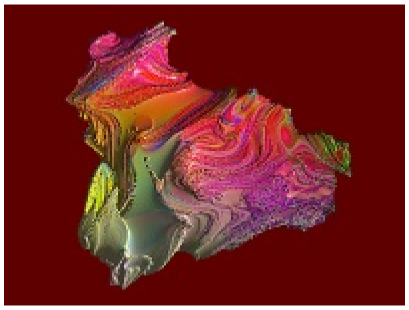
iDynaMaSZ Application Help

Table of Contents

1 Main Index

iDynaMaSZ Help Index



Getting Started

Matrix Algebra, An Introduction by Godwin Vickers

Commands

File menu
Edit menu
Image menu
Type menu
Demo menu
Video menu

Additional

Help Topics

1.1 Hyperalgebras

Tridimensional visualization of 4D Julia sets generated from Latin squares and sign rules

January 2002
Version 1.3
Godwin Vickers

1/ Introduction

This document shows a new method to define 4D algebras in R4 and higher dimensions. Iterative functions can be applied to these algebras and 4D Julia sets other than quaternion

Julia sets can then be rendered and explored. The purpose of this document is to describe these new algebras and how they can be created from a set of rules that can be implemented in a specific algorithm. This is not a tutorial on 4D Julia sets or how to turn 4D Julia sets to 3D pictures (via Z-buffering, ray-tracing or radiosity). Plenty of tutorials explain step-by-step how to render 4D Julia sets, how to calculate the distance between a light source or a viewpoint and a 4D Julia set, how to calculate the normal at each point on the surface of a 4D Julia set and which theorems and formulas are involved in these processes. For more information check the links at the bottom of this document.

In the late 1970's Mandelbrot applied iterative functions to complex numbers and discovered mathematical objects which offered a potentially infinite level of detail at all scales, these details following a rule of self-symmetry as scale decreases or increases. Mandelbrot called these objects fractals. 2D fractals have been extensively visualized for the two past decades but 4D fractals are still hiding some unknown territories that we can now explore thanks to the combination of appropriate algorithms and the increasing speed of microprocessors.

Quaternions are naturally the start of this journey because they are a 4D generalization of complex numbers (or the complex plane). Quaternions are the only normed division algebra of order 4. In fact these algebras are all Clifford algebras and this is the way they are categorized:

Cl(0)	1 dimension	R (real numbers)
Cl(1)	2 dimensions	C (complex numbers)
Cl(2)	4 dimensions	H (quaternions)
Cl(3)	8 dimensions	O (octonions)

Once Cl(8) is reached, we obtain a 256-dimensional algebra which is the building block of all other Clifford algebras since they can all be calculated from Cl(8) using a periodicity theorem. All Clifford algebras are normalized.

The unit quaternion can be considered as a rotation in 4 dimensions and is defined by the following matrix :

$$\begin{matrix} x & -y & -z & -w \\ y & x & w & -z \\ z & -w & x & y \\ w & z & -y & x \end{matrix}$$

Once you apply an iterative function to this matrix you can generate a 3D slice of your 4D Julia set. What we will show in this document is how you can modify the original quaternion matrix algebra and define a new set of 4D matrices which follow several building rules in order to explore new 4D Julia sets.

What we need to do first is dissociate the symbols from the signs so we can create a set of two matrices which, combined together, generate the initial matrix. Here are the two separate matrices :

```
x y z w
y x w z
z w x y
w z y x
```

and

```
+ - - -
+ + + -
+ - + +
+ + - +
```

(where '+' means '+1' and '-' means '-1')

Now that we have a matrix that represents the symbolic structure of the unit quaternion and a sign matrix that defines its sign rules we can analyze the two matrices and find the laws that create them. This reverse-engineering process leads not only to the rules of quaternion algebra but also to the rules of all other similarly built 4D algebras.

2/ Latin squares

The symbolic matrix we have extracted from our unit quaternion matrix is a Latin square. So in order to explore more algebras we'll define all the possible Latin squares of order 4 and keep the essential ones.

While magic squares are made of numeric values only, Latin squares are only made of symbolic values. Latin squares are to magic squares what algebra is to arithmetic.

The rules to construct a Latin square are the following:

-Once you define the order n (n being an integer) of your Latin square, n becomes the number of dimensions of your Latin square: this means there are n rows, n columns, and n different symbols.

-Each symbol must appear once and only once on each row and column of the Latin square.

There are exactly 4!*24 Latin squares of order 4 ((1*2*3*4)*24 = 24*24 = 24^2 = 576). These 576 Latin squares are the result of the symbol permutations of the 24 Latin squares of order 4 starting with a first row in lexicographic order. This means that 24 essential Latin squares can generate 576 (24^2) Latin squares if you just permute their symbols.

Here are the 24 Latin squares of order 4 with their first row in lexicographic order:

```
x y z w   x y z w   x y z w   x y z w   x y z w   x y z w
z w y x   y w x z   w x y z   z w y x   y w x z   y x w z
w z x y   w z y x   z w x y   y x w z   z x w y   z w y x
y x w z   z x w y   y z w x   w z x y   w z y x   w z x y
```

x y z w	x y z w	x y z w	x y z w	x y z w	x y z w
w z y x	w z x y	w z x y	z w x y	w x y z	y x w z
z x w y	y x w z	z w y x	y x w z	y z w x	w z y x
y w x z	z w y x	y x w z	w z y x	z w x y	z w x y

x y z w	x y z w	x y z w	x y z w	x y z w	x y z w
z w x y	w z y x	y x w z	z w x y	z w x y	w z y x
w z y x	z w x y	w z x y	y z w x	w x y z	y x w z
y x w z	y x w z	z w y x	w x y z	y z w x	z w x y

x y z w	x y z w	x y z w	x y z w	x y z w	x y z w
y z w x	z x w y	w z y x	z x w y	y z w x	y x w z
w x y z	y w x z	y w x z	w z y x	z w x y	z w x y
z w x y	w z y x	z x w y	y w x z	w x y z	w z y x

For Latin squares of order 4, there are exactly 4 normalized Latin squares. The correct terminology for normalized Latin squares is 'reduced' or 'standardized': this means their first row and column are in lexicographic order.

3/ Sign rules and sign matrices

Let's analyze first the sign matrix we have extracted from our quaternion matrix. If we follow the rules of sign multiplication of real numbers we can find the sign multiplication rules of quaternions in R4 (+ * + = +, + * - = -, - * + = - and - * - = +). Note that '+' means '+1' and '-' means '-1'.

```
   + + + +

-  + - - -
-  + + + -
-  + - + +
-  + + - +
```

Row 1: + * - * - * - = -
Row 2: + * + * + * - = -
Row 3: + * - * + * + = -
Row 4: + * + * - * + = -

Column 1: + * + * + * + = +
Column 2: - * + * - * + = +
Column 3: - * + * + * - = +
Column 4: - * - * + * + = +

The product of each value of each row is always negative and the product of each value of each column is always positive. These are the simple rules of our quaternion sign matrix. If you replace each + with a - and each - with a + you also get a quaternion. It doesn't matter if rows are negative and if columns are positive as long as rows are always of the opposite sign of columns and that all rows are of the same sign.

