

Fractal!

version 1.4

A program for generating Fractal landscapes.

by
Ed Rotberg

Table of Contents

Introduction.....	3
The File Menu.....	4
Open.....	4
Save.....	4
Save As.....	5
Save PICT File.....	5
Save PICT File As.....	5
Save QT Compressed PICT.....	5
Open Grid File.....	6
Close Grid File.....	7
Script.....	7
Export.....	10
Quit.....	10
The Edit Menu.....	11
Defaults.....	11
The Subdivides Menu.....	13
The Appearance Menu.....	15
Fixed Seed.....	15
Lines.....	16
Polygons.....	16
Noise Type.....	16
Alternate View.....	17
Modify Parameters.....	18
Levels.....	19
Color.....	22
Fog Color.....	22
Water Color.....	23
Mountain Color Range.....	23
Sky Color Range.....	24
Illumination.....	24
Wave Parameters.....	26
Window.....	27
The Rendering Menu.....	30
Render.....	30
Allow Backgrounding.....	30
Auto Render.....	31
Progress Bar.....	31
Other Features.....	33
Theory of Operation.....	34
Acknowledgments.....	38
Disclaimers and other stuff.....	39

Fractal!

Introduction

Fractal! started out as my own personal attempt to learn more about fractal landscapes. I like to think of it as a compensation for my aversion to real gardening! You, however, can use this program to amaze your friends and yourself. With a little bit of tinkering (and a lot of memory and compute time!) you will be able to create some very natural looking landscapes. Got a few spare machine cycles? Put together a portfolio of some of the most beautiful places you've never been!

This program is designed to be an easy-to-use, play and experiment kind of program. You don't need to understand a lot of fractals or higher math in order to use the program. Just play with all of the features. Try changing the various parameters one at a time to see what each does. Most of them can only be set within constrained limits that keep the program from crashing if not always producing desirable results. The main idea here is to feel around and try out different things! If I've done a good job, you'll hardly need this manual at all. After you've experimented with the program and gotten a feel for just what everything does, you can then turn to this manual for further elucidation. If you're interested in the design philosophy behind my implementation, I've included a brief section at the end of this manual for your possible enlightenment.

At this point I'll go through each of the program's menus, and explain each of the commands.

The File Menu

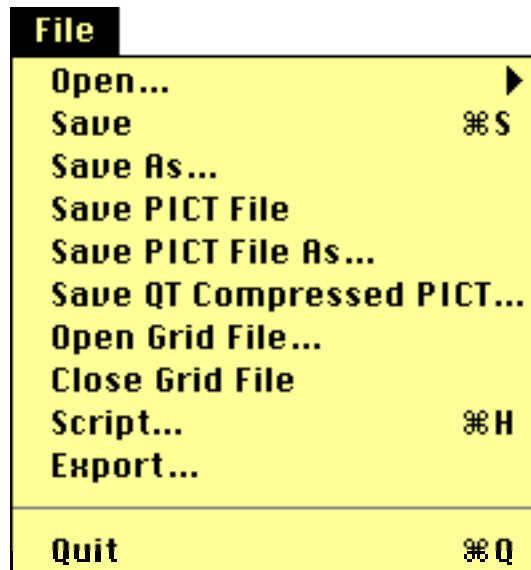


Figure 1

The File Menu provides commands for the input and output of files. Fractal! can input two proprietary types of data files (it's own parameter file and a pre-deformed grid file) and one script file (a standard Macintosh 'TEXT' file). It can output its own parameter file as well as standard 24-bit deep Macintosh 'PICT' files. 'PICT' files may also be saved in compressed QuickTime™ format if the QuickTime extension is present in the System. The file menu also provides the Quit command.

Open... (-O)

The Open command is used to open and read Fractal! parameter files (type 'FPRM'). These files contain all of the data parameters that can be set from the Subdivides and Appearance menus as well as the Auto Render and Progress Bar parameters (see the section on the Rendering Menu below) that can be set from the Rendering Menu. The parameter file contains all the data necessary to exactly recreate a 24-bit deep fractal landscape in just 278 bytes. If an unsaved parameter file is currently open, the user will be asked whether or not to save the current file first.

Save... (-S)

The Save command will cause the current set of parameters to be saved to the currently open parameter file (file type 'FPRM'). If no file is open, the command will proceed as the Save As... command (see below). If the current file is not "dirty", that is, it has not been changed since the file was originally opened or last saved, the command will do nothing. Note that simply requesting a render operation will cause the parameter set to become "dirty" if the Fixed Seed (see below) parameter is not set, as this would cause a new random number seed to be generated.

Save As...

This command will always present the user with the Standard PutFile Dialog, allowing the user to save the current parameter set to a named file.

Save PICT File

This command is used to save the current image to a standard Macintosh, 24-bit 'PICT' file. If the image has not yet been saved at least once, the command will proceed as in the Save PICT File As... command described below. There is no "dirty" concept for 'PICT' files and selecting this menu file will always cause the file to be written to disk.

Save PICT File As...

This command will present the user with a modified PutFile Dialog, allowing the user to save the current image file in standard Macintosh 'PICT' format (24-bit deep).

To save an image of a specific size, set the desired image size directly using the Window... command in the Appearance menu (see below). The fractal will then be rendered to the appropriate size and can be saved as a PICT file at that size.

Save QT Compressed PICT...

This item will only be enabled in the File Menu if the QuickTime extension is present in the System. Otherwise, it will appear as a dimmed (disabled) item. The user is first presented with the QuickTime Compression Settings Dialog (see Figure 2) allowing the choice of available compression techniques and quality level of the compressed image. After the Compression Settings have been selected, the user is presented with a StandardPutFile Dialog, allowing the user to specify the file name and location under which to save the current image file in QuickTime compressed 'PICT' format (24-bit deep). Once this dialog is dismissed, the image will be compressed to the specified file.

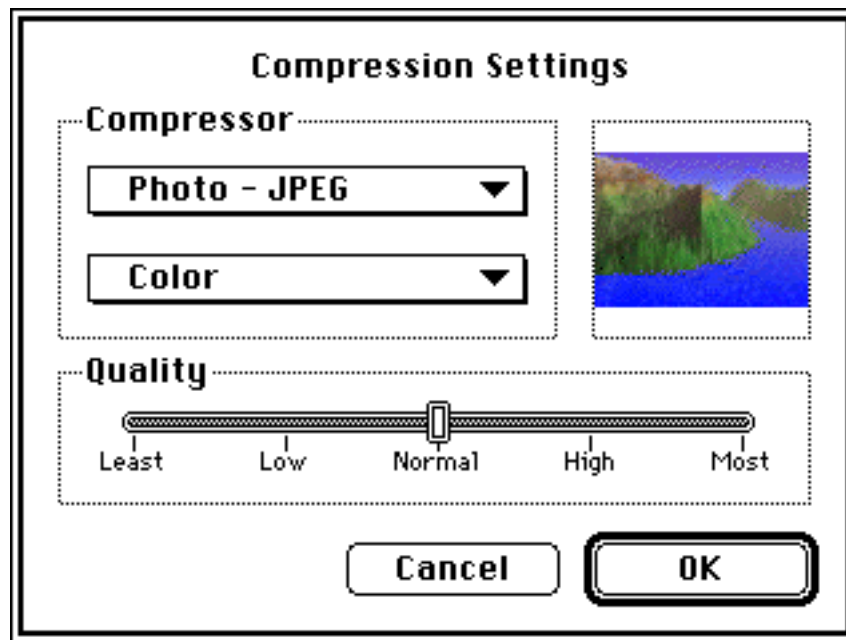


Figure 2

Note that for individual Fractal! images, the JPEG QuickTime compression codec will probably be the most useful choice. QuickTime compression allows you save PICT files in a fraction of the space required for the uncompressed file.

Open Grid File...

This command will allow the input of a pre-deformed array of Y-offsets on a square grid, enabling the program to “fractalize” and render a specified landscape shape.

Currently this feature only works with Carpenter Subdivides (see below)!! The format of this file is as follows. The first 16-bit word (short) specifies the size of the array to follow in terms of subdivides as explained in the section on the Theory of Operation. For example, if the first short is 3, then there will be $(2^3 + 1)^2$ or 81 short words following the 3, making up the remainder of the file. If the first short is 4, there will be 289 words following it, and so on. This array of words comprises the Y-offsets to the square X-Z grid of points with each group of $(2^n + 1)$ consecutive points describing one row of points along the X-axis from smallest X (left) to largest X (right). These rows are presented from largest Z (furthest away, or into the screen) to smallest Z (nearest the screen as seen from the viewing point along the positive Z axis going into the screen).

There is currently no program available for generating such a file, but I have one that I’m working on and may include it as part of this program or in the same package at a future date. Please contact me if you have any questions regarding this file format.

Once the file is open, the user can set the level of subsequent subdivision desired, as

well as the rest of the suite of parameters. The program will continue to use the original grid input as the basis of all subsequent images generated until the Grid File is closed. Note that if the Fixed Seed parameter is not set then the grid will be “fractalized” differently each time it is rendered. This may be a subtle or significant change depending of the rest of the parameters and the level to which the starting grid file is already subdivided. Fixing the seed will force a consistent “fractalization” effect.

Close Grid File

This command will close the currently open Grid File allowing the random number generator to dictate the shape of the landscape.

