Mandelbulber
End User Manual

Version 2.11.0.9.5 (2017-April)

Download:	(https://sourceforge.net/projects/mandelbulber/)
Development:	(https://github.com/buddhi1980/mandelbulber2/)
Community:	(http://www.fractalforums.com/mandelbulber/)

Editors

Krzysztof Marczak:	(buddhi1980@gmail.com)
Graeme McLaren:	(mclarekin@gmail.com)
Sebastian Jennen:	(sebastian.jennen@gmx.de)
Robert Pancoast:	(RobertPancoast77@gmail.com)

background image by Torsten Stier (2017)

Contents

1 About this Handbook

This handbook has been crafted for both new users and experts to assure confidence and ease of usability for Mandelbulber fractal design. We wish you a Happy Experience!

This handbook is still being written. The most recent version can be downloaded from here: https://github.com/buddhi1980/mandelbulber_doc/releases

2 What are fractals?

Fractals are objects with self-similarity, where the smaller fragments are similar to those on a larger scale. A characteristic feature about fractals are subtle details even at very high (up to infinite) magnification.

2.1 Mandelbrot set

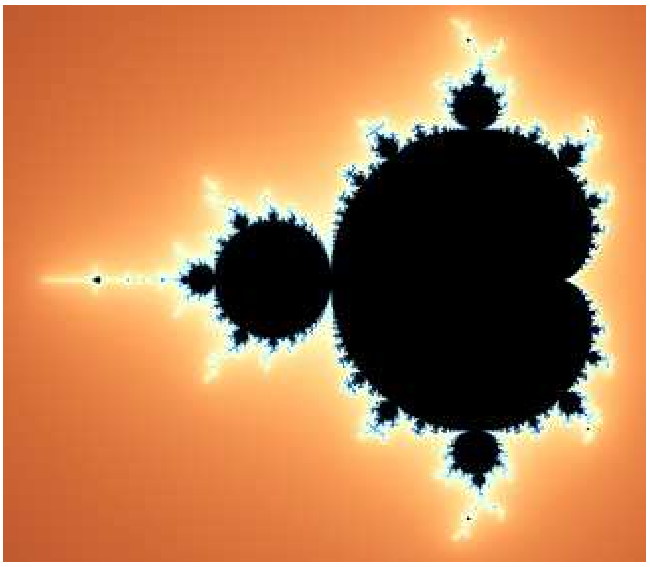


Figure 2.1: Mandelbrot Set

This is a typical example of a two-dimensional fractal generated mathematically. This image is created with a very simple formula, which is calculated in many iterations:

$$z_{n+1} = z_n^2 + c$$

- z is a complex number $(a + \mathrm{i}b)$, where i is the imaginary number.

  $$i = \sqrt{-1}$$

  The number is made of two parts: a the real part and ib the imaginary part.

- c is the coordinates of the image point to be iterated.

In 2D, z is a vector containing two complex number coordinates, x and y, (these points represent the pixel location where x represents the real part of the number [a] and y represents the imaginary part of the number [b]). Because they are complex numbers, they can be positive or negative, but also there will still be a mathematical solution if a function requires the square root of a negative number.

Each original point (pixel position) is tested in the formula iteration loop, to determine if it belongs to the formula specific mathematical fractal set.

The initial value of point z is assigned to equal c, $(z_0 = c)$, this parameter is then used repeatedly in the iteration loop.

$$\begin{aligned} z_{n+1} &= z_n^2 + c \\ z_{n+2} &= z_{n+1}^2 + c \\ z_{n+3} &= z_{n+2}^2 + c \\ &\text{etc.} \end{aligned}$$

The program has to determine if these series are convergent. To do this iterations should be repeated an infinite number of times. But since a computer cannot infinitely repeat in practice the convergence is determined with a simplification.

Termination conditions are applied to ensure the formula does not iterate to infinity. The most common conditions used are called **Bailout** and **Maxiter**.

The **Bailout** condition stops the iteration loop if the formula transforms (moves) the point further than a set distance away from an "origin". This detects if series are convergent (calculated point is outside the fractal body)

Maxiter is simply a condition to stop iterating when a maximum numbers of iterations is reached (just to not do iterations infinite times)

In the Mandelbrot formula, after each iteration, the modulus of a complex number is calculated; in other words, the length of the vector from the origin ($x = 0$, $y = 0$) to the current z point. This vector length is often called r for it is the radius from the center (origin) to the current z point.

In this example, when the length $r > 2$ (i.e $Bailout = 2$), the termination condition has been met, then the iteration process is stopped and the resulting image point is marked with a light color. When, after many repeated iterations, r is still less than 2, then it can be considered for simplicity that such a result will continue indefinitely. Iterations are therefore interrupted after a certain number of iterations (Maxiter). This point is marked on the image with black. This results in a "set" of points that do not reach bailout termination (black) and the rest of the points given lighter colors (dependent on a chosen coloring method).

2.2 3D fractals

The three dimensional fractal type, the "Mandelbulb" is calculated from a fairly similar pattern to the Mandelbrot set. The difference is that the vector z contains three components (x, y, z) or four dimensions (x, y, z, w). As they are part of the z vector, they are denoted as ($z.x$, $z.y$, $z.z$). Examples being Hypercomplex numbers and quaternions.

They can also be created by modification of quaternions or by a specific representation of trigonometric vectors.

Generally, common math operators are used (e.g.: addition, multiplication, squaring, and power) and also conditional functions (e.g., **if** $z.x > z.y$, **then** $z.x = something$).

Some other types of 3D fractal objects are based on iterative algorithms (IFS - Iterated Function

Systems). An example would be the famous Menger Sponge.

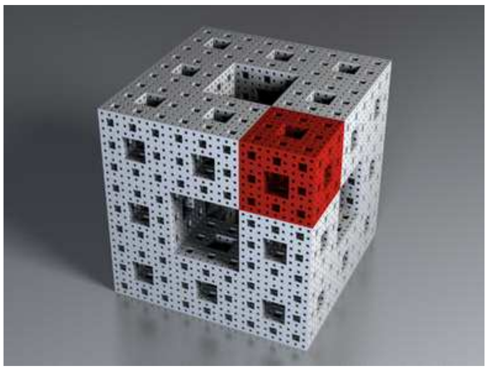


Figure 2.2: Menger Sponge

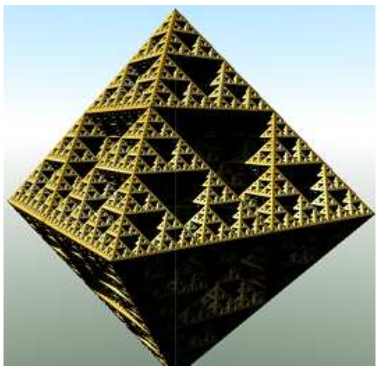


Figure 2.3: Sierpinski

2.3 Mandelbulber Program

Mandelbulber is an easy to use, handy application designed to help you render 3D Mandelbrot fractals called Mandelbulb and some other kind of 3D fractals like Mandelbox, Bulbbox, Juliabulb, Menger Sponge, . . . The following sections cover the program interface and give useful information about how to use it.

3 Distance Estimation

Distance Estimation (DE) is the calculation of an estimated distance from the given point to the nearest surface of the fractal. As suggested by the word 'estimate', it is an approximate value. It is calculated using simplified algorithms based on analytical (Analytical DE) or numerical (Delta DE) calculations of gradients.

DE is the most important algorithm required to render three-dimensional fractals within a reasonable time. It achieves a great reduction in the number of steps needed to find the exact area of the fractal while tracking a "photon" traveling toward the object along a ray (a simulated beam of light from the camera eye). A ray is generated for each pixel (1000 x 1000 resolution = 1,000,000 rays). They match FOV from the camera eye (i.e. they are not parallel).

Without the DE calculation, the proximity of the photon to the fractal surface would need to be repeatedly calculated after each of many very small steps. For example, without an estimate of where the fractal surface is, you may need up to 10,000 steps to trace a ray of light, for every pixel of the image.

Using DE, the size of the steps along the ray of light can be increased, based on the calculated estimate of where the fractal surface should approximately be located. The process of moving along the ray and testing for the surface location is called ray-marching.

Ray marching looks like the illustration below. In each step, an estimation on the distance to the nearest fractal surface is calculated. The photon is moved along the ray by this distance. The next step is re-calculated based on the estimated distance. This distance is less so this time the "photon" is moved a smaller distance. The ray-marching becomes more accurate closer to the surface of the fractal. The ray marching ceases when the "photon" becomes within a set "distance threshold" from the surface or after a maximum number of iteration if the option "stop at maximum iteration" is enabled.

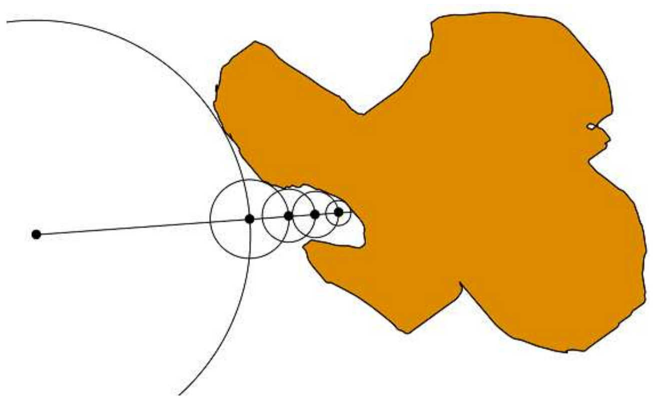


Figure 3.1: Distance Estimation with DE factor 1

Since the estimation contains some error (sometimes quite large), there is a risk that the step of moving the "photon" will be too large, and incorrectly it will flow into the surface of the fractal. This may result in visible noise in the rendered image.

To prevent this, the "photon" can be moved by the estimated distance multiplied by a number between 0 and 1 (*ray-marching step multiplier*). Steps are then smaller, so there is less risk of "overshooting" the surface (better image quality), but the rendering time increases due to more steps being required.

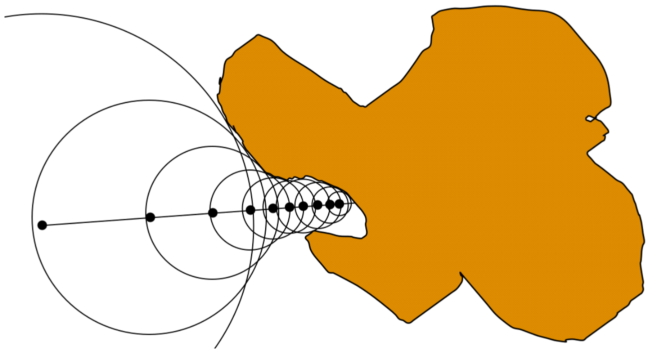


Figure 3.2: Distance Estimation with DE factor 0.5

Each formula has assigned a DE mode and function (“preferred”). In most cases the preferred *mode* is *Analytical DE* (fastest).

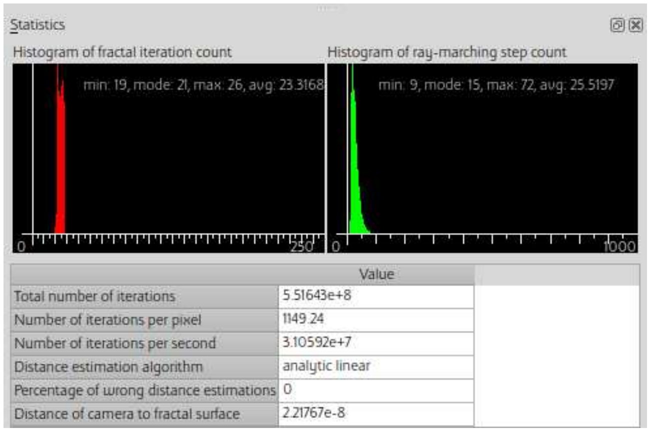
The preferred *function* is assigned based on whether the formula is transforming in a linear or logarithmic manner. These setting can be varied on the *Render Engine* tab.

Analytical DE mode is faster than *Delta DE* mode to calculate. However with some formulas only Delta DE mode will produce a good quality image. The DE modes can be used with either linear or logarithmic DE functions.

Example linear out: $distance = \frac{r}{|DE|}$

Example logarithmic: $distance = \frac{0.5r\log(r)}{DE}$

The quality produced by the DE mode and function combinations is formula specific. The setting of formula parameters can also greatly affect the quality produced by the DE. In some cases the choice of fractal image is determined by what location and parameters can produce good DE quality.


Statistics
Histogram of fractal iteration count
Histogram of ray-marching step count
0
250
0
1000

min: 19, mode: 21, max: 26, avg: 23.3168

min: 9, mode: 15, max: 72, avg: 25.5197

	Value
Total number of iterations	5.51643e+8
Number of iterations per pixel	1149.24
Number of iterations per second	3.10592e+7
Distance estimation algorithm	analytic linear
Percentage of wrong distance estimations	0
Distance of camera to fractal surface	2.21767e-8

Figure 3.3: Statistics Tab with histogram data

In the Statistics (enable in *View* menu) you can see *Percentage of Wrong Distance Estimations* (“Bad DE”). This number is the percentage of image pixels which potentially have big errors in distance estimation calculation (estimated distance was much too high). It is visible as a noise on the image. As a general rule less than 0.1 is good, but it is case specific and 3.0 sometimes is OK and 0.0001 sometimes is not.

4 Ray-marching - Maximum number of iterations vs. distance threshold condition

The *ray marching distance threshold* is the condition where the photon marching along the ray comes within a specified distance from the fractal surface and the ray-marching stops. This controls the size of the detail in the image, and is normally set to vary such that greater detail is obtained for the surface closest to the camera, (in the further regions of the fractal the distance threshold will be larger such that only bigger details are visible). Enabling *Constant Detail Size* on the *Rendering Engine* tab will make the distance threshold uniform.

There are two modes of stopping the ray-marching of each image pixel.

1st case: Stop ray-marching at distance threshold (*Stop at maximum iteration* is disabled).

2nd case: Stop ray-marching at point when a maximum number of iterations is reached (*Stop at maximum iteration* is enabled).

First important note: *Stop at maximum iteration* doesn't control the fractal iteration loop. It controls only ray-marching. The iteration loop always runs to achieve Bailout, (then if bailout is not reached the iteration stops at Maxiter). (see page 6)

On figure 4.1 ray-marching stops at distance threshold. In most cases the fractal iteration loop stops on bailout condition, (because away from surface it is not possible to reach Maxiter). It makes rendering of fractals much faster.

On figure 4.2 ray-marching stops at the photon step when the maximum number of iterations is reached (ray-marching distance threshold is ignored). In many cases iteration loop stops on bailout condition (away from fractal surface), but on the fractal surface the maximum number of iterations is calculated (when bailout is not reached).

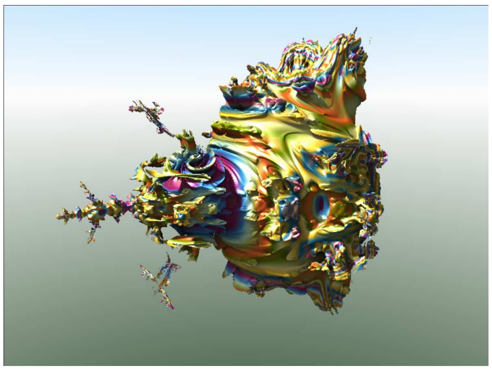


Figure 4.1: Example for 1st case - stop ray-marching at distance threshold

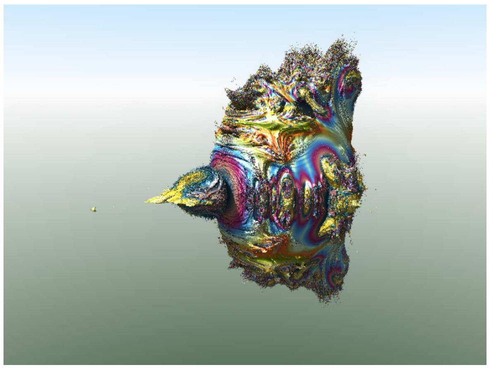


Figure 4.2: Example for 2nd case: Stop ray-marching at Maxiter

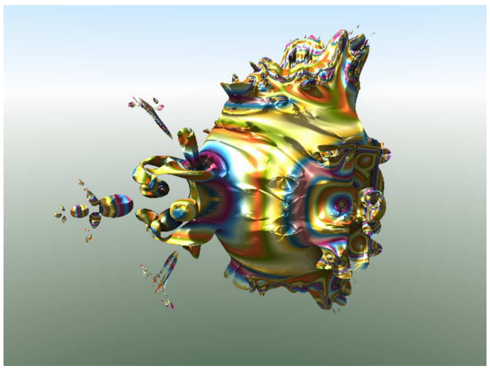


Figure 4.3: Example for 1st case: Stop ray-marching at bailout with low Maxiter

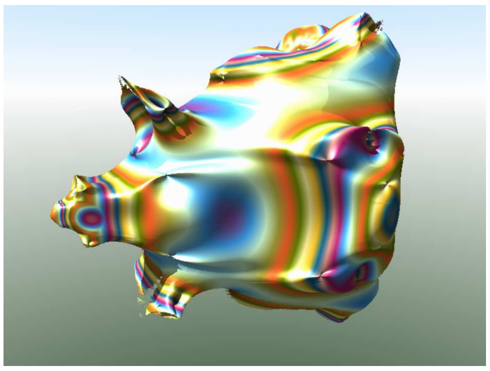


Figure 4.4: Example for 2nd case: Stop ray-marching at maxiter with low maxiter

Figure 4.3 shows what happens if maximum number of iterations is set to 4. Even if Maxiter is reached the ray-marching is continued until the ray marching distance threshold is reached.