All we actually need are two strings of 4 signs to define the sign rules. If 'r' stands for 'rows' and 'c' for 'columns', we can sum up quaternion sign rules like this:

r++++
c----

And since the relationship between the r sign string and the c sign string is "r is the opposite of c", we only need to define one string of signs because the second string will automatically be the opposite of the first one. So '++++' or '----' is enough to define the sign rules of quaternion algebra.

Sign rules and corresponding sign matrices of order 4:

Four pluses:

a + + + +

Three pluses:

b - + + +
c + - + +
d + + - +
e + + + -

Two pluses:

f + + - -
g - + + -
h + - + -
i - + - +
j + - - +
k - - + +

One plus:

l - - - +
m - - + -
n - + - -
o + - - -

No plus:
p - - - -

a & p are opposites : dual sign rule 1 (ap or pa) : ++++ ---- or ---- ++++
b & o are opposites : dual sign rule 2 (bo or ob) : -+++ +--- or +--- -+++

c & n are opposites : dual sign rule 3 (cn or nc) : +-++ -+-- or -+-- +-++
d & m are opposites : dual sign rule 4 (dm or md) : ++-+ --+- or --+- ++-+
e & l are opposites : dual sign rule 5 (el or le) : +++- ---+ or ---+ +++-
f & k are opposites : dual sign rule 6 (fk or kf) : ++-- --++ or --++ ++--
g & j are opposites : dual sign rule 7 (gj or jg) : -++- +--+ or +--+ -++-
h & i are opposites : dual sign rule 8 (hi or ih) : +-+- -+-+ or -+-+ +-+-

Overall 2^4 = 16 sign rules, and 16/2 = 8 dual (or opposite) sign rules.

Sign rule 1	**Sign rule 2**	**Sign rule 3**	**Sign rule 4**
+ + + +	- + + +	+ - + +	+ + - +
- - - -	+ - - -	- + - -	- - + -

```
Sign matrix 1    Sign matrix 2    Sign matrix 3    Sign matrix 4

  + + + +                 - + + +                 + - + +                 + + - +

-  + - - -       +  + + + +               -  - + + +               -  - + + +
-  + + + -               -  - + + +               +  + + + +               -  - + + +
-  + - + +               -  - + + +               -  - + + +               +  + + + +
-  + + - +               -  - + + +               -  + - + +               -  + + - +
```

Sign rule 5	**Sign rule 6**	**Sign rule 7**	**Sign rule 8**
+ + + -	+ + - -	- + + -	+ - + -
- - - +	- - + +	+ - - +	- + - +

```
Sign matrix 5    Sign matrix 6    Sign matrix 7    Sign matrix 8

  + + + -                 + + - -                 - + + -                 + - + -

-  + + + -               -  - + + +               +  + + + +               -  - + + +
-  + + + -               -  - + + +               -  + + + -               +  + + + +
-  + + + -               +  + + + +               -  + + + -               -  - + + +
+  + + + +               +  + + - -               +  - + + -               +  + - + -
```

Overall there are 2^4 = 16 sign rules. If we discard the symmetries there are 16/2 = 8 sign rules left.

4/ Algebras defined as matrices

So far we have dissociated the original unit quaternion matrix in two separate matrices and then we found the laws to create these two matrices and all the other matrices that belong to their group.

Symbolic structure:
There are 576 = 4!*24 Latin squares of order 4.
If we discard permutations we obtain 24 Latin squares of order 4.

Note that among those 24 Latin squares some symmetries or rotations exist. I have read that there are 7 really essential Latin squares of order 4 from which you can calculate the 24 standardized Latin squares and all 576 Latin squares of order 4.

Sign rules:
There are 2^4 = 16 sign rules which are 8 pairs of symmetric sign rules.

If we now combine all the symbolic rules and all the sign rules just like we can combine the symbolic quaternion matrix and its sign matrix we obtain a group in R4 of all the matrices of order 4 which follow both sets of rules. The group we'll define here is composed of 24*8 = 192 algebras.

To combine a Latin square to a sign matrix you must not follow standard matrix multiplication. Instead you have to multiply each value of your Latin square to each corresponding value of your sign matrix.

For example:

x y z w	and	- + + +	gives	-x y z w
z w x y		+ + + +		z w x y
w x y z		- + + +		-w x y z
y z w x		+ - + -		y -z w -x

5/ Iterative functions

The iterative functions commonly used to create Julia sets are polynomials of order 2 and above.

From now on we'll be dealing with standard inter-matrix operations. Addition and subtraction are simple : add or subtract each value of the first matrix to each corresponding value of the second matrix. Squaring a matrix or multiplying it to another matrix is the key thing to understand here:

-To square a matrix (or to multiply it by itself) you have to multiply all its values by its first column.

-To multiply a matrix M1 by another matrix M2 you need to define which matrix is first and which matrix is second because M1.M2 does not necessarily equal M2.M1.
M1.M2 is the first matrix multiplied by the first column of the second matrix.
M2.M1 is the second matrix multiplied by the first column of the first matrix.
This means that depending on which matrix is placed first the product of two matrices involves a full matrix and 1/n (with n being the order of the matrix) values of another matrix.

For example, if you want to apply the classic formula f(M)=M^2+c to the matrix below you need to follow the following procedure to square your original M1 matrix:

x -y z w

y x w z
z -w x -y
w -z -y x

Multiplied with the first column of the matrix:

x
y
z
w

This gives:

x*x -y*y z*z w*w
y*x x*y w*z z*w
z*x -w*y x*z -y*w
w*x -z*y -y*z x*w

Your set of 4 discrete formulas is then (each row is a discrete function):

f(x) = x*x - y*y + z*z + w*w = x^2 - y^2 + z^2 +w^2
f(y) = y*x + x*y + w*z + z*w = 2xy + 2zw
f(z) = z*x - w*y + x*z - y*w = 2xz - 2yw
f(w) = w*x - z*y - y*z + x*w = 2xw - 2yz

If you want to multiply this matrix to the matrix M2 below,

-x y z w
 w z y x
-z w x y
 y -x w z

Then M1.M2 =

x -y z w	*	-x	=	-x*x w*(-y) -z*(z) y*(w)
y x w z		w	=	-x*y w*(x) -z*(w) y*(z)
z -w x -y		-z	=	-x*z w*(-w) -z*(x) y*(-y)
w -z -y x		y	=	-x*w w*(-z) -z*(-y) y*(x)

and M2.M1 =

-x y z w	*	x	=	x*(-x) y*(y) z*z w*w
w z y x		y	=	x*(w) y*(z) z*y w*x
-z w x y		z	=	x*(-z) y*(w) z*x w*y
y -x w z		w	=	x*(y) y*(-x) z*w w*z

To add even more flexibility you can choose not to restrict yourself to the first column and use the second, third or fourth column of your 4x4 real matrix when performing inter-matrix multiplication. This can lead to very interesting Julia sets which you would not be able to see if you only used the first column.

To divide together arbitrary matrices of this group we need to define 1/M first. For transcendental functions (functions using trigonometry) we also need additional rules.

Once we have a squared matrix f(M) we can iterate the function several times like this: f(f(f(f(f(f(f(M))))))) for 7 iterations for instance. According to a pre-defined bailout value (4 is a good choice) we can then find which points belong to the set or not depending on their convergence, divergence or cyclic behaviour.

