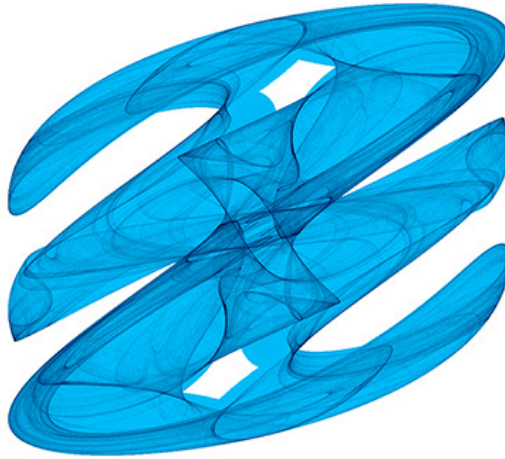


QS Ganymede



About the Program

This program uses the four attractor formulae described in the book *Chaos in Wonderland* by Clifford Pickover (St. Martin's Press, 1994). These formulae generate striking images of chaotic attractors. The book's premise is that alien beings called “Latööcarfians” inhabit Jupiter's moon Ganymede. They visualize the attractors in their dreams. The book describes Latööcarfian civilization in detail, and presents the account of a human visitor and his companion. Within this science fiction are descriptions of chaotic attractors and related mathematical concepts, sample programming code, games and references to mathematical history.

The original program was written in 1994 for DOS. This revision is a 32-bit Windows program which should continue to work in current 64-bit Windows versions. I've added a third-dimensional factor which allows images to be rotated around each of the three axes.

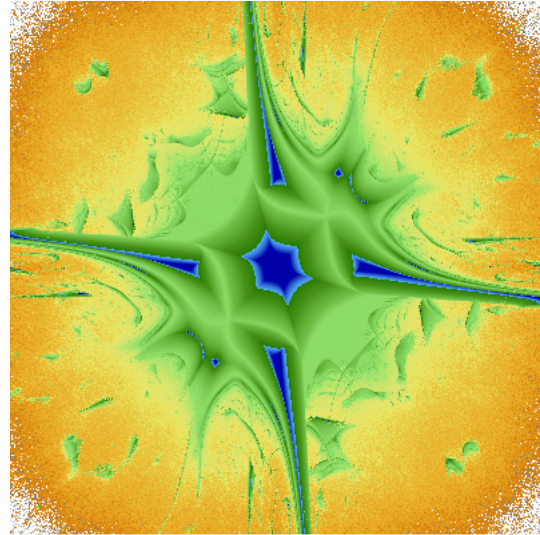
Running the Program

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. Images 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 “GAN” 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.

Parameter sets can be entered in the dialog box, or random sets of parameters can be generated for the currently-selected formula. 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.

A Lyapunov exponent map can be rendered for the selected formula, showing the values for the ranges of the “A” and “B” parameters between -3.0 and +3.0. The “C” and “D” parameters, when applicable to the selected formula, are held constant. Exponents of -2.0 or less are assigned the lowest color index of the palette, and exponents of 2.0 or greater are assigned the highest index. The map can be helpful in showing values of “A” and “B” which might yield interesting images. In this example, the blue “water level” represents values of -2.0 or less, and the white “snowy peaks” represent values of 2.0 or more.



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 “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.

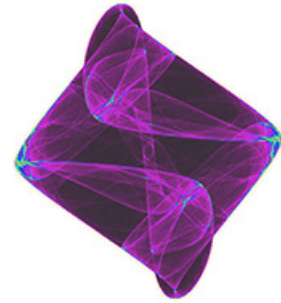
The “Z-plane” algorithm colors pixels according to the position along the Z axis of their associated iterated points. This gives the images a more “solid” character.

Attractor Formulae

A "standard" formula and three "mutations" are iterated to yield successive points (x, y). Each formula utilizes two parameters "A" and "B", in the range of -3.0 to +3.0. The standard formula also utilizes "C" and "D", in the range of -1.75 to +1.75, and the alpha formula utilizes "C". To add a third dimension, each formula also calculates $z_{(n+1)} = \sin(x_n)$.

