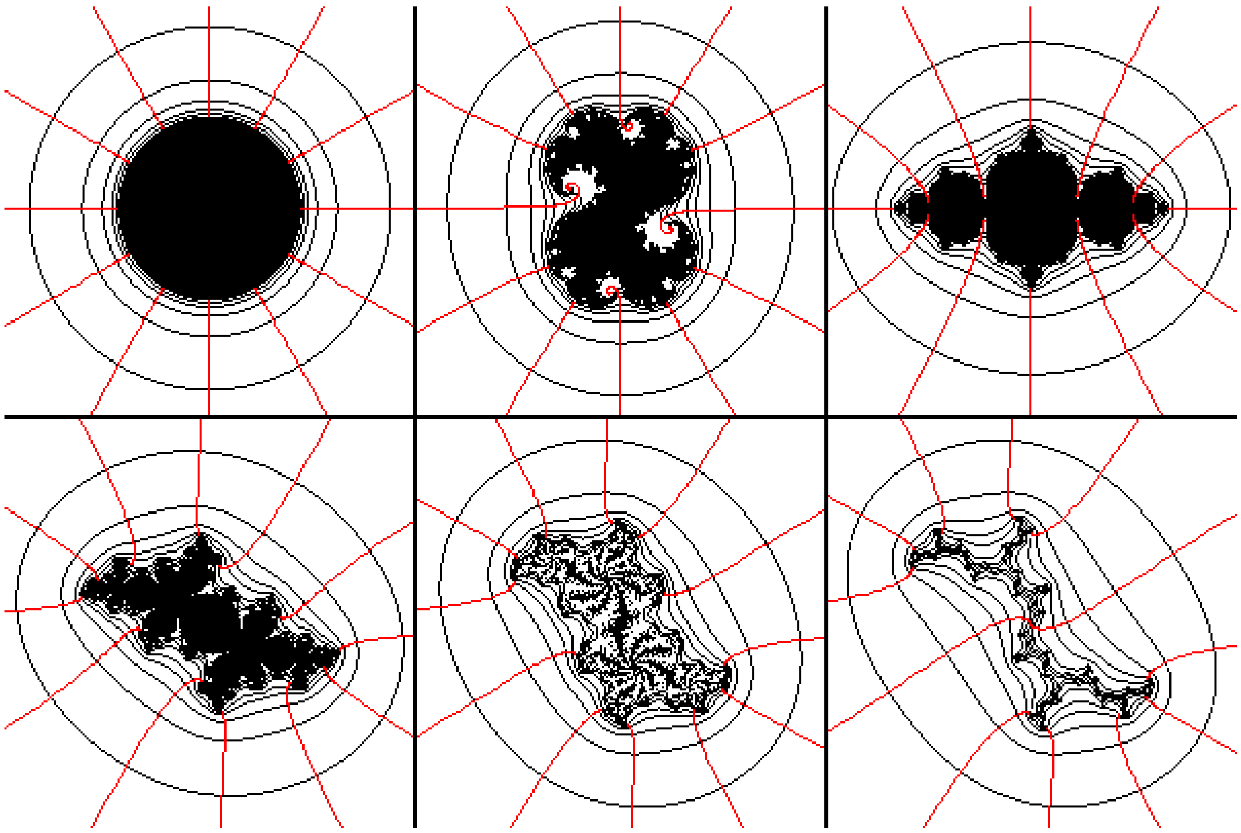
The relationship between binary decomposition and external rays can be seen more clearly in this image. These rays comprise a set with the even denominator 32 and terminate on filament tips or branch points.


$\frac{5}{16} = 0.0100\overline{1}$
$\frac{1}{7} = 0.\overline{001}\,\overline{001}$
01
00
010
001
01001
0010
011
000
100
111
1010
101
110
10
11
$\frac{2}{3} = 0.\overline{10}\,\overline{10}$

