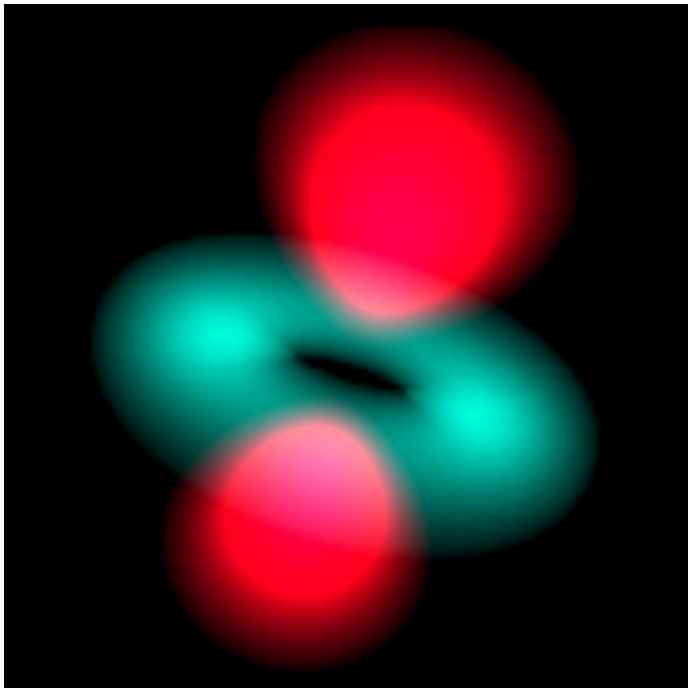
READ ME for

Atom in a Box v1.0

by Dean Dauger



What is Atom in a Box?

Atom in a Box is an application that aids in visualizing the Hydrogenic atomic orbitals, a prime and otherwise unwieldy example of quantum mechanics. Unlike other tools in this category, this program raytraces through a three-dimensional cloud density that represents the wavefunction's probability density and presents its results in real-time (up to 48 frames per second on the latest hardware). The user interface is very interactive and provides a wide degree of flexibility.

It contains all 140 eigenstates up to the n=7 energy level and the allowed spectral transitions between those eigenstates. It also allows a state formed by a superposition of up to eight of those eigenstates allowing for over 3 trillion possible states. The program can display a wavefunction as a picture of a cloud, use color as phase, plot in red-cyan left/right for 3D glasses, and slice the wavefunction.

How do I use Atom in a Box?

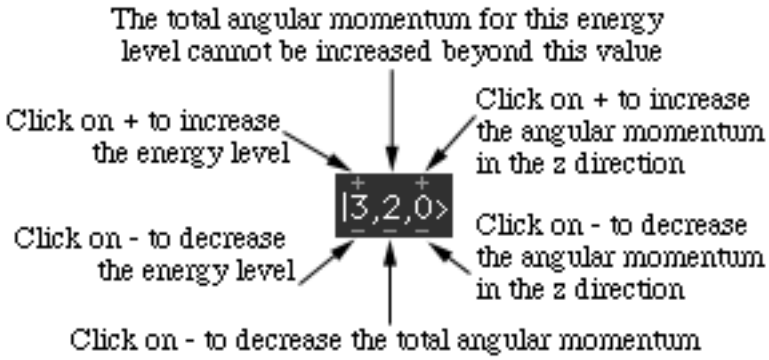
The bottom line is: **PLAY WITH IT!** Go ahead and try anything your heart desires. Most Macintosh users skip manuals anyway, and the program displays help text and help balloons for almost all items in the window. This Read Me is meant for those who want more details about the program but is not required reading.

What are Orbitals?

"Orbital" is the name given to the state of an electron bound to an atomic nucleus, analogous to the orbit a planet occupies around a star. The primary differences are that the orbital is described by Quantum Mechanics and the interaction is electromagnetism. Special states, called eigenstates, have been found that, when properly summed together, can describe any allowed state of the electron bound to the nucleus. By definition, they cannot

be written in terms of one another (i.e., they are said to be "orthogonal"). These eigenstates have their own unique labels. The first refers to the energy of the eigenstate, labeled by n. n can be any integer starting at one, the lowest energy level, then two, three, and so on for higher energies. Within an energy level, there are certain allowed distinct states of angular momentum which are identified by two integers, l and m. l, which identifies how much angular momentum the eigenstate has, ranges from zero to n-1. m, which identifies how much of that angular momentum is in the +z-direction, ranges from negative l to positive l. These three integers together uniquely identify every possible eigenstate, and are often written together in a Dirac ket: |n,l,m>.