Script... (-H)

The Script... command allows the input of a script file which will allow a series of images to be generated and saved to disk as either a QuickTime movie or as a series of PICT files.

Upon issuing the command, a Standard GetFile dialog box is displayed requesting the name of the Script file which will control the animation. Once that file has been selected, a Standard PutFile dialog will appear requesting the name of the output file and the format to be output (see figure 3 below), either QuickTime movie or PICT files. If QuickTime is not present in the system, that choice will be disabled and the PICT files choice will be selected. Otherwise QuickTime will be the default choice. If PICT files are chosen, the filename entered in the dialog will have a 3 digit number appended to it that is incremented for each image saved. If QuickTime movie is chosen, the program will present the QuickTime compression settings dialog (see figure 4 below) allowing the user to select the desired compression technique.

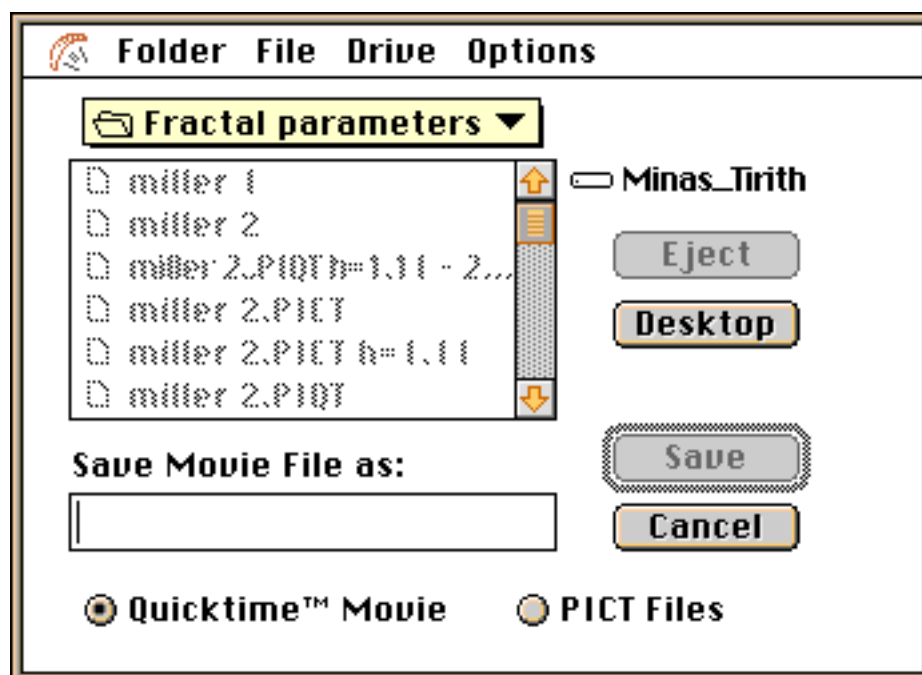


Figure 3

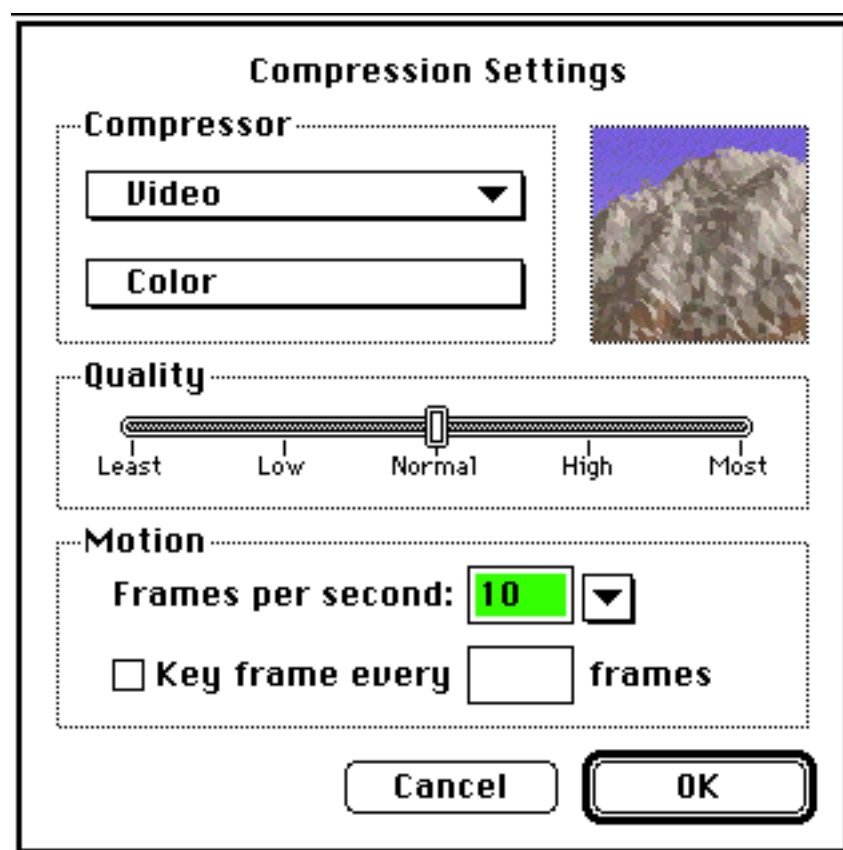


Figure 4

Below is a *very sketchy* description of what the script file looks like. It is vaguely modeled after the Super 3D key frame file format. The file type is TEXT. I recommend using a spread sheet or custom program to generate the series of parameter changes desired, as there is currently no support for algorithmic changes, although this may be added in a future update.

The following are the variables that may be changed by script. Each must start in column 1 and have any parameters it takes on its line followed by white space. Any unrecognized variable is ignored.

To request a new frame to be rendered, put an asterisk in column 1.

<u>Keyword</u>	<u>Data format</u>
brightness	(unsigned short integer)
fog_level	(double)
fog_color	(unsigned short integer triplet - RGB Color)
hwratio	(double)
h_factor	(double)
lower_transition	(short integer)
mountain_bottom	(unsigned short integer triplet - RGB Color)
mountain_top	(unsigned short integer triplet - RGB Color)
num_divides	(unsigned short integer)
sea_level	(short integer)
sky_top	(unsigned short integer triplet - RGB Color)
sky_bottom	(unsigned short integer triplet - RGB Color)
slush	(double)
snow_threshold	(unsigned short integer)
sun	(double triplet)
upper_transition	(short integer)
water_color	(unsigned short integer triplet - RGB Color)
yxlate	(long integer)
yoffset	(long integer)
zoffset	(long integer)
zoom	(double)

*

Note that the sun command is a unit vector for the normal of the light source's vector direction. This value is displayed in the control in the Illumination dialog. As a point of interest, I force whatever triple I read into a valid unit vector, in case the data entered is in fact not a unit vector. This should make it easier to set up a series of values.

Below is a sample section of a script file that starts a sunrise scenario. The only variables changed are sun and brightness.

```

sun    0.833 0.167 -0.250
brightness 16
*
sun    0.825 0.175 -0.247
brightness 17
*
sun    0.816 0.184 -0.245
brightness 18
*
sun    0.807 0.193 -0.242
brightness 19
*
sun    0.798 0.202 -0.239
brightness 18
*
sun    0.788 0.212 -0.237
brightness 21
*
sun    0.779 0.221 -0.234
brightness 22
*
```

Export...

The Export command allows the user to write a text file that describes all of the polygons that make up the fractal landscape. The Save File Dialog that is presented contains two radio buttons allowing this file to be formatted as either Super 3D™ text file or as DXF formatted text file.

Quit (-Q)

The Quit command allows the user to exit the program. If the current parameter set has not yet been saved, the user will be prompted to do so.

The Edit Menu

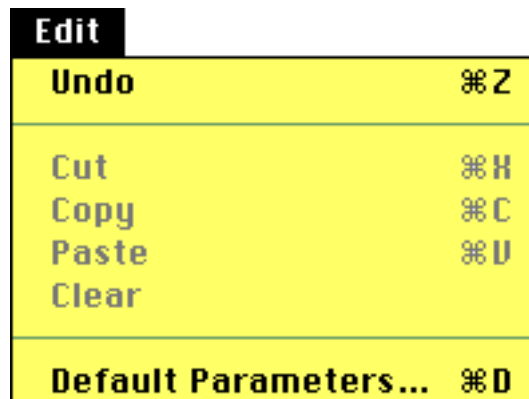


Figure 5

The only “standard” Edit command supported by Fractal! is Undo. This command is supported only one level deep (i.e. you can only undo the last change made with another command/dialog). If a number of changes were made within a single dialog (for example Modify Parameters) then using Undo after that dialog has been exited via OK will cause **all** of those changes to be undone. Additionally, if Fixed Seed is turned off (see discussion of Fixed Seed later in this document) then after initiating a single render, the seed used for that render can be restored by doing an Undo. This can allow the retrieval of a seed that has “gone by”. Once again, for an explanation of how to use this feature, see the discussion of Fixed Seed later in this document.

In addition to the usual and necessary commands, Undo, Cut, Copy, Paste and Clear, there is also one additional command. The Default Parameters... command (see Figure 5).