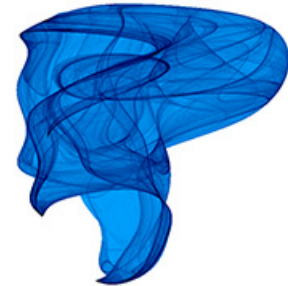
Standard

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



Alpha

$$\begin{aligned}x_{n+1} &= \sin(y_n * B) + \sin^2(x_n * B) + \sin^3(x_n * B) \\y_{n+1} &= \sin(x_n * A) + \sin^2(y_n * A) + \sin^3(y_n * C)\end{aligned}$$



Beta

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



Gamma

$$\begin{aligned}x_{n+1} &= |\sin(y_n * B)| + \sin^2(x_n * B) \\y_{n+1} &= |\sin(x_n * A)| + \sin^2(y_n * A)\end{aligned}$$



Some sample parameter sets, including those from the book, are given in the table on the next page.

Latöocarlian 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		



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*.

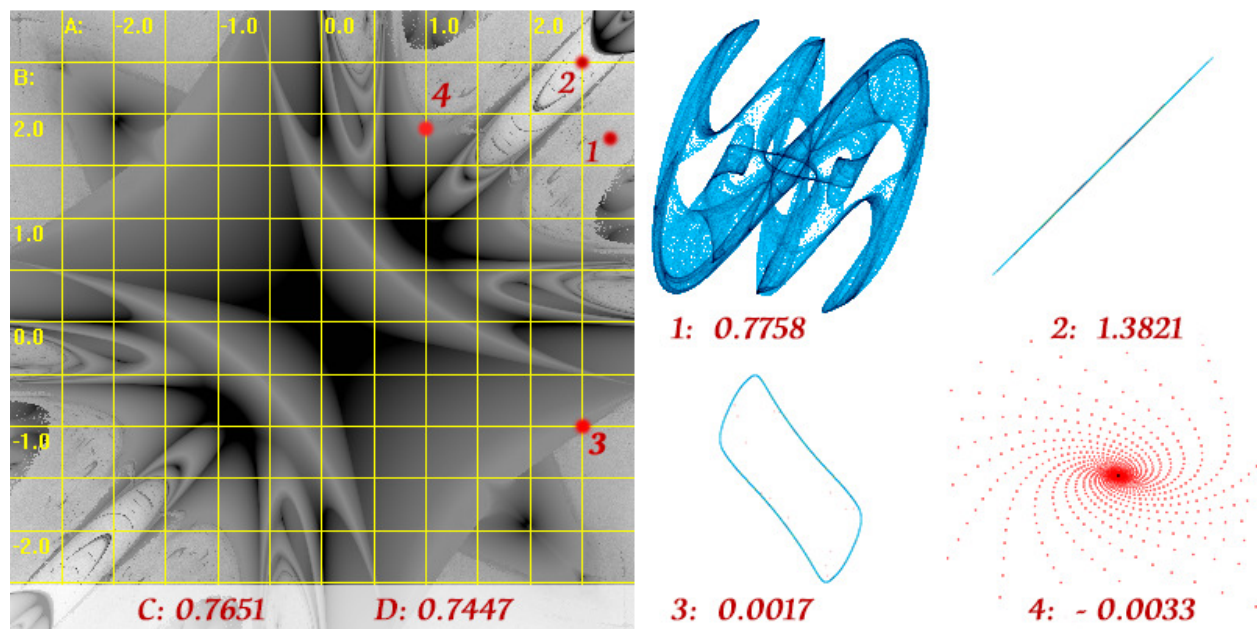
The Latööcarfian formulae cannot exhibit option 3) because the sine functions limit the size of each iterated point. The tendency toward the other options can be characterized by the *Lyapunov exponent*. Calculation of this value is described by Julien Clinton Sprott in various publications, including

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

and by Heinz-Otto Peitgen *et al.* in their text *Chaos and Fractals* (Springer-Verlag, 1992), 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.

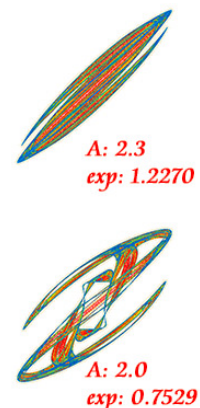
Ultimately, if its Lyapunov exponent is positive, a system will exhibit chaotic behavior, as the distances between the iterations of nearby points will, on the average, diverge. Thus, images of such systems should produce aesthetic, or at least interesting, patterns. In contrast, if the exponent is 0.0 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).

Lyapunov maps provide a graphical way to assess the exponents of a system. The program can plot such maps in the A-B parameter plane, over the range of -3.0 to +3.0, as described above. In these examples for the standard formula, a greyscale palette is used, so that black represents exponents of -2.0 and below, and white represents exponents of +2.0 and above. The selected values of C and D (shown below the grid) are used for the iterations, since the standard formula uses both of these parameters.