In terms of coordinates, we are iterating 4 discrete functions (f(x), f(y), f(z) and f(w), the four rows) of our squared matrix.

6/ Conclusion

This approach allows generalization to n-dimensional matrices. For each level of dimensions the same procedure can be followed: defining various groups of Latin squares of order n (among which are isomorphic and homeomorphic groups/classes, normalized groups, etc.), defining all sign rules or matrices of order n, combining both matrices to define an algebra and then applying an iterative function to visualize them. The larger matrices become the more complex their structure can potentially be and they can therefore reveal some even more interesting Julia sets in hyperspace. As computers get faster and faster it will be possible to slowly increase the number of dimensions at the same time so that larger and larger Julia sets are defined but the number of Latin squares for each number of dimensions increases exponentially and quickly reaches billions of billions.

2 File Menu

File menu commands

The File menu offers the following commands:

New	Creates a new drawing.
Open	Opens an existing drawing.
Close	Closes an opened drawing.
Save	Saves an opened drawing using the same file name.
Save As	Saves an opened drawing to a specified file name.
Save Matrix to [OBJ]	Save polygonized quaternion as Wavefront object.
Save Matrix to [POV]	Save polygonized quaternion as a POV-Ray triangle object.
Save Matrix to [STL]	Save polygonized quaternion as STL solid file.
Save Matrix to [DXF]	Save polygonized quaternion as AutoCad dxf file.
Export Setup	Set maximum number of faces and vertices allocated for quad export.
Load Texture [QTX]	Load texture variables.
Save Texture [QTX]	Save texture variables.
Load Text Format [DZT]	Load a data file in text (platform independent) format.

Save Text Format [DZT]	Save a data file in text (platform independent) format.

2.1 New command

New command (File menu)

Use this command to create a new drawing window in iDynaMaSZ. The image and data file for the opening picture (title.png and title.dmz) are used to create the new window.

You can open an existing data/image file with the Open command.

2.2 Open command

Open command (File menu)

Use this command to open an existing data/image file in a new window.

You can create new images with the New command.

2.2.1 File Open dialog box

File Open dialog box

The following options allow you to specify which file to open:

File Name
Type or select the filename you want to open.
Drives
Select the drive in which iDynaMaSZ stores the file that you want to open.
Directories
Select the directory in which iDynaMaSZ stores the file that you want to open.
Network...
Choose this button to connect to a network location, assigning it a new drive letter.

2.3 Close command

Close command (File menu)

Use this command to close the window containing the active image. If you close a window without saving, you lose all changes made since the last time you saved it.

2.4 Save command

Save command (File menu)

Use this command to save the active drawing to its current name and directory. When you save a drawing for the first time, iDynaMaSZ displays the Save As dialog box so you can name your drawing. If you want to change the name and directory of an existing drawing before you save it, choose the Save As command.

2.5 Save As command

Save As command (File menu)

Use this command to save and name the active drawing. iDynaMaSZ displays the Save As dialog box so you can name your drawing.

To save a drawing with its existing name and directory, use the Save command.

2.5.1 Save As dialog box

File Save As dialog box

The following options allow you to specify the name and location of the file you're about to save:

File Name
Type a new filename to save a drawing with a different name. iDynaMaSZ adds the extension .dsz.

Drives
Select the drive in which you want to store the drawing.

Directories
Select the directory in which you want to store the drawing.

Network...
Choose this button to connect to a network location, assigning it a new drive letter.

2.6 Save Matrix to [OBJ] command

Save Matrix to [OBJ] command (File menu)

Use this command to save a matrix image as a true 3-D object. This uses John C. Hart's Implicit code (Quaternion Julia Set server) to polygonize a matrix formula, and then writes the triangles to a Wavefront object file. The higher the precision, the smoother the finished object becomes but the larger the object file is too. Precision is set with the Steps variable in the Parameters window, where precision=10/Steps.

Notes: some formulas produce asymmetrical object files with this routine, where one side of the quad polygon isn't resolved completely. Usually one side is markedly smoother in this case. If you enter a file name without an extension [obj] when saving an object the requester doesn't check for a file with the same extension before overwriting it. (The file extension is added later.) If you want to make sure you don't overwrite a file with an identical name, enter the filename with the applicable extension.

2.7 Save Matrix to [POV] command

Save Matrix to [POV] command (File menu)

Use this command to save a matrix image as a true 3-D object. This uses John C. Hart's Implicit code (Quaternion Julia Set server) to polygonize a matrix formula, and then smooths and outputs the triangles to a POV-ray file. The file is written as a simple scene, the smooth triangles part of a "union" object, with camera and lighting elements compatible with POV-Ray 3.5. This can be used as a starting point for more complex compositions. The higher the precision, the smoother the finished object becomes but the larger the object file is too. Precision is set with the Steps variable in the Parameters window, where precision=10/Steps.

Notes: some formulas produce asymmetrical object files with this routine, where one side of the quad polygon isn't resolved completely. Usually one side is markedly smoother in this case. If you enter a file name without an extension [pov] when saving an object the requester doesn't check for a file with the same extension before overwriting it. (The file extension is added later.) If you want to make sure you don't overwrite a file with an identical name, enter the filename with the applicable extension.

2.8 Save Matrix to [STL] command

Save Matrix to [STL] command (File menu)

Use this command to save a matrix image as a true 3D object. This uses John C. Hart's Implicit code (Quaternion Julia Set server) to polygonize a matrix formula, and then writes the triangles to a STL solid file. STL files are used with 3D printers and other machinery. The higher the precision, the smoother the finished object becomes but the larger the object file is too. Precision is set with the Steps variable in the Parameters window, where precision=10/Steps.

Notes: some formulas produce unsymmetrical object files with this routine, where one side of the quad polygon isn't resolved completely. Usually one side is markedly smoother in this case. If you enter a file name without an extension [stl] when saving an object the requester doesn't check for a file with the same extension before overwriting it. (The file extension is added later.) If you want to make sure you don't overwrite a file with an identical name, enter the filename with the applicable extension.

2.9 Save Matrix to [DXF] command

Save Matrix to [DXF] command (File menu)

Use this command to save a matrix image as a true 3-D object. This uses John C. Hart's

Implicit code (Quaternion Julia Set server) to polygonize a matrix formula, and then writes the triangles to an AutoCad dxf file. The higher the precision, the smoother the finished object becomes but the larger the object file is too. Precision is set with the Steps variable in the Parameters window, where precision=10/Steps.

Notes: some formulas produce asymmetrical object files with this routine, where one side of the quad polygon isn't resolved completely. Usually one side is markedly smoother in this case. If you enter a file name without an extension [dxf] when saving an object the requester doesn't check for a file with the same extension before overwriting it. (The file extension is added later.) If you want to make sure you don't overwrite a file with an identical name, enter the filename with the applicable extension.

2.10 Export Setup

Export Setup (File menu)

Use this command to set max faces, the target object size, and max vertices, the maximum number of indices that are allocated by the polygonizing routine. Defaults are 50,000 faces and 5,000,000 vertices. Use less to limit the size of the output file or the amount of memory used while polygonizing. Use more if necessary for higher resolution object files. With 50,000 triangle faces the output file is approximately 1.6MB for the [obj] format.