Default Parameters... (-D)

This command initiates a dialog (see Figure 6) that allows you to save the current parameter set as the default, restore the default parameter set, or restore the “factory” default parameter set — although just what factory is being referred to is beyond me! These options all work just as you would expect with a couple of caveats. First, the Fixed Seed parameter is always saved as **off**. That is, your default cannot be a fixed seed of a particular value. If you want to do this, just save a parameter file — that’s what they’re for! Likewise, the seed itself is not saved or restored. Note that restoring either the default parameter set or the factory defaults will not affect either the Fixed Seed parameter or the current seed value.

When you save the current set as the default set, the next time you boot Fractal!, it will be started with those parameters.



Figure 6

The Subdivides Menu

Subdivides	
✓ Miller Subdivides	
Carpenter Subdivides	
3	⌘3
4	⌘4
5	⌘5
✓6	⌘6
7	⌘7
8	⌘8
9	⌘9
10	⌘0
11	⌘1

Figure 7

The Subdivides Menu provides commands for selecting the subdivision algorithm to be used as well as the number of subdivisions for the next render operation.

There are two subdivision algorithms available as indicated in figure 7. Each of these is described in greater detail in the section of Theory of Operation. Basically, Carpenter Subdivides yields very dramatic terrains which are subject to “creasing” artifacts. Miller Subdivides yields more realistic terrains, which have no such “creasing” artifacts, and are therefore the default choice.

The user may select any of the available number of subdivides from the menu, but remember that although higher numbers will yield much nicer looking results, they require more RAM and more compute time. It is very expedient to choose a coarse level of subdivision (4-6 depending upon the speed of your particular Macintosh) in order to “look for” a nice fractal or try a different set of color parameters. Once you’ve found something that looks promising, you can then up the number of subdivisions and kick off a new fractal before you go to bed!

It is worth noting a few things at this point. First of all, for you ResEdit hackers, the Subdivides menu is handled via an algorithm to actually set the parameter for the program. This means that you can add menu items beyond 11 if you have the time and the memory to render an image beyond that level. While I haven’t tried it personally, it would take somewhere around 35 Megabytes of RAM and 12 hours on a Mac IIfx (with backgrounding turned off) to render at 12 subdivisions (I haven’t even thought about 13 levels beyond the fact that it’s an unlucky number!). Fractal! runs fine under VM, and in fact, it is reasonably efficient with it. Because of the method used for rendering, there is minimal disk thrashing once the subdivision

phase is complete (the subdivision phase typically takes less than 1/2 percent of the entire rendering time when rendering polygons). Expect the subdivision phase to thrash a lot!

The Appearance Menu

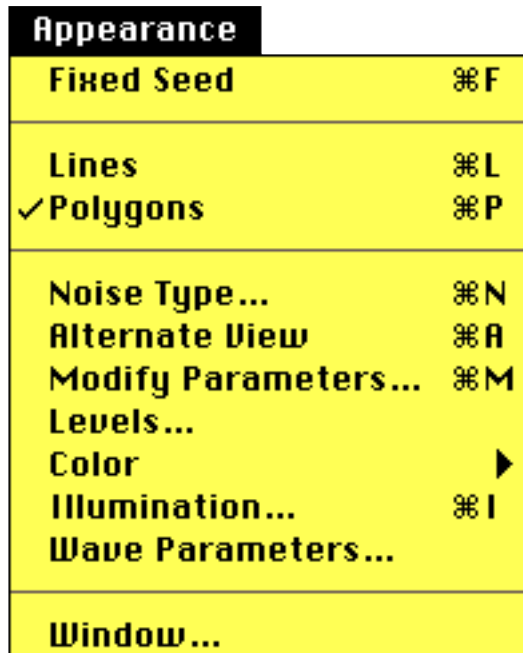


Figure 8

This is the menu with which you will probably spend the majority of your time, tweaking parameters and cursing the atrocious human interface (or lack thereof)! Each of these commands affect the appearance of the final image in some way or another.

Fixed Seed (-F)

If you've found a fractal shape that you like, you can "lock it in" by fixing the seed and then playing with the other parameters in order to generate the final image that you're looking for. You will also note that the seed itself is accessible and changeable from the Modify Parameters menu though you should rarely need to access it from there.

If the seed is fixed, there will be a check mark next to the menu item. When the item is unchecked, each time that a new render is initiated a new random number seed is used, generating a unique fractal. Think of it like rolling the dice!

If the seed is not fixed and you accidentally generate a new seed, the preceding seed can be retrieved by the following method. First, select the Undo command before you do anything else. Now, immediately fix the seed! This will bring back the seed used for the image prior to the one displayed currently. Note that this will only work this way if Fixed Seed is off. If you do not fix the seed after selecting Undo, then you will merely regenerate the currently showing image *and you will no longer be able to retrieve the prior seed*.

Note that even though you have fixed the seed, there are some other parameters that can drastically affect the final image. The most important of these is the Noise Type. As long as the noise type is not changed, fixing the seed will cause subsequent renders to generate an image with the same “basic” shape. Changing the Noise Type, however, will result in a completely different shape even if the seed is fixed. Changing the Noise Type back will of course restore the original shape.

Lines (-L)

Selecting this command causes the rendering engine to merely frame the polygons in the fractal image. As illumination, haze, color, and wave texture need not be calculated, the image can be rendered much more quickly. This can be useful when whiffing through random numbers until a desirable shape is found. The seed can then be fixed and the Polygons command selected so that the rest of the parameters can then be tweaked to produce a final, pleasing image. Remember that beyond a subdivide level of 6 or 7, Lines usually produces an unintelligible clump of lines.

When Lines is selected, a checkmark will appear next to the item in the menu.

Polygons (-P)

Selecting this command causes the rendering engine to use filled polygons complete with an illumination model, a distance hazing model, an altitude based color model, and a wave texture model for water areas. This is the mode to use to produce final images. Significant control of the above mentioned models is possible through the use of the other commands in this menu.

Using this mode can be significantly slower than using the Lines mode, however there is very little difference in the memory used.

When Polygons is selected, a checkmark will appear next to the item in the menu.

Noise Type... (-N)

Selecting this command initiates a dialog that contains three, mutually exclusive options selectable via associated radio buttons (see Figure 9). These options affect the way that the program’s random number routine operates.

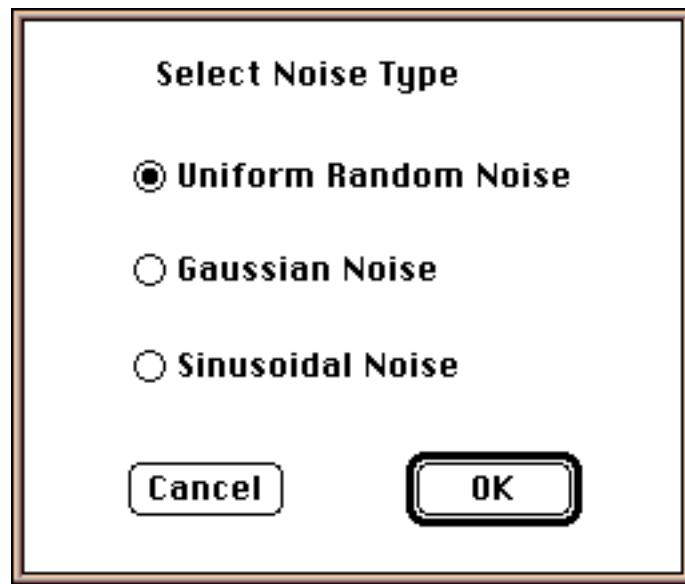


Figure 9

Uniform Noise produces random numbers that are uniformly distributed over the appropriate range. This type of noise tends to produce the most exaggerated landscapes, and generally requires a larger width to height ratio (see Modify Parameters) to somewhat mute the effect.

Gaussian Noise produces random numbers that are distributed in a standard Gaussian bell curve. This is the type of noise called for in most texts on fractals. Due to the reduced chance of extreme random numbers, Gaussian noise results in subtler shapes that might be appropriate for smaller width to height ratios (see Modify Parameters).

Sinusoidal Noise produces random numbers that are distributed in a sine curve. The effects are similar to but different from those of Gaussian noise.

Alternate View

Using this command changes a number of parameters that are accessible from the Modify Parameters menu. This will cause succeeding images to be rendered from a different point of view than the default view with which the program starts. When selected, there will be a checkmark next to the item in the menu. However, due to the interaction with the Modify Parameters menu, this can sometimes be misleading.

Selecting this item when the command is already checked on the menu will return the viewing parameters to the default values.

Modify Parameters... (-M)

This command initiates a dialog that provides access to the bulk of the parameters that affect the resultant fractal landscape (see Figure 10). The dialog will limit the values entered in each field to those that either make sense in context or to those that will keep the program from generating overflow or other obnoxious error conditions. If a value is entered which falls outside of the valid range, the dialog will beep and select the offending value, not allowing you to proceed further until the problem is corrected.

Miscellaneous Parameters		
Height:Width Ratio:	1.000000	
Fog Visibility:	35000.000000	
H Factor:	1.050000	
Zoom:	2053.480000	Conform
Z Offset:	26000	Conform
Y Offset:	-10000	Conform
Y Translate:	67	Conform
Random Seed:	969372044	
Cancel		OK

Figure 10

The first parameter is Height:Width Ratio. This parameter (which should probably be renamed to Width:Height Ratio but isn't for mundane reasons) limits the vertical displacement that happens during the subdivision process. The overall affect of this parameter is to control the general height of the fractal landscape. The smaller this number is, the higher the peaks (and deeper the valleys) will be. The larger this number is the flatter the resulting landscape will be. By having a Script file that starts with a very large Height:Width Ratio and ends with a small one, you can create an animation of "growing mountains". This is a positive floating point number which the dialog box limits to a range between 25.0 and 0.25 inclusive.

