

Fractal Geometry and the Escape-time algorithm as Graphic Art Tools

Daniel E. Lanier, presentation at SDSU Mathematics Summer Seminar, July 2008.

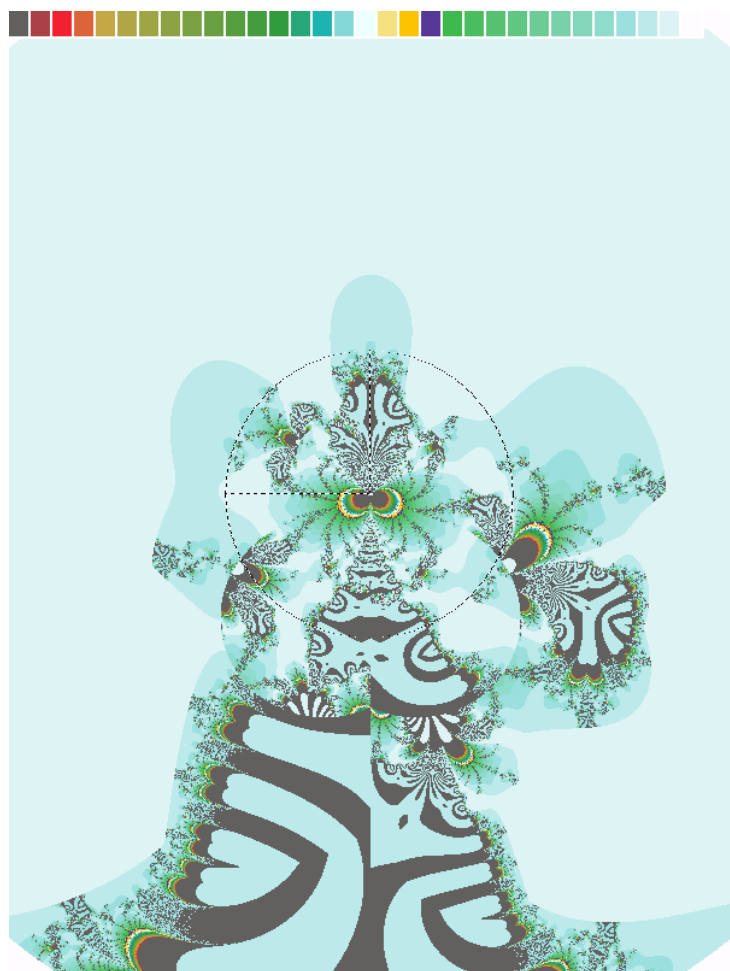


Figure 1. Fractal suggestion of a dog. Imaginary axis points left from the origin between eyes. Real axis points upware inside unit circle.

The Dog above uses a custom color map of 64 colors. The 600 x 800 Demon on the right was generated in about 93 seconds in Matlab using a different function and set of constants. The colormap is Matlab's 'jet' with 64 colors as shown in the backwards colorbar above: 64 is on the left, 0 on the right.

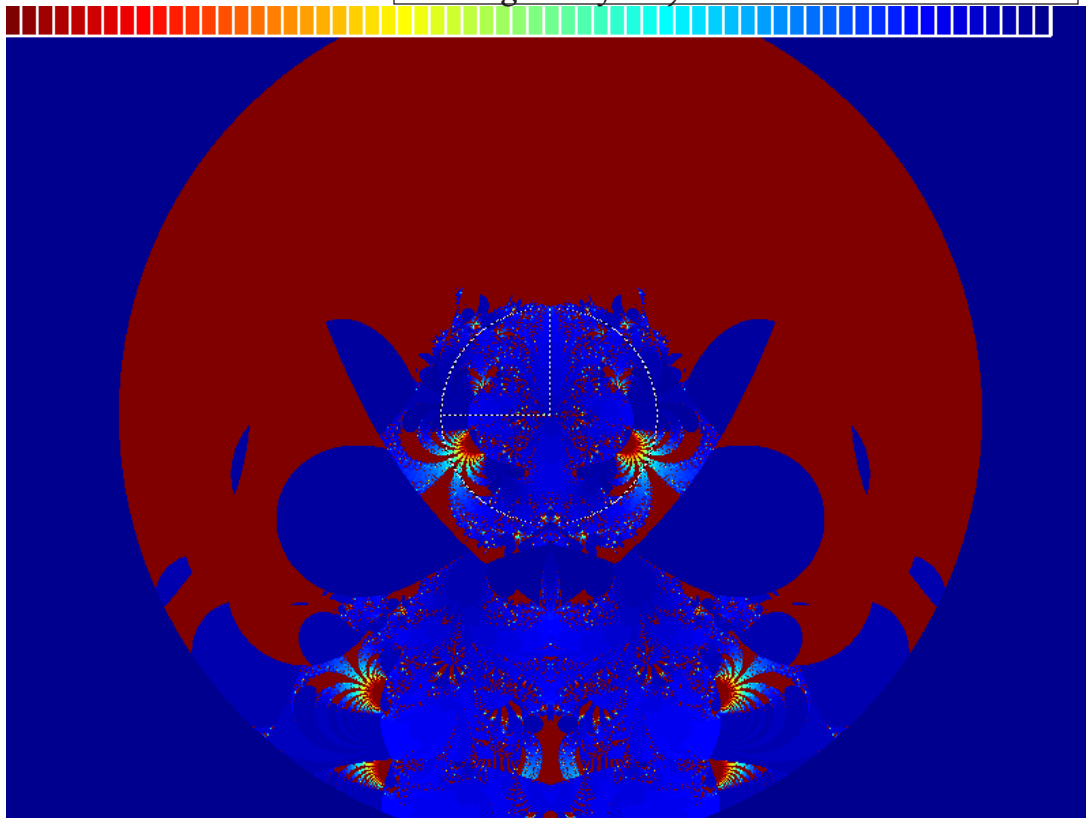
Both figures were discovered by using non-linear equations on the complex plane and using a random search technique to explore parameters.

Fractal Geometry has been called natural geometry or the geometry of nature because of the way that zooming in or out shows more and more detail rather than less. The fractal geometers tool box is still pretty sparsely populated but computer technology is making cpu-abusing algorithms computationally feasible. The 800 by 600 pixel figure on the left was generated by Matlab 7.4.0 (R2007a) in about 23 seconds using the escape-time algorithm on a 2Ghz computer with 2GB of memory with a 150GB hard drive. Until recently generating this kind of figure was not computationally practical on a home computer.

With so much speed and memory it is now practical to explore function and parameter space freely enough to find combinations that produce artistic (if I do say so myself) figures. Fortunately, once such a solution is found, it may be stored in a few lines of code and reproduced in fine detail at any scale. Some may ask 'is it art?' But isn't it just the art that has always been?

None of the images were retouched in photoshop or digitally edited other than to reposition and size them for this pdf.

Figure 2. Scary non-linear difference equation demon grown far beyond the unit circle.



Background and history of the escape time algorithm

The escape–time algorithm takes a point in the complex plane as the argument to a discrete complex function (a difference equation) and recalculates it repeatedly until the variable is beyond some arbitrary distance or until the number of iterations reaches some arbitrary limit. If the variable starts at a 'fixed point' the algorithm might iterate endlessly unless some computationally practical numerical limit is set. The the distance limit, iterations limit, set of starting values, color map and the equation itself all control the appearance of the graphic image. Thus it is an art.

French mathematician Gaston Maurice Julia first published the idea of iterating a rational function in 1918 but not much was done with it graphically until Benoit B. Mandelbrot began producing images he named fractals in the 1970s. Using computers (he worked for IBM) he was able to take Julia's idea to the next level. Mandelbrot retired as Sterling Professor of Mathematical Science at Yale University in 2005, and describes his book – The Fractal Geometry of Nature (still published) as an essay on fractal geometry.

The notion of iterating a function to see where the points in the plane end up has to do with differential equations. Such equations that have no analytic solution can still be made to show where they have attractors, repellers and fixed points by iterating from each starting point in the plane. It is quite suprising to see that two points, infintestimally close to each other can diverge so completely within a few iterations.