Note: the max faces variable has no effect on the size of [Dxf] exports. Dxf-formatted objects are saved without slimming.

2.11 Load Texture [QTX] command

Load Texture command (File menu)

Use this command to load variables that make up the texture and noise parameters. This also loads the palette, coloring filter, orbit trap and coloring options in a texture file [qtx].

2.12 Save Texture [QTX] command

Save Texture command (File menu)

Use this command to save the variables that make up the texture and noise parameters for the current figure. This also saves the palette, coloring filter, orbit trap and coloring options in the texture file [qtx].

2.13 Load Text Format [DZT]

Load a data file in text (platform independent) format.

2.14 Save Text Format [DZT]

Save a data file in text (platform independent) format.

3 Edit menu

Edit menu commands

The Edit menu offers the following commands:

Undo	Undo last edit, action or zoom.
Function and Type	Edit formula/type data.
Parameters	Edit initial image parameters.
Size	Sets the image size.
Lighting	Edit lighting and viewpoint variables.
Palette Editor	Edit palette.

3.1 Edit Undo command

Undo command (Edit menu)

Use this command to undo the last action. An image can be continued after an undo, if continue was enabled before the last action. Color-cycling is disabled after using Undo, though.

Shortcut
Keys: CTRL+Z

3.2 Functions and Type Window

Functions/Type Window

Fun #1 and Fun #2 are combo controls selecting up to two built-in formulas. There are additionally a Type control and a Column control that determine how the above formulas are processed. The Random Latin and Random Sign buttons randomize the Latin square and sign matrix. (For more details on the basis of matrix algebra, see Godwin's tutorial on Matrix Algebra.) The Column control determines which column of the LS matrix is used for bailout purposes. This usually has a moderate effect on the final shape of the image. Values for the Column control are limited to a range of 0 to the number of dimensions in the matrix - 1.

The Type control selects a value of 0 to 9. For a value of 0, the first formula is always used and the second formula is ignored. For a value of 1, the second formula is processed and the first formula is ignored. Type 1 is of use only if you are switching between two functions and don't want to reenter them each time you plot the other one. For a value of 2, the first formula is processed if the iterative result is greater than or equal to .5. For values of 3, the first formula is processed and its output becomes the input of the second formula, which is then processed.

Type 4 takes the average of Fun#1 and Fun#2.

Type 5 alternates between the two functions while iterating.

Type 6 takes the quotient of both functions.

Type 7 uses the lowest iterative results of both Fun#1 and Fun#2

Type 8 uses the highest iterative results of both Fun#1 and Fun#2

Type 9 uses the Formula box to enter up to 150 characters per formula.
Text can be pasted from the clipboard to the formula box by using the keystrokes apple-v.

Clicking on Random Formula, a random formula is generated in the Formula box and the Type is set to 9.

About formula syntax: This applies if you elect to enter your own formula into one of the function boxes and use the parser to generate the plot. User-named-complex variables and constants may be included in a formula. A variable must begin with a letter and may contain numbers and letters only. A variable may be up to 9 characters long. A constant may be up to 20 digits long, including the decimal point. iDynaMaSZ uses syntax similar to Fractint's formula style with an initialization section, followed by the main formula, and an optional bailout routine. Bailout is otherwise handled by the 'Bailout' variable in the Parameters window. Comments may be entered on the same line with a preceding ';'. A ':' terminates the initialization section. Multiple phrases may be entered in the main formula or initialization sections on the same line by using the terminator ',' between phrases. For a complete list of variables, operators and functions recognized by the parser, see Parser Information.

The Title text box is used with the hot key 'T' to annotate a picture with text. Use the Edit/Text command to change font, text color or format text into multiple lines. Text in this box is not saved in a picture's data file, but once entered the same text can be used over and over for different pictures. Useful for adding copyright/author info to batches of pictures. Since the same title text may be used many times, it is shared among views and saved in the file "prefs.txt" in iDynaMaSZ's startup directory.

Click on the Okay button to use the formulas currently displayed in the window, or Cancel to exit the window without making any changes. Click on Draw to apply and draw a new formula, etc. without closing the Formula window.

3.3 Parameters

Parameters

This command opens the formatting parameters window for the 3-D generator.

User-definable constants are ca - ch.

Steps and Fine Tune are pitch adjustments that bear on the quality of the figure or model at the expense of lengthier calculations.

A larger bailout can increase the fullness of a matrix figure. Iterations controls the graininess and esthetics of the matrix object. Higher iterations are generally not needed for most quaternion images, but may be useful for increasing definition on loxodromic and cquat fractals.

Three rotate variables determine the 3-D angle of rotation.

Click on Draw to apply new parameters without closing the window. Click on Stop to halt the drawing. Click on the Okay button to use the parameters currently displayed in the window and redraw the figure, or Close to exit the window without making any further changes.

3.4 Size command

Size (Edit menu)

This allows you to set the drawing area for a picture, independent of the screen size. It also shows which size is currently in use. The aspect for the drawing is based on the ratio of a drawing window's width to vertical height. The size of an image can have a standard 4/3 or 1/1 aspect from 240X180 to 800X800, or a custom aspect set by the W(idth) and H(eight) boxes. Larger bitmap sizes can be created using the Image/Plot to File command

3.5 Palette command

Palette command (Edit menu)

Use the palette editor to modify the palette(s) in use. If the image has been drawn recently, changes to the palette will show up immediately (colors are mapped to the active image.) Otherwise the modified palette will take effect when you close the palette dialog and redraw the image.

There are copy and spread options to smooth or customize the active image palette in iDynaMaSZ.

Colors are shown in 8 groups of 32 colors, or 256 color indexes total. Colors between the indexes are interpolated so there are actually 256X255 or 65000+ possible colors in the iDynaMaSZ palette.

Use the RGB-slider controls to edit any color in the palette. Select Copy to copy any color to another spot in the palette. Select Spread to define a smooth spread of colors from the current spot to another spot in the palette. Copy and Spread take effect immediately when you select another spot with the mouse button. You can cancel the operation with the Cancel button. In iDynaMaSZ, colors do not cycle smoothly when you adjust the RGB/HSV sliders. This would be too slow with true color. The Map button is used to map color changes to an image after you are done adjusting the sliders.

Use Reset to reset the colors of the palette in use, to where it was before it was modified.

Use Reverse to reverse the order of the colors in the palette, excluding color index 1, the background color.

Use Neg to create a palette that is the complement of the current palette, excluding color index 1.

Use SRG to switch the red and green components of all palette colors except color index 1. Use SRB to switch the red and blue components of all palette colors except color index 1. You can use these buttons to form eight different palettes by repeatedly switching red, green and blue components.

Use the Random palette button to randomize the current palette.

Note: unless you click on Reset before exiting the editor, changes are permanent to the palette edited, no matter which way you close the editor (Okay button or Close box.)

3.5.1 Reverse button

Reverse button

Use Reverse to reverse the order of the colors in the palette. This affects only those colors in the start-color to end-color range. This is useful for reversing divide-by-eight palettes, etc., for orbit-trap pictures that require a reversed palette.