This code contains parameters for all the eigenstates up to n=7, which is the energy level for the highest energy electrons in the Uranium atom. That makes a total of 140 orthogonal eigenstates. For each eigenstate, the code computes its wavefunction, a complex function of position. The magnitude squared of this function is plotted as brightness. You can step between these eigenstates by clicking on the + or - above and below the integers at the top of the window.



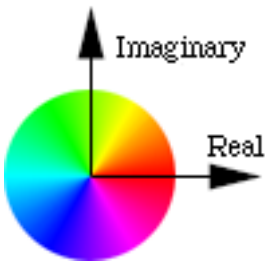
Display

The primary display is the large rectangle on the left side of the window with the orbital in it. You can click and drag there to rotate the orbital. Think as if you're poking your finger at it. If you're still moving it while you release, it will keep rotating. Pressing the space bar stops the motion.

View type menu

The Standard 3-D display mode shows the three-dimensional probability cloud as a two-dimensional image, like photographing a cloud in the sky. With the 3D Red/Cyan modes, the cloud as it would appear for the left and right eyes are shown in red and cyan, so you can use a pair of red/cyan 3D glasses to view the cloud as if it was floating in front of the screen.

How color is mapped onto the complex plane:



With Phase as Color, the phase of the wavefunction is translated into a hue of the cloud. The wavefunction is a complex valued function of position. Zero phase is translated into red, and increasing phase cycles through the color wheel, to yellow, to green, to cyan, to blue, to magenta, and back to red. Opposite phase corresponds to color complements, e.g.,

red and cyan represent opposite phases, yellow and blue are opposite, as are green and magenta.

Also with Phase as Color, you can see the time evolution of the wavefunction. Each eigenstate evolves in time at a rate related to its energy. For example, the states with m not equal to zero exhibit an electron flow around the z axis, like a small superconducting vortex. Also, if you try a superposition (see below) of states of different energies, the probability distributions are likely to change over time as the states go in and out of phase with each other because they evolve at different rates.

Samples menu

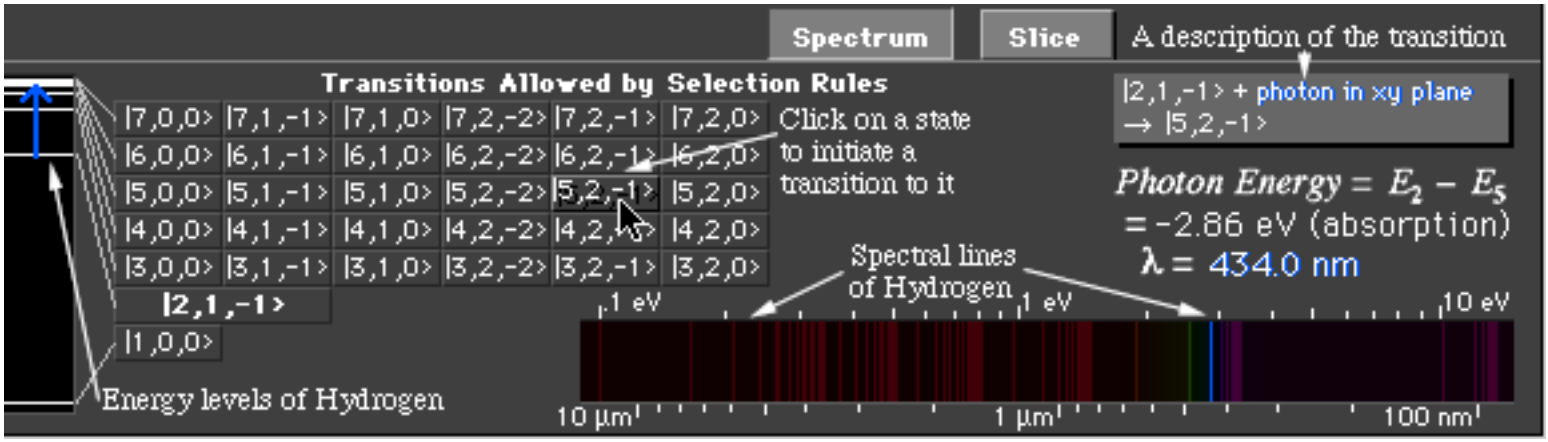
The code displays the wavefunctions by raytracing through the clouds and using a Simpson's rule integration technique to sum the contributions. As for all numerical integration, more samples usually mean higher accuracy, but more computation time spent, so the frame rate decreases. The Samples menu allows you to adjust this tradeoff.

