Helix is a graphics demonstration program by Christopher Tate

This documentation describes version 1.0 of Helix.

Printing This Document

This document uses the Helvetica and Courier fonts. It can be printed on a LaserWriter (select "Font Substitution" and "Text smoothing" for best results) or on an ImageWriter. The documentation was prepared using Microsoft Word. I apologize to all those people who only have access to MacWrite; it was just easier to format this thing in MS Word. I also apologize for the size of the documentation; several people expressed the opinion that being throrough was more desirable that being concise.

The Freeware Stuff

Helix was written in 1988-89 by Christopher Tate, using Turbo Pascal 1.1 at the Pennsylvania State University. It may be used and distributed freely, but may not be sold, nor may it be placed under any organization's anthology copyright, without the author's express written permission. Also, you may not alter this program or its documentation in any way without the express written permission of the author. In short, you may pass copies of this program along to someone else at will, or to upload it for general distribution, but you may not under any circumstances charge money for doing so, nor may you restrict the rights of the recipients to distribute this program under the terms stated here. You must include a copy of this documentation with any copy of the program that you distribute.

What Helix Does

Helix generates its pictures by independently traversing two Lissajous figures. The program keeps track of the current position along each figure, and draws lines connecting the two figures each time it takes a step along their paths. This process generates an intricate ribbon-like pattern which eventually returns to its starting point. The panels in the "About Helix..." dialog give a brief description of the effect of the program's various parameters on the finished picture. The algorithm looks (roughly) like this:

```
x1 := Radius1 * cos(Angle * Factor1);
y1 := Radius2 * sin(Angle * Factor2);
x2 := Radius2 * cos(Angle * Factor3);
y2 := Radius1 * sin(Angle * Factor4);
LineTo(x1, y1);
LineTo(x2, y2);
Angle := Angle + Increment;
```

Helix also includes the capability to save pictures in its own file format. The saved images take up only one block of disk space each, since they are actually only 22 bytes long. This is accomplished by saving only the drawing parameters, rather than the bitmap or the line sequence.

Helix can print its pictures to either an ImageWriter or a LaserWriter. The program automatically scales the screen image to be nearly full-page size when printed. In addition, when printing to a LaserWriter, Helix bypasses the QuickDraw printing interface and sends its pictures directly as PostScript commands. This results in exceptional line definition and clarity.

How to Use Helix

Here's where I tell you about how the parameters determine what the pictures look like.

Radii:
These are two values, Radius 1 and Radius 2, which determine (approximately) the relative sizes of the two Lissajous figures. On a Macintosh SE, 150 is the maximum radius for which the picture will fit in the display window. I'm not sure what the maximum radius is on a Macintosh II; my guess is around 220.

When Helix is launched, the Radius 1 is set to the largest multiple of 50 which will fit inside the display window. Radius 2 is initialized to be 50 less than Radius 1, for a slight "ribbon" effect.

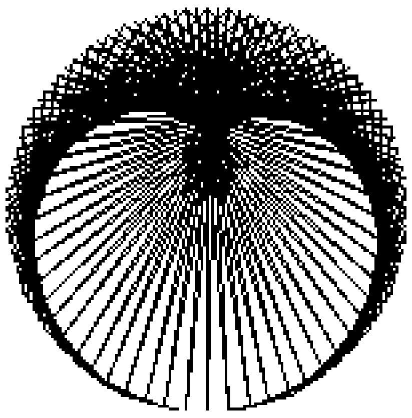
Important note: the Radii are not strictly speaking the relative sizes of the Lissajous figures. In actuality, each Radius defines the x-radius of one figure and the y-radius of the other. This was done to promote a "twisted ribbon" look in the pictures.

Angle Increment:
This value is the common angle increment (given in degrees) used in traversing both Lissajous figures. In general, small angle increments (less than 5) give a dense, smoothly traced single pass through the picture. Large values such as 61 or 137 give a multi-pass structure to the picture, as well as a finer criss-cross pattern.

Factors:
These numbers determine the basic shape of each Lissajous figure. They form two ordered pairs of factors; each pair describes one figure. Factors 1 and 2 are the first figure; Factors 3 and 4 are the second. Setting both factors in a pair to be equal specifies a circle. Setting Factor 1 = 1 and Factor 2 = 2 specifies the "figure eight" of the default pattern. The important thing here is the ratio of the two factors. If instead you set Factor 1 = 2 and Factor 2 = 4, you will get the same figure, traversed twice as fast. In short, these numbers are traversal factors: if you double one of them, you'll traverse that axis twice as fast.