3.5.2 Neg Button

Neg button

Use Neg to create a palette that is the complement of the current palette.

3.5.3 Map Button

Map button

In iDynaMaSZ, colors do not cycle smoothly when you adjust the RGB/HSV sliders. This would be too slow with true color. The Map button is used to map color changes to an image after you are done adjusting the sliders.

3.5.4 H/R Button

H/R button

Change from HSV to RGB mode or back. In the HSV mode, color spreads are based on HSV values instead of RGB values, which in some cases results in brighter color spreads.

3.5.5 Spread Button

Spread button

Select Spread to define a smooth spread of colors from the current spot to another spot in the palette.

3.5.6 Copy Button

Copy button

Select Copy to copy any color to another spot in the palette.

3.5.7 SRG Button

SRG button

Use SRG to switch the red and green components of all palette colors. This is for RGB mode only.

3.5.8 SRB Button

SRB button

Use SRG to switch the red and blue components of all palette colors. This is for RGB mode only.

3.5.9 Okay Button

Okay button

Click on Okay to exit the palette editor, applying unmapped color changes to picture (if color-cycling is enabled.)

3.5.10 Reset Button

Reset button

Use Reset to reset the colors of the palette in use, to where it was before it was cycled or modified. Note: if you change palettes with one of the function keys, any modifications to a previous palette are unaffected by the Reset button.

3.5.11 Cancel Button

Cancel button

You can cancel a copy or spread operation with the Cancel button.

3.5.12 Red Slider

Red slider

Use the RGB/HSV-slider controls to edit any color in the palette.

3.5.13 Green Slider

Green slider

Use the RGB/HSV-slider controls to edit any color in the palette.

3.5.14 Blue Slider

Blue slider

Use the RGB/HSV-slider controls to edit any color in the palette.

3.5.15 Red edit box

Red edit box

Shows red/hue value of selected color index.

3.5.16 Green edit box

Green edit box

Shows green/saturation value of selected color index.

3.5.17 Blue edit box

Blue edit box

Shows blue/value magnitude of selected color index.

3.5.18 Random Palette Button

Random palette button

Use to create a random palette. Fast way to define palettes.

3.6 Lighting

Lighting Window

The LightPoint variables (lightx thru lightz) determine the direction of the light source used in the ray-tracing algorithm. The ViewPoint represents the angle, at which the object is ray-traced, which can affect Phong highlights greatly. This has no effect on the camera view.

The lighting variables Shininess, Highlight, Gamma, Diffuse and Ambient are used to adjust ambient light and highlights. The ranges for these variables appear beside their label. Decreasing the Shininess value increases light reflected by the quaternion and the apparent sheen on the quaternion's surface. The Ambient value controls the amount of ambient light that illuminates the quaternion. The Highlight value increases or decreases the specular (Phong) highlighting, while the Gamma value increases or decreases the intensity of the light source's illumination. Once a plot is drawn (in the current session), the lighting variables and light point can be changed without redrawing the image.

Click the Apply button to redisplay a plot after changing the lighting variables or light point. Click the Okay button to close the lighting window, and apply new settings, if the variables were modified. Click on Close to close the lighting window, keeping the current settings.

4 Image Menu

Image menu commands

The Image menu offers the following commands:

Draw	Draw the figure.
Scan	Scan Mandelbrot border for quaternion Julia set.
Zoom	Zoom into or out of current image.
Plot to File	Plot image to png file.
Reset Ranges	Reset Z ranges.

4.1 Draw command

Draw command (Image menu)

Use this command to draw or redraw the image for the current fractal variables. Clicking inside the draw window with the left-mouse button stops all plotting.

4.2 Scan command

Scan (Image menu)

Enabled when the Type is Mandelbrot. After executing this command, a small quaternion Julia set is generated in the left upper corner, for the current matrix constants and formula, and each time you click in the draw window. Use the Mandelbrot border as a guide to finding Julia sets. Generally if you click too far outside the Mandebrot borders the space occupied by the Julia quaternion set vanishes. Most interesting are the Julia sets found right on the M-border or just outside it.

Quaternion math is used where possible if the Type is set to Quaternion; else hypercomplex math is used for the Hypernion type, etc.

Once you find an interesting quaternion set press the space bar and a new window is opened that sets the fractal parameters to those in the exploratory qjulia window. (The parameters in the scan window revert to their original Mandelbrot settings.)

Click on the status_bar area of the draw window to exit this command without generating a new Julia set. (Parameters and bitmap revert to the previous state of the image.)

4.3 Zoom command

Zoom (Image menu)

Turns on zoom mode, so that detail of the current plot may be magnified or centered. Alternatively, just click inside any drawing window, move the mouse, and the zoom box will appear. Using the mouse, move the zoom box over the portion of the plot you wish to magnify. Hold the left mouse button (or left arrow key) to shrink the box or the right button (or right arrow key) to enlarge it. You may zoom in by defining a box inside the current drawing area. You zoom out by drawing a box outside the current drawing area. Use the up arrow key to enlarge the zoom box X4 for quickly zooming outward or the down arrow key to shrink the zoom box by 4. You start a zoom by pressing the space bar. The program opens a new window and begins a new plot at the zoom coordinates. You abort a zoom by pressing the esc key (shift-esc with OS X 10.4) or click inside the status bar (bottom of draw window.)

4.4 Plot to file

Plot to File (Image menu)

This allows you to plot a large bitmap directly to a .png file without the added system requirements of keeping the whole bitmap in memory. The target bitmapWidth can be 1024 to 14400. Target Height is set by the drawing window aspect, or Target_Height=draw_height/draw_width*Target_Width. Click on Okay to set the target file name and start a new plot to file, or Cancel to exit the window without plotting.

4.5 Reset Ranges

Reset Ranges (Image menu)

The Reset Ranges command resets the real Z and imaginary Z ranges of an image to fit the current drawing window's aspect. This is necessary when changing image size and the drawing window aspect changes also, since a different aspect will otherwise stretch or condense a figure. After reseting the ranges you may need to recenter or rezoom a figure.

5 Type Menu

Type menu commands

The Type menu offers the following commands:

Mandelbrot0	Mandelbrot set (orbit starts at zero.)
MandelbrotP	Mandelbrot set (orbit starts at pixel.)
Julia	Julia set.
3X3 Matrix	Set matrix dimensions to 3 by 3.
4X4 Matrix	Set matrix dimensions to 4 by 4.
5X5 Matrix	Set matrix dimensions to 5 by 5.
6X6 Matrix	Set matrix dimensions to 6 by 6.
7X7 Matrix	Set matrix dimensions to 7 by 7.
8X8 Matrix	Set matrix dimensions to 8 by 8.

5.1 Mandelbrot0

Mandelbrot0 (Type menu)

Mandelbrots base their mapping on varying inputs of complex C, which corresponds to the min/max values set in the Parameters window. With Mandelbrot0, the initial value of Z is set to zero.

