



# Mandelbulber

## End User Manual

**Version 2.10.0.9.4** (2017-February)

**Download:** <https://sourceforge.net/projects/mandelbulber/>  
**Project Website:** <https://github.com/buddhi1980/mandelbulber2/>  
**Forum:** <http://www.fractalforums.com/mandelbulber/>

### Editors:

Krzysztof Marczak ([buddhi1980@gmail.com](mailto:buddhi1980@gmail.com)),  
Graeme McLarekin ([mclarekin@gmail.com](mailto:mclarekin@gmail.com)),  
Sebastian Jennen ([sebastian.jennen@gmx.de](mailto:sebastian.jennen@gmx.de)),  
Robert Pancoast ([RobertPancoast77@gmail.com](mailto:RobertPancoast77@gmail.com))

# Contents

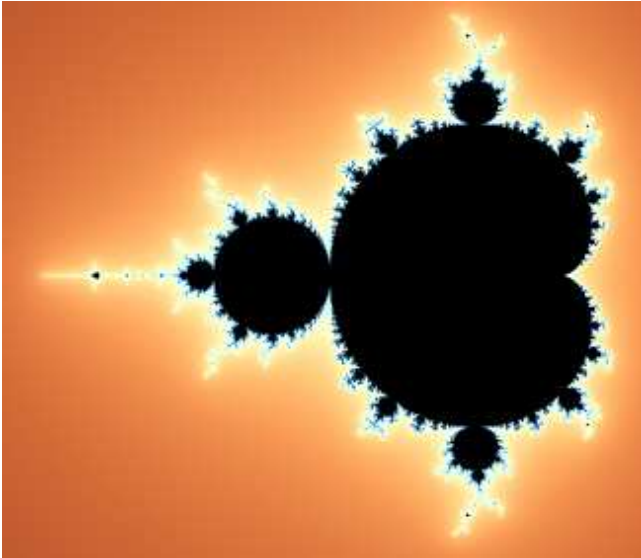
# 1 About this Handbook

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!

## 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 is to have subtle details even at very high magnification.

### 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 ( $a + ib$ ), where  $i$  is 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.

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

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

$$z_{n+3} = z_{n+4}^2 + c$$

etc.

The program has to determine if these series are convergent. To do this iterations should be repeated infinite number of times. But in this way won't be possible to get any image. There is used 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).

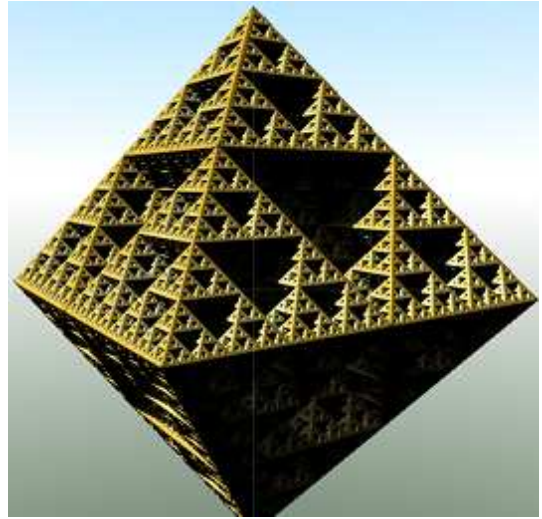
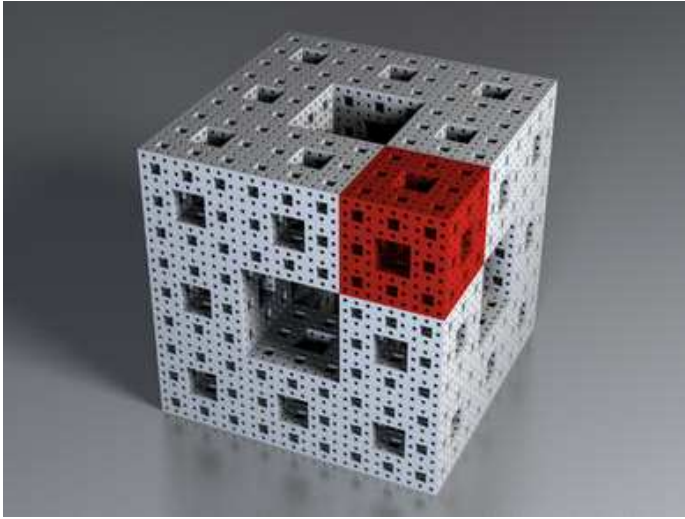
## 2.2 3D fractals

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 complex numbers ( $x, y, z$ ) 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 maths operators are used, e.g., addition, multiplication, squaring, and powers), 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.



## 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, Julibulb, Menger Sponge, . . . . The following sections cover the program interface and give useful information about how to use it.

### 3 Distance Estimation

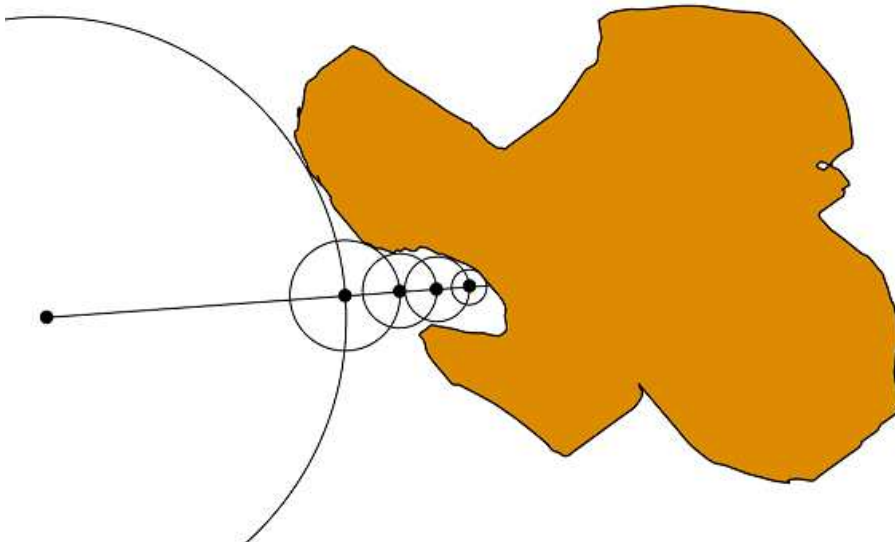
Distance Estimation (DE) is the calculation of an estimated distance from the given point to the nearest surface of the fractal. As suggested by the word 'estimate', it is an approximate value. It is calculated using simplified algorithms based on analytical (Analytical DE) or numerical (Delta DE) calculations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

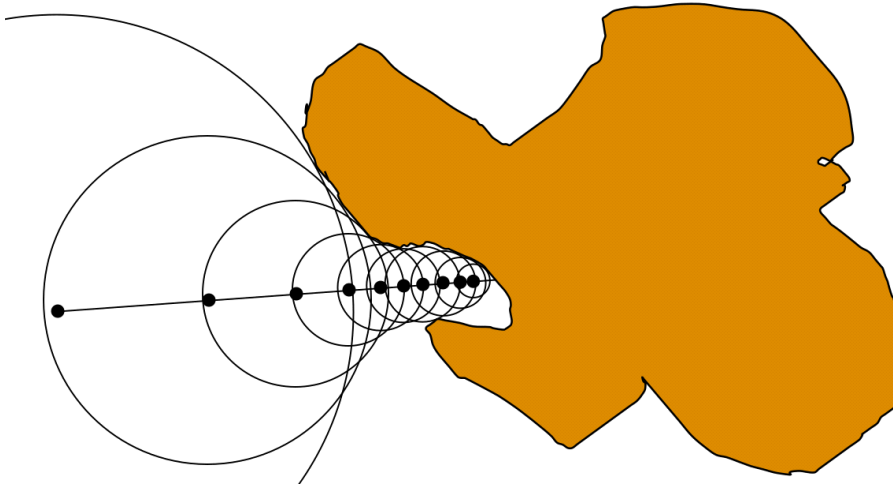


Since the estimation contains some error (sometimes quite large), there is a risk that the step of moving the "photon" will be too large, and incorrectly it will flow into the surface of the fractal. This may result in visible noise in the rendered image.