A good way to learn what the various Lissajous figures look like is to set both of the Radii to be equal, set the Angle Increment to be a small number such as 3, and set corrosponding Factors to be equal. That is, Factor 1 = Factor 3 and Factor 2 = Factor 4. This will generate thin tracings of the pure Lissajous figures. The small Angle Increment makes the figures smooth. If you use a large value, the jumps along the figures will cause interesting cross-hatches to evolve.

The following pictures give an example of the effects of varying angle increments, but they can also serve as a simple example of the designation of Lissajous figures in Helix. They were generated with Helix, then saved to a MacPaint file by the traditional "Shift-Command-3" sequence. From MacPaint, they were clipped out and pasted into this document.



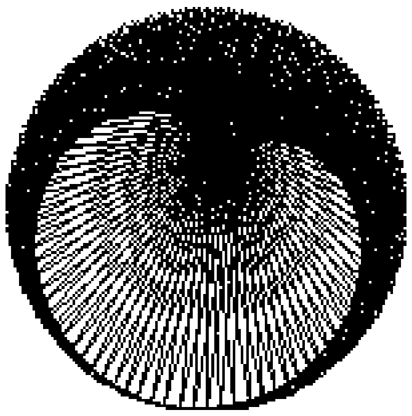
This picture was generated with these parameters:

Radius 1 = Radius 2 = 75 This makes the figure small and aligns the two Lissajous figures ("Ribbon" width of zero).

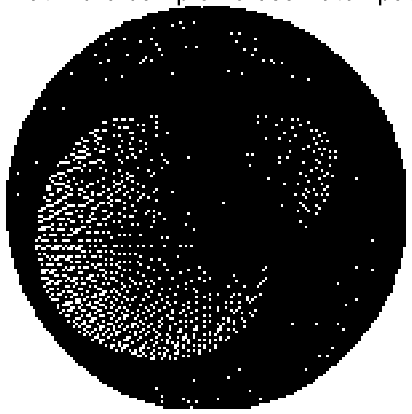
Factor 1 = Factor 2 = 1 This defines the first Lissajous figure to be a circle.

Factor 3 = Factor 4 = 2 This also makes the second Lissajous figure a circle, but it is traversed twice as fast as the first figure. The upshot of this is that lines are drawn fully across the picture, since the second Lissajous figure has completed one full circuit when the first is only half finished.

Angle Increment = 4 Four degrees is a fairly small increment, and it is a factor of 360. So, the picture is fairly simple.



This picture was created using the same parameters as the previous picture, except that the angle increment is 14 degrees. This makes the picture denser, especially since 14 is not a factor of 360. This causes a multiple-pass picture development, with a somewhat more complex cross-hatch pattern emerging.



This final variation (which is a sample parameter file supplied with Helix, called "Mandelbrotish") uses an angle increment of 61 degrees. 61 is prime, so many passes were required to bring both Lissajous figures back to their starting points at the same time. Unfortunately the picture is unclear because of the high line density (it looks much better on screen or printed from Helix). The high angle increment also produces a separation of the "cusp" into two widely separated images. Notice that the separation between the two new cusps is about 61 degrees — get the idea?

Watching how these and other pictures develop on the screen is a useful way to become accustomed to the manner in which the parameters affect the final picture. For this reason several parameter files are supplied in the Stuffit version of this program which is available for downloading.

Helix Menus

Here's a description of what the various menu selections do:

Apple Menu:

About Helix	Calls up a three-pane description sequence of the Helix program, which includes a brief explanation of the program's algorithm and the author's E-mail addresses.

File Menu:

Clear Screen	Erases the screen and redraws the current picture. This is useful if (for example) if you really can't stand watching it fill in the gaps after a portion of the image gets erased by a DA or some such.
Invert Screen	Toggles between black-on-white drawing and white-on-black drawing. Bear in mind, though, that Helix will only print black-on-white.
Save Parameters	Allows the user to save the parameters for the current picture under the filename of his or her choice.
Load Parameters	Allows the user to load picture parameters from a previously saved file. Note that Helix has its own file format and file type.
Page Setup	The standard Page Setup dialog for printer initialization.
Print	Write the picture to the selected printer. The image will be printed at the best possible printer resolution that still maintains the proper aspect ratio.
Quit	Goodbye for now....