Example 1, with a Lyapunov exponent of 0.7758, shows a characteristic chaotic pattern. Example 3, with an exponent near 0, shows a periodic cycle. Example 4, with a negative exponent, shows points spiraling in to a fixed point.

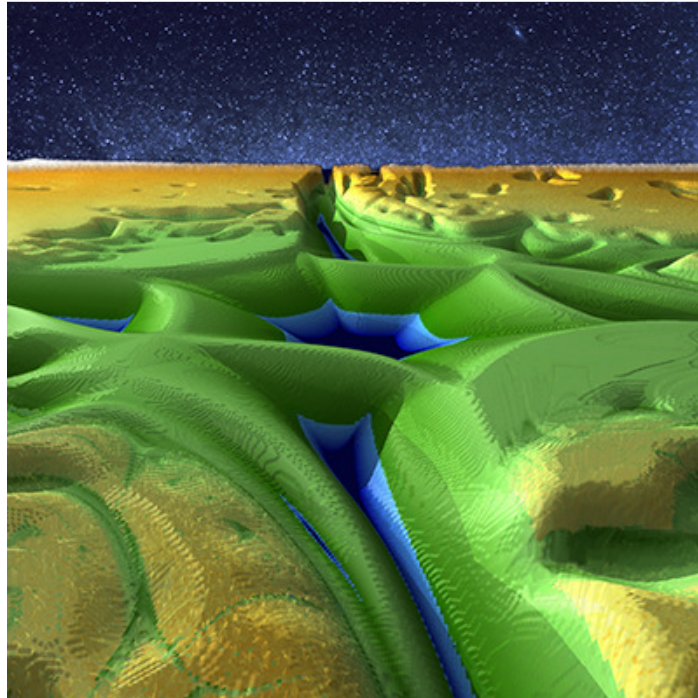
At first glance, example 2, with $A = 2.5$, does not appear very chaotic, even with an exponent of 1.3821. But successive iterated points move back and forth at increasing relative distances along the line. Gradually decreasing the value of A lowers the exponent but “spreads” the image out. Thus, the magnitude of the exponent does not correspond directly with the appeal of the image. Pickover states in his book, “I find that chaotic attractors with an exponent of around 0.5 to be the most pleasing.” But note that as A crosses the dark band around 2.15, the exponent becomes negative, with images consisting of one or a few dots.



Pickover describes several other evaluation tests:

if $|C| * |B| + |A| < 1$ or if $|D| * |A| + |B| < 1$ the system will converge.

if $A = 0$ and $|B| < 1 / |C|$ or if $B = 0$ and $|A| < 1 / |D|$ the system also will converge.



Appendix 2: Image Enhancement

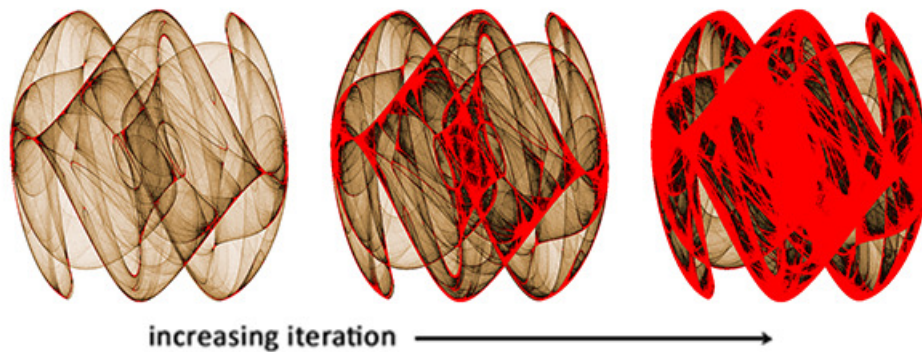
For images of chaotic attractors, an effective coloring method assigns an entry from a color palette according to the number of times the pixel has been “hit” by an iterating point. It should be understood that as iteration continues, an infinite number of different points can land within the boundaries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