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, but the rendering time increases due to more steps being required.

Below is an example for a step multiplier of 0.5:



Raymarching with step multiplier 0.5 produces high iteration count and good DE

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.

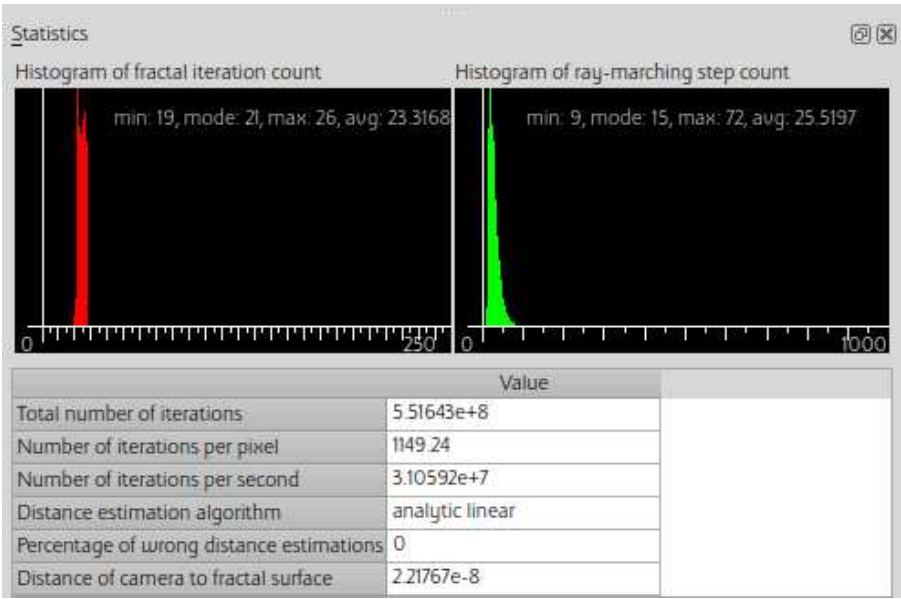
Example linear out:  $distance = \frac{r}{|DE|}$

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 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.

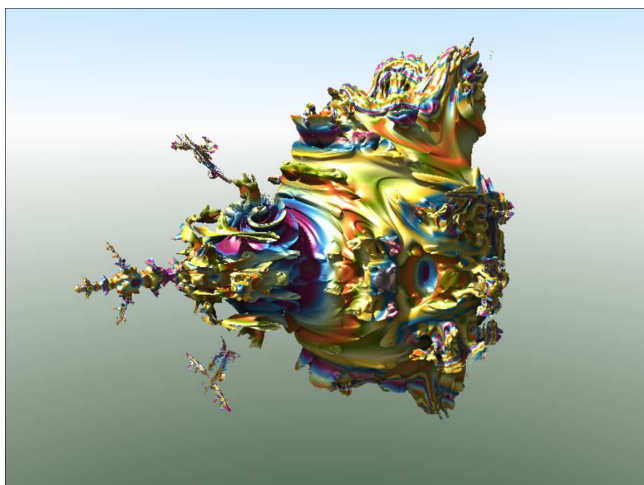
There are two modes of stopping the ray-marching of each image pixel.

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).

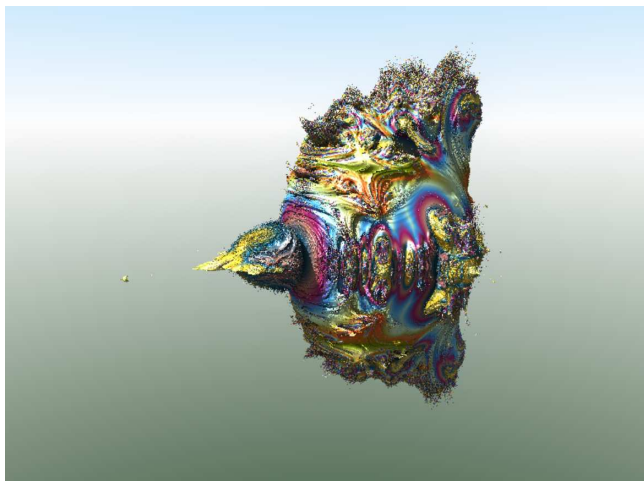
First important note: *Stop at maximum iteration* doesn't control the fractal iteration loop. It controls only ray-marching. The iteration loop always runs to achieve Bailout, (then if bailout is not reached the iteration stops at Maxiter). (see page 6)

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



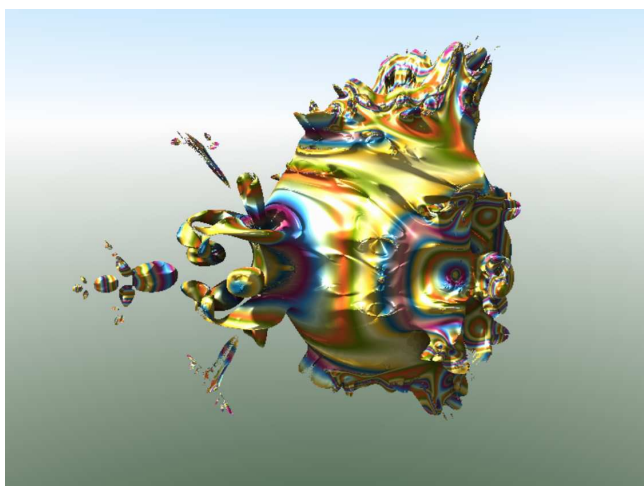
Ray-marching stops at distance threshold. In most cases the fractal iteration loop stops on bailout condition, (because away from surface it is not possible to reach Maxiter). It makes rendering of fractals much faster.

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



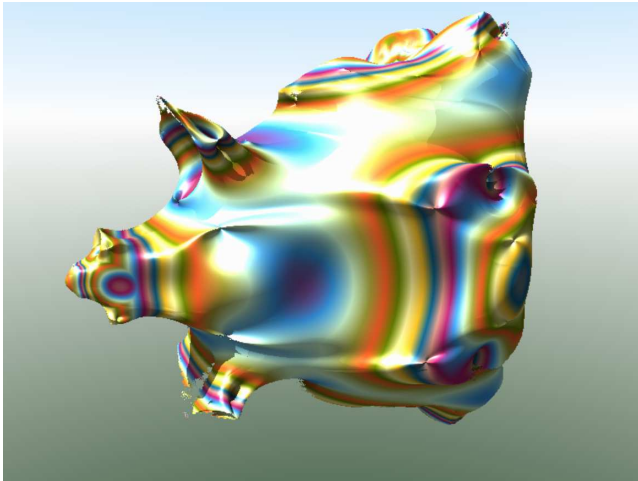
Ray-marching stops at the photon step when the maximum number of iterations is reached (ray-marching distance threshold is ignored). In many cases iteration loop stops on bailout condition (away from fractal surface), but on the fractal surface the maximum number of iterations is calculated (when bailout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

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



When 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.