5.2 MandelbrotP

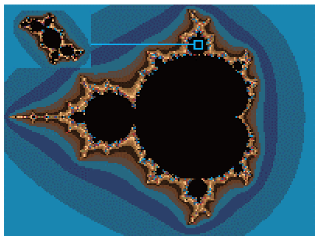
MandelbrotP (Type menu)

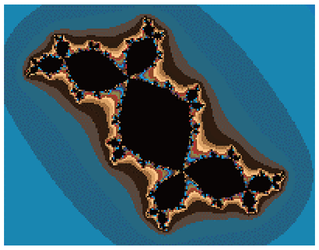
Mandelbrots base their mapping on varying inputs of complex C, which corresponds to the min/max values set in the Parameters window. With MandelbrotP, the initial value of Z is set to the value of the pixel being iterated. Usually used with cubic Mandelbrot formulas.

5.3 Julia

Julia (Type menu)

Julia sets normally have a fixed complex C, with varying inputs of Z, which corresponds to the min/max values set in the Parameters window.


Julia from Mandelbrot


Julia set

5.4 3X3 Matrix

3X3 Matrix (Type menu)

Set the size of th matrix to 3X3. Both Latin square and Sign matrices are resized to the new dimensions. The symbols in the Latin square are set to approximate a quaternion matrix (all zeros in diagonal.) The values of the Sign matrix are randomly set to -1 or 1. Larger matrices take proportionately longer to iterate.

5.5 4X4 Matrix

4X4 Matrix (Type menu)

Set the size of th matrix to 4X4. Both Latin square and Sign matrices are resized to the new dimensions. The symbols in the Latin square are set to approximate a quaternion matrix (all zeros in diagonal.) The values of the Sign matrix are randomly set to -1 or 1. Larger matrices take proportionately longer to iterate.

5.6 5X5 Matrix

5X5 Matrix (Type menu)

Set the size of th matrix to 5X5. Both Latin square and Sign matrices are resized to the new dimensions. The symbols in the Latin square are set to approximate a quaternion matrix (all zeros in diagonal.) The values of the Sign matrix are randomly set to -1 or 1. Larger matrices take proportionately longer to iterate.

5.7 6X6 Matrix

6X6 Matrix (Type menu)

Set the size of th matrix to 6X6. Both Latin square and Sign matrices are resized to the new dimensions. The symbols in the Latin square are set to approximate a quaternion matrix (all zeros in diagonal.) The values of the Sign matrix are randomly set to -1 or 1. Larger matrices take proportionately longer to iterate.

5.8 7X7 Matrix

7X7 Matrix (Type menu)

Set the size of th matrix to 7X7. Both Latin square and Sign matrices are resized to the new dimensions. The symbols in the Latin square are set to approximate a quaternion matrix (all zeros in diagonal.) The values of the Sign matrix are randomly set to -1 or 1. Larger matrices take proportionately longer to iterate.

5.9 8X8 Matrix

8X8 Matrix (Type menu)

Set the size of th matrix to 8X8. Both Latin square and Sign matrices are resized to the new dimensions. The symbols in the Latin square are set to approximate a quaternion matrix (all zeros in diagonal.) The values of the Sign matrix are randomly set to -1 or 1. Larger matrices take proportionately longer to iterate.

6 Demo menu

Demo menu commands

The Demo menu offers the following commands, which illustrate various features of iDynaMaSZ:

Random Matrix	Generate random matrix image.
Random Composite	Generate random composite matrix image.
Random Setup	Options for generating random fractals.

6.1 Random Matrix

Random Matrix (Demo menu)

A random matrix image is generated. Matrices are randomized according to the Random Setup.

6.2 Random Composite

Random Composite (Demo menu)

A random 3-D Julia fractal is generated using one of the composite Types. Two formulas are selected at random and mixed using one of Types 2-8.

6.3 Random Setup

Random Setup (Demo menu)

Here you set options for random-generated images.

There are radio boxes that allow you to customize how random variables are processed to create new 3-D fractals:

Formula -- (default on) check to randomize built-in formula used
Z-Space -- (default on) check to set default z-space
Constants -- (default on) check to randomize the complex constants cj-cK
Sign Matrix -- (default on check to randomize the sign matrix
Latin Square-- (default on) check to randomize the latin square
Column -- (default off) check to randomize the column used for bailout
Custom Formula -- (default off) check to generate a random formula

7 Video Menu

The Video Menu offers the following commands:

Write Movie	Write frames to QuickTime movie file.
Add Frame	Add current image to frame buffer.
Edit Frames	Edit frame buffer.
Load Frames [DFRM]	Load frame buffer.
Save Frames [DFRM]	Save frame buffer.

7.1 Write Movie

With this command the video frame buffer is written to a QuickTime movie. First you choose the width of the video, up to 2048 (height is determined by the current image aspect or height=width*aspect.) A file requester is then opened to choose the name and location of the movie, then the frames are written sequentially in the mov format. Variables are scaled between buffer frames to create the illusion of motion or morphing. The movie is written in the highest quality possible, so there are minimal compression artifacts. (The movie can be compressed later in an external QuickTime program to reduce file size, if necessary.) Most variables that have a numerical value can be scaled between frames.

7.2 Add Frame

iDynaMaSZ uses a frame buffer to compose an animation. You add key frames to the buffer with this command. Each key frame is identical to the active image. Change variables between key frames to create the illusion of motion or morphing. You can edit the frames with the frame editor.

7.3 Edit Frames

Here you can edit any frames in the video buffer. Buttons are supplied to access all the image parameter editors, such as Function, Lighting and Parameters. The Move button allows you to move a frame from one spot in the buffer to another. You can change the frame image being edited by using the Frame slider or Edit box. After changing frames, use the Preview button to display the current frame being edited. In most cases the frame preview is automatically updated when you change an image parameter using the editor or type buttons. The Delete button allows you to delete all but two of the frames, the minimum number of frames to create a movie. (If you want to delete all the frames, use the Video/Reset Frames command.)

7.4 Reset Frames

Delete the current frame buffer. The number of video frames is reset to zero.

7.5 Load Frames [DFRM]

Load a frame buffer that has been previously saved by iDynaMaSZ. The buffer replaces any existing frame buffer.

7.6 Save Frames [DFRM]

This command saves the current frame buffer in a [qfrm] file. A file requester is opened that allows you to choose the location and name of the frame library. The frame buffer files can also be used as image libraries, similar to Fractint's par and frm formats. The frames contain all the information to reproduce an image at any supported size.

8 Help Topics

Help Topics

Getting Started	Tutorial for new users of iDynaMaSZ.
Built-in Formulas	Quick reference to iDynaMaSZ's built-in formulas.
Parser	Quick reference to iDynaMaSZ's parser variables and functions.
Bibliography	Sources for fractal information and complex numbers.
About iDynaMaSZ	Displays the version number and author info for this application.

8.1 Getting Started

Getting Started

iDynaMaSZ uses "whole" matrix algebra for every function that is iterated, which differs from the "column" shortcut used in the original DynaMaSZ. This allows the basic procedures to be extended beyond the four dimensions normally associated with quaternions. Up to eight dimensions are supported by iDynaMaSZ. Unless you have a super fast computer, or are prepared to wait a long time for the plots to finish, it's best to start with one of the lower dimensions <6. The dimension is selected in the Type menu.

For information on the basis of matrix fractals, check out Godwin Vickers **Introduction** on matrix algebra.