Figure 4.4 shows case when maximum number of iterations is reached. Ray-marching is stopped even if distance threshold is not reached.

5 Iteration loop

In section 2.1 it was mentioned that fractals are calculated by repeating a formula (*iterating*) in an iteration loop. The integer i is used to represent the *iteration count* number.

The iteration count starts with $i = 0$, then at the end of each iteration the count number is increased by 1, and the next iteration of the formula commences (e.g. iteration count 0, 1, 2, 3, ...). The iterating continues until *termination conditions* are met, which is either when the iteration count $i = maxiter$ or when the *bailout* condition is achieved.

This section explains the calculations within the iteration loop.

A fractal formula is built from mathematical equations. These equations can be modifications of the Mandelbrot Set equation (e.g Mandelbulb) and also other mathematical equations.

The equations are made from mathematical operators ($+, -, *, /$) and can include mathematical functions (e.g. sin, cos, tan, exp, log, sqrt, pow, abs) and also mathematical conditions (e.g. if x $>$ y then "compute following equation(s)", if i $>$ 4 then "compute following equation(s)").

The equations are applied to vector z or any parts of z (i.e the z.x, z.y and z.z components)

Examples:

`z.x = fabs(z.x);` is using the function fabs() (which is floating-point version of abs()), where *z.x* is assigned the absolute value of *z.x*.

`if (z.x - z.y < 0.0) swap(z.x, z.y);` is using the conditional function if() to determine if the values of z.x and z.y should be swapped.

`z *= 3.0` is using the operator * to multiply the all components of vector z by 3.0.

A set of equations that have a specific function within the formula are called *transforms*, e.g rotation, scale.

The generally a formula is constructed from one or more transforms, which are constructed from equations.

With each iteration of the formula, the point being iterated is *mapped* (moved) to new coordinates as a result of the mathematical equations.

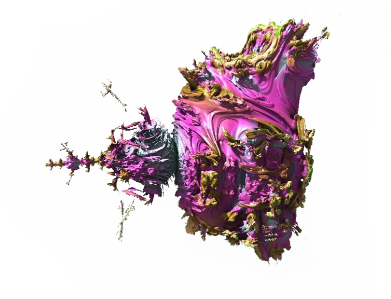
5.1 Single formula fractals

The simplest 3D fractals are calculated by iterating a single fractal formula. More complex fractals are made by iterating a mix of formulas, adding extra transforms, and/or including additional conditions.

Below there are 3 examples of fractals formulas written in C language code

5.1.1 Mandelbulb Power 2

This formula is a modified Mandelbrot Set equation, expanded to 3rd dimension. A cross section at $z_z = 0$ looks exactly the same as Mandelbrot Set.

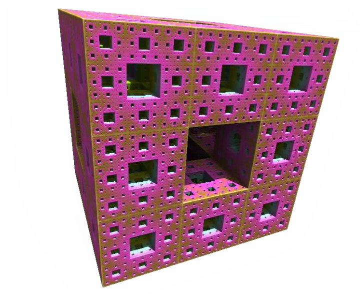


Listing 1: Formula > Mandelbulb Power 2

```
double x2 = z.x * z.x;
double y2 = z.y * z.y;
double z2 = z.z * z.z;
double temp = 1.0 - z2 / (x2 + y2);
double newx = (x2 - y2) * temp;
double newy = 2.0 * z.x * z.y * temp;
double newz = -2.0 * z.z * sqrt(x2 + y2);
z.x = newx;
z.y = newy;
z.z = newz;
```

5.1.2 Menger Sponge

This formula is an Iterated Function System (IFS). It contains several transforms, some of them conditions.



Listing 2: Formula > Menger Sponge

```
z.x = fabs(z.x);
z.y = fabs(z.y);
z.z = fabs(z.z);
if (z.x - z.y < 0.0) swap(z.x, z.y);
if (z.x - z.z < 0.0) swap(z.x, z.z);
if (z.y - z.z < 0.0) swap(z.y, z.z);
z *= 3.0;
z.x -= 2.0;
z.y -= 2.0;
if (z.z > 1.0) z.z -= 2.0;
```

5.1.3 Box Fold Bulb Pow 2

This formula is made from a set of different transforms. It is a good example of how a fractal formula can be more complicated than the *Mandelbrot Set* formula.

First part is a “box fold” transform which conditionally maps the point in x,y,z directions. Second part is a “spherical fold” which does conditional scaling in a radial direction. The end of formula is the same as *Mandelbulb Power 2*.

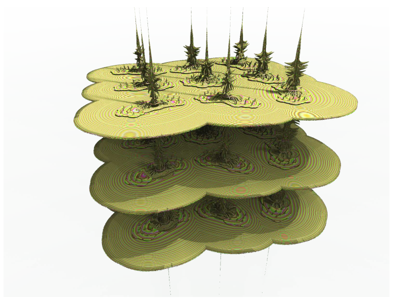
Listing 3: Formula > Box fold Power 2

```
//box fold
if (fabs(z.x) > fractal->foldingIntPow.foldFactor)
        z.x = sign(z.x) * fractal->foldingIntPow.foldFactor
                 * 2.0 - z.x;
if (fabs(z.y) > fractal->foldingIntPow.foldFactor)
        z.y = sign(z.y) * fractal->foldingIntPow.foldFactor
                 * 2.0 - z.y;
if (fabs(z.z) > fractal->foldingIntPow.foldFactor)
        z.z = sign(z.z) * fractal->foldingIntPow.foldFactor
                 * 2.0 - z.z;

//spherical fold
double fR2_2 = 1.0;
double mR2_2 = 0.25;
double r2_2 = z.Dot(z);
double tglad_factor1_2 = fR2_2 / mR2_2;

if (r2_2 < mR2_2)
{
        z = z * tglad_factor1_2;
}
else if (r2_2 < fR2_2)
{
        double tglad_factor2_2 = fR2_2 / r2_2;
        z = z * tglad_factor2_2;
}

//Mandelbulb power 2
z = z * 2.0;
double x2 = z.x * z.x;
double y2 = z.y * z.y;
double z2 = z.z * z.z;
double temp = 1.0 - z2 / (x2 + y2);
zTemp.x = (x2 - y2) * temp;
zTemp.y = 2.0 * z.x * z.y * temp;
zTemp.z = -2.0 * z.z * sqrt(x2 + y2);
z = zTemp;
z.z *= fractal->foldingIntPow.zFactor;
```



5.1.4 Processing of single formula fractals

Single formula fractals are simply iterated several times until termination conditions are met, as shown in figure 5.1.

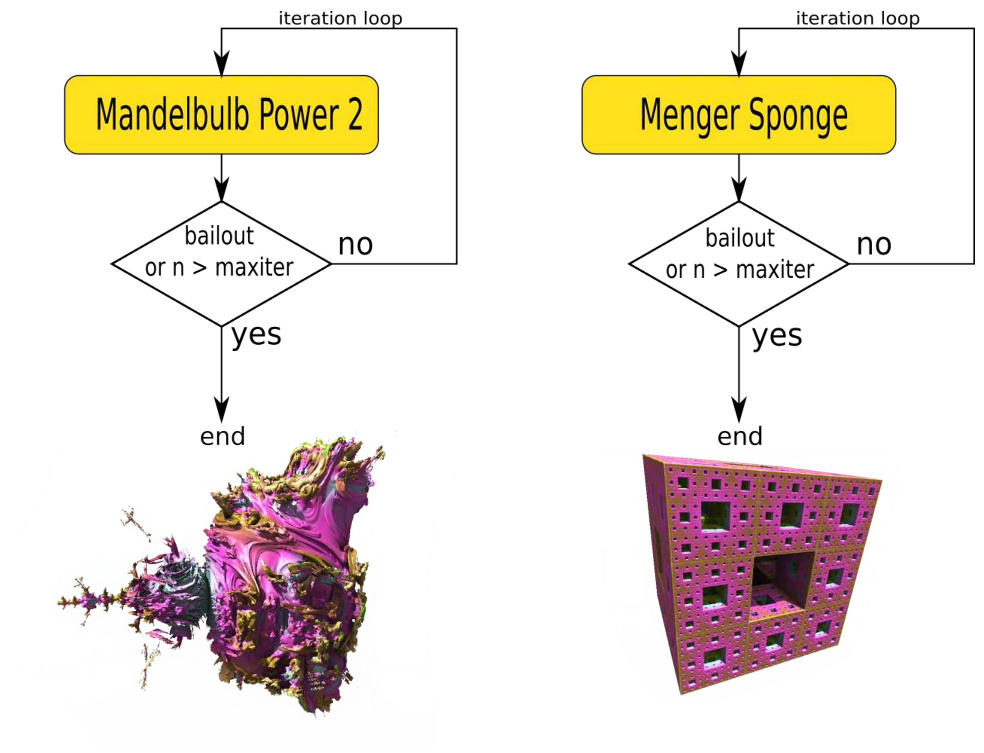


Figure 5.1: Examples of simple Iteration loops with one formula

When the calculation of the iteration loop finishes the resulting final value of z is used to estimate the distance to the fractal body and to calculate the color of the surface.

5.2 Hybrid fractals

Hybrid fractals are constructed by using more than one formula in the iteration loop. This way new variations of fractal shapes can be achieved. There are many different fractal formulas and transforms available in the Mandelbulber program, which allows the user to create a vast variety of hybrid shapes.

5.2.1 Iteration loop of hybrid fractals

In general hybrid fractals are calculated in a similar way to single formula fractals. The calculation consists of the iteration loop, *maxiter* and *bailout* condition. The difference is that when *hybrid mode* is enabled, a user can create a *sequence* of up to nine different fractal formulas (or transforms) inside the iteration loop.

By default the program works in single fractal formula mode, where you can only configure the parameters of the formula tab in the first slot, (#1).

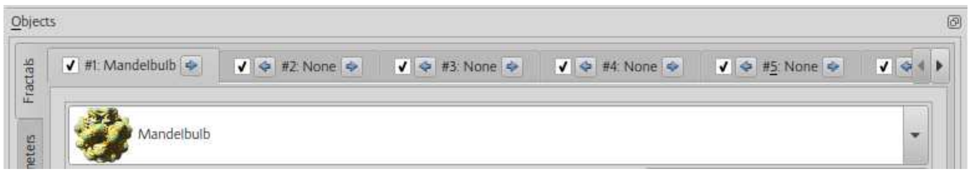


Figure 5.2: Fractal Tab - Formula only in first slot

There are two ways to enable hybrid fractals:

- Click in any slot with a number higher than one. The program will ask if you want to enable hybrid fractals or boolean mode. Select *Enable hybrid fractals*
- Go to *Objects / Hybrid* tab. Tick *Enable hybrid fractals* checkbox.

Once hybrid fractals has been enabled, a user can select additional formulas from the dropdown menus in any of the nine formula slots, as shown in figure 5.3. In this figure *Mandelbulb - Power 2* is selected in slot #1, *Menger Sponge* in slot #2 and *Box Fold Bulb Pow 2* in slot #3. These formulas will be used in the next examples.

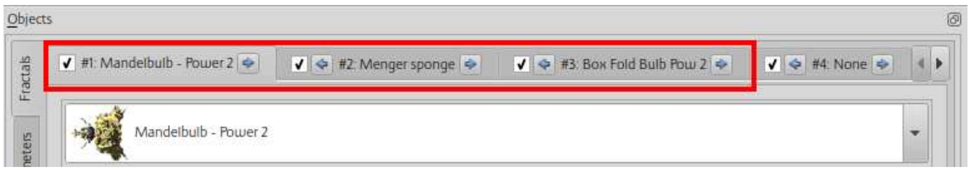


Figure 5.3: Fractal Tab - Multiple formula slots filled

Each formula's parameters can be configured in the formula tab opened in an enabled slot.

The iteration count numbers determine when in the sequence each formula is calculated.

The sequence is in the order of the enabled formula slots from #1 to slot #9, (e.g. If the sequence is calculating formulas in slots #1 and #5, then the iteration loop repeats the sequence of slot #1 calculation followed by slot #5 calculation.)

How the sequence will work depends on the following selections:

- Which fractal formulas are selected in the formula slots
- How many iterations are assigned to each formula
- The range of iteration numbers when the formula will be used
- From which fractal slot the sequence will be repeated

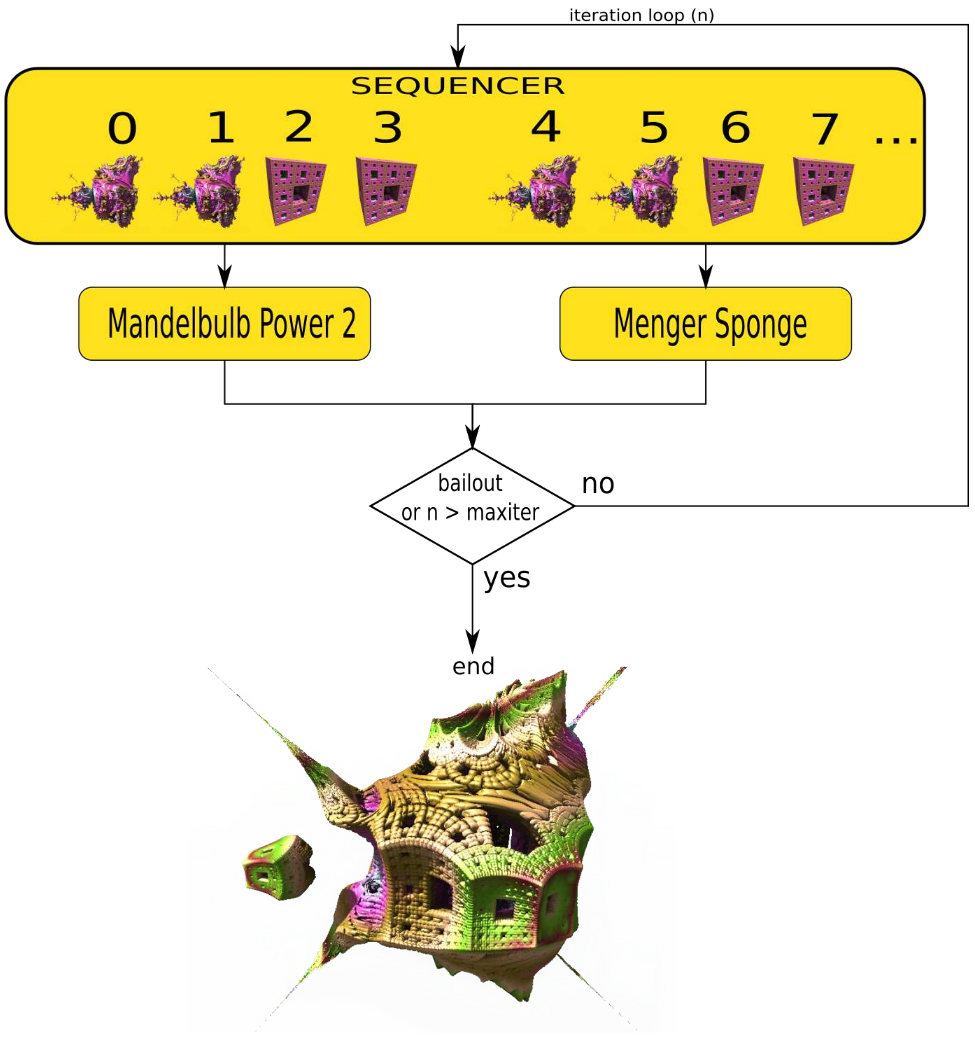


Figure 5.4: Complex Iteration loop with hybrid fractal

5.2.2 One iteration for each slot

The simplest way to create a hybrid fractal is a sequence where formulas are calculated one after another, then the sequence is repeated until termination conditions are met.

In figure 5.5, the sequence consists of one *Mandelbulb - Power 2*, one *Menger Sponge* and one *Box Fold Bulb Pow 2*. The length of the sequence is three iterations, so after every third iteration the sequence repeats from the first slot. The numbers shown are the Iteration Count, starting at i = 0. The count increases by 1 after every iteration performed in the iteration loop.

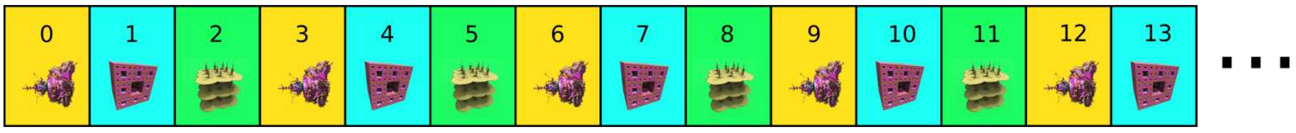


Figure 5.5: Hybrid sequence - Simple sequence using three different formulas

This sequence gives a shape combining properties of all three formulas, see figure 5.6.