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



When maximum number of iterations is reached, then ray-marching is stopped even if distance threshold is not reached.

## 5 Iteration loop

In section 2.1 there was mentioned that fractals are calculated by repeating of iteration loop. The loop is terminated when there is calculated number of iterations equals to *maxiter* or bailout condition is achieved. In this section will be explained what is calculated inside the loop.

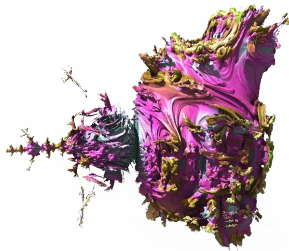
### 5.1 Single formula fractals

The simplest 3D fractals are calculated using single fractal formula which is build from many equations and conditions. These equations can be modifications of Mandelbrot Set equation or can be different mathematical equations with several conditions.

Below there are 3 examples of fractals formulas with C language code

#### 5.1.1 Mandelbulb Power 2

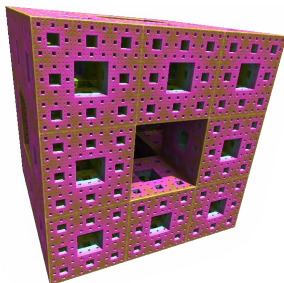
This formula is modified Mandelbrot Set equation, expanded to 3<sup>rd</sup> dimension. Cross section at  $z_z = 0$  looks exactly the same as Mandelbrot Set.



```
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 Iterated Function System (IFS). It contains several transformations where some of them are conditional.



```
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);

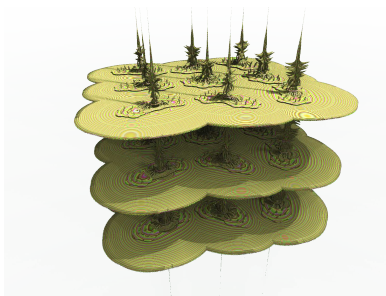
z *= 3.0;

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 a set of different transforms and equations. It's a good example which shows that fractal formula can be much more complicated than *Mandelbrot Set*.

First part is “box fold” transform which do transformations based on box walls. Second part is “spherical fold” which do transformations based on sphere. The end of formula is the same as *Mandelbulb 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;

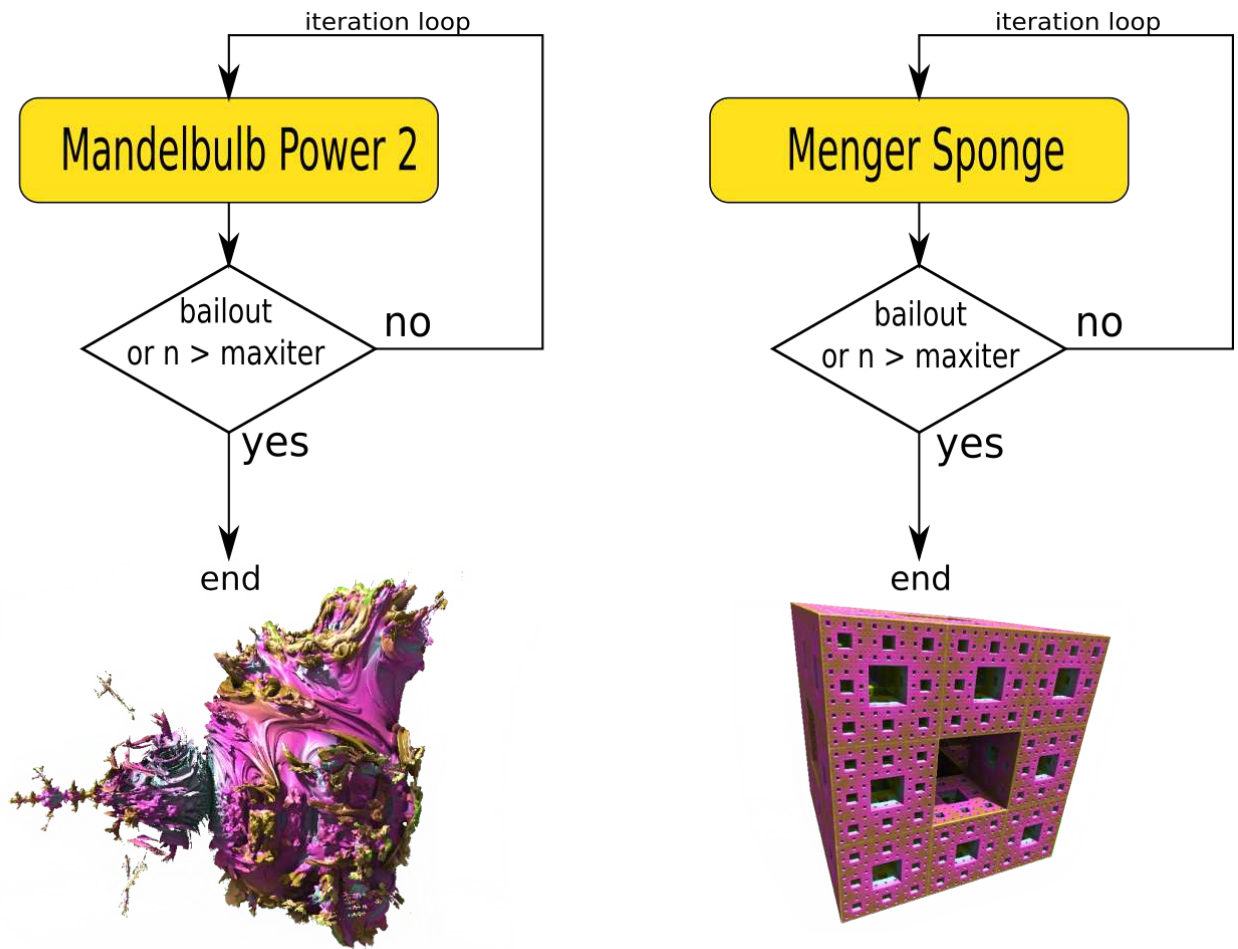
//spherical fold
double fR2_2 = 1.0;
double mR2_2 = 0.25;
double r2_2 = z.Dot(z);
double tglad_factor1_2 = fR2_2 / mR2_2;

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 processed in simple way. Calculation of fractal formulas is repeated several times like it is showed on following diagrams:

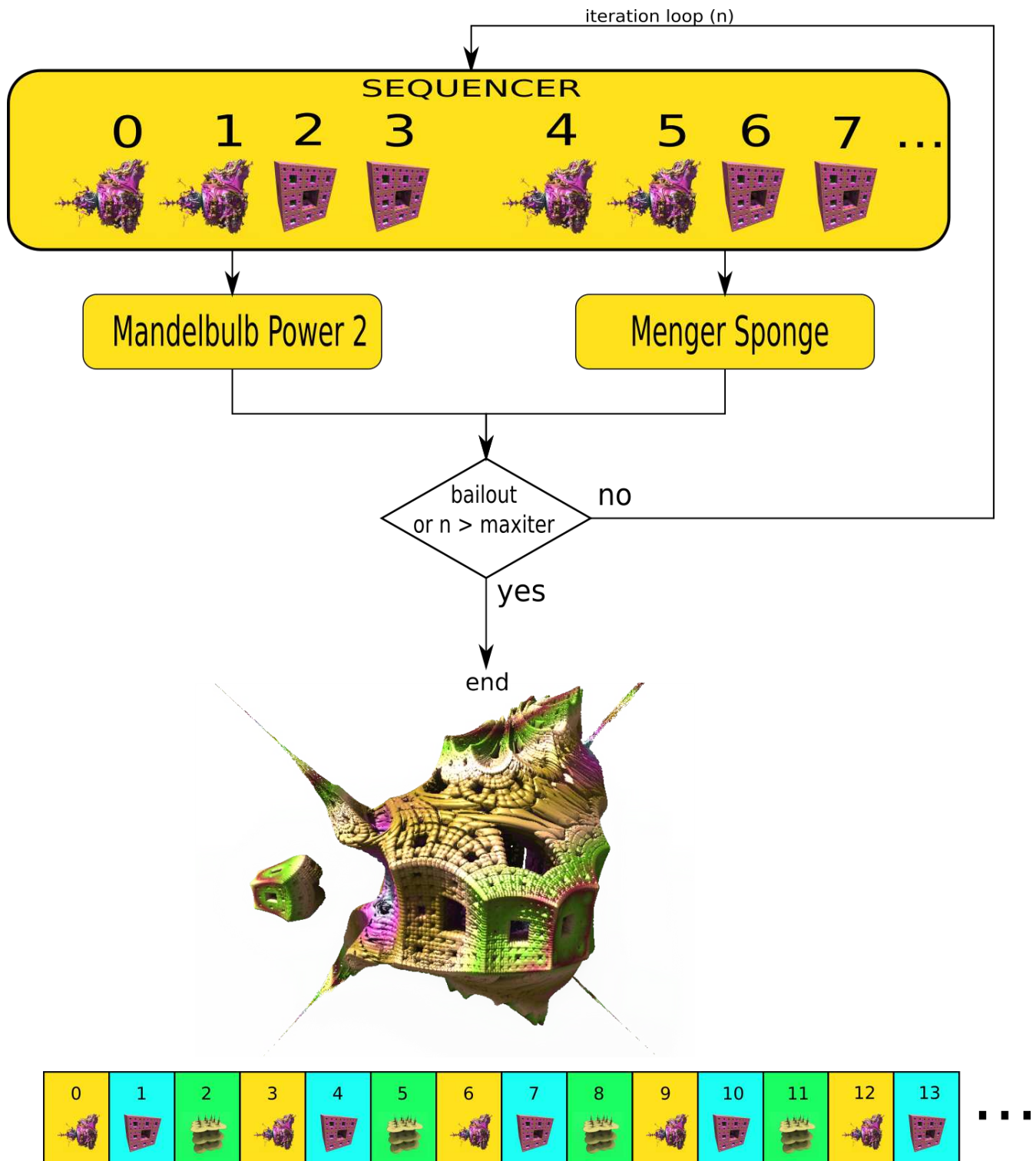


When calculation of the iteration loop finished the result value of  $z$  is used to estimate distance to fractal body and to calculate color of surface.

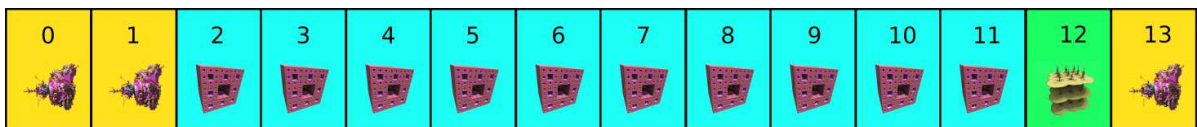
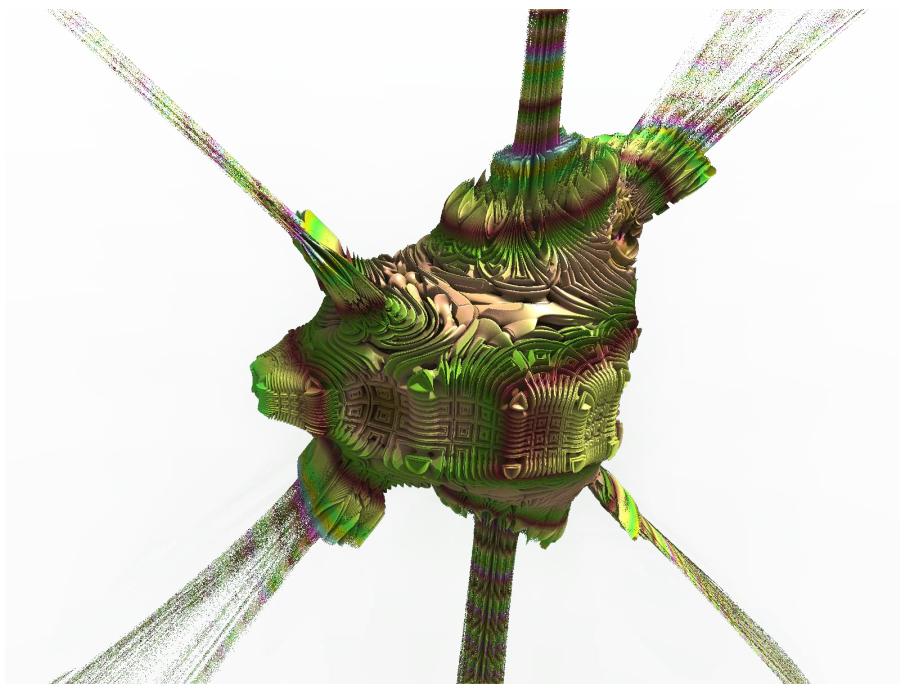
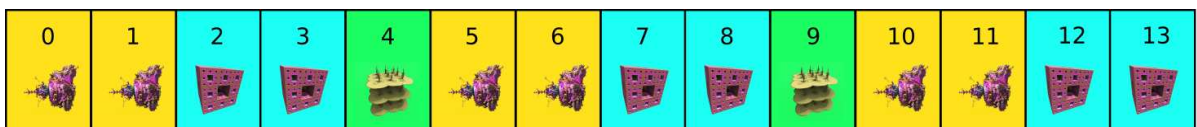
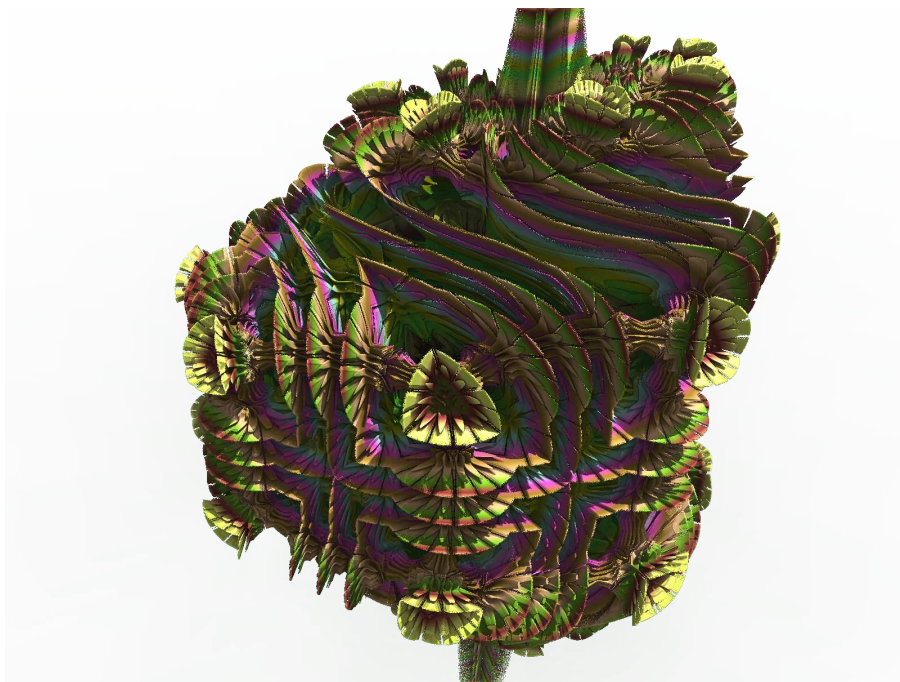
## 5.2 Hybrid fractals