Gamma table menu

The Gamma table menu allows you to adjust the probability density to brightness function. This adjustment makes it easier to see the structure of some wavefunctions. The cutoff gammas cut off the brightness if the density is below a certain value but do no adjustment above that value, and the stepped gammas translate the linear range of density into a stair-stepped set of brightness outputs, giving a more isosurfacish look. (I adopted the "gamma table" term from computer video hardware; video hardware gamma tables are used translate the linear R, G, and B, outputs into a nonlinear voltage output to the monitor to compensate for the nonlinear response of screen phosphor.)

Spectrum

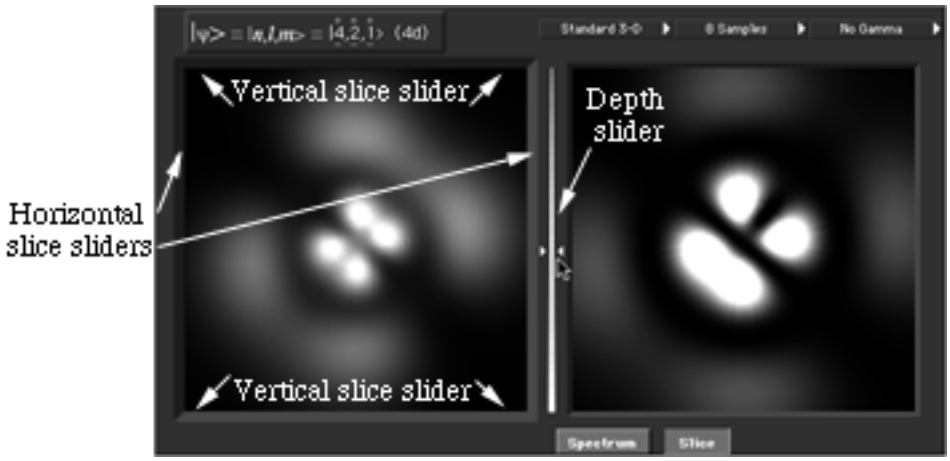
By clicking on the Spectrum tab, you can examine the spectrum of the transitions between these eigenstates. Energy and angular momentum is absorbed or released by absorption or emission of photons, energy packets of light. The Wigner-Eckart theorem describes restrictions on the relative angular momenta of the initial state, final state, and photon.



These restrictions are also known as the "selection rules", whose consequences can be seen in atomic spectra. This code lists all the possible states, up to n=7, that can be reached from the currently displayed state by absorption or emission of a photon. The photon's energy, wavelength, and possible directions of travel are described to the right of the list.

Slice

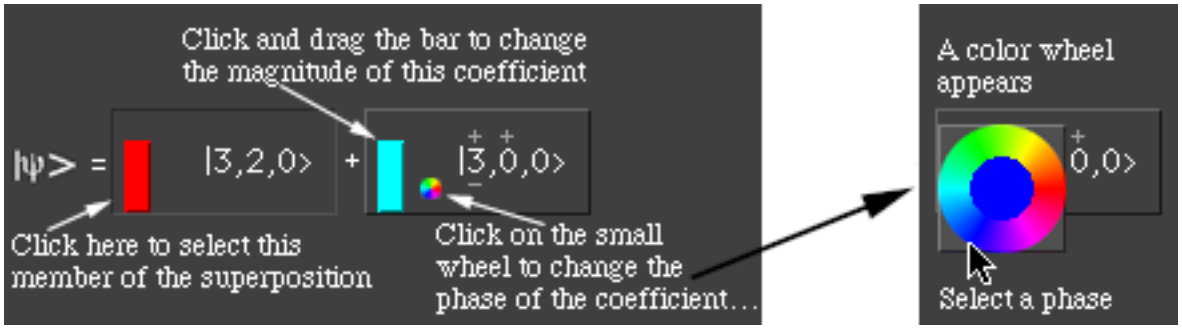
By clicking on the Slice tab, you may examine a slice through the wavefunction. The code can slice vertically, horizontally, or at a depth in the wavefunction. Select vertical or horizontal by clicking on the border around the wavefunction, and at a depth using the long triangular slider.