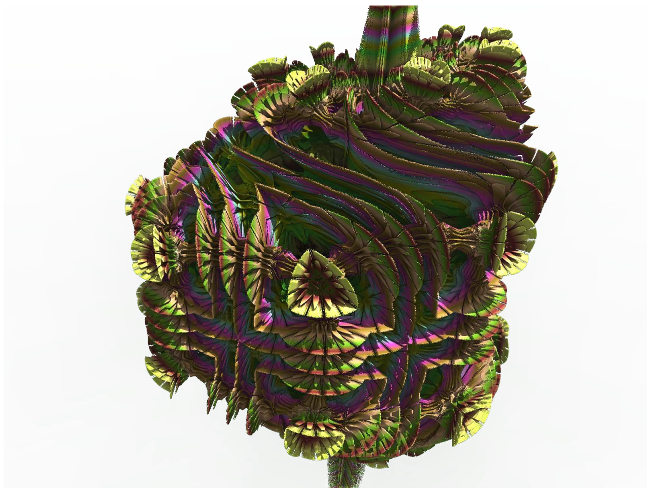


Figure 5.6: Hybrid sequence render - Simple sequence using three different formulas

Because the first iteration is (slot #1) *Mandelbulb - Power 2*, the general shape of the fractal will be simlar in shape to the *Mandelbulb - Power 2*.

Note: Generally, the first few iterations of a fractal strongly influence the final hybrid fractal shape.

In the next iteration *Menger Sponge* formula is used. A single iteration of this formula produces the shape of figure 5.7.

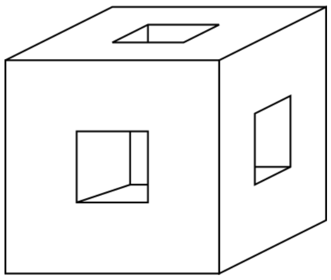


Figure 5.7: Single iteration of the Menger Sponge

Some features of this shape are transferred to the generated shape of the hybrid fractal.

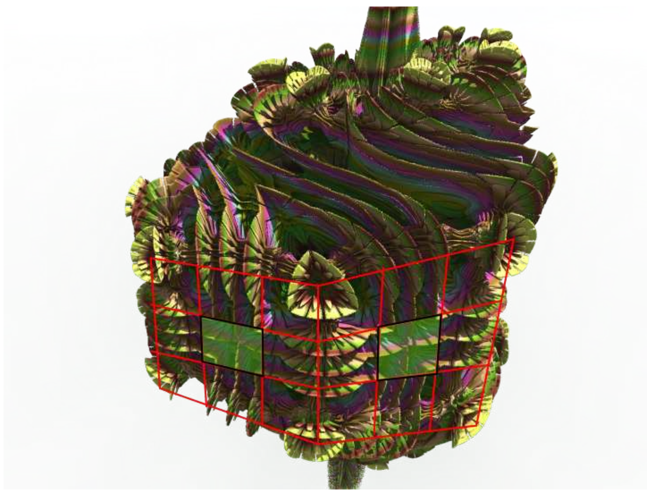


Figure 5.8: Hybrid with Menger Sponge features marked in red

The Menger Sponge shape is distorted, because *Mandelbulb - Power 2* has already deformed the space.

The third formula *Box Fold Bulb Pow 2* adds leaf-like features to the shape.

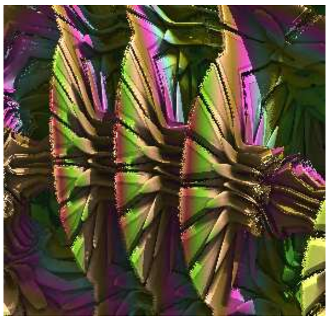


Figure 5.9: Hybrid close up of leaf-like shapes produced by 'Box Fold Bulb Pow 2' formula

5.2.3 More iterations for each slot

With each slot a user can define how many times each fractal formula will be used in the sequence. On each of the formula tabs there is a parameter named *Iterations* which is set to 1 by default. This is the number of iterations (repeats) of the formula performed before the loop moves on to the next formula slot in the sequence. If this value is increased to 2 on the first and second formula slots in our example, then the sequence of the formulas will be as shown in figure 5.10.

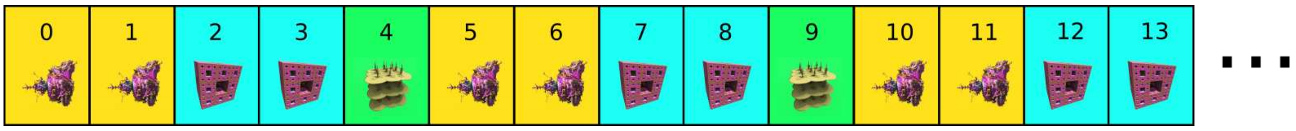


Figure 5.10: Hybrid sequence - The first and second slots set to 2 repeat iterations

The first and second formulas are repeated twice and the third formula only once. The resulting fractal shape is shown in figure 5.11:

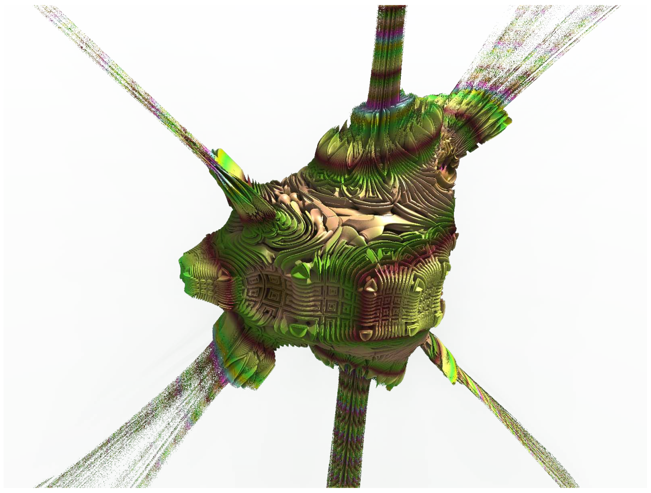


Figure 5.11: Hybrid sequence result - The first and second slots set to 2 repeat iterations

Because the *Mandelbulb - Power 2* calculation is repeated for two iterations at the beginning, the shape of this initial formula strongly influences the final shape of the hybrid fractal.

If the parameter *Iterations* on the second slot is set to 10, then the *Menger Sponge* formula is used from iteration 2 to iteration 11.

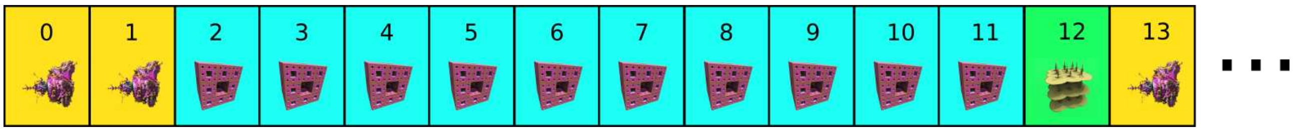


Figure 5.12: Hybrid sequence - The second slot set to 10 repeat iterations

As above, the initial shape is mainly defined by the first two iterations of *Mandelbulb - Power 2*, but the high number of *Menger Sponge* iterations makes the *Menger Sponge* features become more apparent.

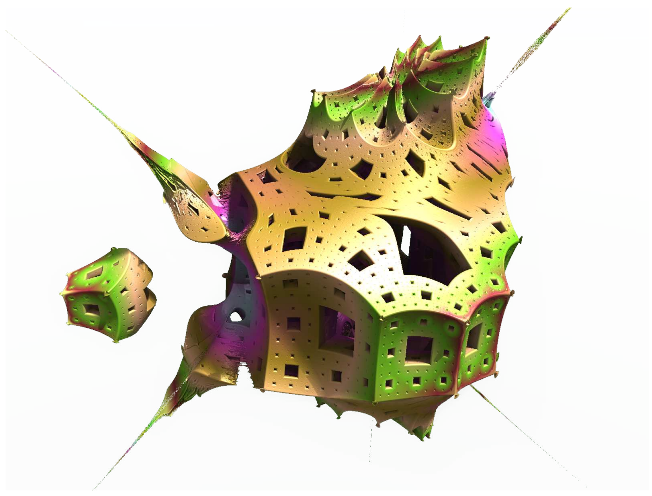


Figure 5.13: Hybrid sequence result - The second slot set to 10 repeat iterations

5.2.4 Range of iterations for slot

The sequences can become more complicated by specifying the range of iterations when a formula will be calculated in the loop.

On each formula tab the parameters *Start at iteration* and *Stop at iteration* are used to define this range.

When computing the iteration loop, the program is moving through the enabled formula slots, checking formula iteration range conditions. If the current Iteration Count number is within the range then a calculation of that formula is performed, and the Iteration Count is increased by 1. If the Iteration Count number is outside the iteration range condition, then the formula is skipped (i.e. no calculation and therefore the iteration count remains unchanged). The program then moves to the next enabled slot in the sequence.

The second formula slot (*Menger Sponge*) in the sequence shown in figure 5.14, has the iteration range set to *from 4 to 250.*

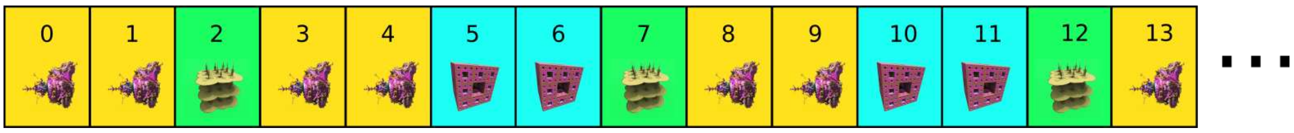


Figure 5.14: Hybrid sequence - Range of iteration set to 4-250 on second slot

During the first pass of the sequence *Menger Sponge* formula could not be used at iterations 2 and 3, because *Start at iteration* was when i = 4 for this formula, and therefore the slot was skipped. During the second pass of the sequence *Menger Sponge* formula was used at iterations 5 and 6, because those iterations were inside the defined range of iterations.

The shape of the resulting fractal is shown in figure 5.15

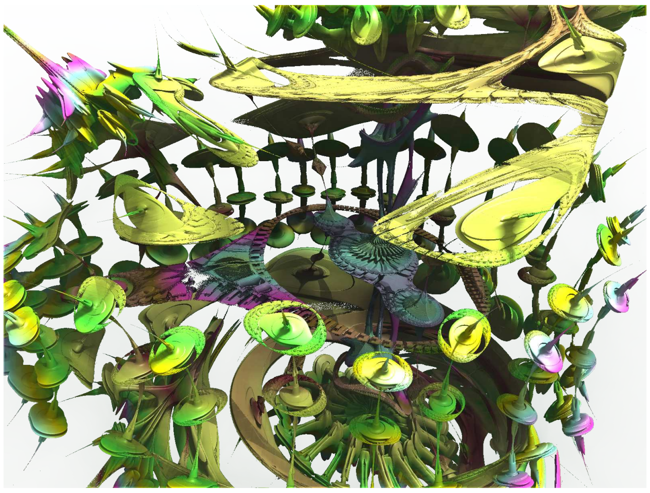


Figure 5.15: Hybrid sequence result - Range of iteration set to 4-250 on second formula slot

Because at iteration number 2 *Menger Sponge* formula was skipped, *Box Fold Bulb Pow 2* formula has much more influence on the final shape.

5.2.5 Changed order in sequence

The order of fractal formulas can be easily changed between slots with the use of the arrow buttons.

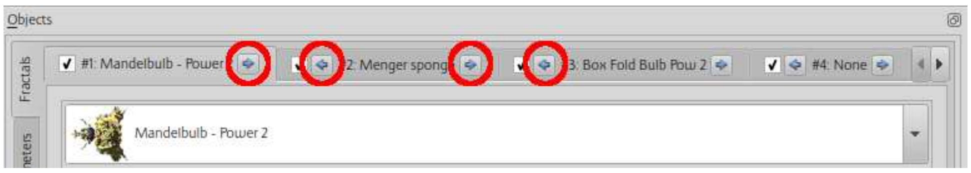


Figure 5.16: Fractal tabs with highlighted tab-arrows

These buttons swap the fractal tabs between the slots, and therefore the formula's position inside the sequence will change. All formula parameters setting are moved in the swap.

Example based on first case shown in section 5.2.3: Swapped *Mandelbulb - Power 2* and *Menger Sponge* creates the sequence shown in figure 5.17.

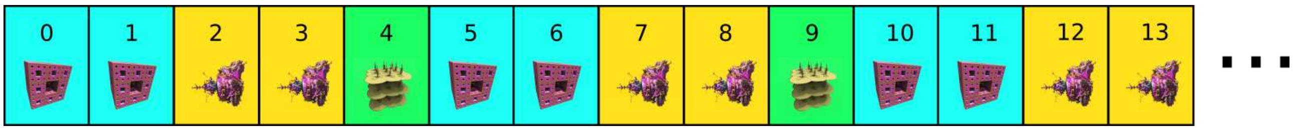


Figure 5.17: Hybrid sequence - Swapped tab one and two

As evident in figure 5.18, the shape of the fractal is completely different.

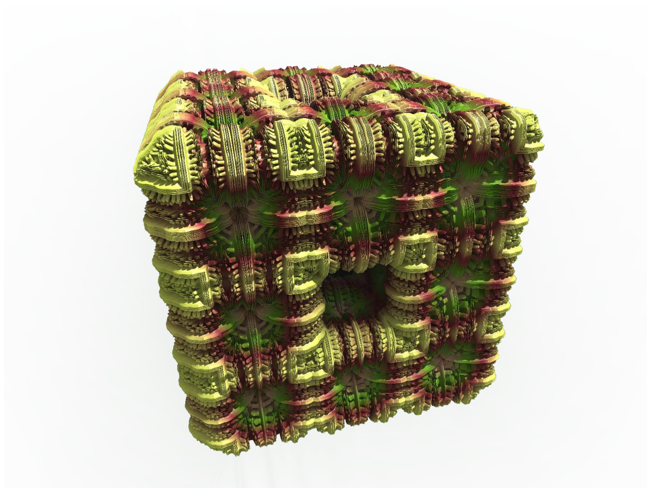


Figure 5.18: Hybrid sequence render - Swapped tab one and two

Now the first *Menger Sponge* formula creates the initial shape of the fractal, and *Mandelbulb - Power 2* only modifies the details.

Even if the same fractal formulas are used in each slot, and for the same number of iterations, the final shape will strongly depend on the parameter settings of the first few formulas that are iterated in the sequence.

There are also a few formulas and transforms which have a very strong influence on the final shape, and these are often run for just 1 or 2 iterations during the iterating of the fractal.

6 Navigation

To set the current view there are two elements:

Camera represents a point where the camera is located

Target represents the point onto which the camera will focus (the camera is *always* looking at the target.)

6.1 Camera and Target movement step

The relationship between the camera point and the target point can be altered manually by changing the numbers in the edit fields, or by navigating with distance and rotation "steps" defined by the user.

For rotations, the camera is moved by the parameter **rotation step** (default 15 degrees). For movements of the camera and/or the target in a linear direction, the parameter **step** (default 0.5) is used. There are two modes for its use:

6.1.1 Relative step mode

The **step** for moving the camera and/or target in a linear direction is calculated relative to the estimated distance from the surface of the fractal. The closer to the surface that the camera is located, the smaller the step. This prevents the camera moving to a location beneath the surface of the fractal.

The actual step is equal to the distance from the fractal multiplied by the parameter **step**.

Example: If the step is set at 0.5 and the nearest point of the fractal is 3.0, the camera will be moved 1.5 (no matter in which direction).

Relative step mode makes navigation easier, because a user does not need to think about the movement size required to avoid the camera moving into the fractal.

In animations this mode is recommended when camera is approaching the surface of the fractal.

6.1.2 Absolute step mode

Step movement of the camera and/or target is fixed. Therefore if the step is set at 0.5, the movement will be 0.5 in the direction of the arrow key or mouse pointer.

This mode is recommended for flight animation with the camera flying at a fixed (or strictly controlled) speed.

6.2 Linear camera and target movement modes using the arrow buttons

A user can navigate by operating the arrow buttons on the Navigation dock, with the user defined steps.

There are three modes for changing the relationship between camera and target:

- move camera and target
- move only camera
- move only target

6.2.1 Move camera and target mode

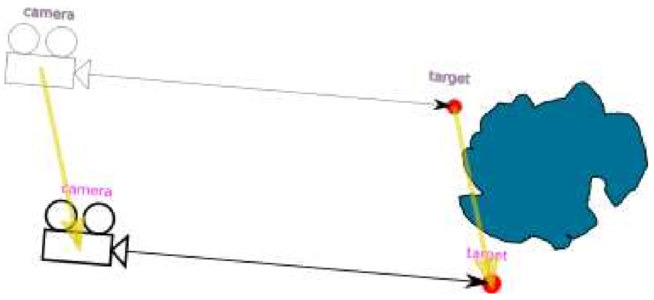


Figure 6.1: Movement mode - camera and target

Arrows move both the camera and the target by the same distance in the same direction. The angle of camera rotation does not change.

6.2.2 Move camera mode

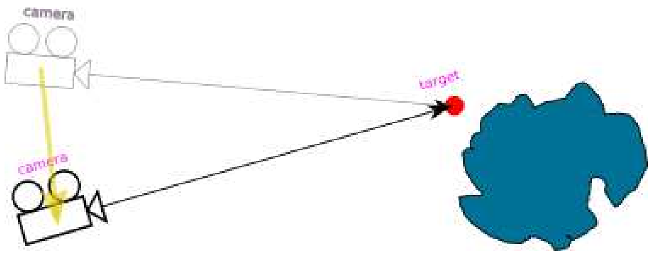


Figure 6.2: Movement mode - camera

Moves only the camera and rotates it in respect to the motionless target.

