JuliaSaver

a fractal screen saver

by Damien M. Jones

Technical Information

For those of you who actually *know* something about fractals, you may be curious about the fractal generation parameters used in this program. This file contains those details completely unnecessary to your enjoyment of the program, unless—like me—you are a detail freak.

The Math

All of the math is done using 80-bit floating-point numbers. "Why," you might ask, "is floating-point math used when integer math is so much faster?" That's a very good question. If you happen to have a copy of FractInt around, try this. Generate a fairly simple Julia fractal image at 1024x768, using integer math. Make a note of the time. Now generate it again, using floating-point math. Make a note of the time. Chances are, if you're using a Pentium-class processor, you won't see too much difference. So I'll let you in on a little secret: on a Pentium processor, 80-bit floating point multiplies are *up to ten times faster* than 32-bit integer multiplies. There's a bit more overhead in working with the FPU for the math, but it evens out in the end. And if it's *at least* as fast, why wouldn't you want the more accurate math? *JuliaSaver* works out to be three to six times faster than FractInt, anyway.

The Guessing Algorithm

JuliaSaver uses a recursive algorithm to generate the image, skipping large areas of the image that are solid colors. If you've used FractInt, you're familiar with the algorithm: it's called **solid guessing**. (No, I didn't look at the FractInt code; I just wrote mine from scratch.) Also, since Julia fractals are symmetrical about the center point, only half the screen is actually calculated; the other half is just copied from that. The amount of acceleration this provides varies, but it helps *enormously* on most of the pictures. For a 1024x768 screen, five guessing passes are made to generate the screen. For a 640x480 screen, only four passes are used. The number scales based on the actual resolution of your screen. If you use the **Resolution** slider to lower the resolution of the generated image, all you're actually doing is eliminating the last passes of fractal generation. Note that this algorithm isn't quite fool-proof—sometimes it allows things to be "dropped out", particularly thin strands poking into the interior of a Julia set. Oh well, nothing's perfect.

The Iteration Cap

The maximum number of iterations performed is 128. This is a somewhat arbitrary limit; I could have used anything. 128 provides a good balance between detail and unreasonable slow-downs. My initial tests were done with only 16 iterations, which does well even without the guessing. (Although on many images, 128 iterations *with* guessing is as fast as 16 iterations *without*.)

Perturbed Mandelbrot

This version of *JuliaSaver* adds the ability to generate perturbed Mandelbrot sets. If you're a fractal fanatic, you know that the Mandelbrot set is simply the amalgamation of all Julia sets, evaluated at their critical points and plotted against their *c* value. Oddly enough, these critical points all work out to be zero. A *perturbed* Mandelbrot set is one in which we don't use the critical value (zero); we use something else. Depending on how close the "something else" is to zero, either a slight distortion (nearly zero) or massive distortion (nowhere near zero) results.

Inverted and Radial Mappings

These new modes change how the complex plane maps to the screen. Ordinarily, the complex plane maps to your screen in a simple, rectangular fashion: distance along the real axis equates to horizontal distance on the screen, and distance along the imaginary axis equates to vertical distance on the screen.

Inverted mapping flips the screen inside out. Imagine a circle drawn on the screen, not quite touching the top or bottom. Everything inside the circle is flipped to the outside, with the center of the circle being flipped all the way to infinity. Everything outside the circle is flipped to the inside, with infinity being flipped all the way to the center. Everything actually *on* the circle will stay right where it's at. Confusing, huh? But it looks neat.

Radial mapping "unwinds" the complex plane around the center. Horizontal distance on the screen corresponds to increasing angles; vertical distance on the screen corresponds to increasing distance from the center. Vertical screen distance and distance from the center to not map linearly; this didn't look very good, and the current funky distortion looks better.

Animations

If you're a fractal freak, you probably *already* know that each Julia set is characterized by a single parameter, *c*. This represents a constant value used in the iterative equation, $\mathbf{z=z^2+c}$. Change *c*, and you change the entire Julia set. All the animation does is slowly change the value of *c*, so that each successive set is a little bit different than the one that was shown before. For perturbed Mandelbrot images, the *c* value is the starting value for each point (normally zero).

The **Sinusoidal Bobbing** method is pretty straightforward; it just moves the real and imaginary parts of *c* (a complex number) using two sine waves each. The **Cardioid Trace** method is a bit more sophisticated: it moves *c* around the heart-shaped area of the Mandelbrot set. The Mandelbrot set is sort of a "road map" to Julia sets, and some of the most interesting Julia sets can be found with *c* points near the heart shape (technically referred to as a *cardioid*). Just so you won't get bored, the current position of the animation is *saved* whenever the screen saver stops, so it always starts on something new. **Random Waves** selects a new real and imaginary component for *c*, and moves towards it using a sine wave. To keep things interesting, it changes the real and imaginary components at different times.