8.2 Parser

Parser Information

Functions (capital letters are optional, and parenthesis are necessary around complex expressions)

The following information takes the form "standard function" ---"form used by iDynMaSZ to represent standard function".

sine z --- sin(z) or SIN(Z)   ; where Z can be any complex expression
hyperbolic sine z --- sinh(z) or SINH(Z)
cosine z --- cos(z) or COS(Z)
hyperbolic cosine z --- cosh(z) or COSH(Z)
tangent z --- tan(z) or TAN(Z)
hyperbolic tangent z --- tanh(z) or TANH(Z)
cotangent z --- cotan(z) or COTAN(Z)
arctangent z -- atan(z) or ATAN(Z)
arcsine z -- asin(z) or ASIN(Z)
arccosine z -- acos(z) or ACOS(Z)
hyperbolic arctangent z -- atanh(z) or ATANH(Z)
hyperbolic arcsine z -- asinh(z) or ASINH(Z)
hyperbolic arccosine z -- acosh(z) or ACOSH(Z) -- implemented as asinh(z)/atanh(z)
vers z -- vers(z) or VERS(z) -- 1- cos(z)
covers z -- covers(z) or COVERS(z) -- 1- sin(z)
exp(z) or EXP(z) -- the exponential function
log(z) or LOG(Z) -- the logarithmic function -- implemented as ln(z+1)
Bessel -- bessel(z) or BESSEL(Z) -- first order Bessel function
Laguerre -- lag(z) or LAG(Z) -- 3rd order Laguerre function
Gaussian -- gauss(z) or GAUSS(Z) -- Gaussian probability function
absolute value of z --- abs(z) or ABS(Z) -- sqrt(x*x + y*y +z*z+w*w)
z squared --- sqr(z) or SQR(Z)
z cubed -- cube(z) or CUBE(Z)
square root of z --- sqrt(z) or SQRT(Z) -- implemented via Newton's method
modulus of z --- mod(z) or MOD(Z) or |z| -- (x*x + y*y +z*z+w*w)
reciprocal of z -- rep(z) or REP(Z) -- inverts the
transpose -- trans(z) or TRANS(Z) -- transposes the matrix
trace -- trace(z) or TRACE(Z) -- sums the diagonal elements of the matrix
rotate columns -- rotc(z) or ROTC(Z) -- rotate matrix columns in clockwise direction
rotate rows -- rotr(z) or ROTR(Z) -- rotate matrix rows in clockwise direction
reverse rotate columns -- rotc(z) or ROTC(Z) -- rotate matrix columns in counter-clockwise direction
reverse rotate rows -- rotr(z) or ROTR(Z) -- rotate matrix rows in counter-clockwise direction
swap columns -- swapc(z) or SWAPC(Z) -- swap outer and inner matrix columns
swap rows -- swapr(z) or SWAPR(Z) -- swap outer and inner matrix rows

if/then/endif – if(argument), then (phrase) endif -- if argument is true then do phrase else skip phrase('then' tag is optional, but a comma should follow argument or put 'if(argument)' on separate line)

if/then/else/endif - if(argument), then (phrase) else (phrase) endif -- if argument is true then do phrase else skip phrase and do alternate phrase('then' tag is optional, but a comma should follow argument or put 'if(argument)' on separate line)

Notes:
If/then/endif and if/then/else/endif loops can be nested only when endifs follow each other at the end of the loops. For example: if(argument) if(argument) then (phrase) endif endif.

All transcendental and trig functions are implemented as power series. In the case of the log and inverse trig functions, since their domain is limited to -1<z<1 there is no guarantee that they will work correctly in all formulas.

Math operators

+ --- addition
- --- subtraction
* --- multiplication
/ --- division
^ --- power function
< --- less than
<= --- less than or equal to
> --- greater than
>= --- greater than or equal to
!= --- not equal to
== --- equal to
|| --- logical or (if arg1 is TRUE(1) or arg2 is TRUE)
&& --- logical and (if arg1 is TRUE and arg2 is TRUE)

The power operator works with whole number exponents only. If you use a matrix variable as an exponent, the integral value of the variable's 'a' element (row 0, col 0) is used.

Constants and variables

complex matrix constant --- c or C, read/write.
e --- e or E -- 1e^1 -- 2.71828, read/write.
i --- i or I -- square root of -1,read/write.
p --- p# or P# -- the cx constant, read-only.
pi --- pi or PI -- 3.14159, read/write.
q --- q# or Q# -- the cy constant, read-only
z --- z or Z -- function value at any stage of the iteration process, read/write.
nn -- any numerical constant up to 20 digits, including decimal point

Note: the parser treats all numerical constants the same as an identity matrix. This allows it to translate '1' correctly when entering reciprocal terms, such as 1/z^2. However, if you want to experiment with constants as regular matrices, attach the postfix 'M' to the constant term. For instance, '1M' would be evaluated as a regular matrix with the same algebraic characteristics as the LS matrix defined.

8.3 Built-in Formulas

Built-in Formulas (enter the following prefix into the Fun #1 or Fun #2 edit boxes)

1	p0; z^2+c
2	p1; (1-z)*zc
3	p2; z(z-1/z)+c
4	p3; cz^2-c
5	p4; z^2+cz^2+c
6	p5; z^3+c
7	p6; ((z^2)*(2z+c))^2+c
8	p7; z^2+j+kzn
9	p8; cz^3+c
10	p9; z^2-cz^3+c
11	r0; z^3-z+c
12	r1; z^2+z^3+c
13	r2; z^3-zn+c
14	r3; z^4-z^2+c
15	r4; (z^2-c)(z+1)
16	r5; (z+j)(z+k)(z^2+1)+c
17	r6; (z+j)(z^2+z+k)+c
18	r7; (z-1)(z^2+z+c)
19	r8; z^5+c
20	r9; z^6+c
21	e0; z(z^5+c)
22	e1; z(z^6+c)
23	e2; z^7+c
24	e3; z(z^7+c)
25	e4; z^7-z^5+z^3-z+c
26	e5; z^3+j+kzn
27	e6; cz^2+zn+c
28	e7; z^4-c^4
29	e8; (z-c)(z+1)(z-1)+c
30	e9; z^2+c^2
31	s0; z^3+c/z+c
32	s1; 1/z^2+c
33	s2; z+z^3/c+c
34	s3; z^3/c+c
35	s4; c/(z*z)+c
36	s5; 1/(z*z)-z+c
37	s6; (1+c)/(z*z)-z
38	s7; c/(z*z*z)+c
39	s8; 1/(z*z*z)-z+c
40	s9; 1/z^2-cz-c
41	"t0; 1/z^2+exp(z)-c"
42	"t1; c*cos(z)+c"
43	"t2; z^3+sin(c)"

44	"t3; 1/z^3+sinh(z)-c"
45	"t4; z^3+tanz+c"
46	"t5; z^3+coshz+c"
47	"t6; z^2-bessel(z)+c"
48	"t7; lag(z)+c --- 3rd order Laguerre function
49	"t8; z^3+Gauss(z)+c -- Gaussian probability function
50	"t9; z^2+z*arcsin(z)-c
51	"a0; (1-z)^2+c

Note: in all built-in fomulas the '1' term is set to the identity matrix regardless of the LS matrix definition.