6.2.3 Move target mode

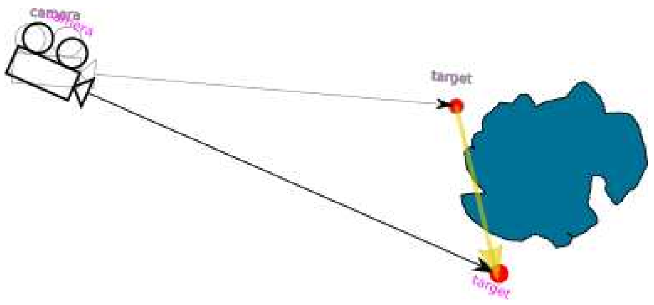


Figure 6.3: Movement mode - target

Moves only the target while maintaining a fixed camera position. The camera rotates following the target. **Note:** In Relative Step Mode, the target is moved by distance related to distance of target to fractal surface. If target is inside the fractal (distance = 0), then this option will not work with Relative Step Mode.

6.3 Linear camera and target movement modes using the mouse pointer

A user can move the camera by selecting a point on the image with the mouse pointer, and then using either left mouse button (move forward) or right button (move backward).

6.3.1 Move camera and target mode

The target is moved to the point selected by the mouse. The camera is moved towards selected point by a user defined step, and rotated.

In relative step mode, the camera is moved by distance equals to *distance_to_indicated_point* multiplied by *step* parameter.

In absolute step mode, the camera is moved by distance equals to *step* parameter.

6.3.2 Move camera mode

The camera is moved by the step (absolute or relative) in the direction of the mouse pointer, rotating the camera to look at the target. The target remains stationary.

6.3.3 Move target mode

The target is moved to the point selected by the mouse. The camera remains at the same point but rotates following the target.

6.4 Camera rotation modes using the arrow buttons

6.4.1 Rotate camera

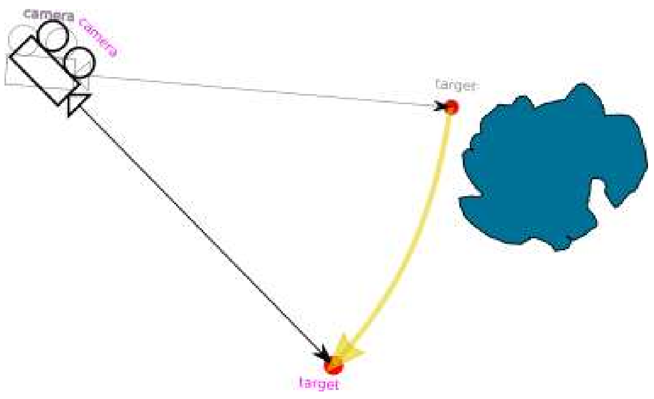


Figure 6.4: Rotation mode - around camera

The camera is rotated by the rotation step around its axis and the target is moved accordingly. This is the standard mode for rotation of the camera.

6.4.2 Rotate around target

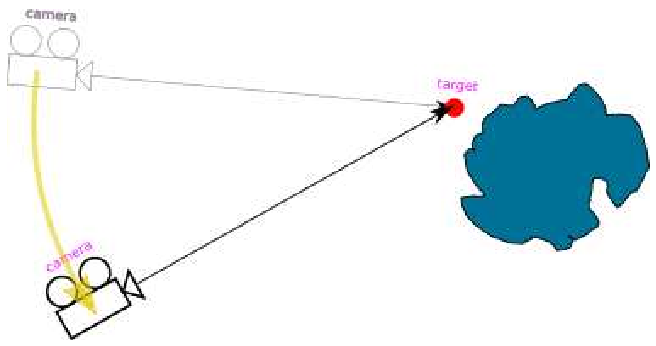


Figure 6.5: Rotation mode - around target

The camera is moved around the stationary target by the rotation step, maintaining a constant distance to the target. The camera is rotated to look at the target.

6.5 Reset View

Camera Position is reset, by being zoomed out from the fractal but still maintaing the camera angles.

If the rotations are changed to zero before using Reset View, the camera will then be zoomed out from the target, and rotated to look down the y axis.

6.6 Calculation of rotation angles modes

6.6.1 Fixed-roll angle

In this mode, the angle gamma (roll) is constant. Pan the camera left or right always takes place around the global vertical Z-axis (not the render window vertical axis).

This mode can be likened to an aircrafts controls, where all turns are relative to the aircraft's axis, not the ground below. Rotate up / down raises / lowers the nose of the aircraft. Rotate left / right turns in the directions of the wings. Tilting the camera buttons tilts the aircraft

When the camera is pointing straight up or down, or when it is upside down in this mode, it is quite difficult to predict the result of the turn.

6.6.2 Straight rotation

The camera is rotated around its own local axis (local vertical axis is the render window vertical axis.)

This mode is a more intuitive way to rotate the camera, e.g., turn the camera left always give the visual effect of the camera rotating in that direction. The rotation angles are automatically converted so they are appropriate for the selected direction. This mode changes the gamma angle (roll).

6.7 Camera rotation in animations

With animation, the camera point and the target point can move independently following their own trajectories, with the camera always looking towards the target point. It is important to be aware that the rotation angle of the camera is the result of the camera coordinates and the target coordinates.

There are various ways of animating, depending on the objective.

The following example is a flight animation, with the camera trajectory approaching a location, with the camera rotating simultaneously so that the location is always observed, (as shown in the figure 6.6). The camera positions represent three keyframes.

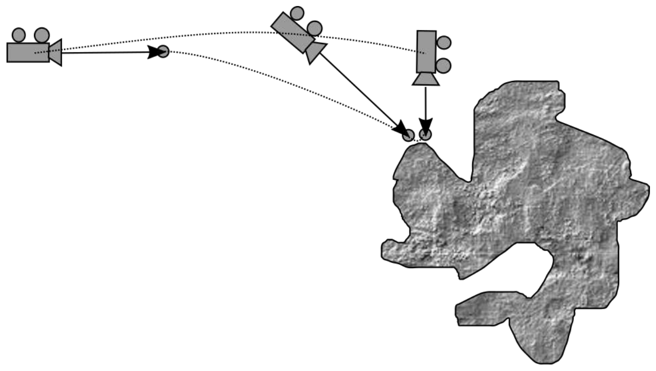


Figure 6.6: Keyframe Animation with differing distances camera to target

Between the first and second keyframes, the camera and target both move large distances. But between the second and third keyframes, the camera moves a much greater distance than the target. This can sometimes lead to unexpected camera rotations between keyframes.

To compensate for this, on the Keyframe navigation tab use the button *Set the same distance from the camera for all the frames* This adjusts all keyframes by setting a constant distance between camera and target. It is important to note that the use of this function does not change the visual effect for the keyframes, and will help correct interpolation. See figure 6.7.

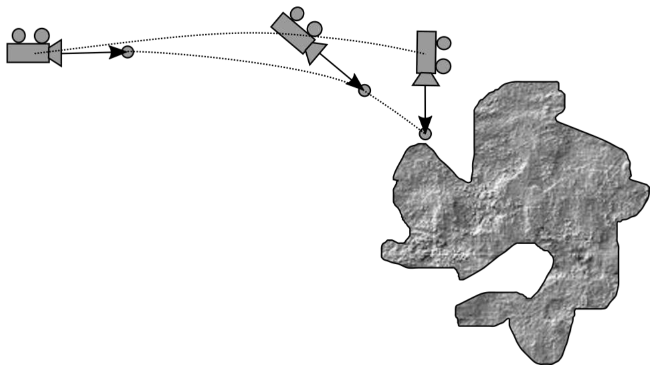


Figure 6.7: Keyframe Animation with equal distances camera to target

7 Interpolation

Interpolation functions, which calculate intermediate values, are used to make smooth parameter transitions between keyframes. There is no need for manual editing of every animation camera position and fractal parameters. frame. A limited number of keyframes is enough to define good looking animation.

7.1 Interpolation types

Parameters in Mandelbulber can be transitioned using several interpolation modes:

1. None
2. Linear
3. Linear angle
4. Akima
5. Akima angle
6. Catmul-Rom
7. Catmul-Rom angle

The chart in figure 7.1 shows a comparison between different interpolation modes.

The choice of mode, greatly effects the animation.

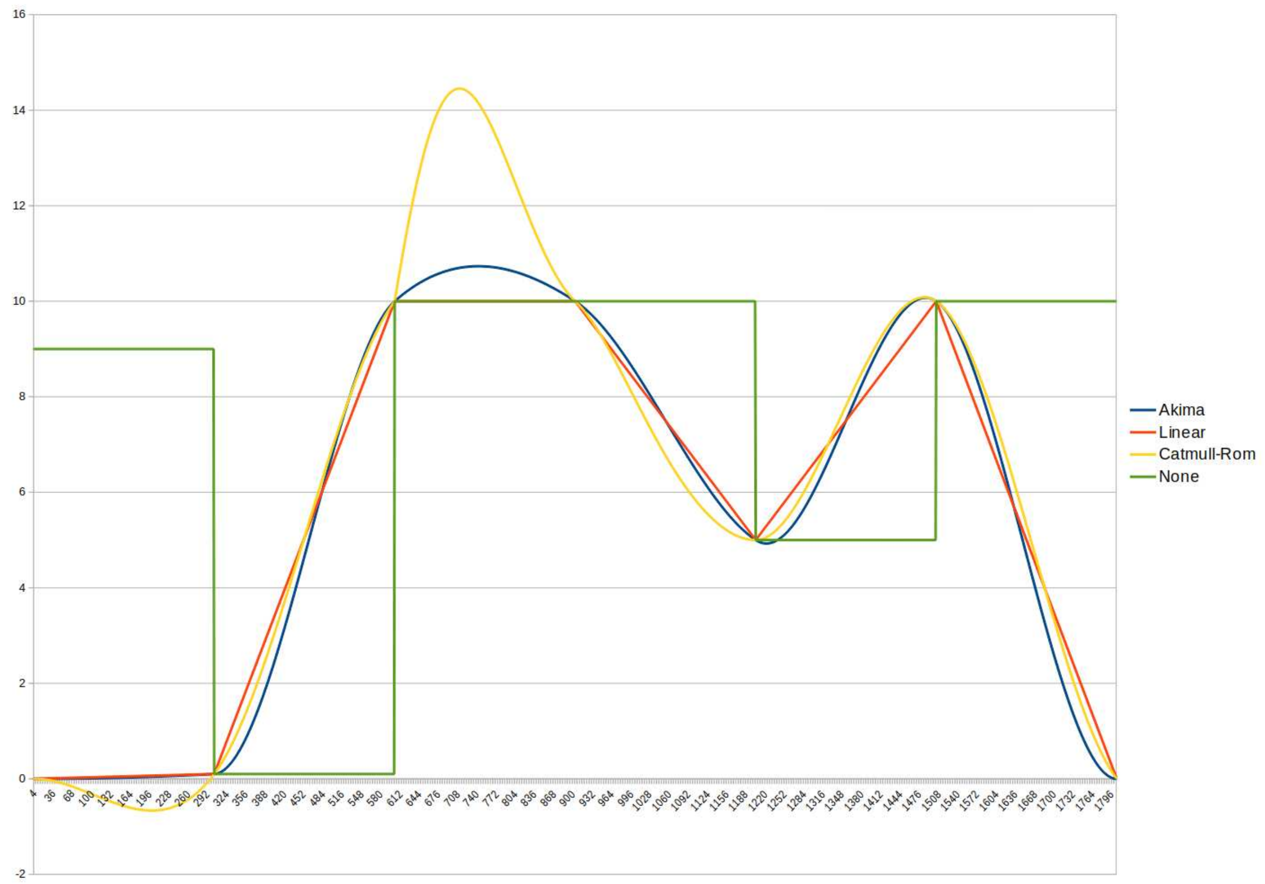


Figure 7.1: Interpolation types

7.1.1 Interpolation - None

The parameter remains constant between keyframes. The parameter will change abrubtly at the any keyframe that has a change in value. This mode can be used with boolean values or with variables which have to be kept at a constant value for a number of keyframes.

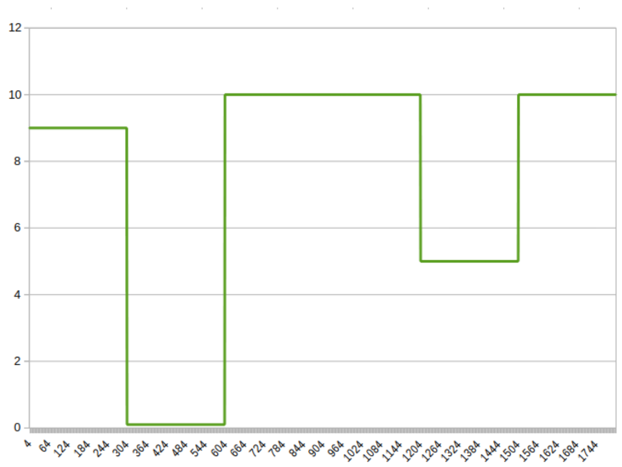


Figure 7.2: Interpolation - None

7.1.2 Interpolation - Linear

The value of the parameter is interpolated using linear functions.

$$y(x) = y_i + (x - x_i)\frac{y_{i+1} - y_i}{x_{i+1} - x_i}$$
$$x_i \leq x \leq x_{i+1}$$

Changes of parameters are easy to predict. There are no overshoots. This interpolation mode is good for fractal parameters and material properties. It is generally not recommended to be used for camera or object movement paths, because of the abrubt changes of speed.

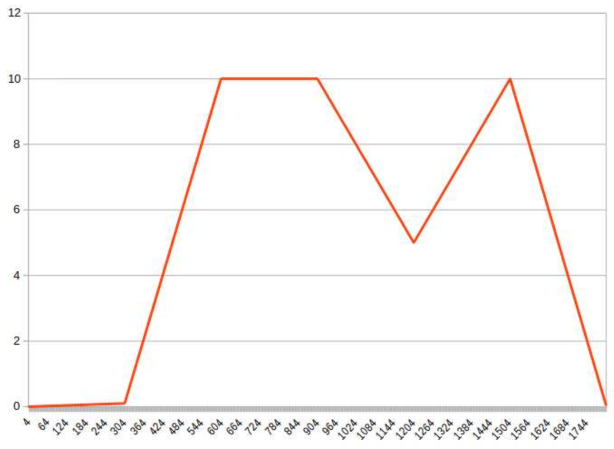


Figure 7.3: Interpolation - Linear

7.1.3 Interpolation - Linear angle

This interpolation mode works like *Linear*, but is prepared of angular parameters. If value exceed 360 degrees, then will go back to zero.

7.1.4 Interpolation - Akima

The Akima interpolation is a continuously differentiable sub-spline interpolation. It is built from piecewise third order polynomials.

$$y(x) = a_0 + a_1(x - x_i) + a_2(x - x_i)^2 + a_3(x - x_i)^3$$
$$x_i \le x \le x_{i+1}$$

This interpolation function produces smooth transitioning through the keyframes. It is very good for most animated parameters. It is used for camera and target animation and for many other parameters which should be animated in a smooth way.

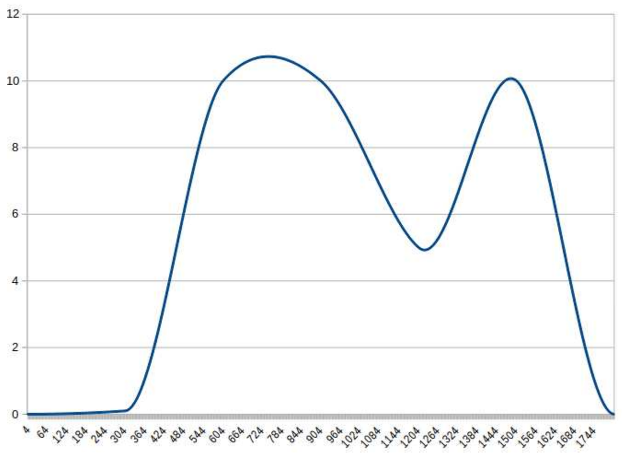


Figure 7.4: Interpolation - Akima

7.1.5 Interpolation - Akima angle

This interpolation mode works like *Akima*, but is written for angular parameters. If a value exceeds 360 degrees, then it will reset back to zero.

7.1.6 Interpolation - Catmul-Rom

Catmull-Rom splines are cubic interpolating splines formulated such that the tangent at each point $y_i(x_i)$ is calculated using the previous and next point on the spline.

$$y(x) = 0.5 \begin{bmatrix} 1 & x - x_i & (x - x_i)^2 & (x - x_i)^3 \end{bmatrix} \begin{bmatrix} 0 & 2 & 0 & 0 \\ -1 & 0 & 1 & 0 \\ 2 & -5 & 4 & -1 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} y_{i-1} \\ y_i \\ y_{i+1} \\ y_{i+2} \end{bmatrix}$$

This interpolation gives very smooth results. Animated objects looks likethey are made of springy materials. It can be used to animate fractal parameters and also the camera path. This interpolation mode can produce oscillations, so it has to be used carefully. Figure 7.5, shows where interpolated values went below zero, where all of the keyframe values were higher than zero. The ocsillation problem is commonly seen near the begining and end of an animation

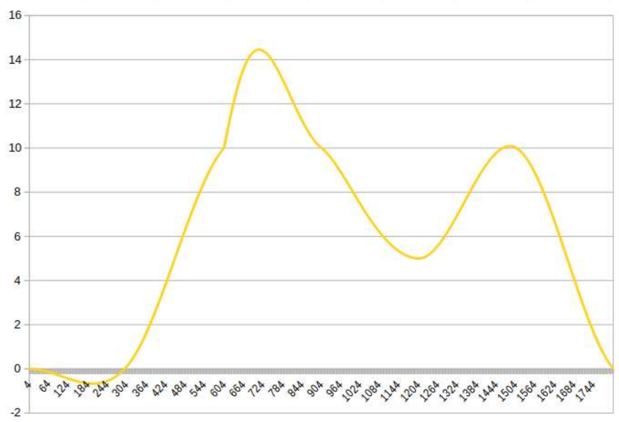


Figure 7.5: Interpolation - Catmul-Rom

7.1.7 Interpolation - Catmul-Rom angle

This interpolation mode works like *Catmul-Rom*, but is prepared of angular parameters. If value exceed 360 degrees, then will go back to zero.