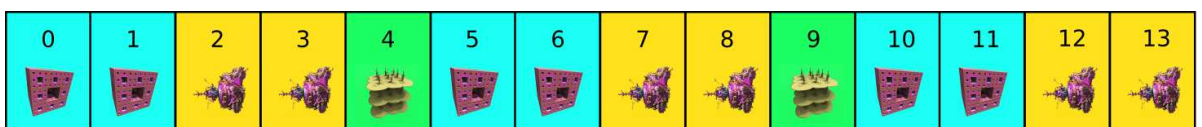
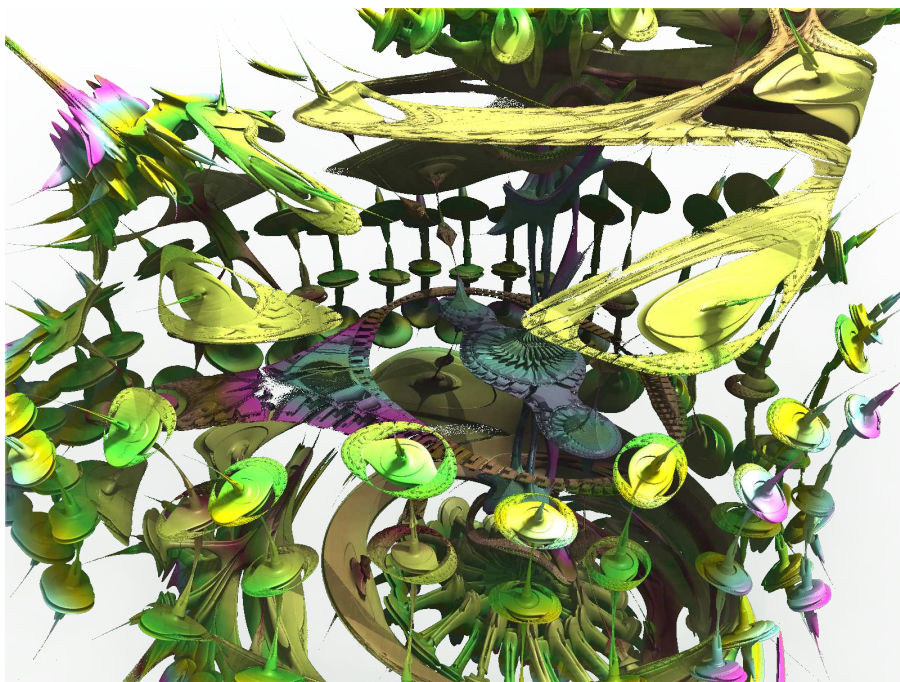
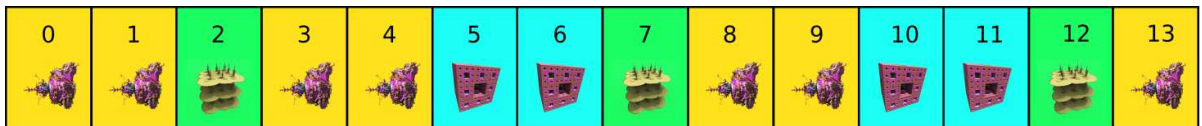
There is possible to mix different fractal formulas by alternating them, to get new fractal shapes. That fractal are named *hybrid fractals*. Mandelbulber program already have many different fractal formulas which gives opportunity to get very big variety of shapes. But using hybrid fractals increases possibilities a lot.

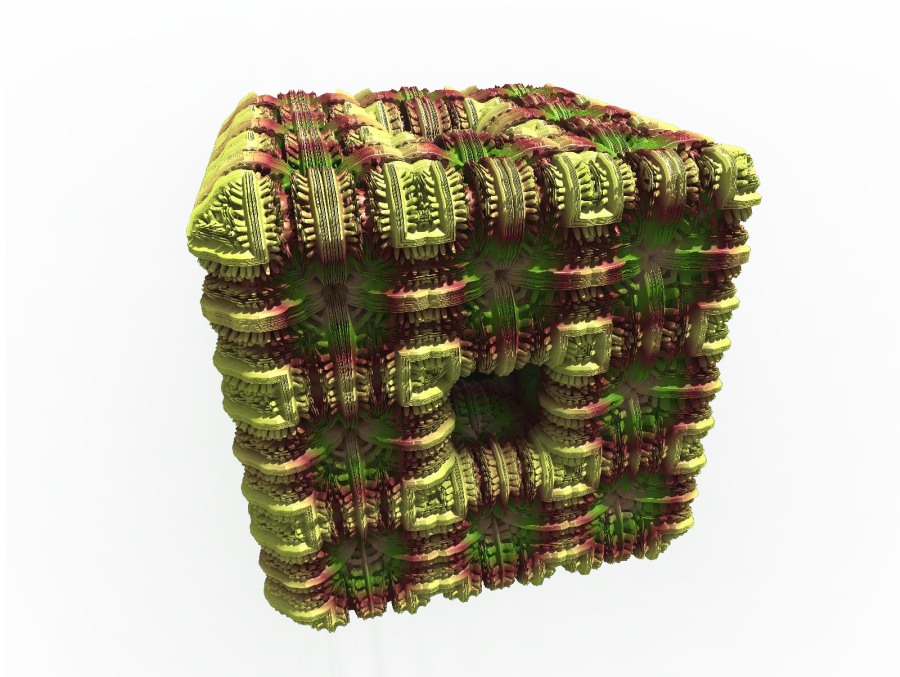
### 5.2.1 Processing of hybrid fractals











## 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 movement of the camera beneath the surface of fractal, because very close to the surface, a step needs to be very small.

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

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

Relative step mode makes navigation easier, because a user don’t need to think how much camera have to be moved to not hit the fractal surface.

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 animation camera flying with a fixed (or strictly controlled) speed.



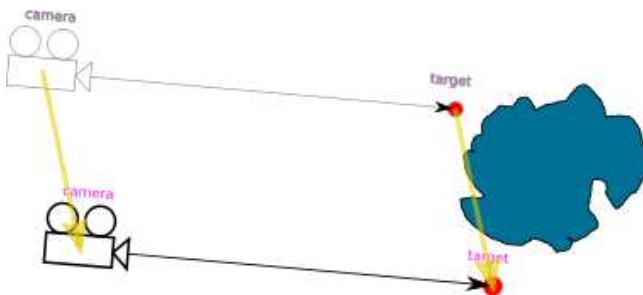
## 6.2 Linear camera and target movement modes using the arrow buttons

A user can navigate by operating the arrow keys 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

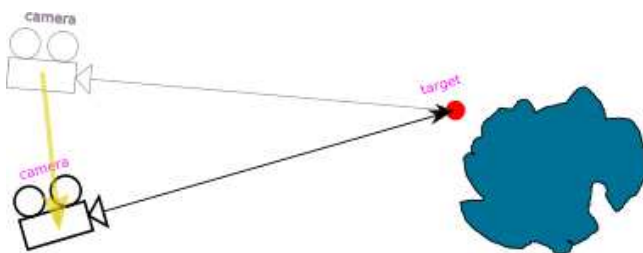
### 6.2.1 Move camera and target mode



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.