8.4 Bibliography

Bibliography

Complex Mathematics
Churchill, Ruel.V. and Brown, James Ward: "Complex Variables and Applications", Fifth Edition, McGraw-Hill Publishing Company, New York, 1990.

Korn, Granino A. and Korn, Theresa M.: "Manual of Mathematics, McGraw-Hill Publishing Company, New York, 1967.

Fractal Theory
Barnsley, Michael: "Fractals Everywhere", Academic Press, Inc., 1988.

Devaney, Robert L.: "Chaos, Fractals, and Dynamics", Addison-Westley Publishing Company, Menlo Park, California, 1990.

Mandelbrot, Benoit B.: "The Fractal Geometry of Nature", W.H.Freeman and Company, New York, 1983.

Peitgen, H.-O. and Richter, P.H.: "The Beauty of Fractals", Springer-Verlag, Berlin Heidelberg, 1986.

Formulas and Algorithms
Burington, Richard Stevens: "Handbook of Mathematical Tables and Formulas", McGraw-Hill Publishing Company, New York, 1973.

Kellison, Stephen G.: "Fundamentals of Numerical Analysis", Richard D. Irwin,Inc. Homewood, Illinois, 1975.

Peitgen, Heinz-Otto and Saupe, Deitmar: "The Science of Fractal Images", Springer-Verlag, New York, 1988.

Pickover, Clifford A.: "Computers, Pattern, Chaos and Beauty", St. Martin's Press, New York, 1990.

Stevens, Roger T.: "Fractal Programming in C", M&T Publishing, Inc.,Redwood City, California, 1989.

Wegner, Tim, Tyler,Bert, Peterson, Mark and Branderhorst, Pieter: "Fractals for Windows", Waite Group Press, Corte Madera, CA, 1992.

Wegner, Tim and Tyler, Bert: "Fractal Creations", Second Edition, Waite Group Press, Corte Madera, CA, 1993.

Whipkey, Kenneth L. and Whipkey, Mary Nell: "The Power of Calculus", John Wiley & Sons, New York, 1986.

8.5 About iDynaMaSZ

About iDynaMaSZ

>>>>> iDynaMaSZ™ v1.01 ©2005-2008 by Terry W. Gintz

iDynaMaSZ requires a true-color video adapter for best results.

Acknowledgements: Special thanks to Frode Gill for his quaternion and ray-tracing algorithms, to Dirk Meyer for his Phong-shading algorithm, and to David Makin for sharing his ideas on quaternion colorings and 3-D insights. Thanks also to Francois Guibert for his Perlin noise example, and to Kerry Mitchell and Mark Townsend for allowing me to incorporate their coloring methods from Ultra Fractal. Kudos to Godwin Vickers for his remarkable work on full-screen quaternion videos, his gracious support and algorithmic input to my programming, and his diligent efforts to expand the realms of 3-D fractal visualization.

For a short history of this program, see Chronology.

8.5.1 Chronology

Chronology

History of this program:

In September 1989, I first had the idea for a fractal program that allowed plotting all complex functions and formulas while attending a course on College Algebra at Lane College in Eugene, Oregon. In November 1989, ZPlot 1.0 was done. This Amiga program supported up to 32 colors, 640X400 resolution, and included about 30 built-in formulas and a simple formula parser.

May 1990 -- ZPlot 1.3d -- added 3-D projections for all formulas in the form of height fields.

May 1991 -- ZPlot 2.0 -- first 236-color version of ZPlot for Windows 3.0.

May 1995 -- ZPlot 3.1 -- ZPlot for Windows 3.1 -- 60 built-in formulas. Added hypercomplex support for most built-in formulas.

May 1997 -- ZPlot 24.02 -- first true color version of ZPlot -- 91 built-in formulas. Included support for 3-D quaternion plots, Fractint par/frm files, Steve Ferguson's filters, anti-aliasing and Paul Carlson's orbit-trap routines.

June 1997 -- ZPlot 24.03 -- added Earl Hinrichs' Torus method.

July 1997 -- ZPlot 24.08 -- added HSV filtering.

December 1997 -- Fractal Elite 1.14 -- 100 built-in formulas; added avi and midi support.

March 1998 --Split Fractal Elite into two programs, Dreamer and Medusa (multimedia.)

April 1998 -- Dofo 1.0 -- supports new Ferguson/Gintz plug-in spec.

June 1998 -- Dofo-Zon -- redesigned multi-window interface by Steve Ferguson, and includes Steve's 2-D coloring methods.

August 1998 --Dofo-Zon Elite -- combination of Fractal Elite and Dofo-Zon

October 1998 -- Dofo-Zon Elite v1.07 -- added orbital fractals and IFS slide show.

November 1998 -- Dofo-Zon Elite v1.08 -- added lsystems.

April 1999 -- Split Dofo-Zon Elite into two programs: Fractal Zplot using built-in formulas and rendering methods, and Dofo-Zon to support only plug-in formulas and rendering methods.

May 1999 -- Fractal Zplot 1.18 -- added Phong highlights, color-formula mapping and random fractal methods.

June 1999 -- completed Fractal ViZion -- first version with automatic selection of variables/options for all fractal types.

July 1999 -- Fractal Zplot 1.19 -- added cubic Mandelbrot support to quaternion option; first pc fractal program to render true 3-D Mandelbrots.

September 2000 -- Fractal Zplot 1.22 -- added support for full-screen AVI video, and extended quaternion design options

October 2000 -- QuaSZ (Quaternion System Z) 1.00 -- stand alone quaternion/hypernion/cubic Mandelbrot generator

November 2000 -- Added octonion fractals to QuaSZ 1.01.

March 2001 -- Cubics 1.0 -- my first totally-3-D fractal generator.

May 2001 -- QuaSZ 1.03 -- added Perlin noise and improved texture mapping so texture tracks with animations.

July 2001 -- QuaSZ 1.05 -- improved performance by converting many often-used dialogs to non-modal type. Added generalized coloring and external rendering library support.

October 2001 -- FraSZle 1.0, QuaSZ formula and algebra compatible version of Fractal Zplot

November 2001 -- DynaMaSZ 1.0, the world's first Dynamic Matrix Systems fractal generator

January 2002 -- MiSZle 1.1 -- generalized fractal generator with matrix algebra extensions

May 2002 -- DynaMaSZ SE 1.04 (unreleased version)-- scientific edition of DynaMaSZ, includes support for user-variable matrix dimensions (3X3 to 12X12)

January 2003 -- Pod ME 1.0 -- first stand-alone 3-D loxodromic generator, Hydra 1.0 -- first 3-D generator with user-defined matrix types and Fractal Projector a Fractal ViZion-like version of DynaMaSZ SE limited to 3X3 matrices

May 2003 -- QuaSZ 3.052 -- added genetic-style function type and increased built-in formulas to 180. Other additions since July 2001: generalized coloring, support for external coloring and formula libraries, and Thomas Kroner's loxodromic functions.

May 2003 -- FraSZle and Fractal Zplot 3.052 -- added random 3D orbital fractals, new 3D export methods, upgraded most frequently-used dialogs to non-modal type and added genetic-style function type. FZ now based on FraSZle except for built-in formula list and Newton support.

Index