The next parameter is Fog Visibility. This basically dictates just how thick the haze is. It is a positive floating point number limited to a range of 1 to 500,000. A value of 1 will look like impenetrable fog, while 500,000 is crystal clear. A more useful low

end value would be in the range to 10,000 to 20,000 which will get you close to San Francisco on a bad day.

While it is possible to “turn off” the haze effect by using a large number, part of the rationale for putting the model in the program in the first place was to provide a form of depth cuing. With images rendered with no haze at all it is sometimes difficult to differentiate near features from far one. A little haze goes a long way to providing ample visual cues to add a real feeling of depth to the images.

For lack of a better term, the next parameter is called H Factor, and basically controls the local area roughness of the terrain. It is related to the “fractal dimension” referred to in the applicable texts. This value is a positive floating point number in the range 0.1 to 10.0. The smaller numbers will yield crazy looking spiky terrain, and the larger numbers will yield soft rolling terrain.

The next four parameters affect the view-point and view-angle of the “camera”. In order to better explain how these parameters function, it will be helpful to imagine that you are looking down on a chess board positioned in front of you, with a transparent screen in between you upon which you can trace the image of the chess board. Zoom affects the projected image in the same way that a zoom lens on a camera would. Z Offset affects how far away from you the chess boards actually is. Y Offset affects how far below you the chess board is. Y Translate works like this: Suppose you have the chess board set far below your eye level (Y Offset). The projected image of the chess board would also appear below you on the imaginary screen that the image is being projected on. Y Translate allows you to “slide” the part of the screen that has the image on it up (or down) to the level that your monitor displays.

Note also that each of these four parameters has a button to the right of it labeled “Conform”. When clicked, the program will attempt to adjust the corresponding parameter such that Sea Level (see the Levels... command) at the front of the landscape grid will match up with the bottom of the window that the image is being rendered to. Sometimes, due to rounding errors, this might be off by a pixel or two. Simply adjust the Y Translate parameter by hand to compensate for this and re-render the image.

The final parameter is the Random Seed. This is the value that actually determines what your fractal landscape will look like. While you can set this seed directly, the program will automatically generate a new seed whenever a render is initiated unless the Fixed Seed parameter is set on.

Levels...

This Command brings up a dialog box that allows you to set 5 separate levels that control aspects of the image rendering and coloring (see Figure 11). The dialog consists of 7 controls and 3 editable text boxes along with the usual Cancel and OK

buttons. This is a fun control to just play with, so try it first and if you're still confused maybe the following explanation will shed some small ray of light.

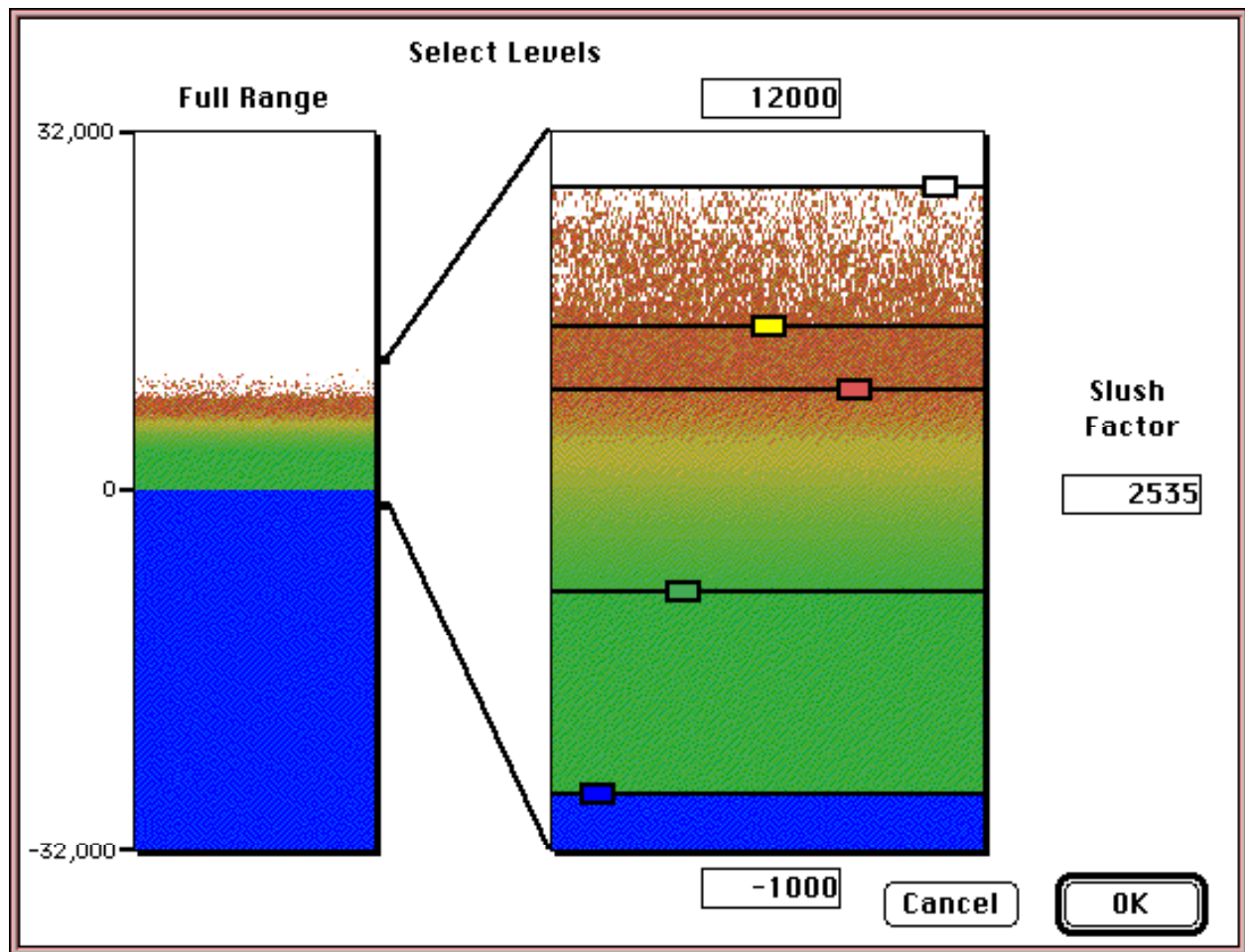


Figure 11

There are two images that dominate this dialog box. The smaller image on the left represents the entire numeric range available for the level parameters. In fact, most of this range is not normally needed (unless you are using some bizarre and twisted viewing and height parameters!!). They are there for completeness and to allow you to effectively “turn off” some features by positioning the appropriate level control to a maximum or minimum position.

On the right side of this leftmost, rectangular image, there are two black “handles” with a line radiating from each one to the top and bottom of the other rectangular image. These handles allow you to set the range of the full numeric space to “zoom into” in the larger, rightmost image (the zoom rectangle). By clicking and dragging these handles up or down, you change the zoom range to the right. The upper handle cannot be dragged below the lower one and vice versa. You will notice that the two editable text boxes above and below the rightmost image also change when

you change the respective handle. These represent the digital values of the top and bottom of the zoom range. They can also be edited by hand with the restriction that the two values may be no closer than 320 units apart, and that they fall in the range +32,000 to -32,000. If you enter an invalid value for one of these, the program will not let you proceed until the error is corrected. When the dialog is first initiated, the zoom range is set so that all of the 5 levels controls will be visible in the zoom image rectangle.

In the larger, rightmost, rectangular image (the zoom rectangle), you will notice 5 horizontal lines, each with its own “handle” such that each of the 5 handles occupies a different horizontal column. These 5 controls allow you to change each of the associated levels. Those levels are, from left handle to right handle, Sea Level, Transition Bottom, Slush Factor, Transition Top, and Snow Level. Their functions are explained below. As each handle is clicked on, its interior color changes to reflect the fact that it is “selected”. Also, the title over the editable text box to the right of the zoom rectangle and the contents of the text box change to indicate which level has been selected. The levels may be changed by simply dragging the handle up or down. Also, the text may be edited directly. Each level has restrictions on what comprises a valid entry, and as above, if an invalid entry is made, the program will not let you proceed until that value has been corrected. As you are not able to enter an invalid value by using the control handles, a simple way to correct an invalid level entry is to click on (and drag) the selected control.

Sea Level is the first editable level. This, quite obviously controls the base level below which Fractal! will clip the terrain flat and force it to be rendered as water. You can play with this value to fill a valley with water — or to empty a lake. The units are arbitrary, but thinking of them as “feet” will give an appropriate feel. This is a signed integer value which is limited from -32,000 to +32,000 but always less than Snow Level. If you drag this control above Transition Bottom, Slush Factor, and/or Transition Top, it will force those levels to its own.