Parameters Menu

Radii	Calls up a dialog to allow the user to change the two Radii values. These are integer values.
Angle Increment	This dialog allows the user to specify the base angle increment discussed above. This number is expected to be an integer number, and is in degrees.
Factors	This dialog lists the current Factors and allows the user to specify new ones. These factors are integers.

Contact Information

As you can see (here I borrow another line from Dave Platt), I'm not making any of money off of Helix. It's essentially been an exercise in Macintosh programming, for the purpose of getting used to all that machine-specific stuff. It's also been a lot of fun. I admit, it's not the tour de force of non-commercial software that MandelZot is (you might have guessed by now that I'm extremely fond of MandelZot...). But I like it, and some of my friends like it, so here it is. If you like it too, just drop me a note saying so. If you have suggestions, drop me a note. If you find a bug, by all means tell me so I can iron it out. I'll be upgrading the versions available on public file servers whenever I have a new one to release.

I can be reached at the following addresses:

Internet:	fixer@faxcsl.dcrt.nih.gov (preferred) cxt105@psuvm.psu.edu
BITNET:	CXT105@PSUVM

U.S. Snail: Christopher Tate
7 Clemson Court
Rockville, MD 20850

The PSUVM addresses are at the Pennsylvania State University; there's no telling how permanent that account or login ID may be. Future versions of Helix will have corrected address information. The faxcsl.dcrt.nih.gov account should be quite stable, though; you should be able to reach me there without any trouble.

Acknowledgments

There have been a lot of people offering helpful suggestions and tips during Helix's somewhat sporadic development. Special thanks to Sonja and John for general enthusiasm; without them this would probably never have become freeware. Thanks also to Dave Platt (indirectly and much to his suprise) for his gracious permission to look at the source code for MandelZot. Without that source, I would never have found out how to get the Mac to send PostScript to the LaserWriter. Thanks to him, also, for telling me what was wrong with that approach, and for pointing out which Tech Notes I should consult.

I am also indebted to the Pennsylvania State University, who allowed me the use of their microcomputer facilities, and encouraged me to release Helix as freeware.

Known Bugs

None.

Update and Version History

Nonpublic versions of Helix were released to a small alpha- and beta-test population. These versions may surface here and there with spurious version numbers. These were not strictly speaking public releases. This is the first freeware version, and is accordingly designated 1.0. Previous versions, as shown below, have been back-numbered to indicate the development of the program (in case anyone's interested).

1.0 First freeware release, after rewritng the printing code for greatly increased device-independance.

0.9.1 "Idiot-proofed" the Angle Increment and Factor dialogs to prevent the user from entering zeros in these cases. Zero values prevent the calculation of the number of line segments to draw.

0.9 Optimized the drawing loop; rewrote the printing routines to improve image resolution on the LaserWriter.

0.8 Cleaned up the screen refresh code and the event dispatcher to properly handle various update situations.

0.7 Rewrote (i.e. corrected) the algorithm which determined how long the program must calculate before the picture was complete.

0.6.1 Fixed a recursion bug which caused a stack overflow whenever a negative Factor was specified.

0.6 Rewrote and tightened the file-handling code; added code to handle activate events from desk accessories.

0.5 Added code to print designs. Added desk accessory support. Reworked the event dispatching routines.

0.4 Nonpublic version: added capability to determine how long to draw before the image was finished. Adding this capability required constraining the angle increment to be integral. (Prior to v0.4, the program would simply overwrite the existing picture *ad infinitum.* This version was intended as a stepping-stone to printing capability.)

0.3.1 Fixed bug in the `extended`-to-string and string-to-`extended` routines.

0.3 Discarded the Turbo Pascal `Num2String` and `String2Num` routines in order to avoid having to include the SANE library. Substituted my own `extended`-to-string and string-to-`extended` routines.

0.2 Rewrote the input-parsing routines for the Parameters dialog boxes.

0.1 First semi-public release, mainly to friends at Penn State.