7.2 Catmul-Rom / Akima interpolation - Advices

7.2.1 Collision

Fast approaching the obstacle may cause inadvertent drag to the camera towards the center of the object. It is recommended to maintain the principle that one keyframe does not reduce the distance to the object more than five times.

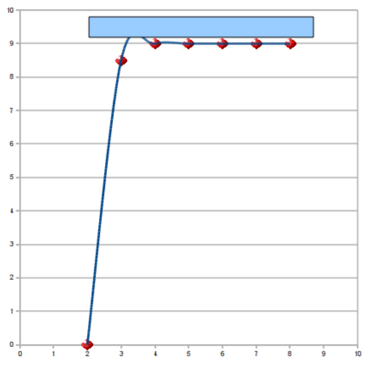


Figure 7.6: Catmul-Rom with collision

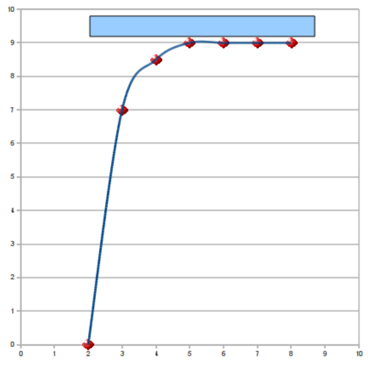


Figure 7.7: Catmul-Rom without collision

7.2.2 Fly through the gap

It is recommended to place a keyframe at the point where the camera flies through a hole / gap in the fractal.

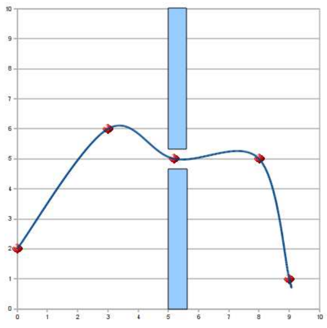


Figure 7.8: Interpolation - Catmul-Rom path through a hole

7.2.3 Proper conduct cameras between objects

Figure 7.9 shows how keyframes should be located between objects to avoid collisions caused by interpolation functions.

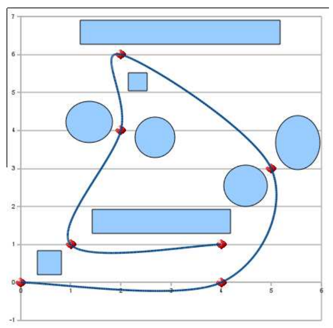
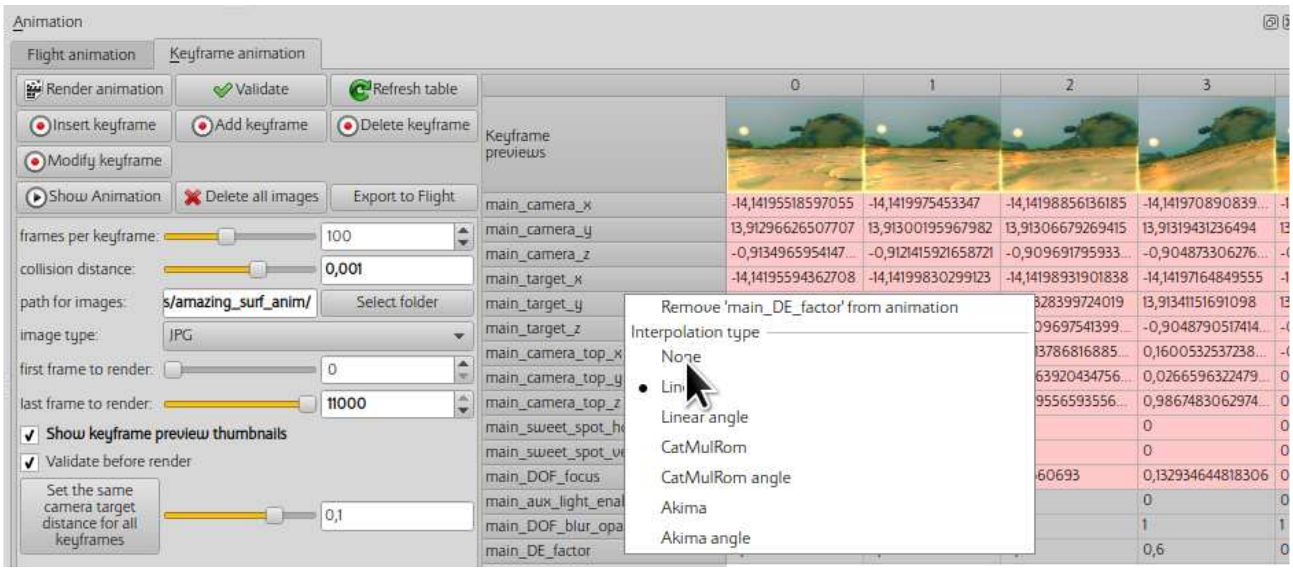


Figure 7.9: Interpolation - Catmul-Rom path evading different obstacles

7.3 Changing interpolation types

To change the interpolation algorithm, right click on the parameter list and the options appear. In this example the *main_DE_factor* have been changed from Akima to Linear. Interpolation type is color coded e.g. Linear parameters are highlighted in grey.



frames per keyframe: 100
collision distance: 0,001
path for images: s/amazing_surf_anim/
image type: JPG
first frame to render: 0
last frame to render: 11000
✓ Show keyframe preview thumbnails
✓ Validate before render
Set the same camera target distance for all keyframes: 0,1

	0	1	2	3
Keyframe previews				
main_camera_x	-14,14195518597055	-14,1419975453347	-14,14198856136185	-14,141970890839...
main_camera_y	13,91296626507707	13,91300195967982	13,91306679269415	13,91319431236494
main_camera_z	-0,9134965954147...	-0,9121415921658721	-0,909691795933...	-0,904873306276...
main_target_x	-14,14195594362708	-14,14199830299123	-14,14198931901838	-14,14197164849555
main_target_y			328399724019	13,91341151691098
main_target_z			09697541399...	-0,9048790517414...
main_camera_top_x			13786816885...	0,1600532537238...
main_camera_top_y			63920434756...	0,0266596322479...
main_camera_top_z			9556593556...	0,9867483062974...
main_sweet_spot_h				0
main_sweet_spot_v				0
main_DOF_focus			60693	0,132934644818306
main_aux_light_enal				0
main_DOF_blur_opa				1
main_DE_factor				0,6

Figure 7.10: Changing an interpolation type

]

8 Animation

8.1 Flight animation - workflow

This section explains the necessary steps required to create flight animation. *Flight animation* in Mandelbulber is like a camera motion track recorded in some kind of flight simulator. The camera can go through interiors of fractal objects. Recording is normally initially done by navigating with the image set at a low resolution (e.g 320 x 240). After previewing and making any changes, the final rendering is undertaken with the image resolution set to a suitable higher size.

The parameters of every single animation frame recorded in Flight Animation mode, can be edited in the animation table.

Workflow:

1. Define fractal object (or many objects).

   Create a fractal with interesting features for the flight animation, (e.g. interesting shapes, geometric structure, texture, coloring, possibly holes where the camera can navigate into).

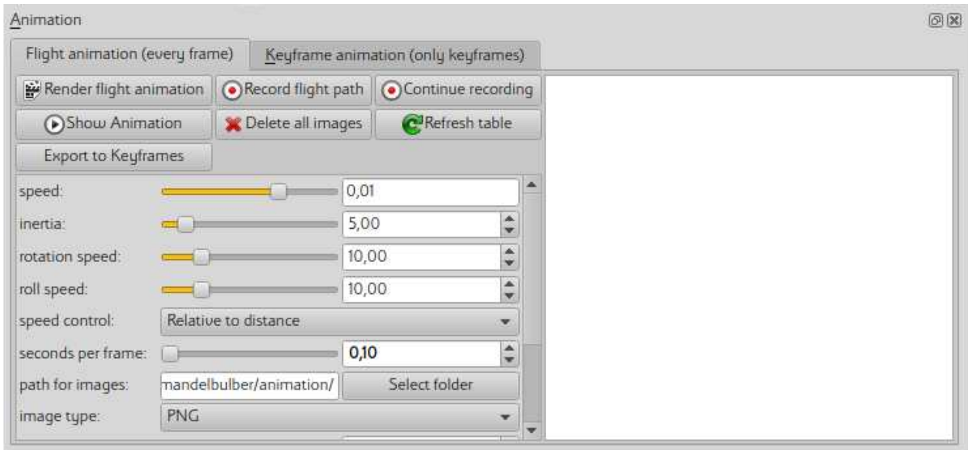
   It is advisable to select a fractal object that is relatively fast to render. Using a fast rendering fractal at a low resolution, results in the navigation and flight path recording being happening at almost real time (or slow motion). A fast rendering fractal will also increase the speed of the final frame rendering process.

2. Place the camera at the point where the flight is to start from.

3. Set a low image resolution. At a low image resolution, the frames-per-second value can be set higher for the flight path recording. It is reasonable to use a resolution like 320x240 or 160x120

4. Disable all effects which can slow down rendering, like ambient occlusion, reflections, transparency, volumetric lights, etc. All these effects can be re-enabled before commencing the final rendering of the animation.

5. Open *Flight animation* editor. It can be opened from top pull-down-menu by activating *View / Show animation dock*. The dock will appear at the bottom of the application window, with *Flight animation (every frame)* tab (showed on picture 8.1)



speed: 0,01
inertia: 5,00
rotation speed: 10,00
roll speed: 10,00
speed control: Relative to distance
seconds per frame: 0,10
path for images: nandelbulber/animation/
image type: PNG

Figure 8.1: Flight animation dock

6. Set parameters of animation

   speed defines how fast the camera will fly. This parameter can be changed during the recording of the flight path.

   inertia defines how heavy is the camera. A heavy camera will make a smoother motion, but it will be more difficult to control.

   rotation speed defines thereaction speed of mouse pointer movements. A higher value will allow for the camera to be turned faster.

   roll speed defines the speed of the reaction for the Z and X keys, which rotate the camera.

   speed control defines how the speed of the camera will be controlled.

   - In *Relative to distance* mode, the camera speed will decrease relatively to the nearness of the fractal surface. This mode will help you to not collide with the fractal. In this mode you can control the relative speed by the speed parameter and by the mouse buttons.
   - In *Constant mode*, the camera speed is only controlled by the speed parameter and the mouse buttons

   second per frame defines the frame rate during flight path recording. Higher values give slower rendering but images are more detailed. The value of this parameter is used only during recording, and is ignored during the rendering of the final animation.

   path for images defines where the rendered animation frames will be stored.

   image type defines the image format for saving the rendered frames. Detailed settings for image format are in *File / Program preferences*

   show thumbnails enables previews of frames in animation table

   add flight and rotation speed to parameters enables possibility to continue recording of animation after recording is completely stopped.

7. Press *Record flight path* button. After 3 seconds recording will be started. During this 3 seconds waiting time the mouse cursor should be placed in the center of image.

8. Use mouse pointer movements to turn camera left / right and up / down. Camera behaves like airplane in flight simulator

   Use Z and X key to rotate the camera

   Use arrow keys to move camera left / right / up / down. Without *Shift* key the camera still goes forward (movement at 45 degree angle). With *Shift* key the camera is not moving forward (movement at straight angle)

   Use the left mouse button to increase flight speed or the right button to decrease speed.

9. Press *space* key to pause recording. When recording is paused, animation parameters can be changed.

10. Press *STOP* button to stop recording. It is good to pause the recording using the (space key) before stopping. This is because by moving the mouse pointer towards *STOP* button, can turn the camera.

11. Recording of animation can be continued if *add flight and rotation speed to parameters* is enabled. If *Continue recording* is pressed, the recording of flight will resume from the point stored in the last frame. The camera linear and rotation speed will be maintained.

12. Increase image resolution and enable all required effects. There can be added light sources, fog, materials, textures, etc.

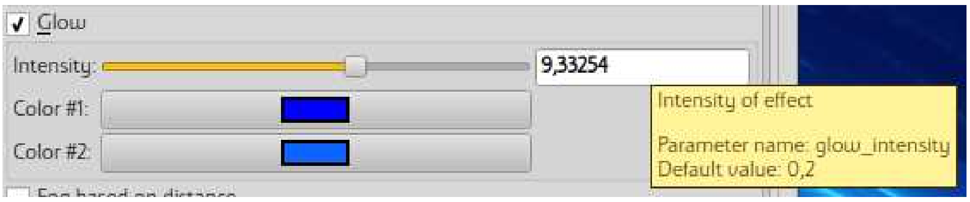
13. Press *Render flight animation* to commence final rendering process for the animation. This can take a very long time depending on the image resolution and the number of frames.

    Rendering of the animation can be stopped at any time and continued later. When *Render flight animation* is pressed, there will be rendered only the frames which are missing from the image frame folder. Any existing rendered frames will be skipped.

8.2 Flight animation - more options

8.2.1 Adding more parameters to animation

It is possible to animate almost any parameter in Mandelbulber. Each edit field has an assigned parameter. If you place the mouse pointer on an edit field a tooltip will be displayed.


Glow
Intensity: 9,33254
Color #1:
Color #2:

Intensity of effect

Parameter name: glow_intensity
Default value: 0,2

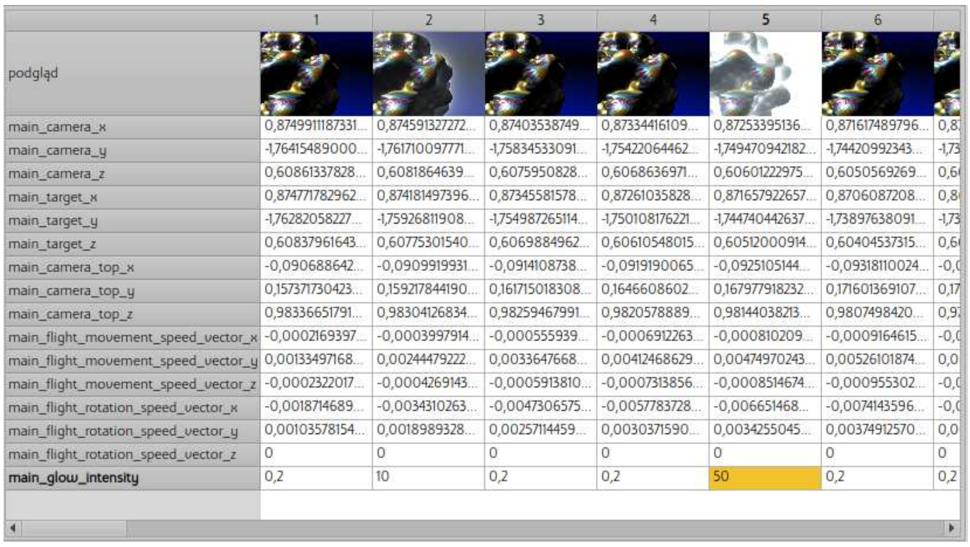
Figure 8.2: Example tooltip with parameter name

Below the parameter description there is the *Parameter name* (in this example it is *glow_ intensity*)

The parameter can be added to *Flight animation*. Right click on an edit field, then use *Add to flight animation* from context menu, and the parameter will appear in the animation table (picture 8.3)

8.2.2 Editing animation in the table

It is possible to edit every single animation frame in the animation table. Every cell in the table is editable. When a value is being edited, the preview is refreshed.