Transition Bottom is the next level. It controls the height below which all polygons that are above Sea Level will be rendered in shades of the Low Mountain Color (see Mountain Color Range command). Polygons that are above this level but below the Transition Top level will be rendered in a color that is intermediate between the Low Mountain Color and the the High Mountain Color based on the polygon’s relative height within that range. This level is in the same units as Sea Level, and is a signed integer in the range -32,000 to +32,000, but must always be less than or equal to Transition Top and greater than or equal to Sea Level. Used in combination with Transition Top, you can control the abruptness of the transition between the two mountain colors, as well as the height at which that transition occurs.

The Slush Factor specifies a range that extends just below the Snow Level. Within this range, each polygon will be rendered as either snow (white) or the mountain color appropriate for that height, depending upon the orientation of that polygon.

The idea behind this is that the flatter (more horizontally oriented) the polygon, the more likely it is to hold snow. The steeper the polygon, the less likely it is to hold snow. This effect is, of course, attenuated by the relative height of the polygon within the Slush Factor range. For example, a polygon that would be rendered as snow near the top of the range, might be rendered as mountain if it were near the bottom of the range. Any polygon that falls below the Slush Factor range will always be rendered as mountain (if it is not below Sea Level). Slush Factor is in the same units as Sea Level, and is a signed integer limited to the range -32,000 to +32,000, but must always be less than or equal to the distance between Snow Level and Sea Level. If the Snow Level is changed, the control for the Slush Factor will stay a constant distance below it unless it is clipped at the bottom by Sea Level.

The next level is Transition Top. It controls the height above which all polygons that are below Snow Level will be rendered in shades of the High Mountain Color (see Mountain Color Range command). Polygons that are below this level but above the Transition Bottom level will be rendered in a color that is intermediate between the Low Mountain Color and the the High Mountain Color based on the polygon's relative height within that range. This level is in the same units as Sea Level, and is a signed integer limited to the range -32,000 to +32,000, but must always be less than or equal to Snow Level and greater than or equal to Transition Bottom. Used in combination with Transition Bottom, you can control the abruptness of the transition between the two mountain colors, as well as the height at which that transition occurs.

The last level is Snow Level, and it specifies the level above which the terrain is *always* rendered as snow (white). It is in the same units as Sea Level and likewise is a signed integer limited to the range -32,000 to +32,000 with the only limitation being that it cannot extend below Sea Level. If you drag this control below Transition Bottom, Slush Factor, and/or, Transition Top, it will force those levels to its own.

Color

Fog Color... (-G)

This command initiates a standard ColorPicker dialog that allows the user to specify the “fog” color. This is actually the color that is used for the distance hazing model. Some unusual effect can also be achieved by using a dark color for the fog color and thus creating a distance “dimming” model. This works well with low light levels (see the section of the Illumination command below).

Water Color... (-W)

This command initiates a standard ColorPicker dialog that allows the user to specify the water color. This is the basic water color that is operated on by the wave model. Try a bright chartreuse for a truly novel and nauseating effect.

Mountain Color Range... (-T)

This command initiates a dialog that allows the user to specify two colors: the High Color and the Low Color. The program uses these two colors to create a smooth range of colors that will stretch from Transition Bottom at the low range to the Transition Top at the high range (see Levels...). Clicking on the box representing either the High Color or the Low Color will bring up the standard ColorPicker allowing the user to select the desired color.

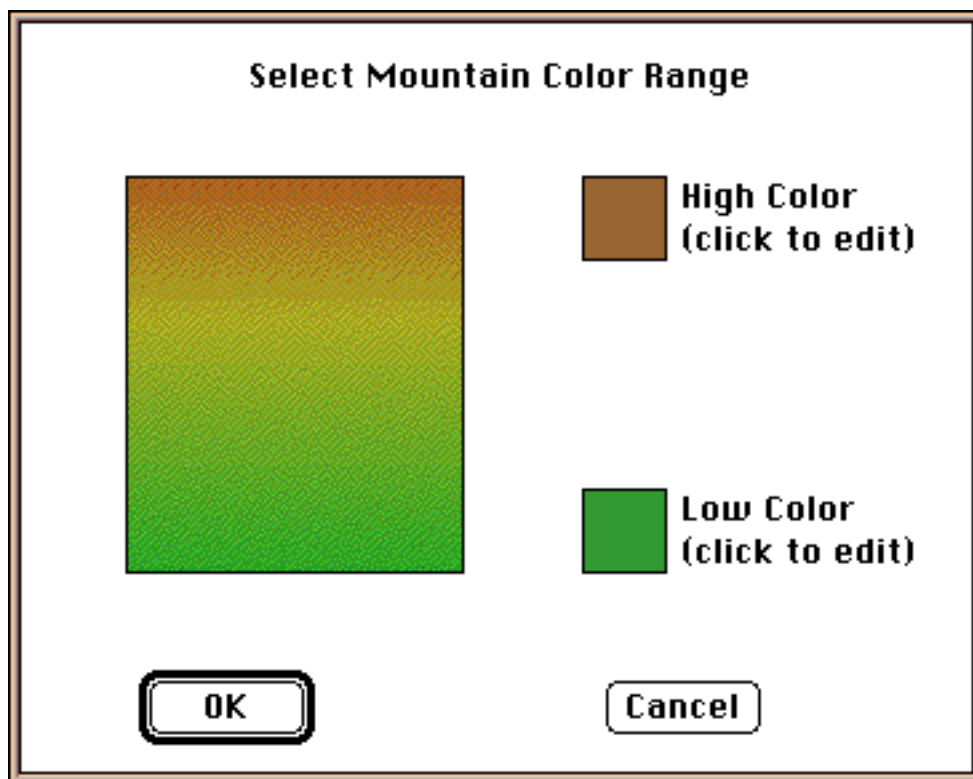


Figure 12

When running on 8-bit color (or any clut type display), Fractal! will normally set the color palette to the most appropriate one for the 32-bit image just rendered. When this dialog is initiated, the palette is temporarily set back to the System Palette.

Note that both Transition Top and Transition Bottom are parameters that can be set from the Levels... command in the Appearance menu.

Sky Color Range... (-Y)

This command initiates a dialog that allows the user to specify two colors: the High Color and the Low Color for the Sky color gradient. The program uses these two colors to create a smooth range of colors that will stretch from Window Top to the Horizon level. This is accomplished with a simple fill prior to the actual rendering of an image (polygon rendering only). Clicking on the box representing either the High Color or the Low Color will bring up the standard ColorPicker allowing the user to select the desired color.

The user also has the choice of either an HSV space gradient or an RGB space gradient (see figure 13 below). This allows for a wider range of effects. RGB space gradients are the default.

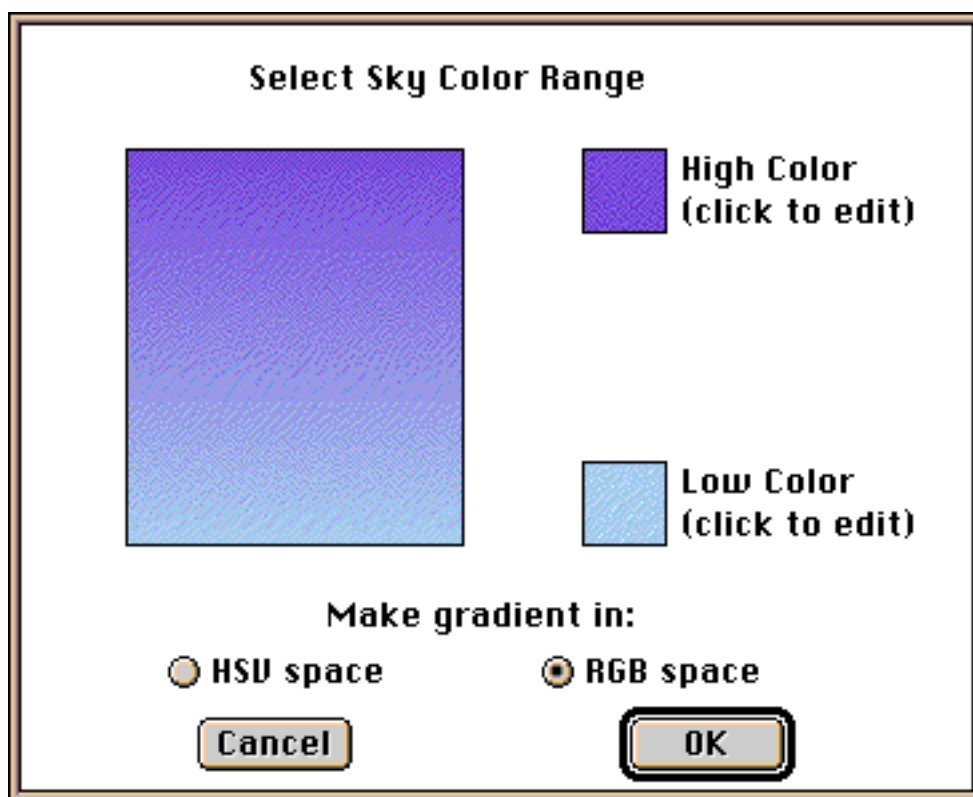


Figure 13

When running on 8-bit color (or any clut type display), Fractal! will normally set the color palette to the most appropriate one for the 32-bit image just rendered. When this dialog is initiated, the palette is temporarily set back to the System Palette.