To summarize the proceedure to make a digital picture: 1) assign a complex number each pixel, 2) iterate each number in the function until it either exits the region of interest or reaches the counting limit, and 3) use the count to color the pixel. Matlab code included here should make the process more apparent to anyone who will take the time to study it.

Complex number rational

The Julia and Mandelbrot sets both use the same simple complex number function:

$$Z_{n+1} = Z_n^2 - c \tag{1}$$

where Z and c are complex numbers of the form $z = x + iy$ with imaginary number $i =$ the square root of -1 . This apparently needless (and tedious) complication of a difference equation is acually a compact, convenient way to express devlishly complicated non–linear systems of equations that are iterated in the while loop of the algorithm. To see how they relate, consider equation (1) in the context of the transform of a system of two difference equations.

$$Z_{n+1} = Z_n^2 - c = (x_n + iy_n)^2 - c_{real} - ic_{imag}$$

$$Z_{n+1} = x_n^2 - y_n^2 + i2x_ny_n - c_{real} - ic_{imag}$$

or,

$$x_{n+1} = x_n^2 - y_n^2 - c_{real}$$

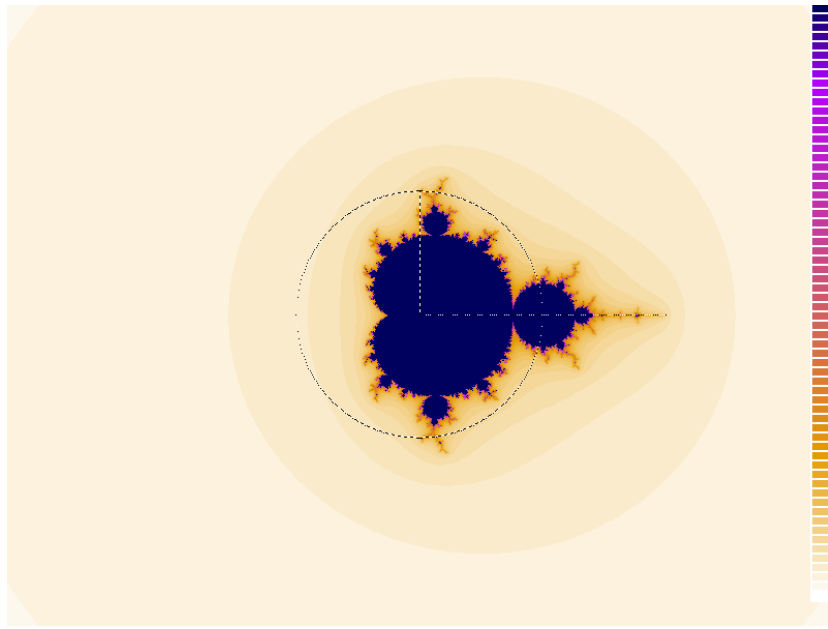
$$y_{n+1} = 2x_ny_n - c_{imag}$$

where Mandelbrot sets use $Z_0 = 0$, with c set to the pixel value, and Julia sets set Z_0 to the pixel value and c is choosen to determine the shape of the figure. Complex number functions really take two arguments and return two numbers, the real part and the imaginary part.

Since one interpretation of the fractal sets is to visually see how initial conditions fare over time (or how the plane is distorted by the function), it is more convenient to use a programming language with native complex number support (Matlab, Fortran) to generate the graphic matrix. The mathematical truth of these figures is that they point to attracting and repelling fixed points and thus show the sensitivity to initial conditions for the function used. If you don't like complex numbers yet, then what are the real plane difference equations for this function.

$$Z_{n+1} = Z_n \left(2Z_n^{2x_n} \right) \tag{2}$$

Background and history of the escape time algorithm



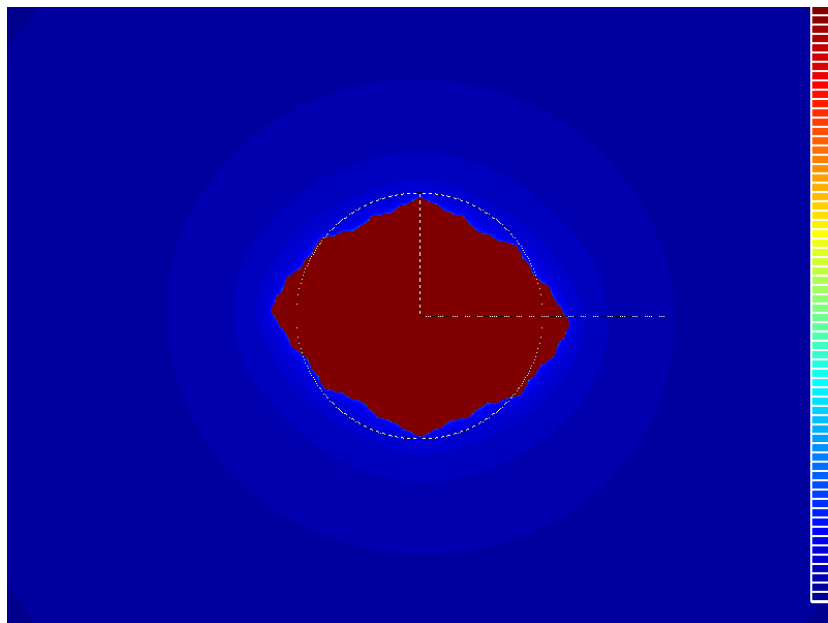
Julia sets and Mandelbrot sets both use the same equation in the escape time algorithm with different starting points and constants. Color = the number of iterations where the highest on the bar = 64 (or more) and lowest = 1. The number of colors = the number of iterations.

Mandelbrot Set

figure 3

Mandelbrot set ($z = z^2 - c, z_0 = 0$) on complex plane showing unit circle, positive real (horizontal) and complex axes.

$$c = x_{\text{col}} + iy_{\text{row}}$$

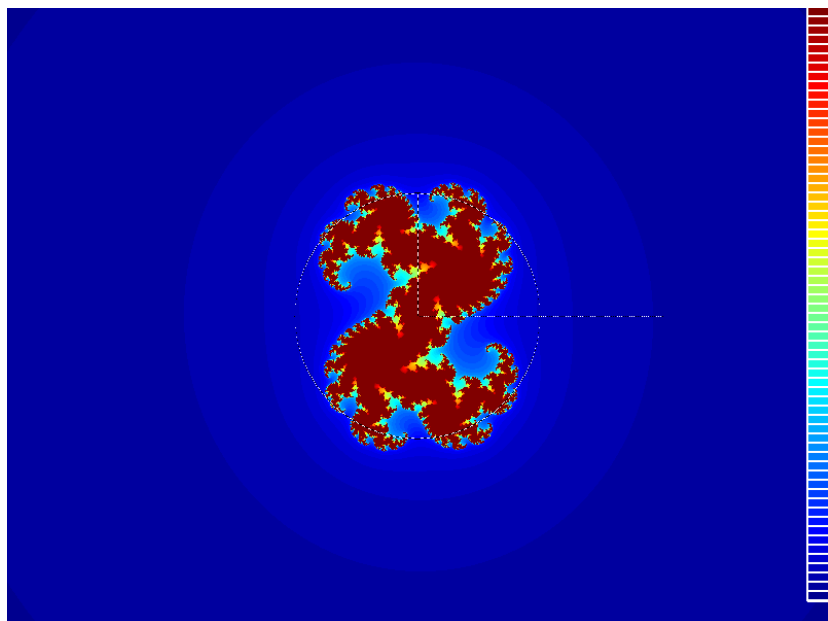


Julia sets have a specific form that depends on the value of the complex constant c . Many values of c just produce a potato or a cloud, but certain ranges produce interesting fractal-geometric figures. That is to say that zooming in on them show more and more detail, with self-similarity.

Julia Potato

figure 4

Julia set ($z = z^2 - c, z_0 = x_{\text{col}} + iy_{\text{row}}$) on complex plane showing unit circle, positive real (horizontal) and complex axes. $c = -0.18687260 - 0.48976440i$.