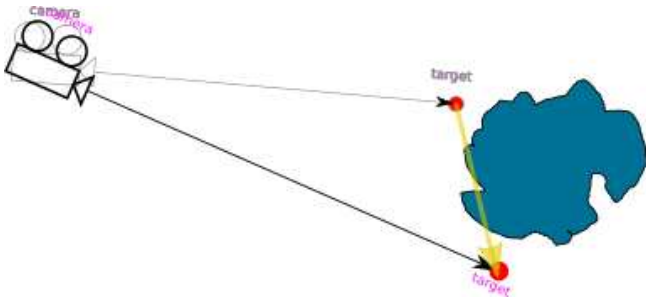
### 6.2.2 Move camera mode



Movement mode - *camera*

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

### 6.2.3 Move target mode



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 indicating point on image and use left mouse button (go forward) or right button (go 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

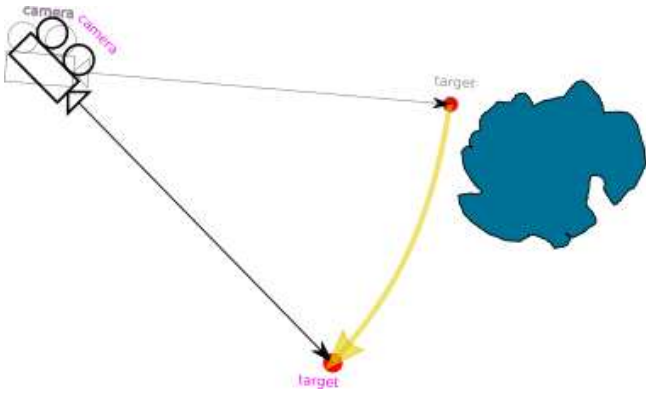
The camera is moved by the step (absolute or relative) in the direction of the mouse pointer, rotating the camera to look at the target. The target remains stationary.

### 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

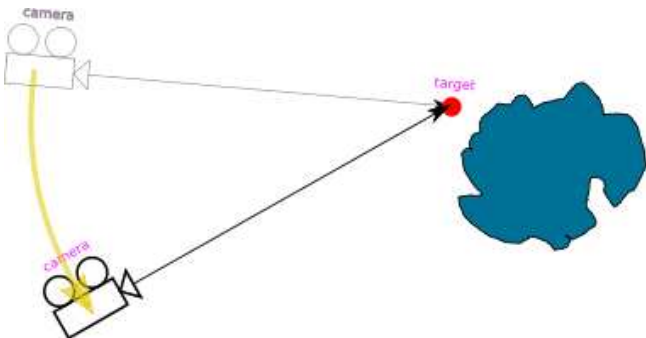
### 6.4.1 Rotate camera



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



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 maintaining 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 aircraft's 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).

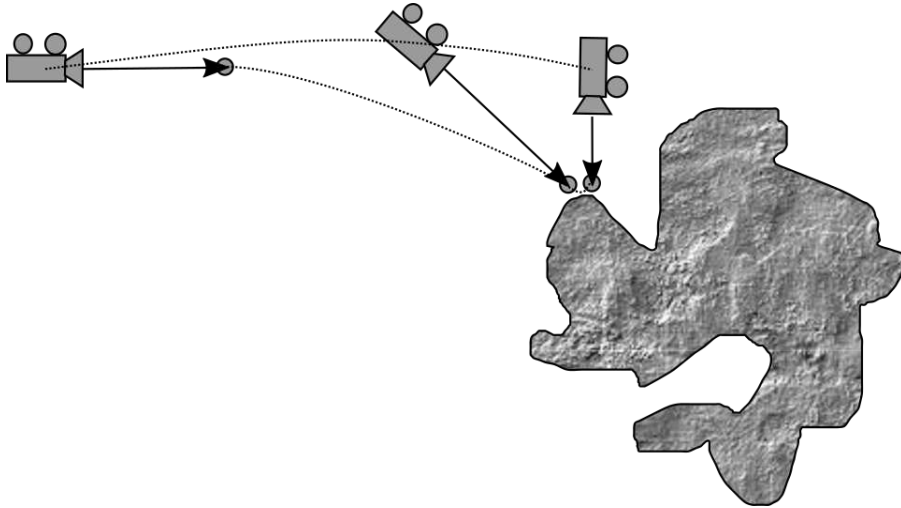
## 6.7 Camera rotation in animations

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 below). The camera positions represent three keyframes.

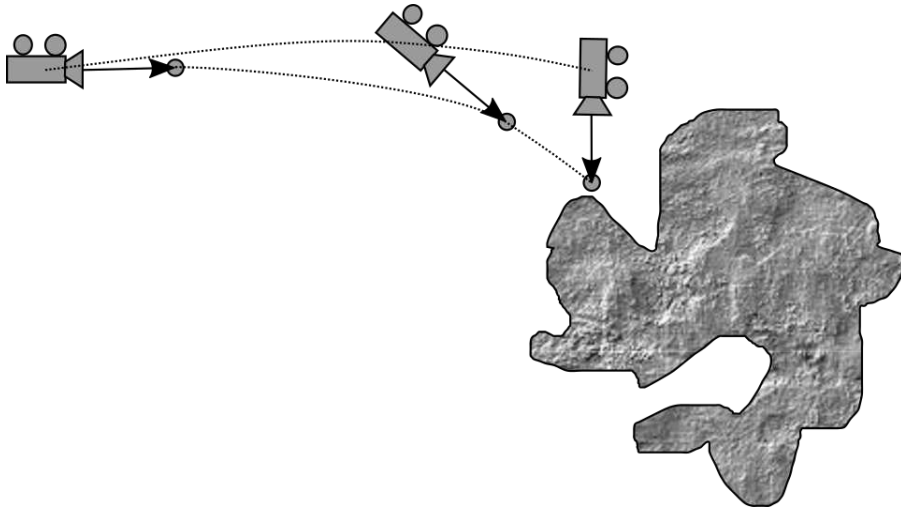




### Keyframe Animation with differing distances camera to target

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.