Illumination... (-I)

This command initiates a dialog that allows the user to control various aspects of the lighting model used for rendering (see Figure 14). There are 3 basic controls in this dialog box.

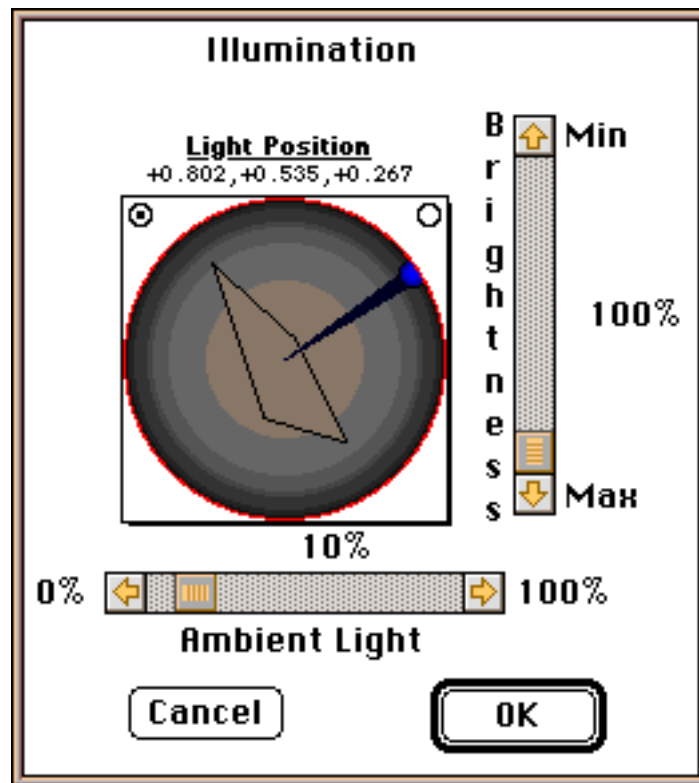


Figure 14

The Brightness control along the right hand side of the dialog box is a standard scroll bar that allows the setting of the overall brightness of the the light source. Its operation is self-explanatory.

Like the Brightness control, the Ambient Light control is a standard scroll bar. This control affects the minimum level of illumination on a surface that would normally receive *no* light from the light source. Setting this control to zero would cause such a surface to be rendered true black (assuming no haze).

The most interesting control is the large control that dominates most of this dialog box. This is an ingenious control devised by Bob Flanagan at Atari Games Corporation to be used for a number of 3D control applications. Having little or no pride myself, I begged Bob for permission to use it in Fractal! to control the direction of the light source. He graciously agreed and helped me integrate it into my program.

The operation of this control is pretty simple. By clicking and dragging on the knob of the “stick” you can tilt this stick about a full hemisphere. There are two ways to move this “stick” between the front and rear hemispheres. First, click on the stick and hold the mouse button down. Then, keeping the mouse button depressed, simply move the cursor well outside of the control circle. Still keeping the button depressed, move the cursor back into the control circle and the stick will have

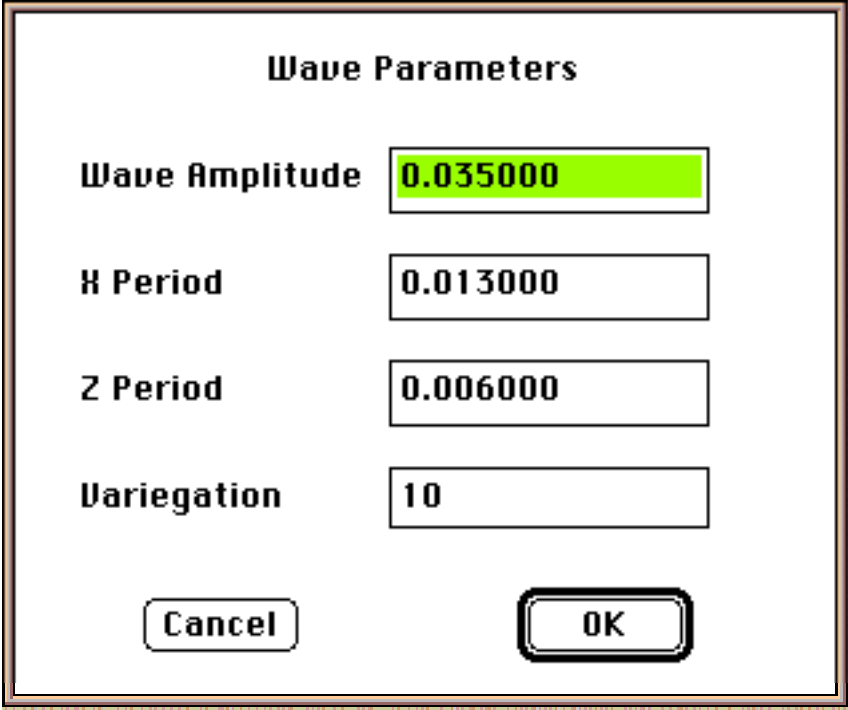
flipped to the other hemisphere. To return to the prior hemisphere, just repeat the process.

The second method is even simpler. Clicking on the miniature radio button in the upper left-hand corner of the control area will also cause the stick to flip hemispheres. Note that the mini radio button in the upper right-hand corner of the control is used to force the illumination vector to be straight into the Z-axis (positive Z), sort of a head-light effect.

Simply move the stick such that the stick itself represents the desired vector for the illumination direction. That is, the vector from the knob of the stick to the rectangular polygon to which the stick is attached will represent the vector for the light source.

Wave Parameters...

This command initiates a dialog that allows the user to change 4 parameters that affect the wave texture model (see Figure 15).



The image shows a dialog box titled "Wave Parameters". It contains four input fields, each with a label to its left: "Wave Amplitude" with a value of "0.035000", "X Period" with a value of "0.013000", "Z Period" with a value of "0.006000", and "Variegation" with a value of "10". The "Wave Amplitude" field is highlighted in yellow. At the bottom of the dialog are two buttons: "Cancel" on the left and "OK" on the right.

Wave Parameters	
Wave Amplitude	0.035000
X Period	0.013000
Z Period	0.006000
Variegation	10
<div>Cancel OK</div>	

Figure 15

The entire wave texture model is pretty much a hack that I came up with after many attempts to do *something* to make the water look better than just a solid color. The effect is achieved by varying the saturation level of the Water Color. If I tried to explain just how it works, I'd not only confuse you, but myself as well, so just accept on faith that you can get some interesting effects with these parameters.

In my opinion, the texture improves the water appearance at any sub-division level, but is most pleasing at sub-divisions of 8 and higher.

The first of the 4 parameters, Wave Amplitude, is pretty self-explanatory. It determines how subtle (or obnoxious) the wave texture will be. Setting this parameter to zero will effectively “turn off” the wave texture effect.

The X Period and Z Period are used in a function to cause a “beat” effect. Try varying these to make the distance between the “waves” larger and smaller.

Variegation is a parameter that controls some randomness at each polygon. This allows for occasional “whitecaps”.

Window...

The Window... command allows you to specify the final size of the image being rendered in a choice of pixels, inches, centimeters, millimeters, points or picas. You can also set the resolution, both vertically and horizontally, for any PICT files to be output. Two mutually exclusive radio buttons are available allowing you to choose between a Full Screen image of a fixed size (depending upon the size of your display) and a Standard Window of any size (depending upon memory constraints) (see figure 16 below).

Full Screen windows are always placed on the “deepest” display device connected to the system at the time the program was started. If this device is also the Main Device (i.e. the display with the menu bar) then the image size will be the size of that display minus the size of the menu bar. If not, the image size will be the size of the full device. In any event, this size is displayed in the dialog box.

Standard Windows are initially placed on the deepest device and centered in that display. Depending upon the size of the image selected, the window will be fully expanded and have inactive scroll bars, or will only display part of the image and show active scroll bars. This window may be dragged onto any display on the system, even straddling multiple displays. The window may also be sized using the grow box in the lower right-hand corner of the window. Note that when a ‘PICT’ file is saved, the full image is saved, not just the portion displayed in the Standard Window.

To the right of the edit fields for Image Size, there is a pop-up menu to allow you to choose in what units you would like to specify your final image. Since Full Screen images are a fixed size, selecting this menu (or entering a new size value in the edit fields) will cause the radio buttons to switch to Standard Window if it is not already selected. For Standard Windows, you may select any units in which to specify the size of your final image.

Below this area is the specification for output resolution. These values will directly affect either the size of the image or the number of total pixels in the image. By using the Image Size and resolution fields you can correctly specify the output format for a wide range of different applications from screen presentation to film recorder output.

Below the resolution fields is an information area that will show (in appropriate units) what affect any changes you make will have.

Window Options

☐ Full Screen (832 x 604)

☒ Standard Window

Image Size

x ▼

(Horiz.) (Vert.)

Output PICT File resolution (DPI) :

Horizontal Vertical

x

PICT file image will be 750 x 600 pixels .
Screen image will be 10.42 x 8.33 inches on
deepest display (72 x 72 dpi).

☐ Preserve image (needs lots of memory)

Figure 16

Whenever a change in window style or size is desired, a check box at the bottom of the dialog box becomes enabled. If you desire to have a scaled version of the current image copied to the new window that will be created, you can check this box. Note that doing this will temporarily require additional memory as both the original offscreen GWorld and the new offscreen GWorld must both be present at the same time. Also, because of the scaling and the potential dithering required to a clut type of display, this may take a considerable amount of time, especially if one or both of the image sizes are large.

