L-System Help System

Orientation Commands

+	Turn left up and around vector
+(x)	Turn x left up and around vector
-	Turn right up and around vector
-(x)	Turn x right up and around vector
&	Pitch down around left vector
&(x)	Pitch x down around left vector
^	Pitch up around left vector
^(x)	Pitch x up around left vector
<	Roll left (counter clockwise) around forward vector
<(x)	Roll x left (counter clockwise) around forward vector
>	Roll right (clockwise) around forward vector
>(x)	Roll x right (clockwise) around forward vector

Special Orientation Commands

|	Turn 180 deg around up vector
%	Roll 180 deg around forward vector
$	Roll until Horizontal
~	Turn/Pitch/Roll
~(x)	Pitch down around left vector
t	Pitch x down around left vector
t(x)	Pitch up around left vector

Movement Commands

		When { } Active
F	Move forward and draw full length	Record Vertex
F(x)	Move x forward and draw full length	Record Vertex
Z	Move forward and draw half length	Record Vertex
Z(x)	Move x forward and draw half length	Record Vertex
f	Move forward full length	Record Vertex
f(x)	Move x foward	Record Vertex
z	Move forward half length	Record Vertex
z(x)	Move x foward	Record Vertex
g	Move forward full length	Don't record Vertex
g(x)	Move x foward	Don't record Vertex
.	Don't move	Record Vertex

Structure Commands

[	Push current state
]	Pop current state
{	Start polygon shape
}	End polygon shape

Increment / Decrement Commands

"	Increment length by 1.1
"(x)	Multiply length with x
'	Decrement length by 0.9
'(x)	Multiply length with x
;	Increment angle by 1.1
;(x)	Multiply angle by x
:	Decrement angle by 0.9
:(x)	Multiply angle by x
?	Multiply thickness by 1.4
?(x)	Multiply thcikness by x
!	Multiply thickness by 0.7
!(x)	Multiply thcikness by x

Additional Commands

c	Increment color index
c(x)	Set color index to x
@	End of file
#	Comment

What is an L-System ?

Lindenmayer system, or L-System, was introduced in 1968 by the biologist Aristid Lindemmayer, primarily conceived as a mathematical theory on plant development. In the 'bible' of L-Systems, "`The Algorithmic Beauty of Plants" (ISBN 0-387-97297-8) - or ABOP for short-, Lindenmayer and Prusinkiewicz wrote:

"The central concept of L-systems is that of rewriting. In general, rewriting is a technique for defining complex objects by successively replacing parts of a simple initial object using a set of rewriting rules or production."

An l-system is a rule like description of a 3d form. It contains descriptions of parts and how they should be assembeld together. The program reads a l-system description in and processes it into a 3d form which can then be outputed in several formats, including DXF and VOL.

The description is applied to itself a number of times (= recursion levels) so fractal and recursive forms are very easy to describe in an l-system. That's why they are used a lot for plants and natural looking organic forms. By increasing the recursion level the form slowly 'grows' and becomes more complex.

A word from the author

This program was written as a project to bring the DOS based L-System program of Laurens Lapre (notation and C-based code) into the Windows environment utilizing Delphi. This release is not intended as a final product, but as a continuing development project for my own learning.

This project has evolved through its phases of development. Initially, my objective was to import the lparser engine into the Windows environment. After Version 1 release it was painfully obvious that additional viewing capabilities were required. Version 2 addressed some of Version 1's limitation, but lacked the ability to see a quality picture within the program, hence this Version.

With the release of Version 3, I am postponing my graphics development in leiu of attacking the incorporation of file generation features such as:

- Mutation of files
- Random generation of files
- Utilizing a Genetic Algorithm to "breed" a randomly generated population.

I encourage you to visit www.thinkpiece.com. Bryan Smith has dedicated a significant portion of his site (and time) to support L-Systems. You can send your seed files or final graphics which utilize L-System objects to him via e-mail at: bryan@thinkpiece.com.

In closing, please forward any suggestions or complaints to me at: tperz@hotmail.com.