## 7 Interpolation

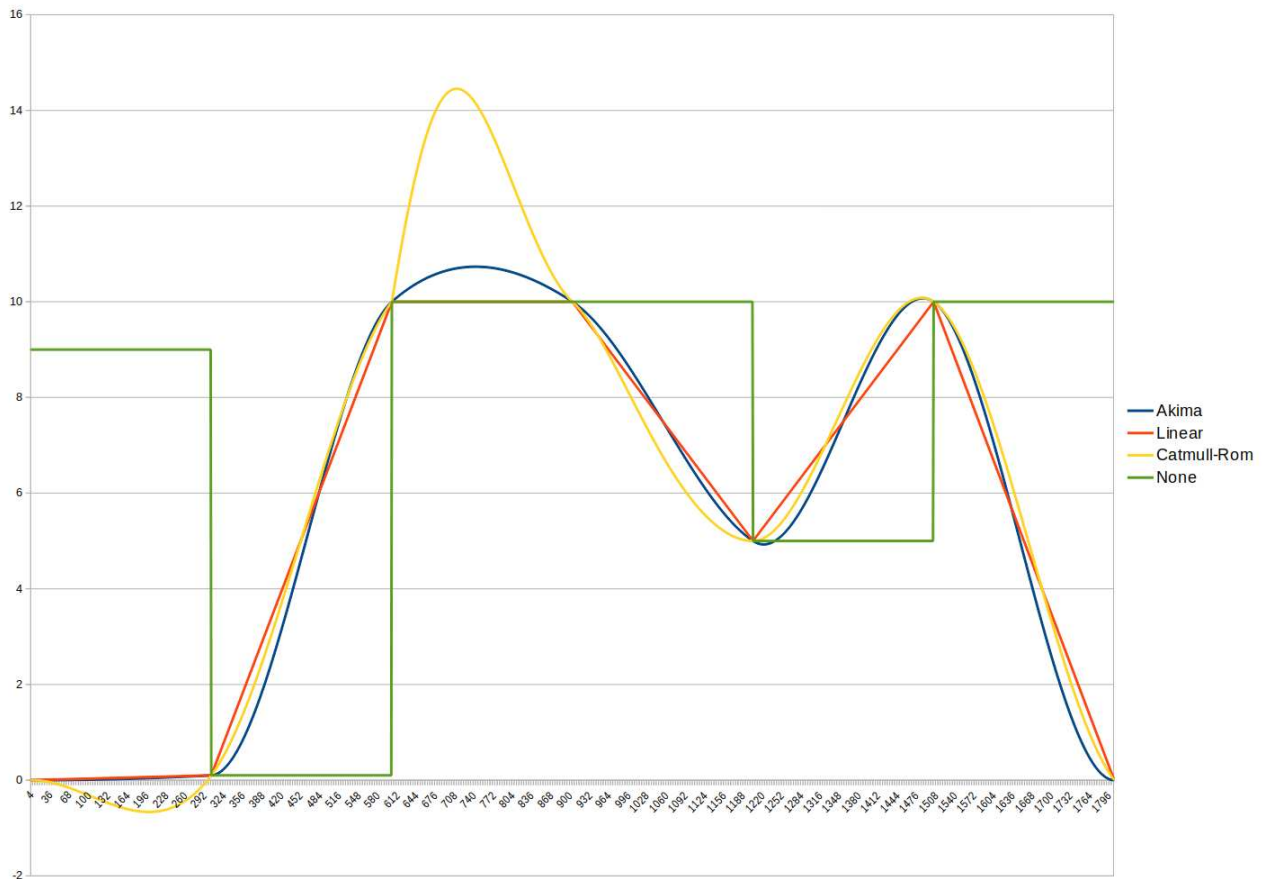
To get smooth transitions of parameters between keyframe there are used interpolation functions, which calculates intermediate values. There is no need for manual change of camera position and fractal parameters for every animation frame. Limited number of keyframes is enough to define good looking animation.

### 7.1 Interpolation types

There are implemented several interpolation functions:

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

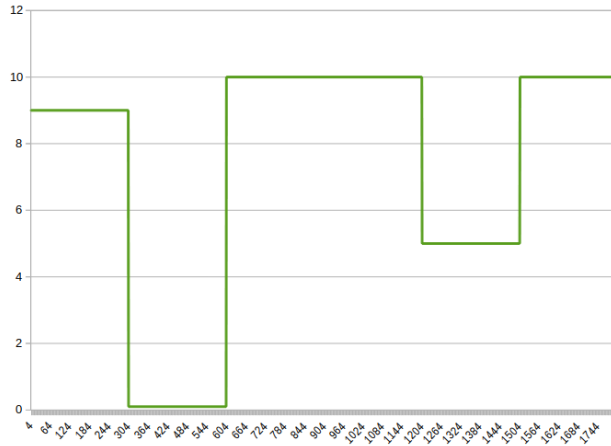
The chart bellow shows comparison between different interpolation modes



As it is visible, depending which interpolation function will be used, the final result of animation will be different.

### 7.1.1 Interpolation - None

Parameter is not interpolated. After every keyframe the value of parameter will have step change. This mode can be used with boolean values or with variables which have to be kept at constant levels for some number of keyframes.



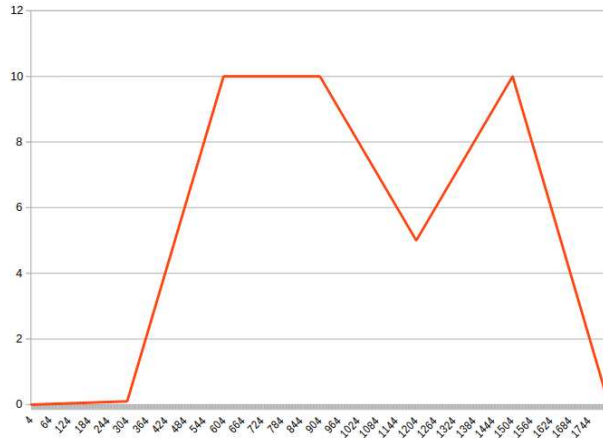
### 7.1.2 Interpolation - Linear

Value of 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's not recommended to use it for camera or objects movement paths, because of rapid changes of speed.



### 7.1.3 Interpolation - Linear angle

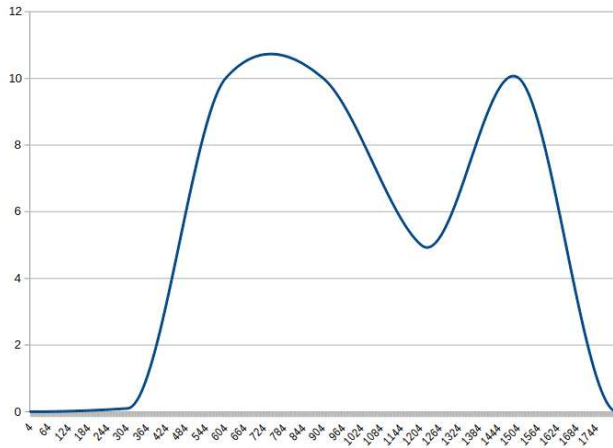
This interpolation mode works like *Linear*, but is prepared of angular parameters. If value exceed 360 degrees, then will go back to zero.

### 7.1.4 Interpolation - Akima

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

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

This interpolation method is very good for most of animated parameters. It can be used for camera and target animation and for many other parameters which should be animated in smooth way.



### 7.1.5 Interpolation - Akima angle

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

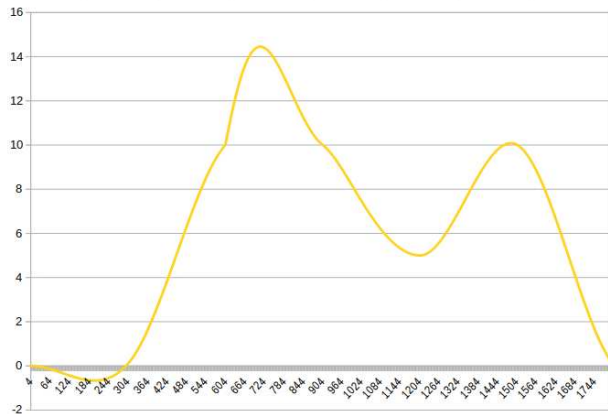
### 7.1.6 Interpolation - Catmul-Rom

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 like made of springy materials. It can be used to animate fractal parameters and also camera path. This interpola-

tion can produce oscillations, so has to be used carefully. As it is visible on the chart below, values went below zero when all of keyframe values were higher than zero.