When the slice is vertical, depth extends to the right in the slice plane. When the slice is horizontal, depth extends upwards. When using the depth slider, whose metaphor is a road extending into the distance, a deeper slice is higher on the slider. Hold down shift to constrain the slider to one eighth increments.

Superposition

The electron does not have to be in only one of these eigenstates, but can be in a state composed of many different eigenstates. A complex number multiplies each distinct eigenstate in the superposition. The sum of the magnitudes squared of these coefficients must be equal to one for the state to be normalized.

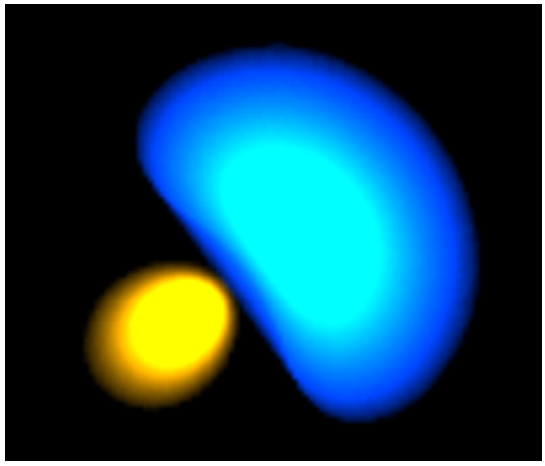


Click on the Superposition button to turn the state into a superposition. Use the + and - buttons to change how many terms there are in the sum. Click in the rectangle around a member of the superposition to select that member. Once selected, you can edit the magnitude of the coefficient by clicking on the bar and adjusting it. You can edit its phase by clicking on the small color wheel. That brings up a larger one that displays the phases, using the same color convention as above, you can select from. The code displays detailed information about the selected coefficient in an information pane at the bottom of the window. You can click in this pane to edit the coefficient more precisely, and you can lock the coefficient so that it won't change while you're editing other coefficients.

Clicking on the Simplify button will cancel all locks and sum together nonorthogonal terms. Zero terms will then be eliminated, and the new superposition will be renormalized. This turns the state into the simplest and most well behaved form of the superposition, minimizing the number of eigenstates used and providing the best display of the state.

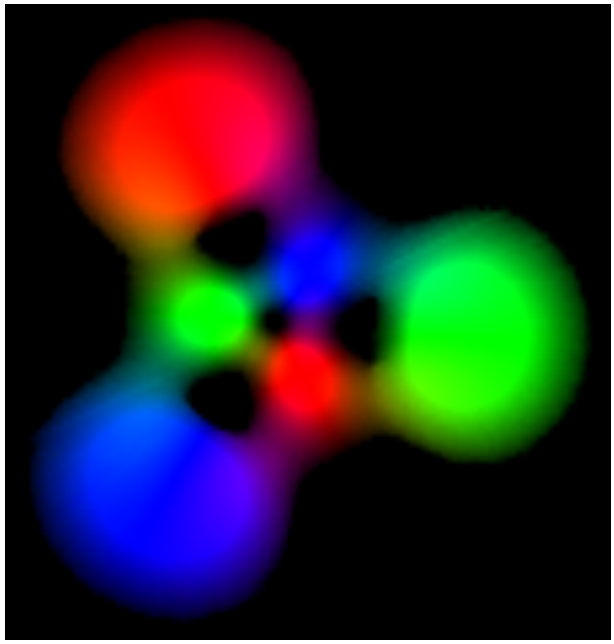
Here's a short gallery of superposition states:

This is
(|2,1,0> + |2,0,0>) √0.5,
an sp state.