Timothy C. Perz
28 October 1998

SEED FILE FORMAT

An L system file format is completely structured. The file is a standard ASCII text file with a defined order for passing paramters from the text file to the program which acts on those parameters to create an output file. (Note: Any line beginning with a "#" sign is assumed to be a comment and everything on that line is ignored.)

The parameters which are passed to the progam are:

- **w**The number of recursions (Line 1)
- **w**The default angle used for orientation commands (Line 2)
- **w**The default line thickness used for drawing (Line 3)
- **w**Axiom (Start string) (Line 4)
- **w**Rule 1 (1st Substring which acts on axiom) (Line 5)

.

.

.

- **w**Rule N (Nth Substring which acts on axiom) (Line N+4)
- **w**"@" (End of Rules Marker) (Line N+5)

TUTORIAL

The following tutorial was taken from the tutorial originally written by Cees van der Mark jr. . References to the LParser and LViewer programs have been edited since this program is intended to serve the same functionality as both of those programs.

```
The first thing to understand is that a lsystem has its own orientation in 3D
space.

We start with the symbols for left and right:

       ^
  +    |   -
       |
-----------------

The arrow points at the "draw direction", to move left: use +, to move right: use -

An example: (after the "#" sign the parser knows the rest is comment)

2      # recursion depth
25     # angle
50     # thichness as % of length
F      # draw a full length
@      # end of file mark
```

If you view the result of file, you just see a red bar standing up straigth.
Now try the next file:

```
2
25
50
+F
@
```

Viewing the result shows the same red bar, but now tilted in an angle of 25 degrees. Watching the picture you see that the object is tilted somewhat towards the screen. So "+F" in this case means "turn left by the angle and draw a full length", "-F" would mean "turn right by the angle and draw a full length". Remember we defined the angle to be 25 degrees. Try the file with different angle and thickness settings, so you get used to it.

Now the recursion depth: recursion is a loop, the file loops in itself as many times as defined.

An example:

```
4
25
50
A      # the axiom (the final definition of the object)
A=+FA
@
```

This file will loop 4 times, lets do that on paper:

```
loop 1      +F        so A=+F
loop 2      +F+F      so A=+F+F
loop 3      +F+F+F    so A=+F+F+F
loop 4      +F+F+F+F  so A=+F+F+F+F
```

In every loop the A gets an "+F" added because we defined A to become +FA. Try to imagine what object this file should generate. First an angle of 25 deg, followed by a full length, again an angle change of 25 deg and again a full length and so forth. If you repeate this long enough it should become a circle. View the file ..you'll see an open ended circle.

We also have the "-" option to try out, so:

```
4
25
50
AB      # the axiom
A=+FA
B=-FB
@
```

Reading this file tells us (as final string): +F+F+F+F-F-F-F-F (which is equal to AB). Again: try to imagine what this object would look like, than try it.

As you probably noticed the program draws every next F ON the previous one, this is because we just typed "AB"; so B follows A. We can alter the definition in such a way that this will not happen, we use the "[]" symbols for it. Everything between the "[]" symbols is drawn but not counted for..i.e. the program draws but does not remember where it left off between the brackets, it remembers only where it left off before the brackets.

A file to explain this:

```
4
25
50
[A][B]    # the axiom
A=+FA
B=+FB
@
```

The final string will be: [+F+F+F+F][-F-F-F-F] So the program starts at the bottem, draws the fist string [A], than again starting from the bottom it draws the second string [B].

"+" and "-" is left and right (around up vector:), but there are more orientation commands, try this:

```
4
25
50
[A]c[B]c[C]c[D]
A=+FA
B=-FB
C=^FC
D=&FD
@
```

The final string will be: [+F+F+F+F][-F-F-F-F][^F^F^F^F][&F&F&F&F]. The "^" and "&" are pitching up and down around the left vector. So this should be left and right on the 3D axis in the space around the object. Also "c" is added in the final definition, this is the color command, every time it is used the color changes. By editing them out one by one it becomes clear which definition becomes what object in the drawing.