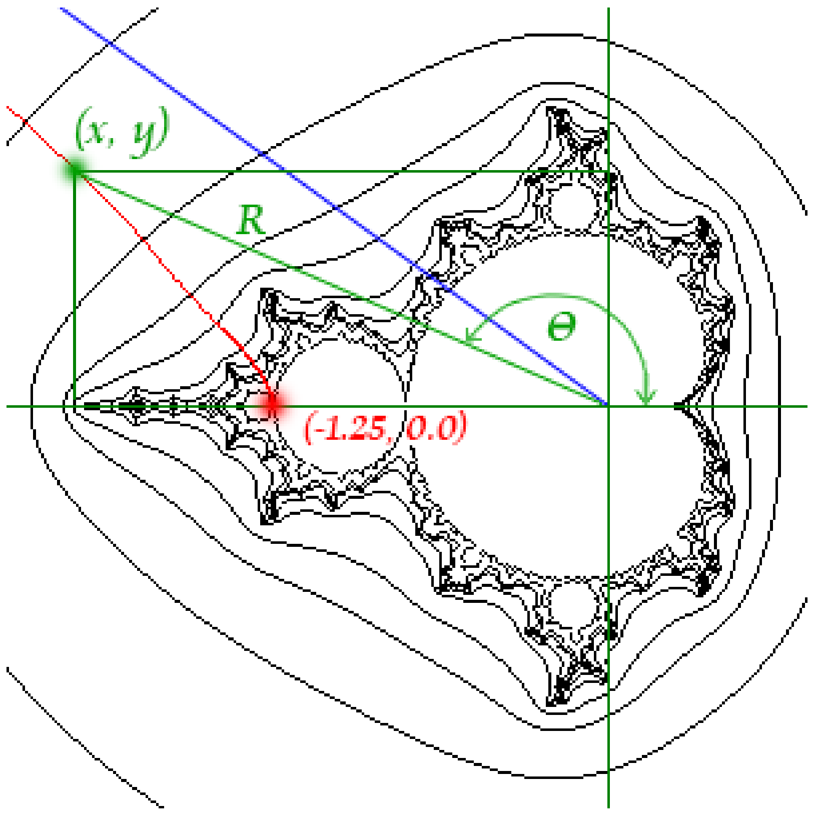
In this image, the binary decomposition cells are labeled in a counter-clockwise direction. Several external rays are shown. The rays with odd denominators (1/7 and 2/3), which are periodic, go to root points of buds (the “pinch” points where buds are attached to larger buds). The ray with an even denominator (5/16) is pre-periodic and attaches to the point at the tip of a filament; such points are known as *Misiurewicz points*, and their orbits also are pre-periodic. In both cases, the rays travel through the binary decomposition cells whose labels correspond to the binary expansions of the external arguments of the rays.



In the main cardioid, the pairs of periodic rays attached to the root of a bud of period *p* have the denominator (2^p - 1). Pre-periodic rays which develop a period *p* after *n* iterations have the denominator 2^n * (2^p - 1). Similar relationships can be observed in the smaller buds. Misiurewicz points at branching points of filaments have two or more rays attached to them, as seen at the point with rays of 9/56, 11/56 and 15/56.



This image shows some external rays in a series of Julia sets.



While binary decomposition is a simple way to demonstrate external rays, it is not so easy to draw them individually. This image shows the external ray with the external argument 2/5 in red, and a straight ray from the origin in blue, with an argument of (2/5 * 2π) radians or 144º and a slope of tan(144º) = -0.727. The two rays meet at infinity, from which the external ray extends toward the root point of the period 4 bud at (-1.25, 0.0).

At (x, y), the length "R" of the green segment from the origin is 2.192 and its argument "Θ" is 2.723 radians or 156º. Its position in polar coordinates is (R, Θ), which also can be expressed as the complex number R * $e^{(\Theta * i)}$. From this, we can return to x = R * cos(Θ) = -2.0 and y = R * sin(Θ) = 0.897. The goal is to find such points (x(n), y(n)), for the given argument Θ and a series of radii R(n), which when iterated approximate the points R(n) * $e^{(\Theta * i)}$. Since the Mandelbrot function is M(z) = z^2 + c (or similarly for the Julia function), and a point in polar coordinates is squared by squaring the radius and doubling the argument, this means that

$$M^{n}_{(z)} = R^{2^{n}} e^{(2^{n} \theta i)}$$

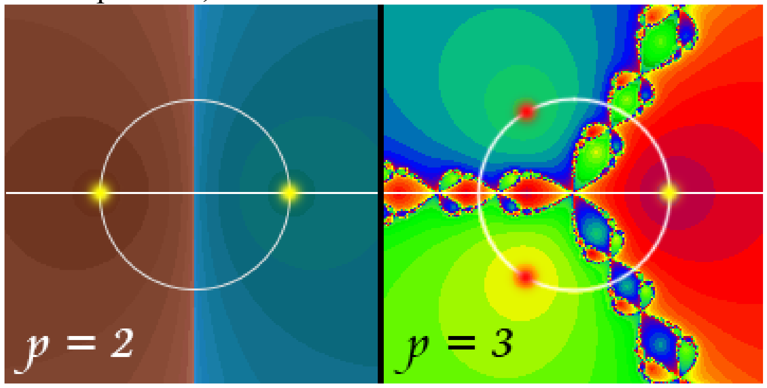
We will return to this equation below.

A rough approximation of an external ray can be obtained by asking the computer, after iterating each point and calculating its external argument, to color it distinctly if that argument happens to be the desired one. Problems with this approach include the fact that the screen is divided into pixels, and the desired argument, unless it is 0 or 1/2, probably does not fall directly onto the pixel's value. So a range must be specified which captures too many pixels in some locations in order not to have gaps in the ray. But since the argument values are tightly compacted close to the M-set, that range will cause significant spillover artifacts in this part of the image.