If there is insufficient memory to create the new window, the operation will be aborted and the original screen style and size will be reset. If there is insufficient memory to create the offscreen GWorld, the entire operation is aborted and the program exits. For this reason, it is recommended that you **do not** select the Preserve Image option unless you are sure of what you are doing.

Note that while the size, window style, and resolution of an image are saved in the parameter file, if a Full Screen parameter file is used on a system with a different size display from the one on which it was created, it will still be treated as a Full Screen image with the image size being that of the new display. Because of this, a “conform zoom” operation (see Modify Parameters... above) is always performed upon the opening of a Full Screen parameter file and the total number of pixels may vary from display to display. (It is assumed that Full Screen images are intended for screen display purposes.)

The Rendering Menu

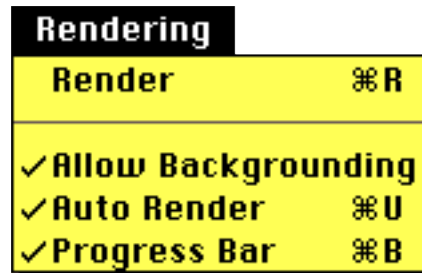


Figure 17

This menu contains the commands that control the rendering of the fractal image.

Render (-R)

Selecting this command initiates the rendering operation. Depending upon the state of the Allow Backgrounding parameter, this operation can be aborted by pressing (-.). Also the current state of the image can be copied to the window by clicking the mouse anywhere in the window if backgrounding is allowed.

Note that while a render is active, the Render menu item will always be highlighted. This is one way to determine if the program is busy during a render. Also, if backgrounding is on, the Render menu is the only program menu left active although the Render command itself is obviously not.

Allow Backgrounding

When checked, Fractal! behaves as a MultiFinder friendly application, yielding the processor after rendering 2 polygons. This means that Fractal! can do its rendering in the background while other applications are running. It also allows the user to abort a render by pressing (-.), or request the current, partially complete image to be displayed in the window by clicking the mouse anywhere within the window. Note that either of these operations will be postponed until after the subdivision phase which, depending upon the processor being used and the level of subdivisions, can be anywhere from a few seconds to a couple of minutes.

If Allow Backgrounding is *unselected* (the item is unchecked), once a render is started it cannot be aborted or interrupted until it has finished (or until Allow Backgrounding has been re-enabled), and no other application will receive processor time during the render. The advantage of turning off backgrounding is that the render operation will proceed much more quickly as the application hogs the entire cpu (much as I hogged the TV last night!)

Note that as the Render menu (and the room service menu) is the only menu active during a Render operation, it is possible to turn off backgrounding once a render has started via the menu item. The reverse is possible only via the

command key sequence (-K) as all menus are inactive with backgrounding turned off. Pressing (-K) will allow backgrounding to resume, recheck the menu item to indicate this, and, if the Progress Bar is visible, the Cancel button will reappear (see below). The reason that there is no command key combination to turn backgrounding off is that doing so inadvertently can lock up the processor for extended periods of time until the render operation has completed.

Also note that because of the possibility of locking up the cpu for significant amounts of time, this parameter is not saved to the parameter file. In addition, whenever a parameter file is opened, Allow Backgrounding is automatically enabled.

Auto Render (-U)

When this option is checked, any changes made with the Subdivides menu or the Appearances menu will automatically initiate a render operation. As you first work with Fractal!, having this feature enabled is a good idea. This way you can see the effects of each change as you make it. While Auto Render can be very convenient, once you are familiar with all of the programs features it will probably be preferable to make many changes before initiating a new render.

This parameter is saved with the parameter file.

Progress Bar (-B)

When this option is checked, anytime a render is initiated, a floating window will appear containing a Progress Meter (see figures 18 and 19). This displays the percentage of rendering that has completed. It appears after the sub-division phase, so that when subdividing to a high level, this window might not appear for a minute or so. If Allow Backgrounding is turned on, the window can be temporarily dismissed by clicking in the close box. Also there will be a Cancel button as shown in figure 18. If Allow Backgrounding is turned off, the Progress Meter will have no Cancel button as the rendering operation cannot be canceled. In this case it will look like figure 19. The Progress Meter may be turned off completely by selecting the menu item such that the check marks disappears.

This parameter is saved with the parameter file.

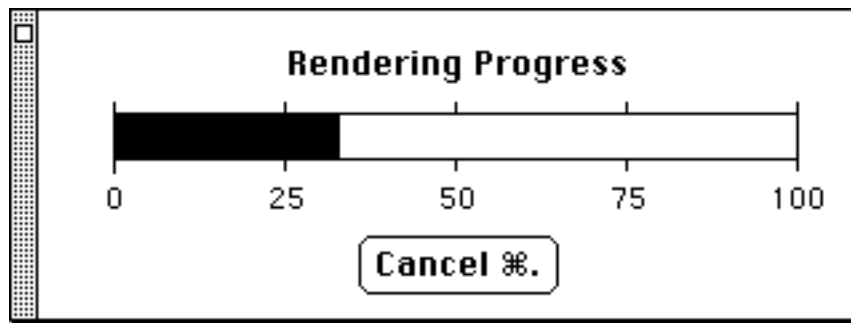


Figure 18

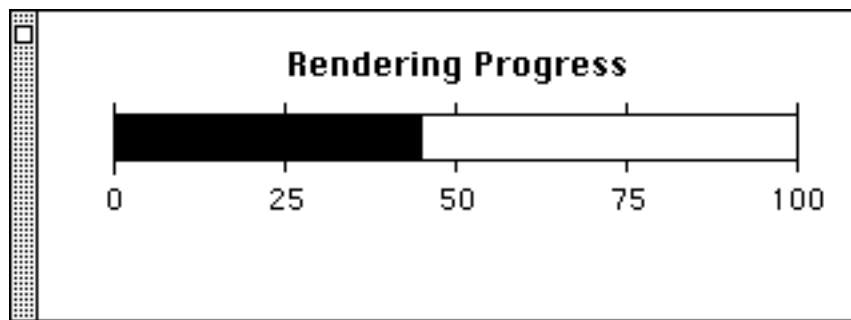


Figure 19

Other Features

Online help (such as it is) is accessible either by selecting the option under the Help Menu (see Figure 20) or by pressing the help key (on the extended keyboard). If you can't figure out the operation of the help system it probably wouldn't be of any use to you anyhow.

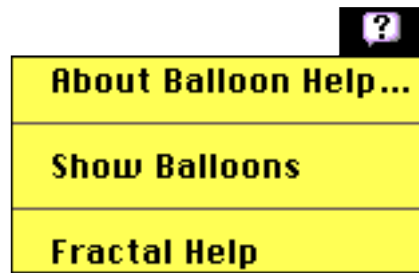


Figure 20

You can find out a little about this program by selecting the About Fractal... menu item under the Apple Menu.

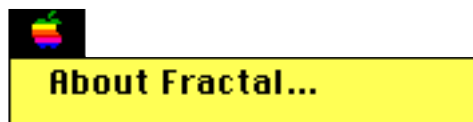


Figure 21

Theory of Operation

Much has been written on the subject of fractal landscapes. My math, however, has never been particularly strong, and given my game designing background, I'm always looking for ways to "cheat" in order to optimize. During the process of writing the code, I often ran into places where it seemed prudent to compromise purity for the sake of speed or space considerations. In order to more fully explain my approach to these problems, a little background is necessary.

As Freeman Dyson put it, "*Fractal* is a word invented by Mandelbröt to bring together under one heading a large class of objects that have [played] ... an historical role ... in the development of pure mathematics."¹ In his book *The Fractal Nature or Geometry*, Mandelbröt himself points out, "...fractal geometry reveals that some of the most austere formal chapters of mathematics had a hidden face: a world of pure plastic beauty unsuspected till now."²

Fractal! uses two different subdivision algorithms to create the grid of points that represent a landscape.

Carpenter Subdivides

These fractal landscapes are achieved by the successive subdivision of, in this case, a rectangular polygon. Start with a planar square in the X-Z plane. Next find the mid-point of each of the four sides of this square and draw a line connecting each pair of opposing mid-points, thus dividing the original square into four squares (see figure 22). Take the five new points generated by this method (diamond shaped points in figure 22) and displace each along the Y axis by a random amount constrained by the length of the side being subdivided. Now do this same operation to each of the four squares just created. By continuing this subdivision process you will create a fractal landscape. The more subdivisions, the more detailed the landscape - and the more memory and processing time required.

