JuliaSaver

a fractal screen saver

by Damien M. Jones

Contents

I. System requirements
II. Installation
III. Configuration
IV. Technodweeb Details
V. Registering

I. System Requirements

- Pentium-class processor *(not required, but **strenuously** recommended)*
- Windows 95 or Windows NT *(these instructions assume Windows 95)*
- 640x480x256 or better display mode

II. Installation

Run `SETUP.EXE`. This will install the screen saver file into your `WINDOWS\SYSTEM` directory.

Once you've done that, right-click on your desktop, select the **Screen Saver** tab, and use the drop-down box to select **JuliaSaver**. If everything works well, you should see Julia fractals appearing in the little preview window. If you don't, something very likely went wrong. Try installing it again.

III. Configuration

Now that you've got this thing installed, you probably want to play with it a bit. Go ahead and click the **Preview** button and see the fractals full-screen. Moving the mouse or pressing a key will shut off the screen saver. Don't panic if the image doesn't disappear right away; sometimes it can take a couple of seconds. This is *normal*. After all, calculating fractal images makes your CPU break out in a sweat.

Okay, you want to tweak the settings on this thing, maybe make it do something a little bit different. I can understand that. Click the **Settings...** button and you'll get a nifty *configuration* dialog with a bunch of options in it.

The **Colors** drop-down lets you choose from eight different color schemes. I suggest checking them all out; my personal favorite is the grey-cyan-purple scheme (the default).

The **Animation** drop down lets you choose the animation method. Animation works by varying the *c* parameter in the basic Julia equation—see the **Technodweeb** section for more information. Your options here are **Sinusoidal Bobbing**, which moves the *c* point in a complex lissajous pattern over the Mandelbrot set; **Cardioid Trace**, which moves the *c* point around the heart-shaped core of the Mandelbrot set; **Random Waves**, which moves the c point around to random places, in sinusoidal-like patterns; and **Totally Random**, which selects a different point at random for each frame.

The **Speed** slider controls how quickly the Julia shapes change. This *only* affects how quickly the *c* point varies. Some images take longer than others to generate, but the *c* point (for **Sinusoidal Bobbing**, **Cardioid Trace**, and **Random Waves**) moves at the same speed, no matter how fast the screen is updated. This slider lets you control that speed. Move the slider to the left to slow down the animation; move it to the right to speed it up. (Technodweebs: each notch on the slider represents a factor of 2.)

The last two sliders offer ways to help the screen update faster. The **Iterations** slider sets the maximum number of times the Julia equation can be used for each point. Less iterations (move the slider to the left) mean faster results, but you lose some detail. More iterations (move the slider to the right) give more detail, but take longer. The **Resolution** slider sets the final resolution of the image. Lower resolution pictures (move the slider to the left) take *considerably* less time to generate. Higher resolution pictures (move the slider to the right) look nicer.

The next option is **Animation Trails**. If this option is selected, rather than simply refresh the entire screen with a new fractal, it is *blended* with previous frames so you get a "trails" effect. This looks exceptionally nifty. The one drawback is that you can't use **Animation Trails** with the highest **Resolution** setting—that would require some *serious* computation that would slow things down enormously. The good news, though, is that when **Animation Trails** is on, the fractal is effectively generated at a higher resolution anyway, and you get the added benefit of *speed* from the lower resolution setting. Keep a napkin handy to mop up the drool.

To the right of **Animation Trails** is **Shrink Palette**. The color schemes ordinarily stretch to 128 colors; if you move the **Iterations** slider to a setting lower than maximum, though, not all the colors will be used. If **Shrink Palette** is checked, as it normally is, the colors will br shrunk to fit into the actual number of colors used.

The bottom option is **Allow screen captures**. If this box is checked, you can press `ALT + PRINT SCREEN` to grab a snapshot of the fractal image and put it on the clipboard. You can then paste it into any graphics program, such as Windows Paint. However, when this option is on, that means pressing the `ALT` key won't exit the screen saver. That's why you can turn this "feature" off, if you like.

IV. Technodweeb Details

For those of you who actually *know* something about fractals, you may be curious about the fractal generation parameters used in this program. This section contains those details unnecessary to your enjoyment of the program.

First, 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?

Second, the program 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, four guessing passes are made to generate the screen. For a 640x480 screen, only three 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.

Third, 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 most images, 128 iterations *with* guessing is as fast as 16 iterations *without*.)

Now let's talk about those 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. 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.

V. Registering

Well, okay. The program isn't much. So I'm not *asking* much. If you like **JuliaSaver**, just send me $10. Just make that check out to me (Damien M. Jones), put it in an envelope, and drop it in the mail. Speaking of the mail, here's the address you should put on the envelope:

Damien M. Jones
3207 SW Bessey Creek Trail
Palm City, FL 34990-1801

If you have a question, complaint, or suggestion regarding **JuliaSaver**, you can send a letter, or send e-mail to **dmj@emi.net** (preferred). Updates to the program can be found on our web pages, at **http://www.emi.net/~dmj**

Enjoy!