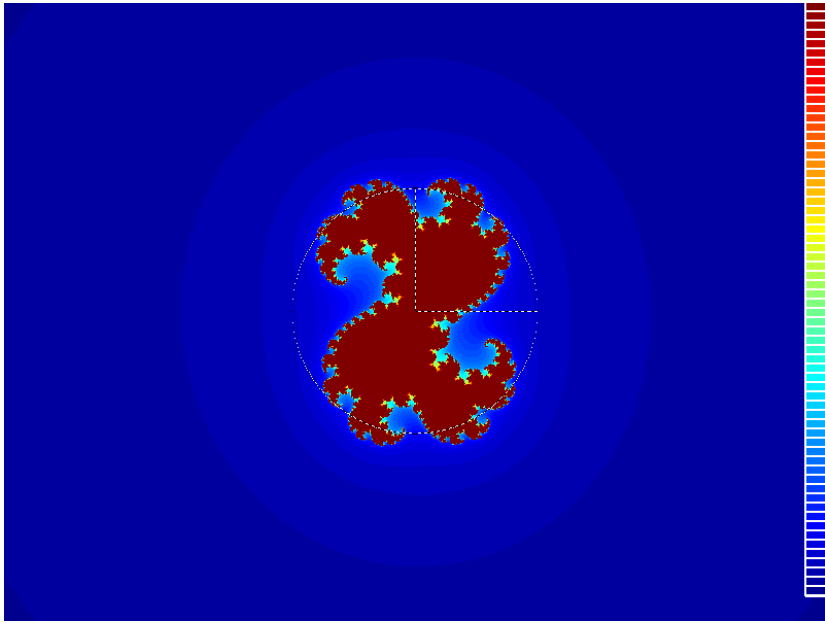


Julia Dragoon

figure 5

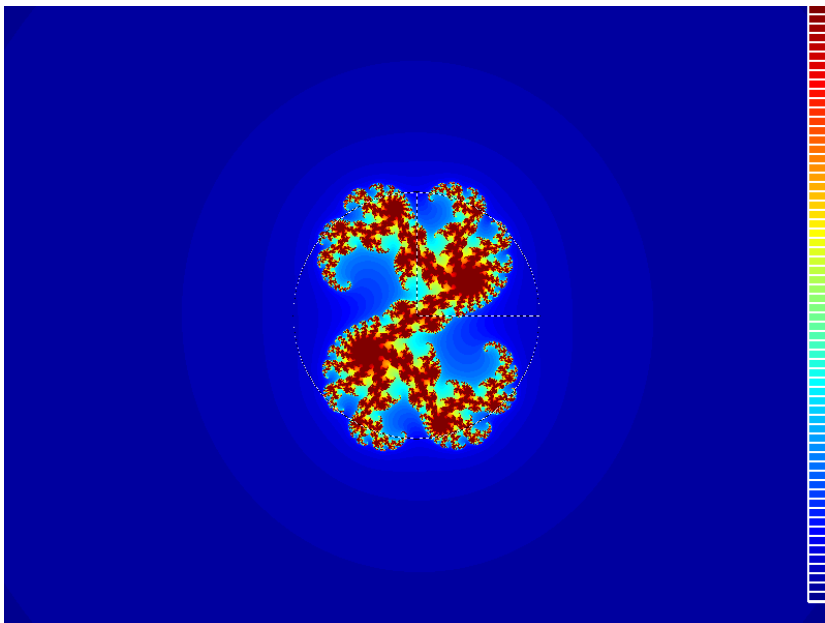
Julia set ($z = z^2 - c, z_0 = x_{\text{col}} + iy_{\text{row}}$) on complex plane showing unit circle, positive real (horizontal) and complex axes. $c = -0.32 - 0.043i$.

Background and history of the escape time algorithm



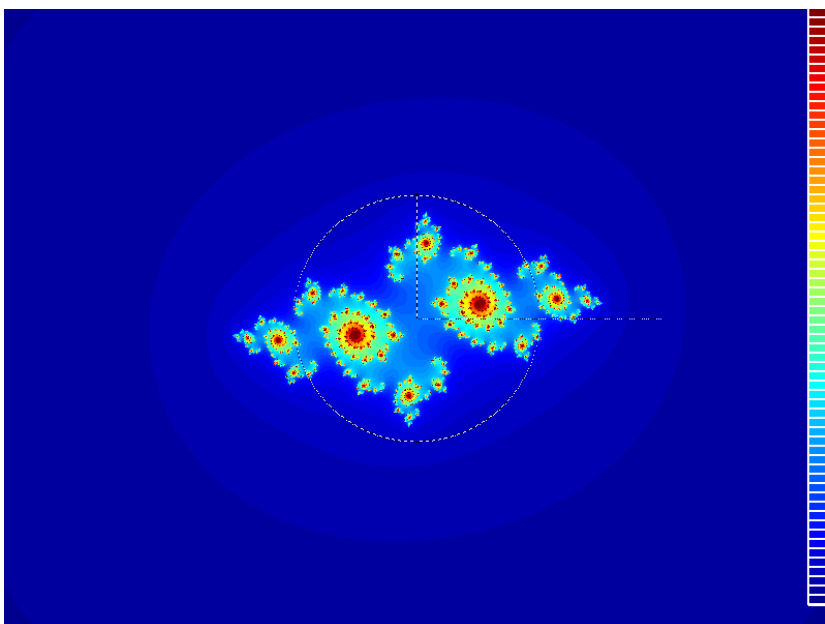
Julia Fat Dragon
figure 6

Julia set ($z = z^2 - c$, $z_0 = x_{\text{col}} + iy_{\text{row}}$) on complex plane showing unit circle, positive real (horizontal) and complex axes. $c = -0.32 - 0.0471i$.



Julia Skinny Dragon
figure 7

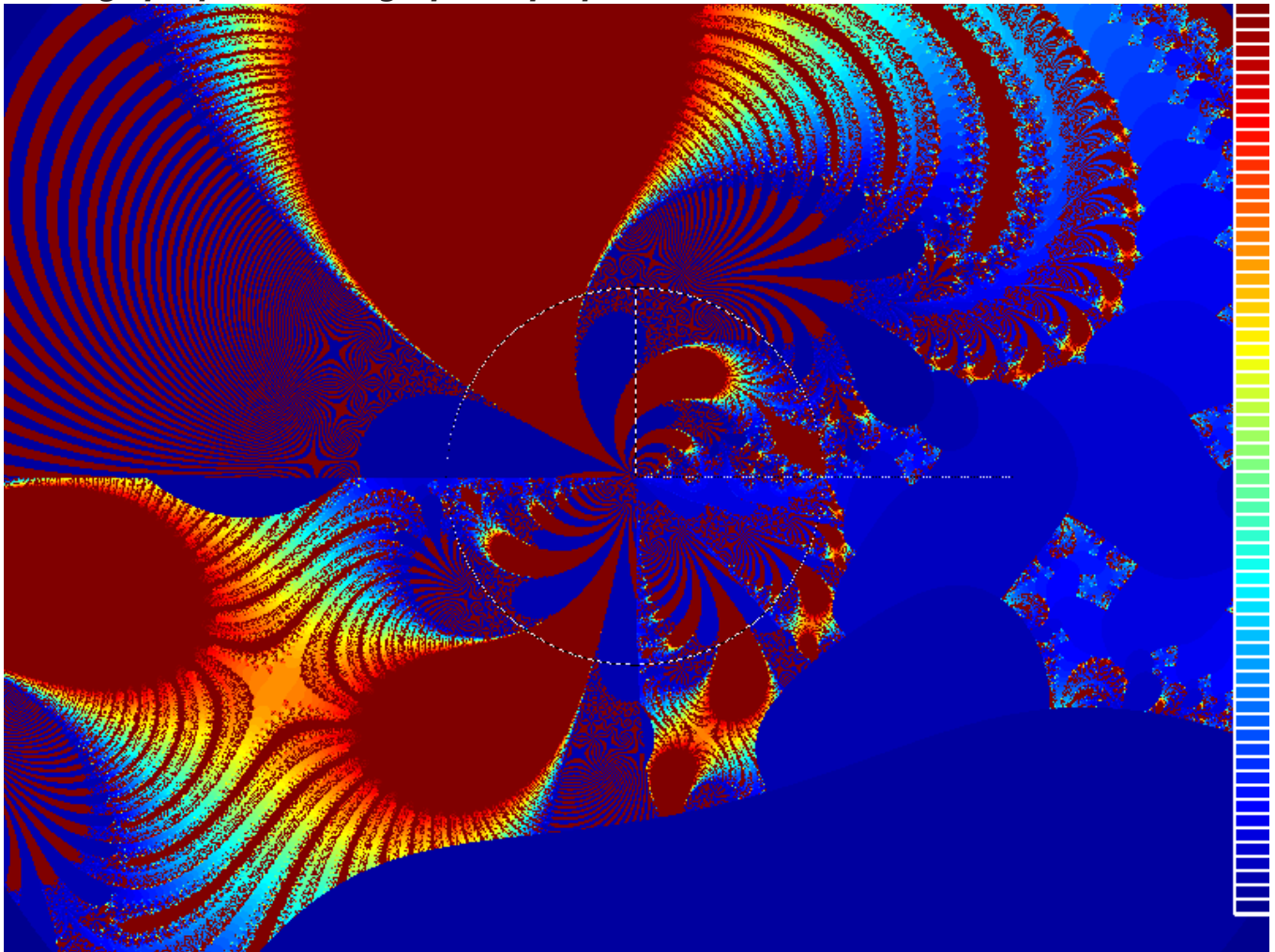
Julia set ($z = z^2 - c$, $z_0 = x_{\text{col}} + iy_{\text{row}}$) on complex plane showing unit circle, positive real (horizontal) and complex axes. $c = -0.32 - 0.0371i$.