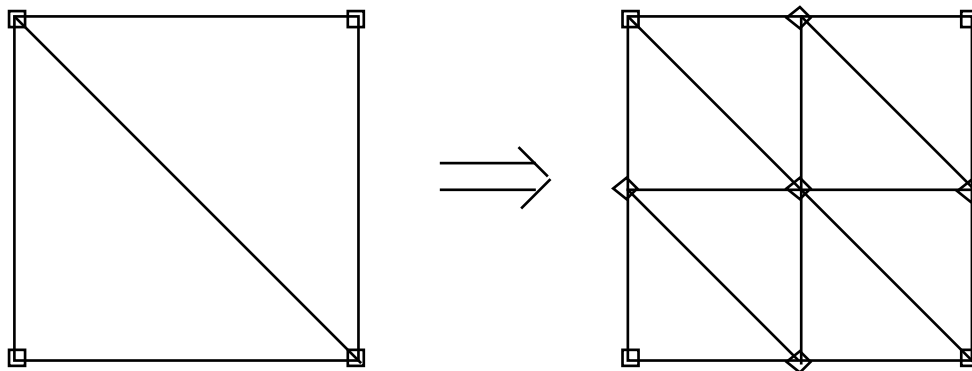


Figure 22

¹ From "Characterizing Irregularity" by Freeman Dyson, *Science*, May 12, 1978

² From *The Fractal Nature of Geometry* by Benoit B. Mandelbröt, © 1977, 1982, 1983

This is the basic idea behind fractal landscapes created with the Carpenter subdivision algorithm. By dividing each square into two right triangles created by the diagonal line between two opposing points we create polygonal surfaces that cover the vertices in the grid, avoiding the problem of non-coplanar polygons as no three points can ever be non-co-planar.

Miller Subdivides

While the above method generates interesting landscapes, there are certain artifacts that arise from it. After the first level of subdivision, each triangle is divided into other triangles which are coplanar with it. This yields a “rolling pyramid” effect which when shaded with an illumination model exhibits very noticable creases along the x and y axes and the diagonals.

In 1986, Gavin S. P. Miller described another algorithm which is not subject to these “creasing” artifacts³. Rather than calculating and displacing the midpoints of each line segment, new points are not generated along connecting lines at all. Instead they are generated within the original square with points at $1/4$ and $3/4$ the distance along the X and Z axes generating 4 points within each square (see figure 23 below). To find the starting Y value from which to displace each of the new points we take $9/16$, $3/16$, $3/16$, and $1/16$ of the adjacent points with the nearer points having the greater weighting as seen in figure 23. The new square created by these four points is $1/2$ the size of the original square as in the Carpenter algorithm, however, the new grid of points shares no points in common with the original grid.

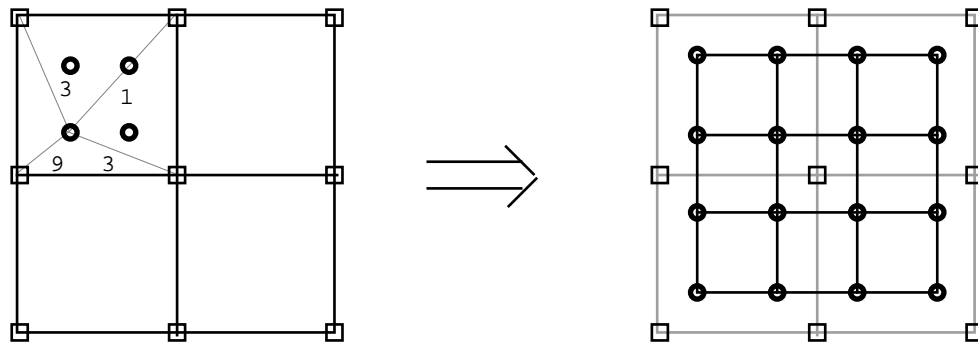


Figure 23

³From the 1986 SIGgraph proceedings “*The Definition and Rendering of Terrain Maps*” by Gavin S. P. Miller. August 1986

This then is the basic theory behind Fractal!'s algorithm. The program starts with a single square in the X-Z plane. Before either subdivision process is started, I first displace the original four points in Y in such a way that those furthest away from the view point in Z tend to be higher than those nearer our view point. This technique favors the generation of landscapes that proceed from a height in the background to the bottom of the screen (and usually sea-level) in the foreground. Experimentation will, however, confirm that this situation is not guaranteed — hey, life's a crapshoot!

The original grid, therefore, starts with four points and two triangles. With Carpenter's algorithm, after the first subdivision, we will have 9 points and 8 triangles. The second subdivision yields 25 points and 32 triangles and so on such that at the n^{th} subdivision there will be:

$$(2^n + 1)^2 \text{ points and } 2^{(2n+1)} \text{ triangles.}$$

Since the co-ordinates of the four original points in X and Z are fixed and known, the X and Z co-ordinates of any point at any level of subdivision are always calculatable and therefore never need to be stored in memory. We only need to store the Y co-ordinate of every point generated. In Fractal!, I have constrained my universe such that each Y co-ordinate may be stored in a single short (2 byte) variable. This expedient means that my program needs 1/2 the memory that it would have taken if I had used longs. Simple math will now provide us with the memory requirements for the grid which describes our landscape at any level of subdivision n as $2*((n + 1)^2)$ points (remember, 2 bytes per point!) giving us the following table:

3.....	162 bytes
4.....	578 bytes
5.....	2,178 bytes
6.....	8,450 bytes
7.....	33,024 bytes
8.....	132,098 bytes
9.....	526,338 bytes
10.....	2,101,250 bytes
11.....	8,396,802 bytes

With Miller's algorithm our numbers are somewhat different. Unlike the case shown in figure 23, I do not start with a grid of 9 points. Therefore, at any level n of subdivision, there will be 2^{n*2} points which will require $2*(2^{n*2})$ bytes of storage.

The table for this algorithm then looks like:

3.....	128 bytes
4.....	512 bytes
5.....	2,048 bytes
6.....	8,192 bytes
7.....	32,768 bytes
8.....	131,072 bytes
9.....	524,288 bytes
10.....	2,097,152 bytes
11.....	8,388,608 bytes

However, since with Miller's algorithm, each successive grid of points does not share any points with the preceding grid, at any level of subdivision, the program would require the sum of the above storage requirements for that level and the preceding level. Thus for 11 subdivision, Miller's algorithm would require 2,097,152 + 8,388,608, or 10 megabytes (10,485,760 bytes) *just to create the grid!*

Now, add to this the memory required for the offscreen GWorld used to hold the 24-bit deep image (actually 32 bits per pixel). If we are using a full screen window on the standard 13 or 14 inch Macintosh display, the pixmap for this GWorld alone is 1,177,600 bytes ($640 * 460 * 4$). There is, of course other memory needed by the program for various structures in the Mac environment, plus that needed by the program code itself. In trying to conserve memory, I only maintain screen coordinates for two rows of pixels at any one time during the rendering process. Since this is an additional 4 bytes for each of these pixels (2 bytes each for screen h & v position), that's an additional 4 bytes * 2 rows of $(2^n + 1)$ points per row. For 11 subdivisions, this adds up to an additional 16,392 bytes. As you can see, these totals add up quickly, so keep this in mind as you decide how big a partition to allocate the Fractal!. If you run short, cut back the number of iterations - honestly, this program is not a ploy to generate revenues for memory manufacturers!

Acknowledgements

I'd like to thank a number of people for their help with this program. First of all, I'd like to thank **Lyle Rains** for his inspiration, and invaluable assistance with much of the math. Lyle was working on his own fractal landscape program at the time I started Fractal!, and many of my ideas and at least one algorithm can be directly attributed to his help.

As I mentioned earlier in this manual, **Bob Flanagan** was gracious enough to let me use his wonderful 3D CDEF in my Illumination Dialog. This little gem is easily the most elegant control of its type that I have seen in any 3D program, including the heavyweight commercial ones.

Owen Rubin, the "Black Thumb" of software testing, provided many crashes for my perusement. But he went far beyond this, nagging and goading me into many changes that have resulted in a far better program. As I think back on where this program was when I first put it into his hands, I'm amazed at how many of the features that I now think of as basic parts of the program are directly attributable to his incessant pestering.

Whenever I hit a wall with Mac graphics problems, **Konstantin Othmer** was always willing and able to help. The "Godfather" of Quickdraw, Kon pulled my neck out of the noose on more occasions than I care to think about. Kon, for all you do, this Bud's for you!

Dave Theurer, the author of DeBabelizer, (the most amazing bitmap graphics utility available anywhere on any platform!) was also an incredible help. His many travails down the myriad paths of Macintosh graphics enabled him to answer many of my questions as I trod into unfamiliar waters.

For general help in testing and encouragement I'd also like to thank:

Bruce Burkhalter

Kurt Clark

Peter Commons

Sam Comstock

Jon Fuelleman

Ed Logg

Gavin Miller

Gabe Rotberg

Dave Surovell

Joe Williams

and of course Benoit Mandelbröt, without whose work, none of this would have been possible!

Disclaimers and other stuff

This program is **freeware**. That means no-one is allowed to charge anything for it! *Unmodified* copies may be freely distributed as long as all original accompanying documentation is included. It may be posted to public bulletin boards and online information services as long as no fee is charged for downloading beyond the normal connect time fees.

It may **not** be included in any collection of freeware or shareware without the express written permission of the author (this means **you** EduCorp!). It may **not** be posted to any online service that charges a membership fee beyond the normal connect time fees of the host system (this means **you** ZMAC!). I personally resent the above two approaches to money-grubbing with no consideration to the authors.

The author reserves all rights to the program and all accompanying documentation.

I would love to hear your comments about Fractal! Please feel free to send them and any neat fractals (parameter files only please!!) that you've created with the program to:

Ed Rotberg
308 Ramon Drive
Los Altos, CA 94024

or

AppleLink: ROTBERG
Internet: gonzo@3do.com
CI\$: 71121,3053