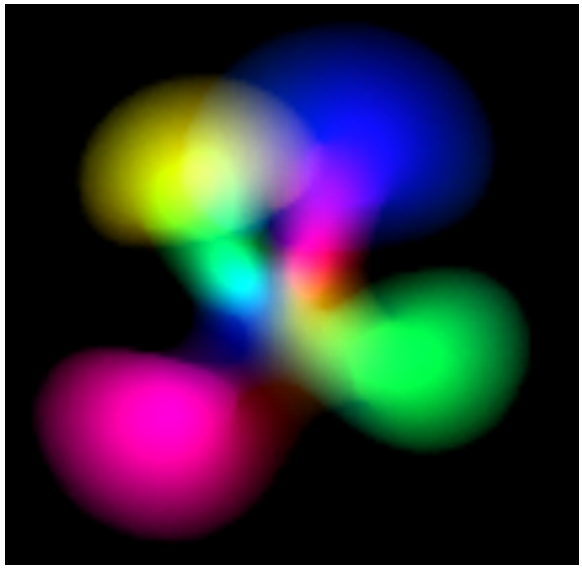


I was amazed when I saw these two:

This is
(|3,2,2> + |3,1,-1>) √0.5



and
(|3,2,1> + |3,1,-1>) √0.5.



Can you predict the shapes of others?

Superposition is a feature I have not seen in any other program for visualizing orbitals. (I've seen a few precomputed isosurface plots of an sp state). This code allows a superposition of up to 8 different eigenstates out of 140 available. That means there are 3 trillion available combinations of eigenstates. (If only 1% of those were interesting, then there would be one unique interesting superposition for each person on the planet, with enough leftover for everyone born in the 21st century!) And then that number of

possibilities is multiplied by the number of allowed combinations of the coefficients. So I figured that was enough.

Save into PICS File (under the File menu)

This program can also save a series of images, based on the current rotation and settings, into a series of PICTs stored in a PICS file of your choice. This file can then be read by any PICS player or converted into a QuickTime movie using free utilities from Apple. I recommend that you set the Samples setting to its highest level and turn off time evolution if you are using Phase as Color for best results.

Behind the Scenes - How does this code do what it does?

The simplest brute force way of doing it, like how one might do it in Mathematica, would be to fill a large three-dimensional array of values using the mathematical solution of the wavefunction. Suppose you wanted a resolution of 256 x 256 x 256 in x, y, and z. Well, that's a little big, so you can probably interpolate from 128 x 128 x 128. That's about 2 million elements, each of which should require at least single-precision floating point numbers, which are 4 bytes big. Then that table would be 8 megabytes big. Of course, the wavefunction is complex, so that requires two floats per position, so that's 16 megabytes of data. And then you have to rotate your coordinates and raytrace through all of that. Even doing just an isosurface of that data can get slow without special hardware.

So, obviously, that's not it, because the entire application requires barely over a megabyte, much less than 16 megabytes. Another way might be to compute the wavefunction only when it is needed, so minimal data space is required. Well, you can have a look at the equations that the program displays. They have cosines, sines, and exponentials, all of which are nasty to numerically evaluate over and over again. The PowerPC is fast at math, but it's not quite fast enough for that approach. (And neither is the Pentium series. Maybe a bunch of 900 MHz DEC Alphas....)

So what's the trick? In a programming tour de force for speed, this code utilizes lookup tables to an extreme. All operations more complicated than add and multiply are done using tables. That is, divides, square roots, exponentials, cosines, sines, arctangents, hue to RGB, etc. are all deep inside a combination of tables. And the tables are folded up in such a way so that all the tables used at any one time can fit inside the Level 1 cache onboard the PowerPC 604e chip (64 kilobytes).

So let me give you the rundown for how much memory is used for what:

- Only 7 kB of code is running at any one time. (The application is separated into many sections, e.g., one for Standard 3D, another for Phase as Color, etc., but typically only 7 kB is used for each combination of settings.) I've tried to optimize the code heavily, including manually unrolling loops. This code should fit in the L1 cache too, otherwise the chip fetches instructions from slower memory, like the Level 2 cache. My tests indicate that L1 cache is at least 30% faster than the L2 cache and at least twice as fast as main memory on my PowerTower 200e.
- 4 kB for a single eigenstate in the Standard 3D and Red/Cyan 3D modes. That's a far cry from the 8 megabytes mentioned above.
- 3 kB of RGB tables and another 0.5 kB of additional phase information for the Phase as Color mode.
- 8 kB for the gamma table, if used, although perhaps only half of it might be used.

• 5 kB is used to describe completely each member of the state when Superposition is on. Again, this is much less than 16 megabytes.
• And 6 kB of other tables essential to making it all work.