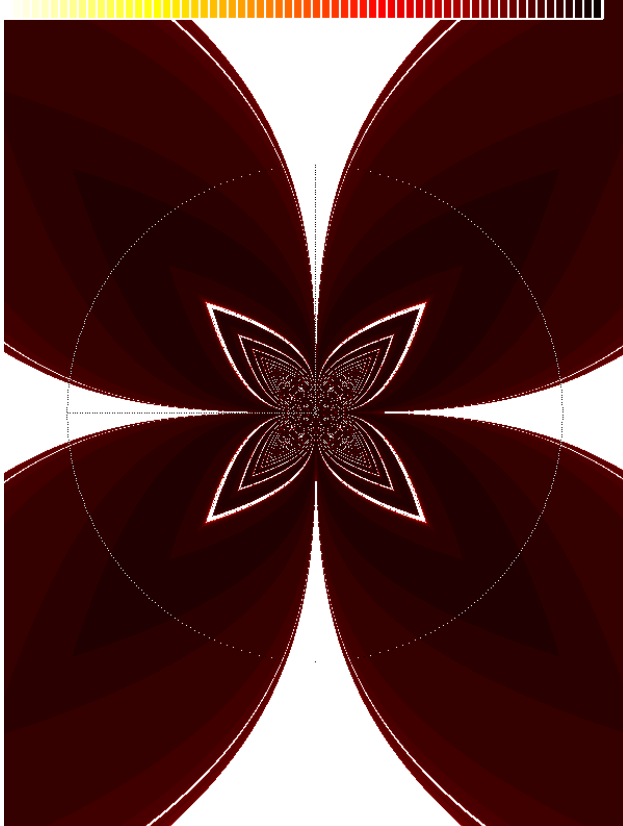
Julia Spiral Florets
figure 8

Julia set ($z = z^2 - c$, $z_0 = x_{\text{col}} + iy_{\text{row}}$) on complex plane showing unit circle, positive real (horizontal) and complex axes. $c = 0.75126706 + 0.25509512i$

Making up equations for graphical purposes



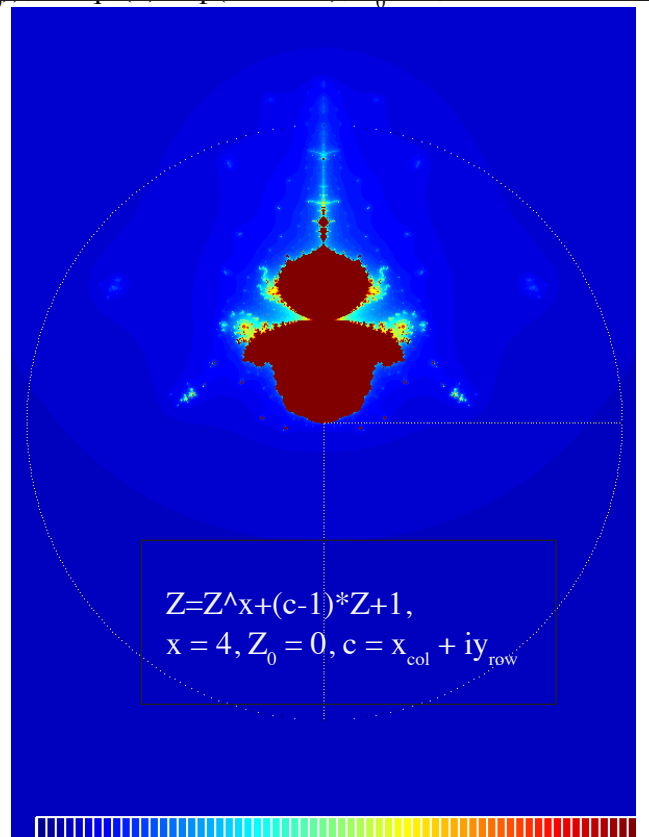
Above: A Julia set with function, $Z_{n+1} = Z_n \left(2Z_n^{2x} \right)$, $x = -0.64514600$, iteration limit = 64 and colormap jet.
 Below left: $n = \max(\text{real}(z_0), \text{imag}(z_0))$, $m = \min(\text{real}(z_0), \text{imag}(z_0))$, $k = \sqrt{3} - \exp(i * n / m^c)$, $Z_0 = k$, $Z = Z^c - k^Z + k$



Above
figure 9

Left
figure 10

Right
figure 11



Sample Matlab code for generating Julia Set figures

Must be saved in a file with the same name as the function in a directory that is in the current path.

```
1 function A = SimplyFractalJulia(h,w,c)
2 %
3 % Usage:
4 %   A = SimplyFractalJulia(h,w,c);
5 %
6 % sample command line call:
7 %   A = SimplyFractalJulia(300,400,0.32+0.043i);
8 %   try: c = -0.11+0.67i; or c = -0.39054-0.58679i;
9 % Input:
10 %       h = height in pixels
11 %       w = width in pixels
12 % Output:
13 %       Image matrix A is an unsigned 8 bit, h by w by 3 matrix
14
15 % set constants here: change to experiment
16 ETMAX = 32; % set maximum escape time & # of color
17 MAXDIST = 2; % distance limit for Escape Set
18
19 % define x and y axes as linear arrays
20 Xaxis = linspace(-2.1,2.1,w); % 'w' divisions on -2.1 to + 2.1
21 Yaxis = linspace(-2.1*h/w,2.1*h/w,h)*i;
22
23 % matlab functions define colormap matrix (ETMAX rows by 3 columns
24 mp = colormap( hot(ETMAX) );
25 mp = cast(mp*255,'uint8');
26
27 % allocate the matrix -> use unsigned 8-bit integers for images
28 A = uint8( zeros(h,w,3) ); % think three pages of h by w
29
30 % for each row, then for each column in that row...
31 for m = 1:h % there are h = height-of-image rows
32     for n = 1:w % and w = columns in the image
33
34
35         Z = Xaxis(n) + Yaxis(m); % Z is the intersection of m,n
36
37         esctime = 1; % initialize escape time counter
38         while esctime < ETMAX && abs(Z) < MAXDIST
39
40             Z = Z^2 + c;
41
42             esctime = esctime + 1; % count: # times through while loop
43         end
44
45         A(m,n,:) = mp(esctime,:); % set three bytes with the color map
46                                 % using escape time as the index
47     end
48 end
49
50 % use matlab to display the image with a colorbar & pixel info
51 imshow(A); colorbar; impixelinfo;
```

With the above file in the current path type the following at the Matlab command prompt to run.

```
>A = SimplyFractalJulia(300,400,0.32+0.43i);
```

Matlab will either produce a figure showing the Julia set fractal or an error message if something is wrong.