External rays for Julia sets can be drawn by inverse iteration, since the value of c remains constant. A point with the desired argument near “infinity” is selected, so that $z^2 + c$ can be considered equivalent to z^2. Iterations of sqrt(z - c) gradually approach the Julia set along a ray. (Depending on the selected *maxiter*, the points land on one of the rays described by period doubling of the argument.) A challenge is that at each step there are 2 roots. The correct one can be selected by forward iterating the previous value of z while backward iterating the current z and saving the values, comparing them at each step to the 2 roots and selecting the closer one. Another problem is that as the points approach the set boundary, their differences become small enough to exceed the floating point limit of the computer, necessitating multiple precision calculation algorithms to draw the ray relatively close to the boundary.

A more satisfactory approach, which works for both Julia and Mandelbrot sets, is to use Newton's method, for which I've adapted Wolf Jung's code from his program *Mandel*. This technique converges on the root(s) of a function “F” which equals zero, by starting with a “reasonable” approximation “X(0)” and iterating X(n+1) = X(n) - F(X(n)) / F' (X(n)), where F' is the derivative of F. A popular use of this technique produces fractals from the equation $z^p = 1$. Thus $z^p - 1 = 0$ and with a bit of manipulation,

$$z_{n+1} = \frac{(p - 1) * z_n^p + 1}{p * z_n^{(p-1)}}$$



With p = 2, there are two roots, 1.0 and -1.0, on the real number axis. Along this axis, and also anywhere on the complex plane, all points where $x < 0.0$ (brown) converge to (-1.0, 0.0), all points where $x > 0.0$ (blue) converge to (1.0, 0.0), and points where $x = 0.0$ are indeterminate, requiring division by 0. With p = 3, there is only one root on the real axis, but there are two additional roots $(-1/2, \pm \sqrt{3}/2)$ on the complex plane, all falling on the circle with radius = 1. Indeterminate points result in chaotic orbits which make the fractal interesting.

Returning to external rays, we revise the previous equation:

$$F^n_{(z)} = M^n_{(z)} - R^{2^n} e^{(2^n \theta i)} = 0$$

$$z_{n+1} = z_n - \frac{F(z_n)}{F'(z_n)}$$

A large initial value of R is selected to represent infinity, and an initial approximation z(0) is set at R * $e^{(\Theta * i)}$, with Θ being the desired external argument converted from turns to radians. z(0) then is iterated, which is as challenging to program as it looks. When the difference between z(n) and z(n+1) is less than a predetermined value, z(n+1) is accepted as a point on the ray. Drawing the ray on the screen consists of connecting the point to the previous point with a line segment. R is progressively decreased and the process is repeated to generate new points.

Drawing external rays in the program: Select the menu item **Options → External Rays**. A dialog box will let you select the external argument of the desired ray as *N / D*. The maximum denominator is 256, and the numerator must be between 0 and (*D* - 1). You can choose to draw just the single ray, or all rays with the designated denominator, in which case the value of *N* does not matter. You can choose the color that the ray(s) will be drawn with. The ray(s) are automatically drawn when you click the *OK* button. You can draw as many rays as you like. To erase the rays, redraw the fractal with the **Plot Fractal** menu command. If for some reason you zoom into the image extremely deeply, Newton's method might misfire and jump to the wrong root.

9. Conclusion

Considering the “informational content” of a fractal image, Douady emphasizes that while a textual description would be “enormous”, the program that produces it is very short, with the mathematical part taking only a few lines within it. He draws an analogy with the “phenomenon of developing information” from a compact key (DNA) in biology. A complete transcription of an organism's DNA would be markedly brief compared to ponderous volumes of anatomy, physiology and psychology. He imagines “scientists faced with the collections of Julia sets without knowing where they came from”, recording lengthy observations and giving “a description of the specific features attached to each term of the classification.” But he clarifies, “I am not claiming that Julia sets can provide a model for any biological phenomenon, but they are a striking example of how a very simple dynamical system can develop the small information contained in a key, and produce various highly organized structures.”

Map 36, *The Beauty of Fractals*

Mathematical truths, ranging from basic laws of arithmetic to expressions involving imaginary numbers that can produce the beauty of fractals, are neither matter nor energy. They can be considered to exist independent of space and time, and to predate the Big Bang and the density of matter that exploded, regardless of where it came from. They seem to represent well the sentiment of Keats's Grecian urn, that “Beauty is truth, truth beauty, - that is all / Ye know on earth, and all ye need to know."

Programming and help file authoring by:

Michael Sargent
South Burlington, Vermont
September 2015

10. Disclaimer