	1	2	3	4	5	6	
podgląd							
main_camera_x	0,8749911187331...	0,874591327272...	0,87403538749...	0,87334416109...	0,87253395136...	0,871617489796...	0,87
main_camera_y	-1,76415489000...	-1,761710097771...	-1,75834533091...	-1,75422064462...	-1,749470942182...	-1,74420992343...	-1,73
main_camera_z	0,60861337828...	0,6081864639...	0,6075950828...	0,6068636971...	0,60601222975...	0,6050569269...	0,6
main_target_x	0,874771782962...	0,874181497396...	0,87345581578...	0,87261035828...	0,871657922657...	0,8706087208...	0,8
main_target_y	-1,76282058227...	-1,75926811908...	-1,754987265114...	-1,750108176221...	-1,744740442637...	-1,73897638091...	-1,73
main_target_z	0,60837961643...	0,60775301540...	0,6069884962...	0,60610548015...	0,60512000914...	0,60404537315...	0,6
main_camera_top_x	-0,090688642...	-0,0909919931...	-0,0914108738...	-0,0919190065...	-0,0925105144...	-0,09318110024...	-0,
main_camera_top_y	0,157371730423...	0,159217844190...	0,161715018308...	0,1646608602...	0,167977918232...	0,171601369107...	0,17
main_camera_top_z	0,98336651791...	0,98304126834...	0,98259467991...	0,9820578889...	0,98144038213...	0,9807498420...	0,9
main_flight_movement_speed_vector_x	-0,0002169397...	-0,0003997914...	-0,000555939...	-0,0006912263...	-0,000810209...	-0,0009164615...	-0,
main_flight_movement_speed_vector_y	0,00133497168...	0,00244479222...	0,0033647668...	0,00412468629...	0,00474970243...	0,00526101874...	0,0
main_flight_movement_speed_vector_z	-0,0002322017...	-0,0004269143...	-0,0005913810...	-0,0007313856...	-0,0008514674...	-0,000955302...	-0,
main_flight_rotation_speed_vector_x	-0,0018714689...	-0,0034310263...	-0,0047306575...	-0,0057783728...	-0,006651468...	-0,0074143596...	-0,
main_flight_rotation_speed_vector_y	0,00103578154...	0,0018989328...	0,00257114459...	0,0030371590...	0,0034255045...	0,00374912570...	0,0
main_flight_rotation_speed_vector_z	0	0	0	0	0	0	0
main_glow_intensity	0,2	10	0,2	0,2	50	0,2	0,2

Figure 8.3: Animation table with all recorded frames and one additional parameter

This picture (picture 8.3) shows changes to *main_glow_intensity*. On frame 2 it is changed to 10 and on frame 5 it is changed to 50.

To get a smooth change of value within the selected range of frames, there is an option to do linear interpolation of values.

Example: We would like to have smooth transition of glow intensity starting from frame 30 ($glow = 0.2$) and ending on frame 100 ($glow = 10$)

- Right click in the table on cell for glow intensity at frame 100
- Use option *Interpolate next frames*
- *Last frame number* set to 100
- *Value for last frame* set to 10
- When you press OK, you should see that in the table the glow intensity parameter is increasing to 10 from frame 30 to frame 100

To cut the animation at the start or at the end, there is an option to delete a range of frames. If you right click on the frame number, there are two options:

- *Delete all frames to here* - deletes all frames from first to selected frame
- *Delete all frames from here* - deletes all frames from selected frame to the end

8.3 Keyframe animation - workflow

This section will be written soon

9 NetRender

NetRender is a tool that allows you to render the same image or animation on multiple computers simultaneously. If you have multiple computers connected to an Ethernet network, you can greatly increase overall computing power.

One of the computers (server) manages the process of rendering. It sends a requests to the connected computers (clients) and collects the results of rendering. Other computers (clients) render different portions of the image and send it to the server. There can be only one server (master) but clients (slaves) can be any number. The more clients, the faster the rendering will be.

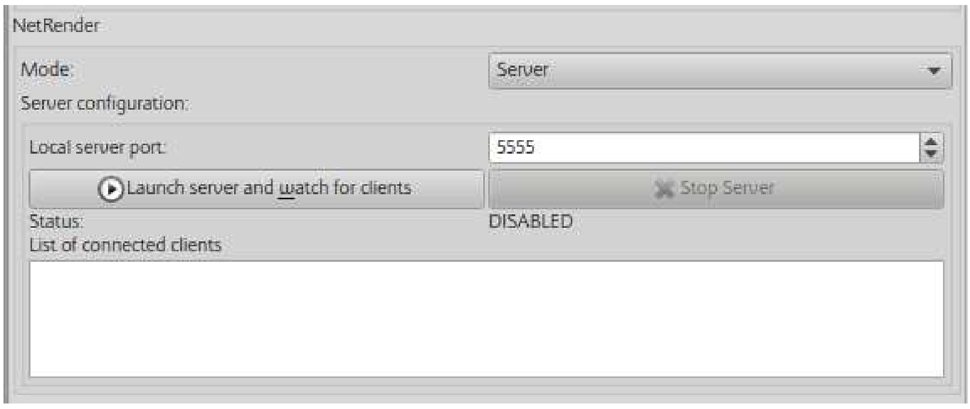
The Server is also the computer which renders the combined image .

The total number of CPUs (cores) used is the sum of server's CPUs cores + all client's CPUs cores.

9.1 Starting NetRender

9.1.1 Server configuration

On the computer which will be used as the Server, Mode is set to *Server.*


NetRender

Mode: Server

Server configuration:

Local server port: 5555

Status: DISABLED

List of connected clients

Figure 9.1: NetRender in 'Server' mode

Local server port should be set to one which is not used by other applications, and is passed through routers (if any are used) and firewall. The default is 5555.

If settings are correct, press *Launch server and watch for clients* button to connect server to existing clients.

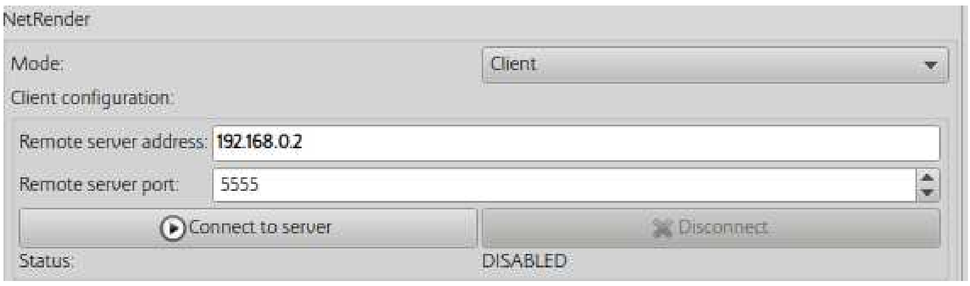
At this point, the server is ready to work

Alternative way to launch the server is to use command line. Example:

```
$ mandelbulber2 --server --port 5555 pathToFileToRender.fract
```

9.1.2 Configuring the clients

On the computers which will be used as Clents, *Mode* is set to *Client*


NetRender

Mode: Client

Client configuration:

Remote server address: 192.168.0.2

Remote server port: 5555

Status: DISABLED

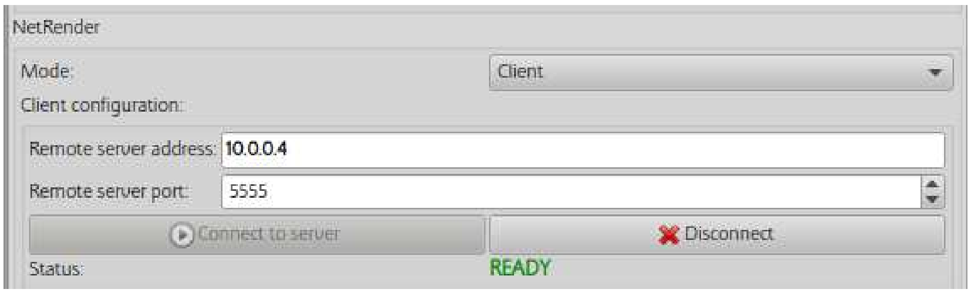
Figure 9.2: NetRender in 'Client' mode

The remote server address must be set to the same as the Server computer which is running Mandelbulber in Server mode. The address can be given as an IP address or a computer name.

The remote server port number must be exactly the same as the setting on the Server.

Press *Connect to server* button to connect to the server

Once the connection is established correctly, the client application should show the status *READY*


NetRender

Mode: Client

Client configuration:

Remote server address: 10.0.0.4

Remote server port: 5555

Status: READY

Figure 9.3: NetRender in 'Client' mode connected to the server

Alternative way to establish NetRender client is to use command line:

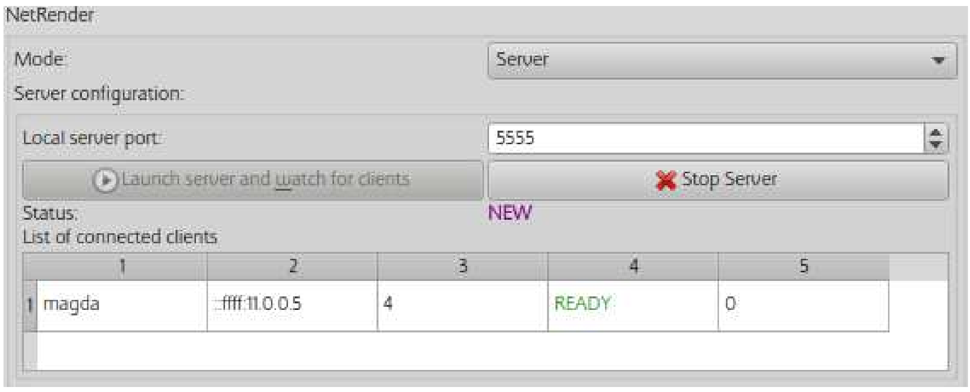
```
$ mandelbulber2 --nogui --host 10.0.0.4 --port 5555
```

when connection is successfully established the program should return following message:

```
NetRender - Client Setup, link to server: 10.0.0.4, port: 5555
NetRender - version matches (2090), connection established
```

On the Server computer, in the table "List of connected clients" should be shown the name and address of the connected clients and the number of available processors (cores).

i.e in figure 9.4 “magda” computer has 4 cores and is *READY*.


NetRender

Mode: Server

Server configuration:

Local server port: 5555

Status: NEW

List of connected clients

	1	2	3	4	5
1	magda	::ffff:11.0.0.5	4	READY	0

Figure 9.4: NetRender in 'Server' mode with a connected client

9.1.3 Rendering

Only the Server can initiate rendering on the computers connected using NetRender. When the *RENDER* button is pressed on the server, all the connected computers commence rendering.

In the table *List of connected clients* in the column *Done lines* will be shown the number of lines the image rendered by each of the computers.

when rendering is finished, the Server computer, will display the complete image. On the Client computers, only that portion of the image which was rendered by that client will be displayed.

10 Case study

10.1 Examples

10.1.1 Example of MandelboxMenger UI

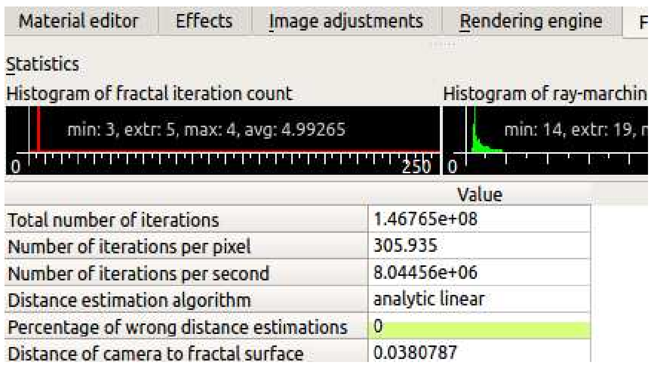
Example settings

(copy to clipboard, then load in Mandelbulber using : *File – Load settings from clipboard*):

```
# Mandelbulber settings file
# version 2.08
# only modified parameters
[main_parameters]
ambient_occlusion_enabled true;
camera 1.872135433718922 -2.023030528885091 1.871963531652841;
camera_distance_to_target 0.005814178381115117;
camera_rotation -28.76425655707408 26.3550335393397 3.450283685696816;
camera_top -0.1604796308669786 -0.4174088010201082 0.894436236356597;
DE_factor 0.6;
dont_add_c_constant_1 true;
flight_last_to_render 0;
formula_1 91;
formula_2 61;
formula_iterations_2 5;
formula_start_iteration_2 4;
formula_stop_iteration_2 5;
fractal_constant_factor 0.9 0.9 0.9;
fractal_enable_2 false;
fractal_rotation 0 -90 0;
keyframe_last_to_render 0;
main_light_beta 44.34;
main_light_intensity 2;
mat1_coloring_palette_offset 12.83;
mat1_coloring_palette_size 255;
mat1_surface_color_palette fd6029 698403 fff59c 000000 0b5e87 c68876 a51c64 3b9fee d4ffd4 aba53c;
SSAO_random_mode true;
target 1.874642452030676 -2.018463533070165 1.874544631933419;
view_distance_max 28.58330790625501;
volumetric_fog_colour_1_distance 3.55841069795292e-06;
volumetric_fog_colour_2_distance 7.116821395905841e-06;
volumetric_fog_distance_factor 7.116821395905841e-06;
[fractal_1]
fold_color_comp_fold 0.3;
mandelbox_color -0.27 0.05 0.07000000000000001;
mandelbox_rotation_main 9 1.74 3;
mandelbox_scale -1.5;
transf_addCpixel_enabled_false true;
transf_int_1 12;
transf_scaleB_1 0;
transf_scaleC_1 0;
transf_start_iterations_M 4;
transf_stop_iterations_M 5;
```

In the example the MengerSponge part is run only on iteration 4. A single iteration of another fractal to make a hybrid is often the best practice.

In the Statistics (enable in View menu) in figure 10.1 you can see Percentage of Wrong Distance Estimations ("Bad DE") is 0, which is good! As a general rule less than 0.01 is good, but it is case specific and 3.0 sometimes is OK and .0001 sometimes is not.



	Value
Total number of iterations	1.46765e+08
Number of iterations per pixel	305.935
Number of iterations per second	8.04456e+06
Distance estimation algorithm	analytic linear
Percentage of wrong distance estimations	0
Distance of camera to fractal surface	0.0380787

Figure 10.1: Statistics tab of the MandelboxMenger formula

The *Raymarching step multiplier* (*Rendering Engine* tab) or *fudge factor* is set at 0.6, which is good for a hybrid. If I change it to 0.7 the Percentage of Bad DE leaps up to 0.25 and you can see the areas of quality loss on your image.

Now if we disable the *addCpixel Axis swap Constant Multiplier*, we find we can now increase the *Raymarching Step Multiplier* to 0.9, and get a faster render and visually the same quality. So monitoring Percentage of Wrong Distance Estimations is a guide to managing quality. (Note when doing animations you may want to drop the Raymarching step down a bit to allow for what might happen between keyframes.)

MandelboxMenger Hybrids can behave a bit differently to a lot of hybrids, in the fact that the *Percentage Bad DE* often improves when you zoom in.

Optimizing of *maximum view distance* Located : Rendering Engine - Common Rendering settings

It is important to optimize this setting to minimize render time. You can reduce until the furthest part of the 3D object(s) starts to disappear. However with animation an allowance should be made for changes between keyframes.

Note! When navigating in Relative step mode, mouse click on spherical_inversion, camera zooms out, tand maximum view distance becomes set on 280. If you don't reset it your render times will be increased.

Magic Angle Benesi Mag Transforms

In mathematics the Magic Angle = 54.7356° .

When rendering basic mag transforms the image does not render parallel to the standard x,y,z global axis. On the fractal dock, in "Global parameters" set y-axis rotation to 35.2644° (= 90° - 54.7356°). The fractal will then render parallel to the x-y plane.

10.1.2 Example of using Transform Menger Fold to make Hybrid

```
# Mandelbulber settings file
# version 2.08
# only modified parameters
[main_parameters]
ambient_occlusion_enabled true;
camera -1.528388569045064 -1.23063017895654 -0.0251755516595821;
camera_distance_to_target 0.0004503351519815117;
camera_rotation -14.07789975269277 -44.28785609194563 3.773777260910995;
camera_top 0.2333184436621841 0.6598138513697914 0.7142885869084139;
DE_factor 0.7;
flight_last_to_render 0;
formula_1 1052;
formula_2 1010;
formula_3 1052;
formula_4 1009;
formula_iterations_1 5;
formula_start_iteration_4 45;
formula_stop_iteration_2 12;
formula_stop_iteration_4 5;
fractal_constant_factor 0.9 0.9 0.9;
fractal_enable_4 false;
hdr true;
hybrid_fractal_enable true;
keyframe_last_to_render 0;
main_light_alpha 2.6;
main_light_beta 1.59;
mat1_coloring_palette_offset 46.51;
mat1_coloring_palette_size 255;
mat1_coloring_random_seed 647723;
SSAO_random_mode true;
target -1.528310155903731 -1.230317492741513 -0.02549000429402527;
volumetric_fog_colour_1_distance 3.55841069795292e-06;
volumetric_fog_colour_2_distance 7.116821395905841e-06;
volumetric_fog_distance_factor 7.116821395905841e-06;
[fractal_1]
transf_addition_constantA_000 -0.071633 0 0;
transf_function_enabledy false;
transf_int_1 12;
transf_scale 0.5;
transf_scaleC_1 0;
transf_stop_iterations_1 2;
[fractal_2]
transf_scale3D_333 1.055556 1.027778 0.861111;
[fractal_3]
transf_function_enabledx false;
```

On this transform UI, the standard menger sponge formula is split into a start and end function. The simplest way to use this transform is in Hybrid Mode, having the menger fold transform in slots 1 and 3. In slot 2 place any linear type formula or transform. (ie more mengers, kifs, mboxes, amazing surf, folds, rotation , Benesi T1 etc).

In slot 1 disable the stop function and in slot 3 disable the start function, resulting in a standard menger sponge with something in the middle.

BTW in fact you can mix around with the start and stop functions have all enabled if you wish. Generally linear functions all work well together in making hybrids.

In Statistics, maximum is approx. 80 iterations. Generally hybrids take longer to render than standard formulas. As well as adjusting formula parameters, you can use the iteration controls to tweak hybrids. In this example the first slot is set to repeat for 5 iterations before moving to slot 2. Slot 2 is set to stop at iteration 12, whereas slots 1 and 3 can continue to termination conditions are met (bailout or maximum number of iterations).