Sample Matlab code for generating Mandelbrot set figures

Must be saved in a file with the same name as the function in a directory that is in the current path.

```
1 function A = SimplyFractalMB(h,w)
2 %
3 % Usage:
4 %   A = SimplyFractalMB(h,w);
5 %
6 % sample command line call:
7 %   A = SimplyFractalMB(300,400);
8 %
9 % Input:
10 %       h = height in pixels
11 %       w = width in pixels
12 % Output:
13 %       Image matrix A is an unsigned 8 bit, h by w by 3 matrix
14
15 % set constants here: change to experiment
16 ETMAX = 32; % set maximum escape time & #of colors
17 MAXDIST = 2; % distance limit for Escape
18
19 % define x and y axes as linear arrays
20 Xaxis = linspace(-2.1,2.1,w); % 'w' divisions on -2.1 to + 2.1
21 Yaxis = linspace(-2.1*h/w,2.1*h/w,h)*i;
22
23 % matlab functions define colormap matrix (ETMAX rows x 3 columns)
24 mp = colormap( hot(ETMAX) );
25 mp = cast(mp*255,'uint8');
26
27 % allocate the matrix -> use unsigned 8-bit integers for images
28 A = uint8( zeros(h,w,3) ); % think three pages of h by w pixels
29
30 % for each row, then for each column in that row...
31 for m = 1:h % there are h = height-of-image rows
32     for n = 1:w % and w = columns in the image
33
34         Z=0; % Z starts the loop at the origin
35         c = Xaxis(n) + Yaxis(m); % c is the intersection of m,n
36
37         esctime = 1; % initialize escape time counter
38         while esctime < ETMAX && abs(Z) < MAXDIST
39
40             Z = Z^2 + c;
41
42             esctime = esctime + 1; % count: # times through while loop
43         end
44
45         A(m,n,:) = mp(esctime,:); % set three bytes with the color map
46 % using escape time as the index
47     end
48 end
49
50 % use matlab to display the image with a colorbar & pixel info
51 imshow(A); colorbar; impixelinfo;
```

With the above file in the current path type the following at the Matlab command prompt to run.

```
>A = SimplyFractalMB(300,400);
```

Matlab will either produce a figure showing the Julia set fractal or an error message if something is wrong.

Sample Matlab code for generating Julia and Mandelbrot figures - explanation

The two functions (programs) are identical except for the signature (line 1) and lines 34 & 35. The lines that begin with the percent sign and show up in green are comments and do not need to be typed in if you are a one finger typist. After typing in one or the other, doing a Save As and changing the three lines (1, 34 and 35) will produce the other.

The choice of parameters (named c in the Julia program) is critical. Most choices for the parameter will produce something like a blob of the smallest number's color in the middle of the largest number's color. Three settings for c are given in the comments at the top of the Julia code. Small changes in those parameter settings produce changes in the graphical output. Raising the Z squared term to a higher power works to produce more points on the figure.

Making up equations and substituting them for Z squared plus c sometimes makes very interesting graphics, sometimes not. One that works is,

$$Z_{n+1} = \frac{aZ_n^2 + b}{cZ_n^2 + d} \quad (3)$$

where

$$ad - bc = 1 \quad (4)$$

Writing a function to return a , b , c and d with the constraint of equation (4) is a good Matlab learning project. The figure below was produced with equation (3) using an escape time iteration limit = 128, escape boundary limit = 8, and these settings for a , b , c and d .

$a = 0.84071 - 0.25428i$, $b = -1.76241 - 1.55060i$, $c = 0.61446 - 0.50774i$, $d = -0.92926 - 0.34998i$

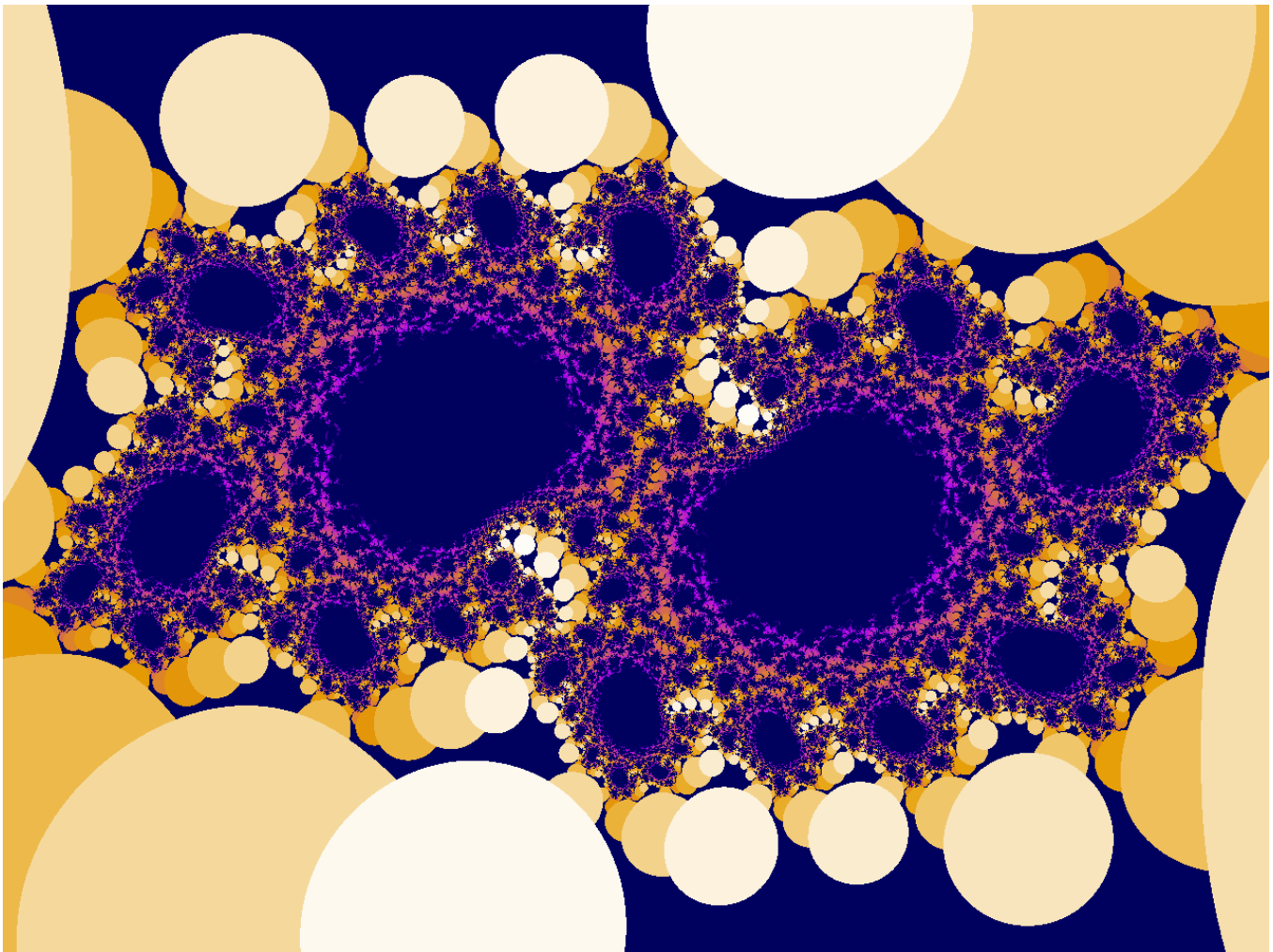


Figure 12, using a function suggested in Mandelbrot.

After practicing with Matlab and the code just given for a little while the limitations of the programming will become apparent. There is no way to zoom in or out, move the point of view around, or externally change the escape time limits. The three listings on the next two pages show one way to overcome these limitations.