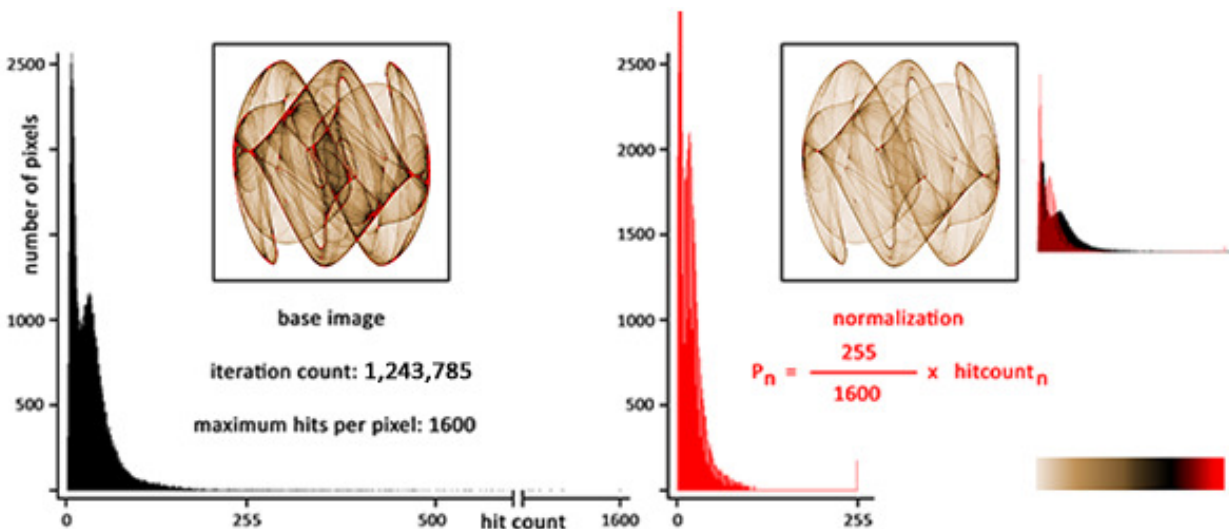
These files were developed for the pioneering fractal rendering software *FractInt* as an easy way of creating, using and exchanging palettes. The first color of the palette is pure black. This is commonly done so that `palette[0]` can be used for the background by a program. Therefore, *QS Ganymede*, 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 standard 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.

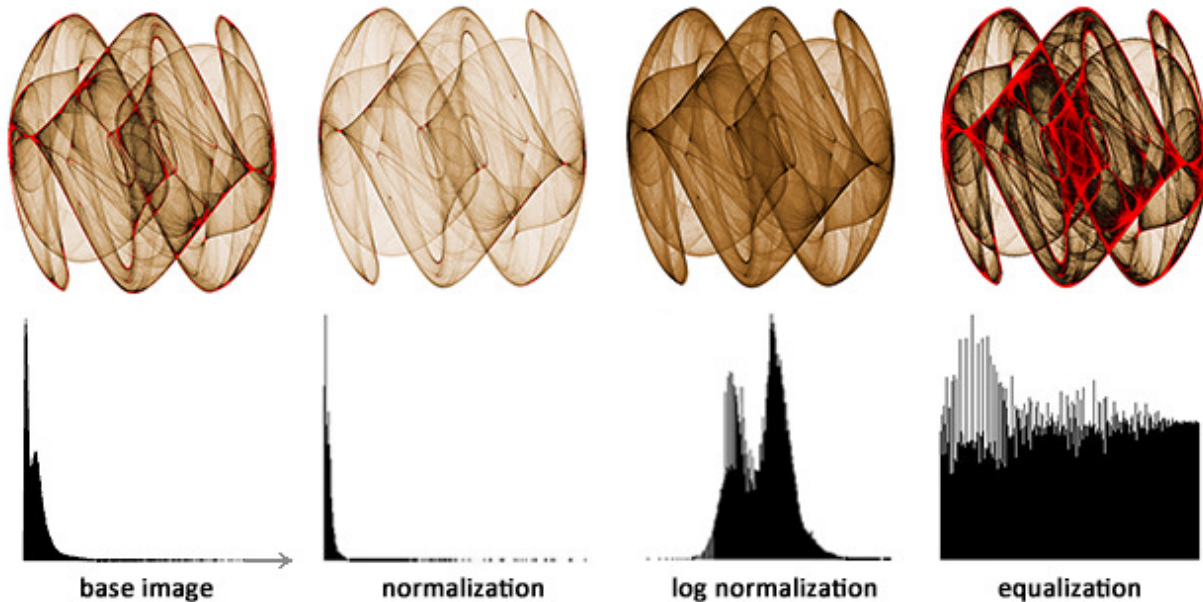


One method of compensating for this situation is *normalization* of the hit counts, in which the count for each pixel is multiplied by a factor consisting of 255 divided by the maximum hit count of all the pixels. Thus, all hit counts will be normalized to a range of 1 to 255. But this method also has 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 table on the next page shows representative data from various hit counts for another image in which rendering was stopped at a maximum hit count of 1600.

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

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.