View the file, and try deeper recursions. What happens if the recursion gets above 14 ? What happens if the "c" command is in between the brackets instead outside them? Why?

Notice that +F in these files means "left 25 degrees, draw F", it can also be defined as +(25)F. That way you can use angles different from the predefined angle.

"E" (after the lparser has gone through all the loops!) can be inserted into the the form itself, imagine what that would do: placing the form on each of

its own branches!

try this file:

```
8
25
50
E
E=[A][B][C][D]
A=c+FAE
B=c-FBE
C=c^FCE
D=c&FDE
@
```

As you can see: "E" is inserted onto every branch. This is a fractal form, i.e. a form ever repeating itself in itself. There are some differences, we pre-defined the first form, but hey...the same thing goes for the famous snowflake fractal!

Lets do that Snowflake as well!

By now you will know what is done in the first steps creating a .ls file setup:

Lets do it by the book:

"The construction of the Koch curve proceeds in stages. In each stage the number of segments increases by a factor of four." The initiator (the smallest form of the fractal) is "F" of course, the generator (the smallest fractal part still having the form of the fractal) is a line like:

```
  _/\_
```

The angles are clearly 45 degrees, so the generator should be defined as:

```
4      # no need for deep recursion yet.
45     # the angle of 45 degrees.
50     # well... 50 is a good thickness.
A      # The axiom (for now).
A=++F-F++F-F--
@
```

So, that was simple, now it is getting a bit more complex: We just made the generator, this is the form that must be repeated in inself, keeping the same form, now this generator is used to create the same form again:

```
4
45
50
B
A=++F-F++F-F--
B=A-A++A-A
@
```

As you see, the generator is used to create the same form as the generator itself, the beginning of the snowflake fractal is made. We could get carried away and

define C D E (not F!) G etc. every time including the growing form.... like:

```
4
45
50
D
A=++F-F++F-F--
B=A-A++A-A
C=B-B++B-B
D=C-C++C-C
@
```

If you view this file the snowflake KOCH fractal is there! And without recursion, we defined every bit of the form. That is not the way.
Assume you want it to be repeated 100 times...some typing to do!, not even speaking of the symbols you have going to use to define them all, the alphabet (with use of upper and lower case letters) is to short to define all the forms!
So we have to use recursion depth, like this:

```
4
45
50
C
A=++F-F++F-F--
B=A-A++A-AC
C=cB-B++B-B
@
```

As you read the file, you will see that the "C" form is included into the "B" form, which itself is part of the "C" form, etc. etc. Now we can use recursion depth!
Try it, but keep the level below 12 :) Every new "C" form has a new color, defined with the c at the beginning of the C definition.

Maybe it is a good idea to make another fractal like object. It's a simple one, called the Pythagoras tree. It starts with the mean branch, then on it, under an angle of 45 degrees, two branches with a length of 1/SQR(2) times the mean branch. On them again two branches under 45 degrees with a length of 1/SQR(2) times the PREVIOUS length and so on, all in the same plane...
Starting from the base form (F) and working our way up:

```
5
45
10
F
@
```

This is the mean branch, now we have to make the two branches on it, with a length

of 1/SQR(2) the mean branch, we use the command '(0.7071) for it. Using this the file should look like this:

```
5
45
10
A
A=F[+'(0.7071)F][-'(0.7071)F]
@
```

As you see, I used the branching commands [] to do the job. Now the only thing that remains is: get it recursive! Calculating the lengths is a rule like 0.7071*0.7071*0.7071.....etc, each recursion the length is decreased by the same factor, this is the first thing that could be made recursive.