In the example above, slot2 of the hybrid sequenced ended at iteration 12. 12 was chosen because how it fitted into the iteration sequence, as follows:

- **Slot 1** x 5 – iterations: 0, 1, 2, 3, 4 (note first iteration is iteration number 0)
- **Slot 2** – iteration 5
- **Slot 3** – iteration 6
- **Slot 1** – iterations 7, 8, 9, 10, 11
- **Slot 2** – iteration 12 (last use of **Slot 2**)

Sequence continues **Slot 1** x 5, **Slot 3**, . . . to bailout.

As you see **Slot 2** is used only twice in the iteration process. If I had entered 11 instead of 12 for **Slot 2**'s *stop iterations*, then the slot would have been used only once, if I entered 19 then it would run three times.

10.2 Q&A . How do you get different materials on different shapes?

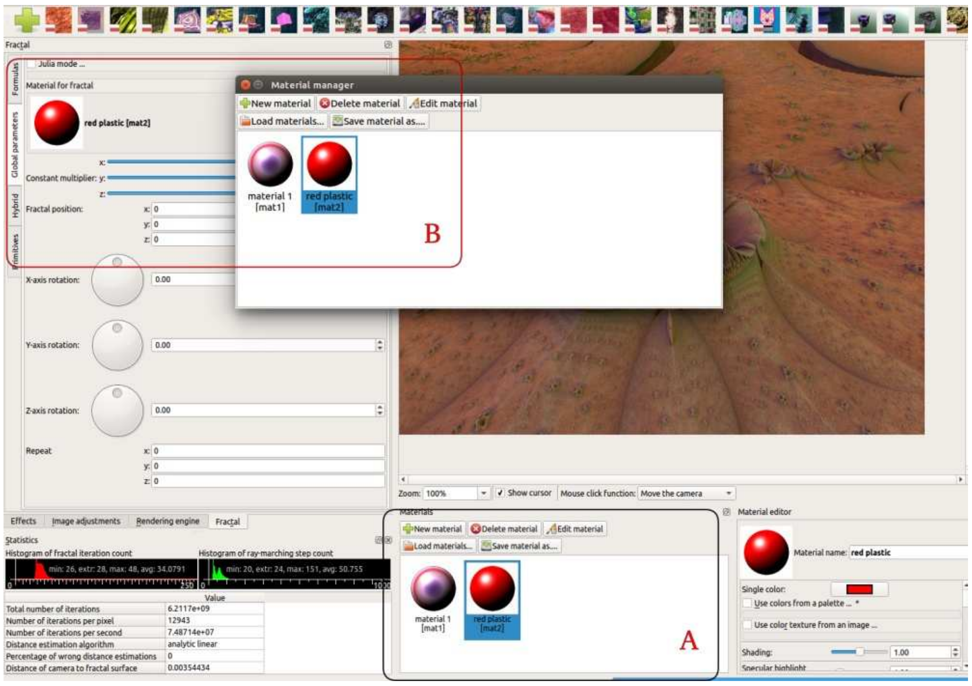
This is how I have been doing it, see also figure 10.2:

Rectangle at the bottom marked A.

This is where you start a new material or load an existing. The active material is highlighted in blue. Meaning it is active in the **material editor** where you create or modify the material.

Rectangle at top left marked B.

One way to use a material is to go to Global Parameters, click on the material preview image, and the **Material Manager** UI will appear with the materials you have loaded or created. Click on the one you want to use, then close that UI.
Similarly with primitives, click on the material preview image. And with Boolean Mode each fractal/transform has it's own material preview image when you scroll down.

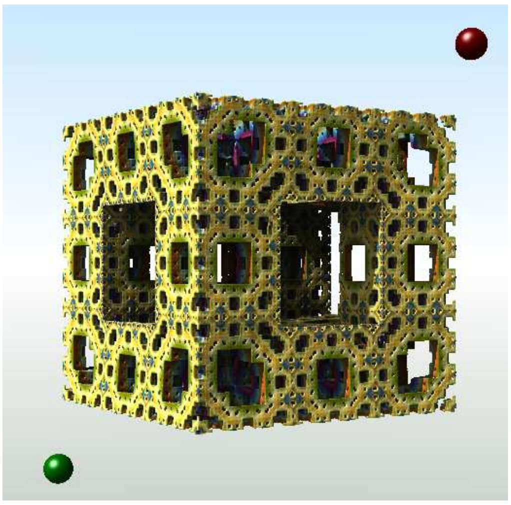


	Value
Total number of iterations	6.2117e+09
Number of iterations per pixel	12943
Number of iterations per second	7.48714e+07
Distance estimation algorithm	analytic linear
Percentage of wrong distance estimations	0
Distance of camera to fractal surface	0.00354434

Figure 10.2: Material selection

11 Using “Anim By Sound” with multiple tracks

Note. The following "walk through" tutorial is for using Anim By Sound with multiple tracks. This tutorial is based on my initial experiments with Anim by Sound and may be revised as I gain more experience. The tutorial demonstrates using sound to animate a fractal offset parameter and the material color. The settings file also includes animation of some other parameters, and produces an animation just over a minute long, at 300 x 300, 45 minutes to render.



Animations can take many hours to render, so it is best when learning the controls, to keep it simple. Choose fractals and/or primitives that render fast. Do not use slow effects like Volumetric Light of Multi Ray Ambient Occlusion. I drop the resolution to 400 x 300 Detail Level 0.5, or 200 x 150 Detail Level 0.25.

The animation value of an object (or an effect) at each frame, is the sum of the parameter value, (generated from the keyframe animation table), and the sound value at that frame.
animVal = paraVal + soundVal.

This tutorial is about the basics of using soundVal to vary animVal. So I will keep paraVal constant and use only sound data to direct the animation of parameters.

A cool thing about using only soundVal (no keyframe animation) is that we can set up a trial with just two keyframes (beginning and end).

Note. You can create an audio file for the single purpose of directing animation, where the audio file is not used at all in the final song mix. You can use Anim by Sound to create silent videos.

Keyframe animation requires changes to be made at keyframes. It is possible with Sound animation to make changes at any frame, (i.e. a change at any 1 / 30 of a second time interval, when at 30fps.)

Previously, choreographing parameters with spreadsheets was very time consuming and I was limited to what I could achieve, so I stopped and have waited. Anim by Sound has made this process much more simpler, and has infinite possibilities.

All files used in this example can be downloaded from mandelbulber.org

http://cdn.mandelbulber.org/doc/audio/9%20tut.zip unzip and place “9 tut” folder in home/mandelbulber/animations/

11.1 Audio Files

The following formats are supported:

- *.wav (wave form audio format)

- *.ogg (Ogg Vorbis)

- *.flac (Free Lossless Audio Format)

- *.mp3 (MPEG II Audio Layer 3) - supported only under Linux

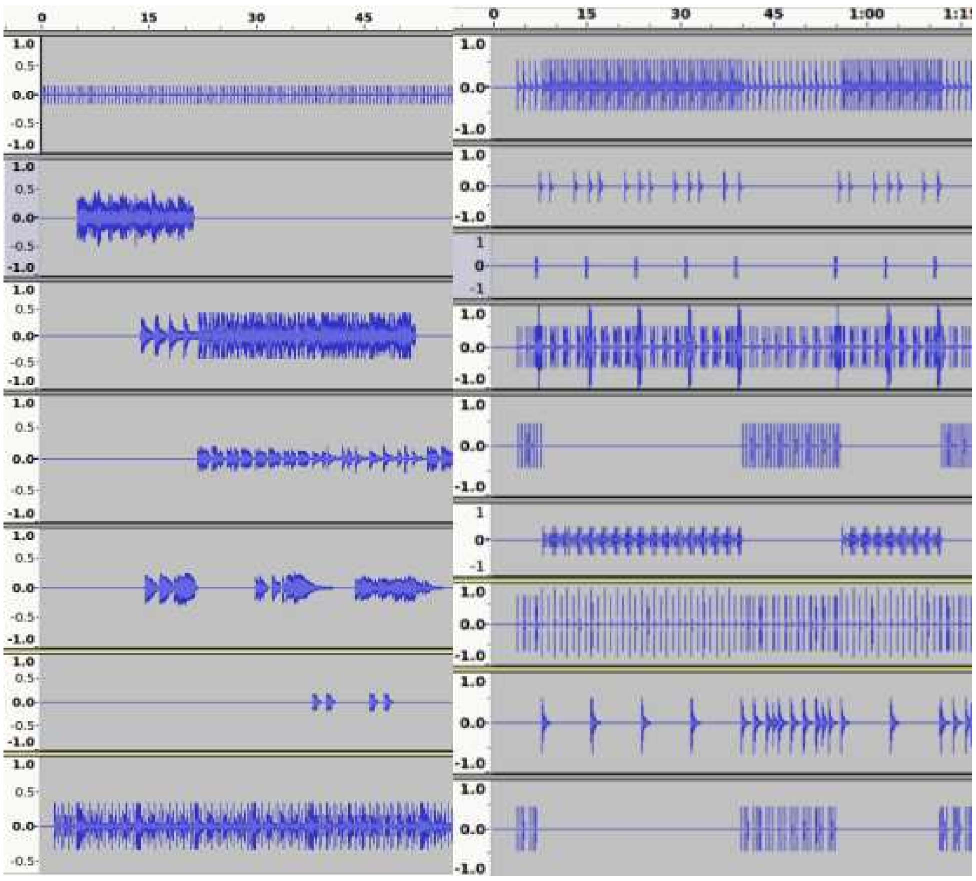
I used Hydrogen Drum Kit emulator to make all the individual drum track files. These are .wav files but as I am using Linux I converted them to .mp3 to use in Mandelbulber. These are mono working files, when I make the video in VirtualDub I then use the final song mix .wav file.

The guitar tracks have also been recorded as .wav, and a .mp3 copy made to use with Mandelbulber.

The audio file data is sampled at every frame point, the data is then converted to a *sound* number ranging between 0 (silent) and 1 (maximum) which can represent Amplitude or Pitch. This value is shown in the Sound Animation chart on the Audio Selector UI. We can use either the default Amplitude mode or choose Sound Pitch mode to animate.

For this example I used Pitch with lead guitar (melody line) creating the fractal movement, and Amplitude for a drum to alternate the color. The fractal shape will respond to the free flowing melody line and the color change as a repetitive rhythm event.

This is a screen-shot from Audacity showing some of the instrument tracks I had available. I only used one drum (a kick drum), but normally I would be using more percussion instruments.

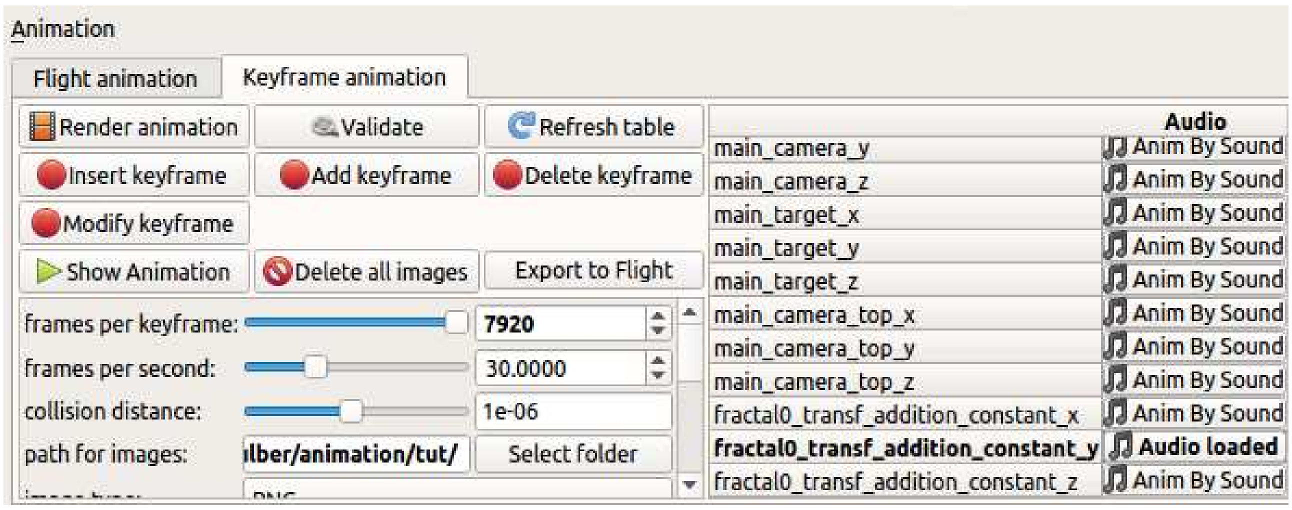


11.2 Adding a parameter.

Firstly, have the Animation Dock open. (i.e. from menu select View - show animation dock.)

Then go to the dock or tab for the parameter you wish to animate (e.g. fractal, material, effect etc). Right mouse click on the parameter field, and select Add to Keyframe Animation.

The parameter will then be listed in the keyframe animation table, with Anim By Sound in the next column.



frames per keyframe: 7920
frames per second: 30.0000
collision distance: 1e-06
path for images: ılber/animation/tut/

	Audio
main_camera_y	Anim By Sound
main_camera_z	Anim By Sound
main_target_x	Anim By Sound
main_target_y	Anim By Sound
main_target_z	Anim By Sound
main_camera_top_x	Anim By Sound
main_camera_top_y	Anim By Sound
main_camera_top_z	Anim By Sound
fractal0_transf_addition_constant_x	Anim By Sound
fractal0_transf_addition_constant_y	**Audio loaded**
fractal0_transf_addition_constant_z	Anim By Sound

Here I have chosen to animate parameter Menger_Mod1 offset y fractal0_transf_addition_constant_y

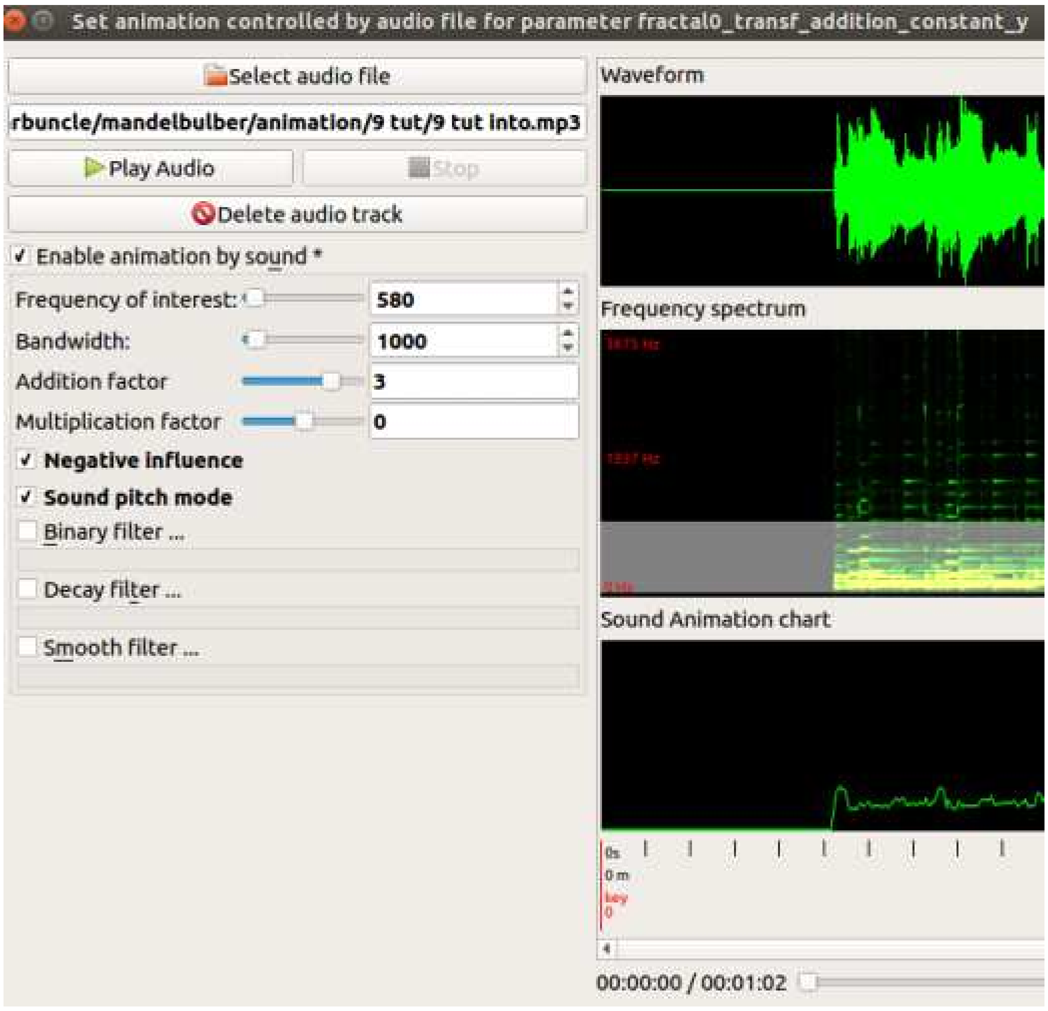
i.e parameter name *transf_addition_constant_y;* from formula slot *fractal0.*

Parameters x & z are also added because this parameter is part of a vector3.