The first listing, RadSpace.m separates the while function from the for loops so that new functions may be created more simply and called by using a pointer to a function called a file handle in matlab. New or old colormap functions may be passed in the same way. The RadSpace function has another aid to exploration in that it will return a randomly chosen parameter to test if it is called without a parameter list. Pretty tricky but very useful if you have a new function and no idea where the bifurcation points (ask a mathematician) might be.

The next listing, frcScrp.m is a script rather than a function, and sets the variables needed to call the for looping function GenFractal.m. Looking at the signature line of GenFractal shows that it would be a lot of typing to call from the command line without the script. Calling GenFractal with viable numbers will write a tif file (maybe a big one) to your current directory, so take care to notice what directory you are in if you want to be able to find it later. The name includes the time (to the tenth of a millisecond) the file was calculated to make the filenames more unique (to keep from overwriting files).

Error messages are the bane of a Matlab programmers existence because they rarely describe what is really wrong and often lead to messing things up through misunderstanding. **??? Undefined function or method 'GenFractal' for input arguments of type 'double'.** or some message like that in red probably doesn't mean you used the wrong input arguments or that there is anything wrong with the function itself (although there might be). Type 'path' at the command line and make sure that the directory containing 'GenFractal.m', and the rest is listed. If not use the addpath command to cause Matlab to look in your directory or click on the three elipsis button at the top of the window.

```
>>addpath('/Users/myHomeDir/WhereTheCodeIsDir/'); % like this line
```

The Error messages that stop the program once it is running are useful for finding where the program stopped but very cryptic in terms of what is really wrong. Usually the line number and function name at the top of the red list is the place to start and most likely the error source is a typo. Rather than lose your mind, have someone else help you spot your own invisible typos. If that doesn't work then logical tracing back through the call chain will find the problem and you must then struggle to understand it. Matlab is very convenient to debug with in this respect because any function or line of code may be called by itself from the command line.

For instance to see how a colormap works type `mp = jet(16)` (without a semicolon) at the command prompt and Matlab will return a 16 by three matrix of decimal numbers. For that matter most any line in the programs listed here may be entered (with valid numbers substituted for names) at the command line to get a better feel for the program. Try `linspace(-1-i,1+i,10)` for instance. Following any command with a semicolon suppress output, speeds up the program and leaves a much cleaner command window. If you call GenFractal.m without the semicolon Matlab will display all the pixels as numbers scrolling madly down through the command window.

Matlab also has a very good help section with examples included in the explanation for each built in function. Note that if you name one of your functions by the same name as built in function, and yours is in the current path the built in function will be overridden and funny things may happen. Typing 'help GenFractal' at the command prompt will demonstrate the old style quicker way to get an explanation, and clicking on the help → Matlab Help tab is the graphical way.

```

1 function count = RadSpace(Z,c,ETM,ETB)
2 % Usage: da_pixelValue = RadSpace(Z,S_lim,B_lim,a);
3 % returns the escape time (beyond B_lim) < S_lim at Z in the complex plane
4 % Inputs:
5 %   Z = the location in the complex plane
6 %   c = a (possibly complex) parameter
7 %   ETM = the escape time limit or maximum count loop limit whatever
8 %   ETB = the boundry to escape: abs(Z) < B_lim --> terminate loop
9 % Output:
10 %   count = escape time, 1 <= escape time <= S_lim
11
12 % function returns a useable parameter if called with no arguments
13 if nargin == 0
14     count = rand;
15     return
16 end
17
18 % otherwise find the escape time
19 count = 1;
20 while count < ETM && abs(Z) < ETB
21     Z = Z^(2*Z^(2*c^(2*Z^c)));
22     %Z = Z^(2*Z^(2*x^(2*Z^x)));
23     count = count + 1;
24 end

```

```

1 % frcScrp.m      A script to set the inputs to GenFractal.m
2
3 h = 200;        % height of figure in pixels
4 w = 320;        % width of figure
5
6 CP = 0+0*i;    % Center Point in the complex plane
7 ZM = 1;        % Zoom factor - must not equal zero
8
9 ETM = 32;      % Escape Time Maximum
10 ETB = 4;      % Escape Time Boundry
11
12 theta = 0;    % figure rotation angle
13
14 fh = @RadSpace; % or reset to new escape time function
15 cfh = @jet;    % or reset to new colormap function
16
17 whos;
18
19 % how to generate a random parameter c
20 % c = feval(fh), [A S] = GenFractal(h,w,CP,ZM,ETM,ETB,theta,fh,c,cfh);
21 % or to keep the current c, make twice as big, and rotate 90deg ccw
22 % [A S] = GenFractal(2*h,2*w,CP,ZM,ETM,ETB,pi/2,fh,c,cfh);
23
24 % use the following two commands to retrieve the parameter list from a .ti
25 % fnAAinfo = imfinfo('fnAA.tif'); disp(fnAAinfo.ImageDescription);

```

```

1 function [A S] = GenFractal(h,w,CP,ZM,ETM,ETB,theta,fh,c,cfh)
2 % Usage:
3 %[Image,descriptionString] = GenFractal(h,w,CP,ZM,ETM,ETB,theta,fh,c,cfh);
4 % Inputs:
5 %   h = height of figure in pixels
6 %   w = width of figure in pixels
7 %   ZM = zoom factor
8 %   ETM = Escape Time Maximum
9 %   ETB = Escape Time Boundry
10 %   theta = rotation angle of complex plane inside the figure
11 %   fh = file handle of the function to iterate
12 %   c = the array of constants to pass to the function
13 %   cfh = colormap file handle - the function to generate colormap matrix
14 % Outputs:
15 %       A = Image matrix, is a unsigned 8 bit, h x w x 3 matrix (uint8)
16 %       S = parameter string - a string to store in the tif file header
17 c1 = clock; % record system clock (seconds)
18
19 % define complex X & Y axis end points:
20 RIGHTB = CP - exp(i*(-theta))/ZM; % e^itheta =
21 LEFTB = CP - exp(i*(pi-theta))/ZM; % cos(theta) + i*sin(theta)
22 TOPB = CP - exp(i*(pi/2-theta))*h/(ZM*w); % define four points
23 BOTTOB = CP - exp(i*(-theta-pi/2))*h/(ZM*w); % around center
24
25 % linear arrays to define real and imaginary number line axis
26 Xaxis = linspace(LEFTB,RIGHTB,w); % image X axis to w many complex #s
27 Yaxis = linspace(TOPB,BOTTOB,h); % image Y axis to h many complex #s
28
29 % define colormap and allocate image array
30 mp = colormap(feval(cfh,ETM)); mp = cast(mp*255,'uint8');
31 A = uint8(zeros(h,w,3)); % unsigned 8-bit integer == image
32
33 % for each row, for each column in that row...
34 for m = 1:h % there are h = height-of-image rows
35     for n = 1:w % and w columns in the image
36         Z = Xaxis(n) + Yaxis(m); % Z is the intersection of m,n
37         esctime = feval(fh,Z,c,ETM,ETB);
38         A(m,n,:) = mp(esctime,:); % set three pixel bits to color#
39     end
40 end
41 % finish chores: set description string, display image etc.
42 S = ['Center =',num2str(CP),char([10 13]),...
43     'ZM =',num2str(ZM),char([10 13]),'c = ',num2str(c)];
44 imshow(A); colorbar; impixelinfo; % show image with colorbar
45 c2 = clock; nSeconds = etime(c2,c1); % real clock running time
46 fprintf('elapsed time = %f\n',nSeconds); % not cpu cycles
47 S = [char([10 13]),'Image Runtime=',...
48     num2str(nSeconds,'%0.2f'),char([10 13]),S,char([10 13])];
49 CURTIM = clock;
50 fname = ['mefile',num2str(CURTIM(4),'%2.0f'),'.',...
51     num2str(CURTIM(5),'%2.0f'),'.' ,num2str(CURTIM(6),'%2.4f'),' .tif'];
52 imwrite(A,fname,'tif','Compression','none','Description',S);