Doing just that, we get:

```
5
45
10
A
A=F[+BF][-BF]
B='(0.7071)
@
```

B is now incorporated into A, but.... viewing the output shows an object with only two branches. We still have to get the branching recursive. This is done by incorporating the form into itself. (This sounds cryptic huh...:)

Like:

```
5
45
10
A
A=[+BFA][-BFA]
B='(0.7071)
@
```

Try this at recursion level 10, looks nice does it! But it is still all in one plane, and the program is much to powerful to let it stay that way! If we add branching in the other two directions as well, maybe that will do the trick:

Like:

```
5
45
10
A
A=[+BFA][-BFA][^BFA][&BFA]
```

```
B='(0.7071)
@
```

Viewing the output shows a 3D tree, but the BASIC form of the fractal is lost. Clearly this is not the way, to get a good 3D model one should make a so called rotation model of the fractal, i.e. turning it 90 degrees (in this case!) on its central axis, continuously writing the 2D model, just like rotation models with mathematical integration.

The next file shows the form into four directions. As you can see, I coded the form twice (definitions A and B), enclosed them with [], so the startingpoint does not alter, and put them together just by adding them to each other in the AB command.

```
9
45
10
AB
A=[F[+FCA][-FCA]]
B=[F[^FCB][&FCB]]
C='(0.7071)
@
```

Also possible :)

```
#--------L-System--C.J.van der Mark-----
#--------Pythagoras semi 3D form--------
5
5
10
Y
Y=X>X
X=A>B>A>B>A>B>A>B>A>B>A>B>A>B>A>B>A>B
A=[F[+(45)FCA][-(45)FCA]]
B=[F[^(45)FCB][&(45)FCB]]
C='(0.7071)
@
```

Cees van der Mark jr. cvdmark@xs4all.nl / http://www.xs4all.nl/~cvdmark

ACKNOWLEDGEMENTS

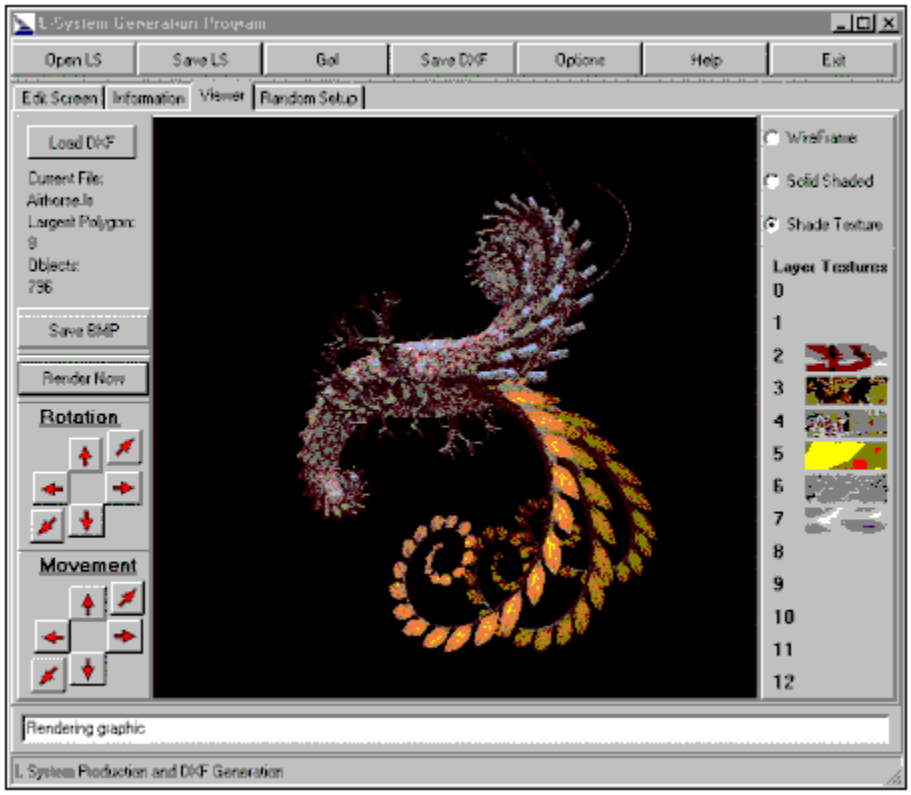
This implementation of an L-System development enviroment is based on the work of Laurens Lapre. Visit his Website for the original program (and associated files) as well as the c-based code for the Lparser program which he developed.