11.3 Loading the Audio File

Left mouse click on Anim By Sound and the Audio Selector UI will open. The name of the parameter will be in the description along the top.

Select an Audio file and three charts will appear.
Enable Animation by Sound and the options will appear.


Set animation controlled by audio file for parameter fractal0_transf_addition_constant_y

rbuncle/mandelbulber/animation/9 tut/9 tut into.mp3

✓ Enable animation by sound *

Frequency of interest: 580

Bandwidth: 1000

Addition factor 3

Multiplication factor 0

✓ Negative influence

✓ Sound pitch mode

Binary filter ...

Decay filter ...

Smooth filter ...

00:00:00 / 00:01:02

Animation of a parameter is created by applying an addition-factor and/or a multiplication-factor.

animVal = paraVal + soundVal.

soundVal = (paraVal * multiplication-factor * *sound)* + (addition-factor * *sound*)

It is less complicated when learning, to use only the addition factor, so we change multiplication factor to 0.0.

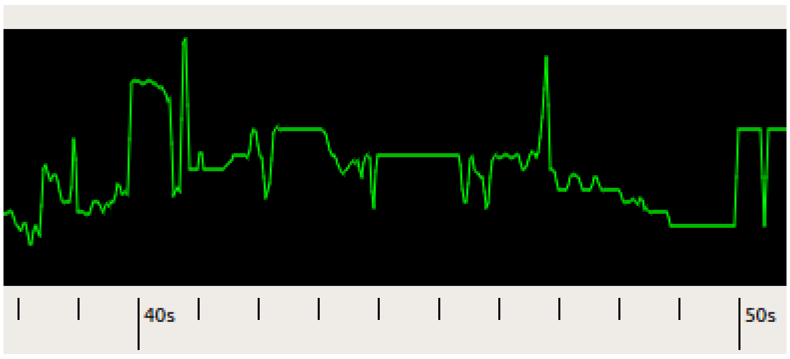
11.4 Using Sound Pitch mode.

I set frequency at 580Hz and bandwidth to 1000Hz, this covers the range of the fundamental frequencies of my lead guitar notes (I am removing higher harmonic frequencies, although this may not be necessary).

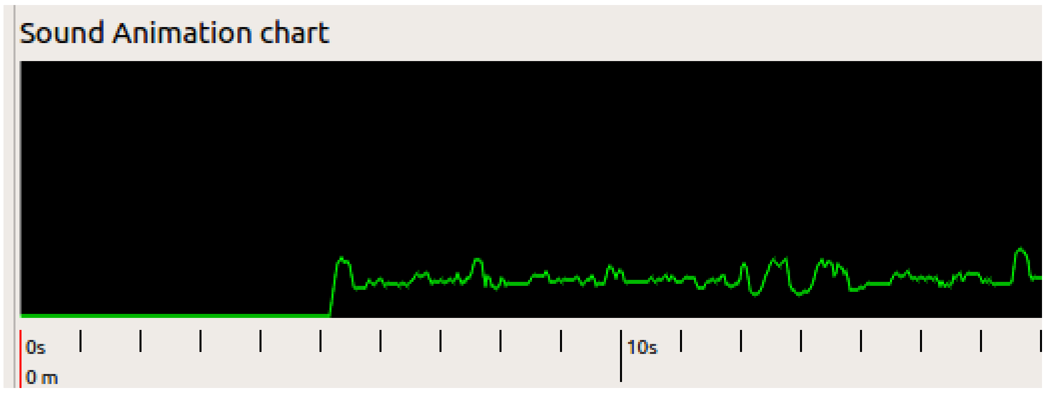
Make further adjustment if the Sound Animation charts shows that the pitch is contained only in the top or bottom of the chart, (resulting from a melody line only using high notes or only using low notes.) Push the Play Sound button and check that the chart line is following the pitch of the audio track.
The main point is not to limit the Pitch by having a small band width that does not cover the full spectrum of the fundamental notes used in the audio track.

In Pitch mode the Sound Animation chart rises from silent to high pitch.

In this image the *sound* varies from about 0.2 up to almost 1.0 (maximum *sound*). Therefore the *sound* will have a wide effect on the parameter animation.



In this image the *sound* is fairly constant at around 0.2, so it will have a narrow effect on the animation.

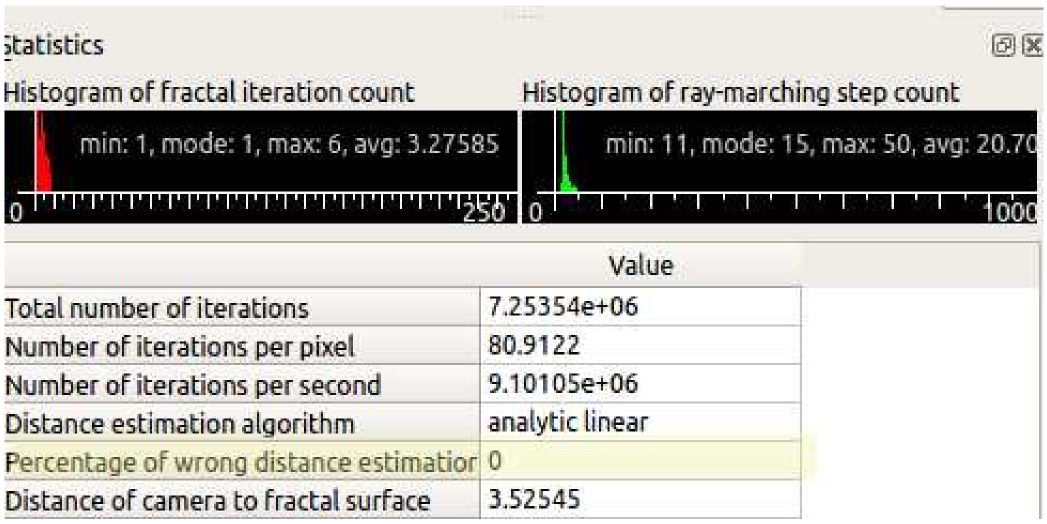


11.5 Testing the parameter

Close the Audio Selection UI and go the Menger_Mod1 fractal tab and test the parameter through a range of values.

If a parameter belongs to a fractal formula, you may notice that the ray marching step multiplier needs to be adjusted to produce a good image throughout the animation range.

As a guide for adjusting the ray marching step multiplier, open View - Show statistics, and monitor Percentage of wrong distance estimations.


Statistics

Histogram of fractal iteration count

min: 1, mode: 1, max: 6, avg: 3.27585

Histogram of ray-marching step count

min: 11, mode: 15, max: 50, avg: 20.70

	Value
Total number of iterations	7.25354e+06
Number of iterations per pixel	80.9122
Number of iterations per second	9.10105e+06
Distance estimation algorithm	analytic linear
Percentage of wrong distance estimatior	0
Distance of camera to fractal surface	3.52545

Now I decide on the appropriate size of the addition factor to use for animating the parameter.

I have paraVal "offset y" set at a constant value of 0.0, and I test an addition factor of 3.0. The *sound* will increase the soundVal in the range of 0.0 to 3.0 maximum. However for the effect I want, I am be using Negative influence mode, which will subtract the soundVal from paraVal instead of adding it, therefore the possible range to be tested is 0.0 to -3.0.

The audio file I used only creates *sound* between 0.0 and about 0.2, so the offset values I will be testing are the actual range between 0 and -0.6, i.e. 0.0 = 0.0 * -3.0 max, -0.6 = 0.2 * -3.0 max.

Note: There are two types of parameters in this program. The first type can have negative values entered, the second type cannot. It is important when animating a parameter of the second type, that the functions and settings used, do not result in a negative number.

View the Sound Animation chart to see the what values you are likely to get from *sound* at different parts of the instruments music.

Remember to set the parameter back to the original value when you have finished testing.

11.6 Using Amplitude

Example: Animate the color of the fractal on every beat of the kick drum.

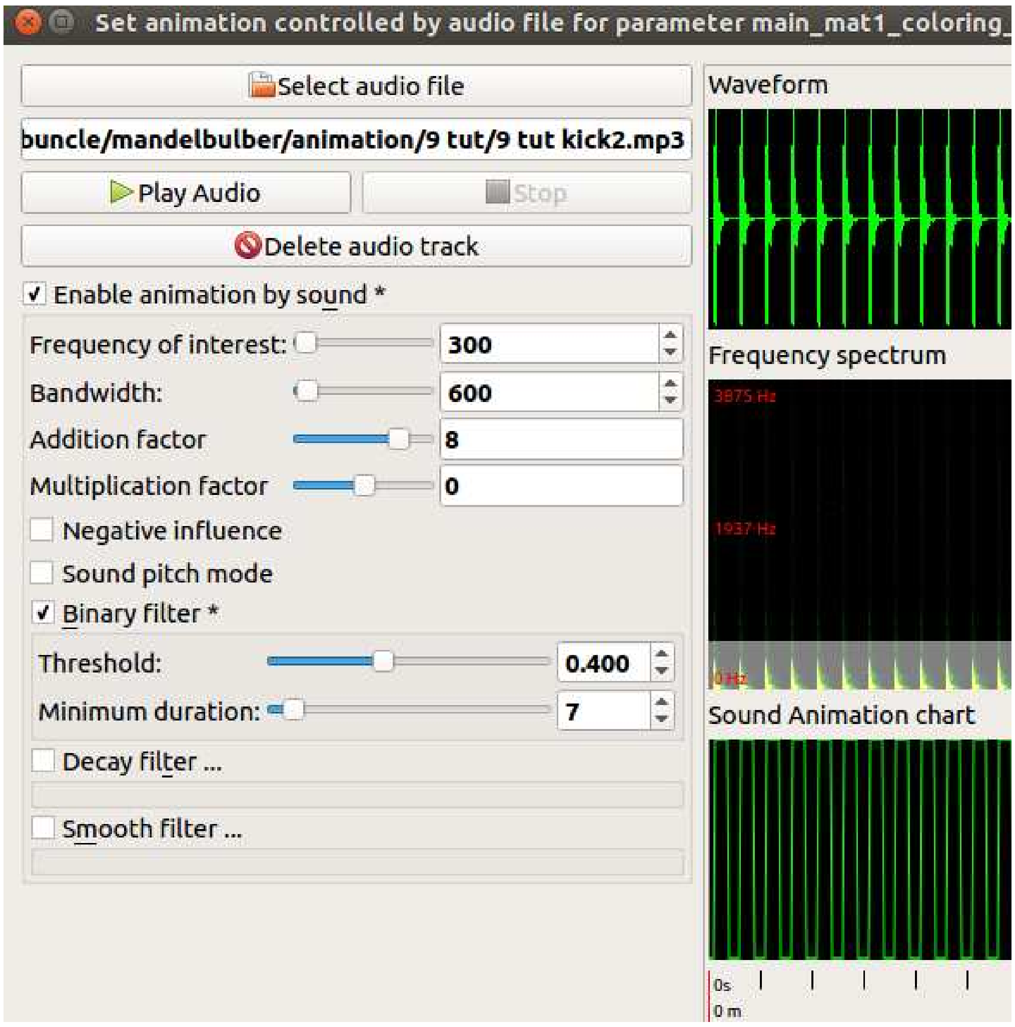
Add material 1 parameter "Palette_offset" to the Keyframe animation table.

Open Anim By Sound, load audio file and Enable Animation by sound.

Adjust Frequency of interest and bandwidth.

Enable "Binary filter" and adjust the threshold so that all the beats are shown in the Sound Animation chart.

Here I am creating an event (at every kick drum beat), that lasts for a minimum duration of 7 extra frames (7/30 seconds). The event is using Addition factor * sound and is triggered every time the sound amplitude increases above the threshold (0.400) and will last for 7/30 seconds. Events that last less than "say" 7 frames are difficult to observe at 30 fps.


Set animation controlled by audio file for parameter main_mat1_coloring_

Select audio file

buncle/mandelbulber/animation/9 tut/9 tut kick2.mp3

Play Audio | Stop

Delete audio track

☑ Enable animation by sound *

Frequency of interest:	300
Bandwidth:	600
Addition factor	8
Multiplication factor	0

☐ Negative influence

☐ Sound pitch mode

☑ Binary filter *

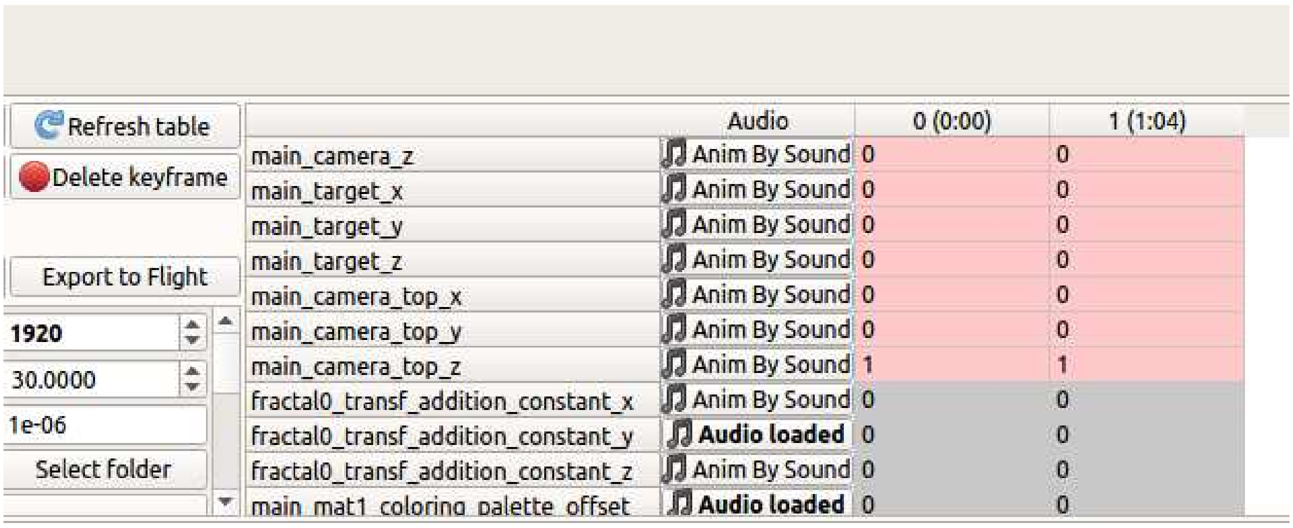
Threshold:	0.400
Minimum duration:	7

☐ Decay filter ...

☐ Smooth filter ...

11.7 Rendering the animation.

First add two identical keyframes to the Keyframe Animation table,



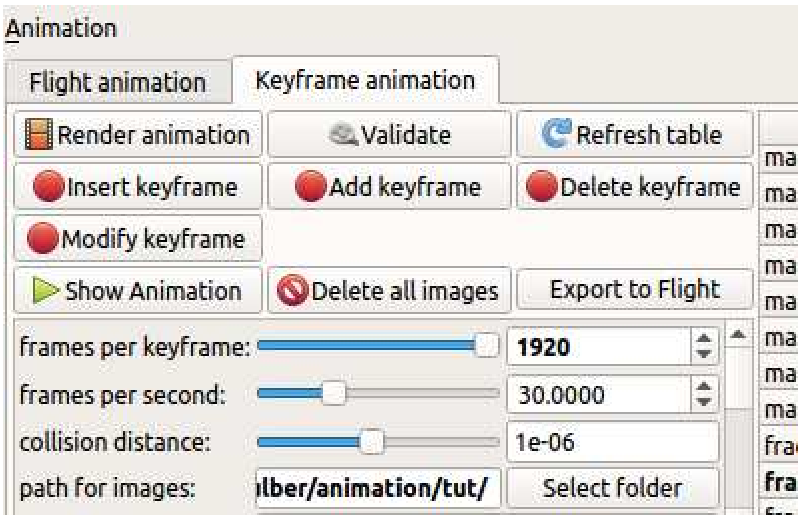
1920	
30.0000	
1e-06	

	Audio	0 (0:00)	1 (1:04)
main_camera_z	Anim By Sound	0	0
main_target_x	Anim By Sound	0	0
main_target_y	Anim By Sound	0	0
main_target_z	Anim By Sound	0	0
main_camera_top_x	Anim By Sound	0	0
main_camera_top_y	Anim By Sound	0	0
main_camera_top_z	Anim By Sound	1	1
fractal0_transf_addition_constant_x	Anim By Sound	0	0
fractal0_transf_addition_constant_y	**Audio loaded**	0	0
fractal0_transf_addition_constant_z	Anim By Sound	0	0
main_mat1_coloring_palette_offset	**Audio loaded**	0	0

and set "frames per keyframe" to a number that will cover the length of the trial, i.e. for 64 seconds at 30 fps it would be:

64sec * 30fps = 1920 frames per keyframe.

Make sure that the “path for images” is linked to the correct folder, and ensure that the folder is empty, “Delete all images” button.



frames per keyframe: 1920
frames per second: 30.0000
collision distance: 1e-06
path for images: lber/animation/tut/

11.8 Now render the trial animation.

When rendering is finished (note the time it took) , press “Show animation” button for a preview (you can also use “Show animation” while rendering is in progress.)

I also create a video with VirtualDub and the audio file, to ensure that the animation is working correctly with the sound. If the animation is satisfactory, then set the resolution and detail level to your final settings and wait.

12 Thanks

Thanks to the fractal community for your ongoing support!

Sincerely,

Mandelbulber Team

Listings

List of Figures

Index