```

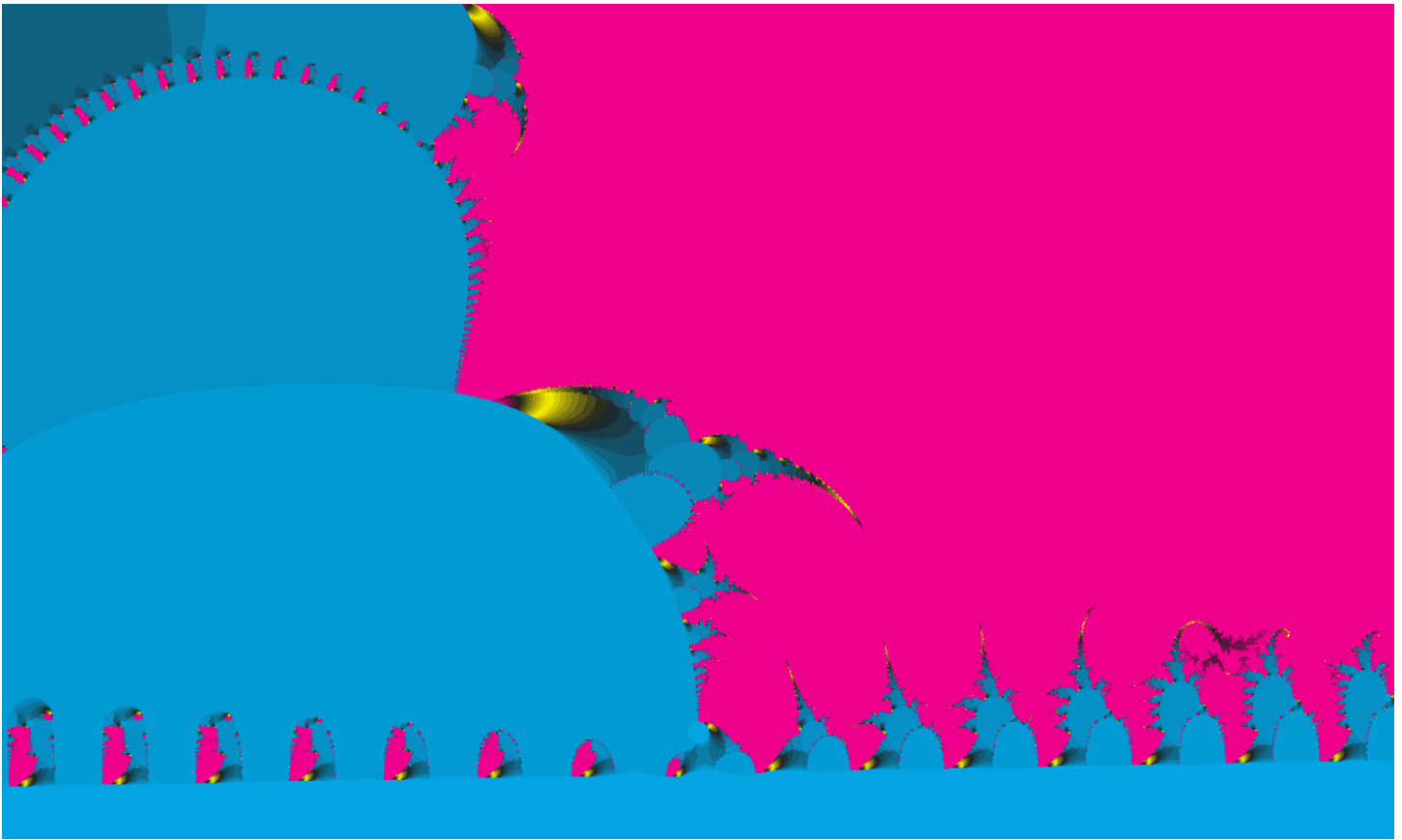


Figure 13. I don't know what they are but I wouldn't want to scare them off of whatever it is they are perched on. In the complex plane Upper Left = $-0.13085-5.833i$, U.R. = $-5.833-0.13085i$, L.L = $3.433-2.2691i$, L.R. = $-2.2691+3.433i$. Cyan = escape in 1, Magenta = prisoner set after 64 iterates.

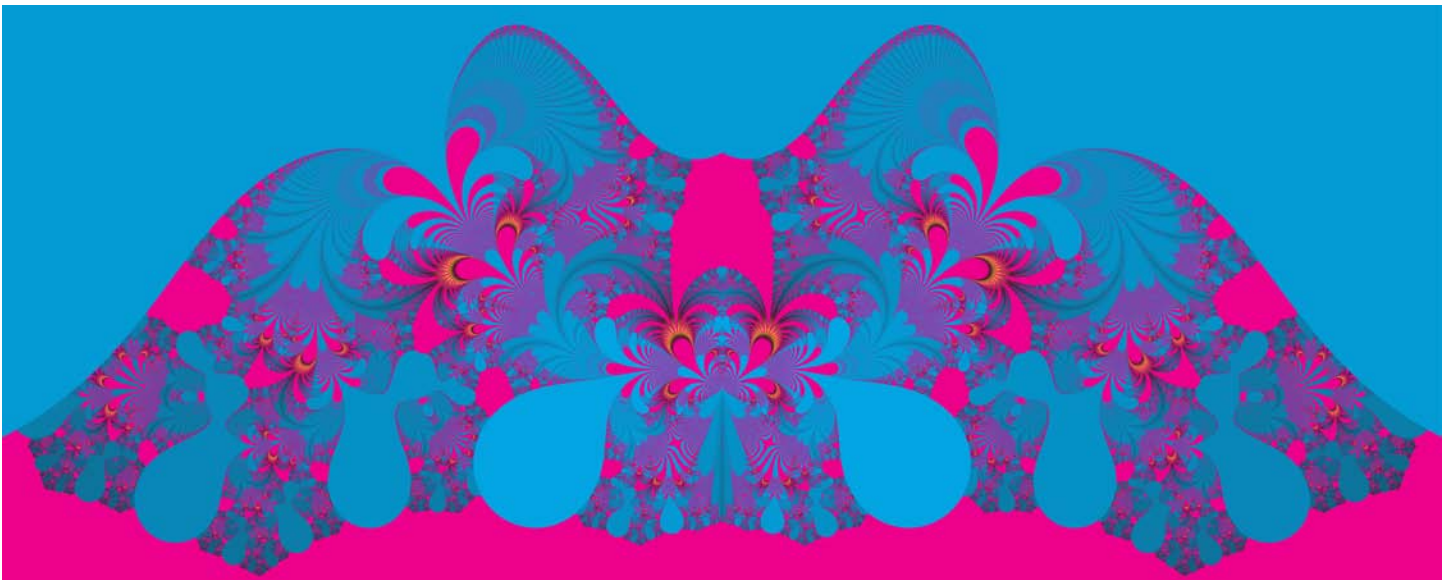


Figure 14. Working title is *Angles de los Conejitos (Angels of the Bunnies)*, and if you look carefully there are bunnies. In the complex plane Upper Left = $2.9233+5.4783i$, U.R. = $2.9233-5.4783i$, L.L = $-1.4593+5.4783i$, L.R. = $-1.4593-5.4783i$. Using the same colormap as figure 13. This set seems to extend indefinitely, that is to say that step-zooming out to $\pm 5,000$ real and complex, reveals more and more larger, similar sets with mind-boggling detail. This is all more or less on the negative imaginary side of the plane which is shown upside down here.