The Viewer is based on a series of articles by Peter Dove and Don Peer in the Delphi Informant. Great articles and great magazine.

DISCLAIMER

GRAPHICS VIEWER

L-System Version 3 includes an integrated viewer which allows the user to rotate and move the object generated by the seed file.



Grahics Features:

Viewing of object in wireframe, shaded, or shaded texture mode.

Capability to open previously generated objects which were saved as DXF files.

Each layer is assigned a texture.

Reassignment of textures for each layer (just click on a texture to call up a selection dialog).

Capability to save viewer window to BMP file.

(Note: I have not had a problem opening the BMP output file in either Microsoft Paint or Paint Shop Pro, but I have heard that there is a problem with the BMP file opened in Photoshop.)

FUTURE DEVELOPMENTS

Planned features to be incorporated in future releases of this program include:

- Output to several file formats
- Mutation, Random, and Genetic generation of seed files
- New building blocks
- Expansion of the rules to include special capabilities
  - Selection of different blocks within the Rules
  - Final iteration commands
  - etc.
- Incorporation of L-Systems Extensions (Circles & Spheres, Generalized Cylinders, Surfaces, etc.)
- A better help file

FEEDBACK

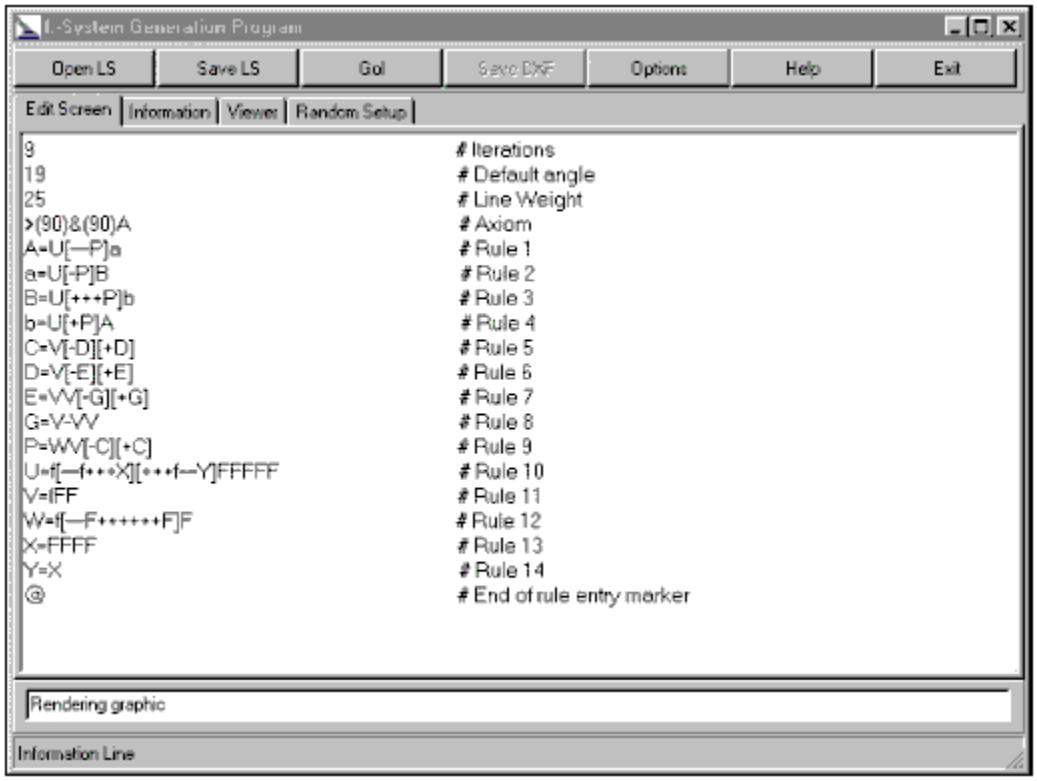
Please give me some FEEDBACK. Whatever you think of the program let me know. I would really like to know where there is a problem, and more importantly, functionality that you would like to see.