So all this has to fit within the Level 1 cache on the chip, otherwise it slows down. By the way, I have a strong suspicion that a part of the L1 cache is locked and allocated for the 68k emulator by the Mac OS. I don't know how to unlock it, but I don't think I should because that might crash something.

With the default settings when you start the program, it's taking up 7k for code + 4k for the eigenstate + 6k for other tables, totaling at about 17 kB. That fits quite nicely even on the PowerPC 601's 32 kB L1 cache. I've gotten about 10 fps on a 8100/100 (601 at 100 MHz), and on 31 fps on a 8600/200 (604e at 200 MHz).

If you're using Phase as Color, add about 3.5 kB. So that's under 21 kB.

With a superposition of two eigenstates, subtract out 4.5 kB and add two times 5 kB. So a two-state superposition with Phase as Color is about 26 kB. Add 5 kB for each additional eigenstate.

There really is a lot more work that the CPU has to do when Phase as Color or Superposition is on, so speed becomes more dependent on the CPU than in the other modes. Actually, a few months ago, I didn't think I could get either of these features working in real-time on current Power Macs. I had estimated that I would need a 1.2 GHz CPU before I could get Superposition working in real-time, but here we are.

Now, L1 cache and CPU speed are not the only bottlenecks. The bus speed has a significant effect. Typically, I write the image data to a 32-bit offscreen pixmap, then, once the image is done, dump it to the screen using QuickDraw, which handles details like dithering to a lower bit depth and windows overlapping mine. Interacting with those pixmaps, which are about 256 kB each, is dependent on the bus and off-CPU memory speed. If your monitor is at Millions, you can try bypassing QuickDraw (under the Display menu), so that my code writes directly to the VRAM and skips the QuickDraw dump of 256 kB through the bus. On my PowerTower 200e and the Power Mac 8600/200, the frame rate jumps from 31 fps to 42 fps.

So why did you do this?

The *Mechanical Universe* is a series of videotapes produced at Caltech that runs the gamut of first year undergraduate physics. It's an excellent source for visualizing and understanding basic concepts in physics. In one episode near the end, entitled "Atoms to Quarks", they show probability density clouds for the n=1 and 2 states, and stop at the |3,2,m> states. I loved it, but it all goes by very fast. The tapes date from the 80's, so I'm sure they computed all the images on some computer and dumped them to videotape. That's the only time I remember seeing the orbitals plotted as probability clouds. All the other portrayals of orbitals I've seen are scatter plots and single isosurfaces, which really hide a great deal of the structure of orbitals, or, even worse, just the Ylm spherical harmonic functions from Mathematica without the radial dependence.

So I decided that I wanted to generate volumetric (i.e., clouds) plotting of the orbitals in real-time on my computer. My earliest attempts were on my //ci (68030 at 25 MHz) back in 1994, but there I only got as fast as one frame per minute (!) with no tables. My first PowerPC attempt with tables was in 1995, but it failed miserably (I wasn't using the tables correctly). Then in 1997, I resurrected the code and was a lot more careful and got

Standard 3D and Red/Cyan 3D working and working fast. In January 1998, while I was at the American Association of Physics Teachers conference, I figured out how to quickly translate the phase of the wavefunction into color using tables, and then in April 1998 I applied similar techniques for superposition and spruced up the user interface.

Why the funky grayscale look?

Well, it's grayscale for two reasons: 1. When in the 3D red/cyan mode, anything other than grayscale around the orbital is very distracting and hurts some people's eyes after a while. 2. It makes the color in the PhaseColor mode stand out more. So the theme I thought of here was like a piece of stereo equipment.

And as for the funky part: From 1992 to 1994, I worked for a company, known today as MetaCreations, and was one of the two programmers who wrote Kai's Power Tools, a set of filters that plugged in to Adobe Photoshop for artists and advertisers. Besides doing neat things with images and being the first filter set with real-time previews, it had a very 3D-ish interactive user interface, coded by us two programmers (the other one still works there) and designed by Kai Krause (he's done Yes album covers, among many things). Some users thought the UI was psychedelic (almost literally). So, I suppose my UI artistic style was influenced by my experience working with the people there, but perhaps I'm a little more reserved (and definitely more physics oriented).

Are there any easter eggs?

Yes, there is one easter egg.

Thanks goes to Robert M. Zirpoli for critiquing the program and this Read Me.