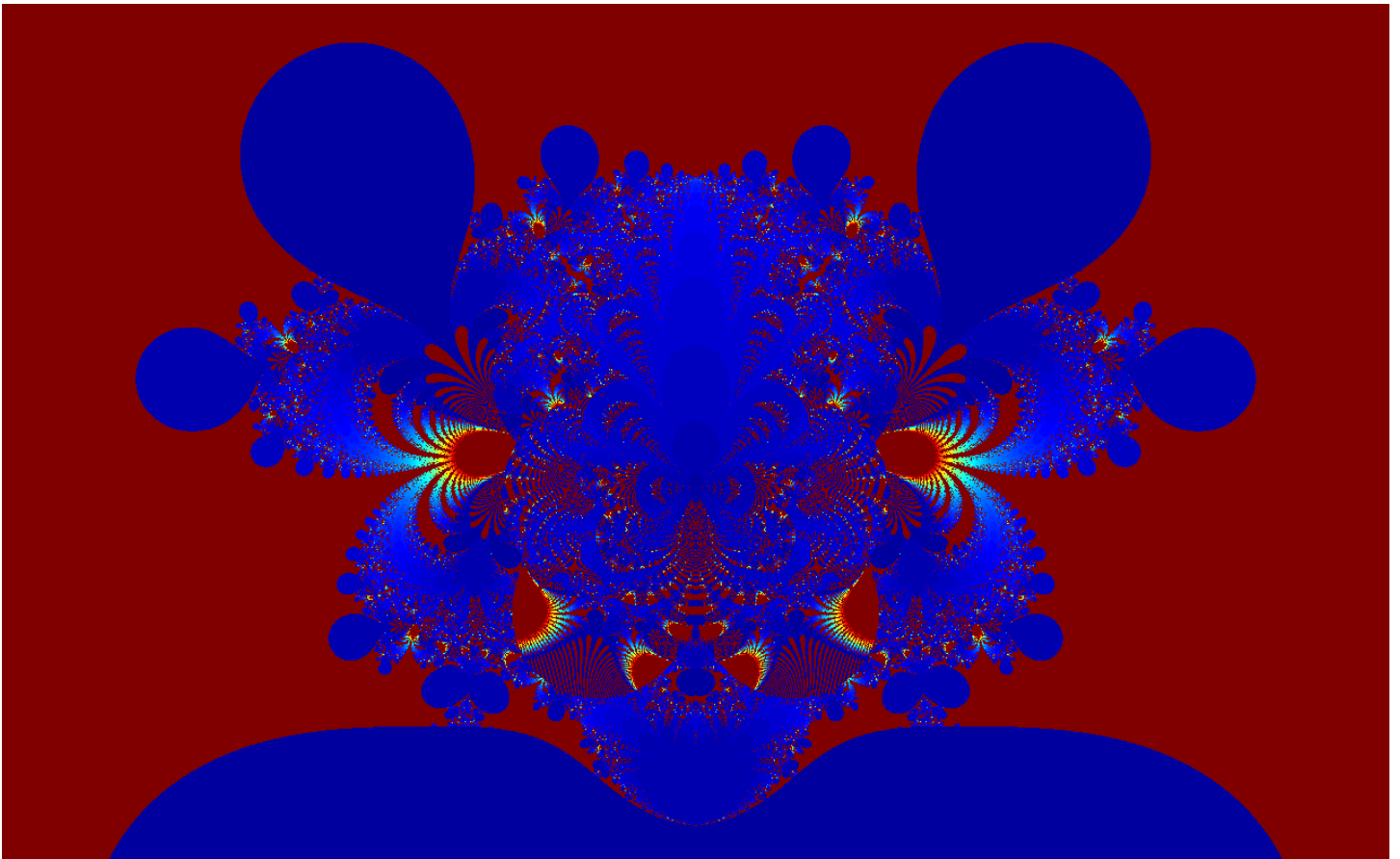


Figure 15. Using ordered sets of parameters in exponential functions seems to produce symmetrical figures, not all of which look friendly. This is more or less on the complex unit circle.

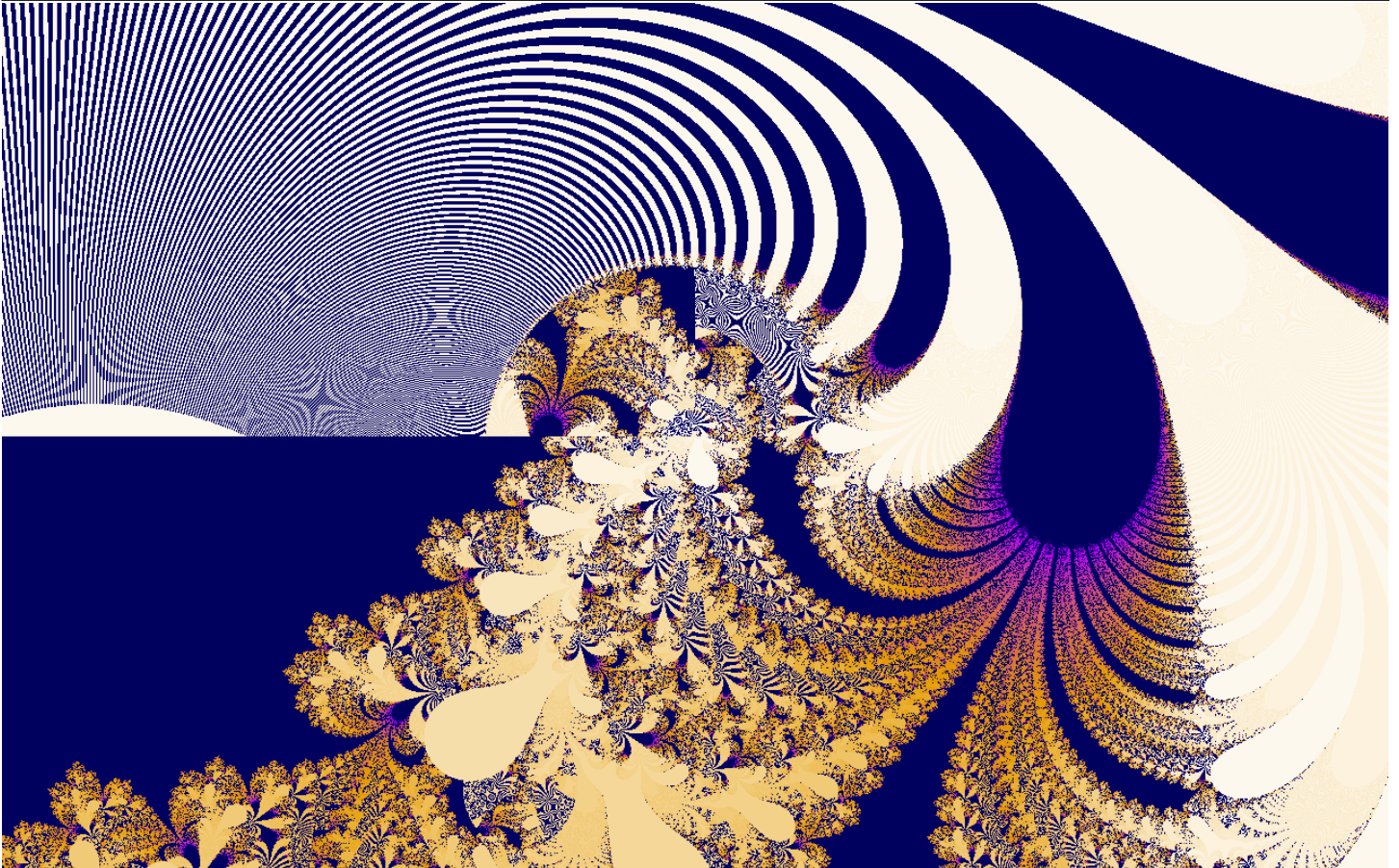


Figure 16. I don't know how to explain this. The complex unit circle is in the middle.

Useful References

Barnsley, M., (1993), *Fractals Everywhere 2nd Ed.* San Francisco, Morgan Kaufmann

Brown, J. W., Churchill, R. V., (2004), *Complex Variables and Applications 7th ed.*, N. Y., McGraw-Hill.

Mandelbrot, B., (1977, 1983, 21st printing 2006), *The Fractal Geometry of Nature*, New York, W. H. Freeman and Company

McAndrew, A., (2004), *Introduction to Digital Image Processing With MATLAB*, Boston, Thomson Course Technology.

Peitgen, H., Jurgens, H., Saupe, D., (2004), *Chaos and Fractals, New Frontiers of Science 2ND Ed.*, N. Y., Springer