E-MAIL: tperz@hotmail.com

I will try and reply to all the e-mail messages I get. For those of you who cannot send e-mail, sorry. I am working in Saudi Arabia, so the most expeditious means of reaching me is via e-mail.

PROGRAM FUNCTIONS

Program Functions are available via the overhead toolbar and the tabs.



Open LS File:

The Open LS File Button calls an Open Dialog Box listing all files with an "LS" extension. The Defualt directory for the Open Dialog Box is "\lsystem\lsfiles".

Save LS File:

The Save LS File Button calls a Save File Dialog Box. The program will prompt the user if a filename is already in use. The Defualt directory for the Save Dialog Box is "\lsystem\lsfiles".

Go!:

The Go! Button begins the process which creates the output file. The program:

- **w** Reads the input parameters listed in the Edit Screen
- **w** Checks the validity of the input parameters (somewhat)
- **w** Generates a production string based on the input parameters
- **w** Generates an output file based the Options set by the user and the values in the production string.

View DXF:

The View DXF Button calls the DXF viewer process.

Save DXF:

The Save DXF Button saves the output of the process into the "\lsystem\dxffiles" subdirectory.

Options:

The Options Button calls the Options Dialog Box.

Help:

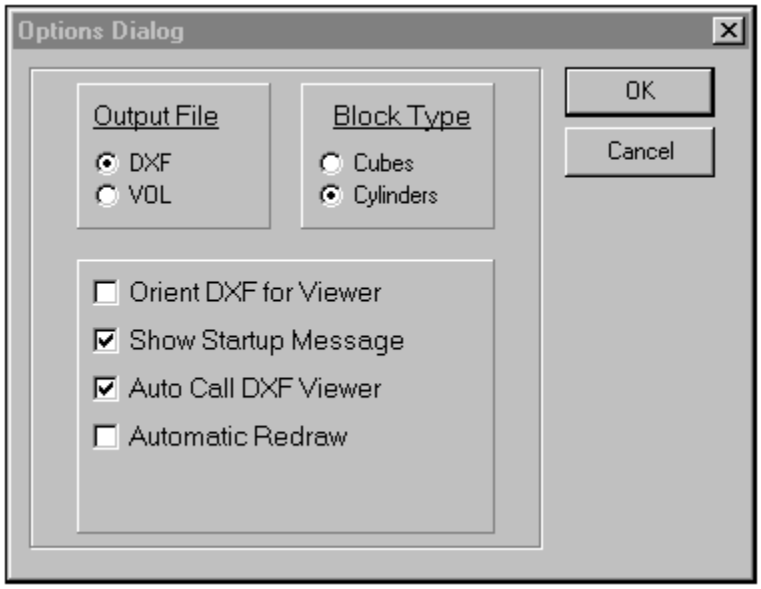
The Help Button calls this help file.

Exit:

The Exit Button closes the program.

OPTIONS

Program operation Options are available via the Options Menu.



Output File:

The LSystem Program outputs to either a DXF or VOL file format. Other file formats are planned to be added based on demand. The default Output is DXF.

Block Type:

The LSystem Program has two basic building blocks, cubes and cylinders. Other building block definitions are planned to be added based on demand. The default building block is a cylinder.

Orient DXF:

Checking this Option will add two rotational characters which I have found aids in viewing the DXF Output. This feature is more or less obsolete since the user now has the capability to rotate objects.

Show Startup Message:

This Option is to bypass the opening Startup Message.

Automatic Callup of the DXF Viewer:

This Option is to automatically start the DXF viewer file after a successful generation of a DXF output.

Automatic Redraw:

This Option is to automatically render the object once the user has stopped rotating or moving the object for a few seconds. If not selected, the user will have to press the "Render" button to redraw the object.

(Note: If this option is selected, the "Movement" and "Rotation" text above the spin buttons will be red, when not selected, the text will be black.)