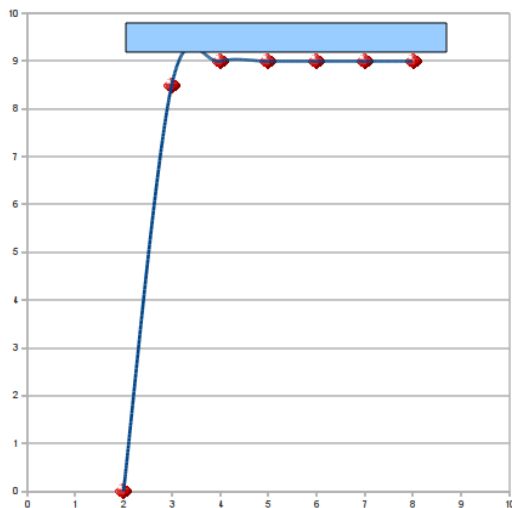
### 7.1.7 Interpolation - Catmul-Rom angle

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

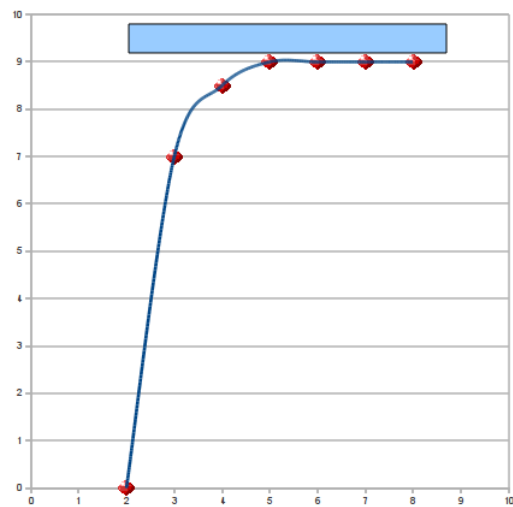
### 7.2.1 Collision

Fast approaching the obstacle may cause inadvertent drag to the camera towards the center of the object.



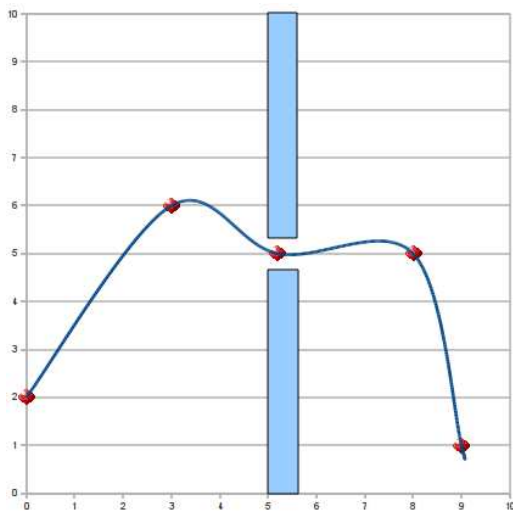
It is recommended to maintain the principle that one keyframe does not reduce the distance

to the object more than 5-times.



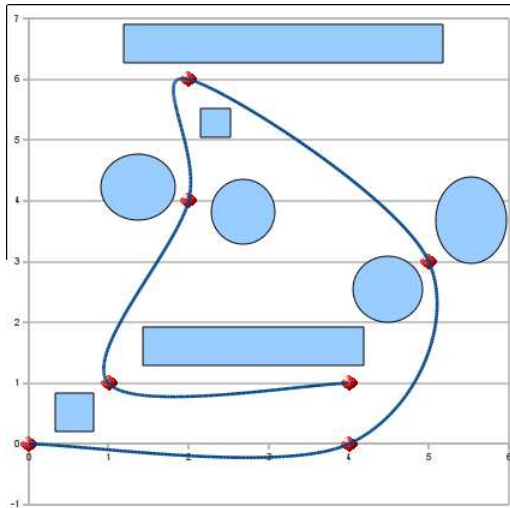
### 7.2.2 Fly through the gap

It is recommended to place a keyframe at the point where the camera flies through the gap



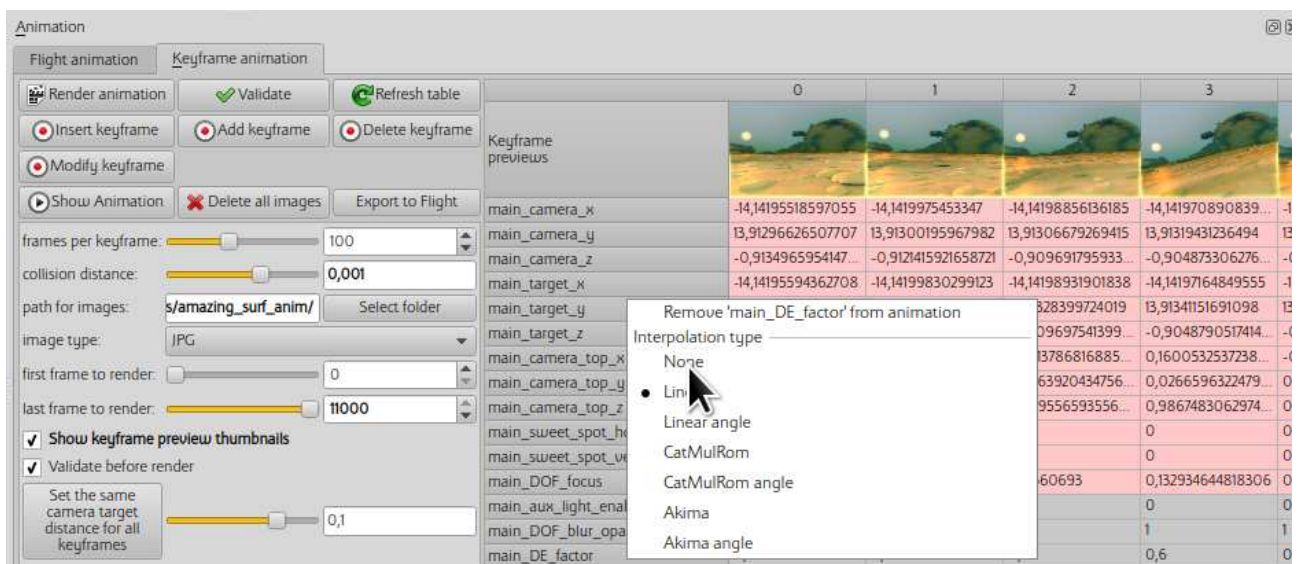
### 7.2.3 Proper conduct cameras between objects

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



## 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\_formula\_weight parameters have been changed from Akima to Linear. Interpolation type is color coded e.g. Linear parameters are highlighted in grey.



Animation

Flight animation | Keyframe animation

Render animation | Validate | Refresh table

Insert keyframe | Add keyframe | Delete keyframe

Modify keyframe

Show Animation | Delete all images | Export to Flight

frames per keyframe: 100

collision distance: 0,001

path for images: s/amazing\_surf\_anim/ | Select folder

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

Keyframe previews

	0	1	2	3
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	13,91300195967982	13,91306679269415	13,91319431236494	13,91341151691098
main_target_z	-0,909691795933...	-0,904873306276...	-0,900532537238...	-0,89697541399...
main_camera_top_x	13,91300195967982	13,91306679269415	13,91319431236494	13,91341151691098
main_camera_top_y	-0,909691795933...	-0,904873306276...	-0,900532537238...	-0,89697541399...
main_camera_top_z	-0,909691795933...	-0,904873306276...	-0,900532537238...	-0,89697541399...
main_sweet_spot_h	0	0	0	0
main_sweet_spot_v	0	0	0	0
main_DOF_focus	0	0	0	0
main_aux_light_ena	0	0	0	0
main_DOF_blur_opa	0	0	0	0
main_DE_factor	0,6	0,6	0,6	0,6

Interpolation type

- None
- Linear
- Linear angle
- CatMulRom
- CatMulRom angle
- Akima
- Akima angle

## 8 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.

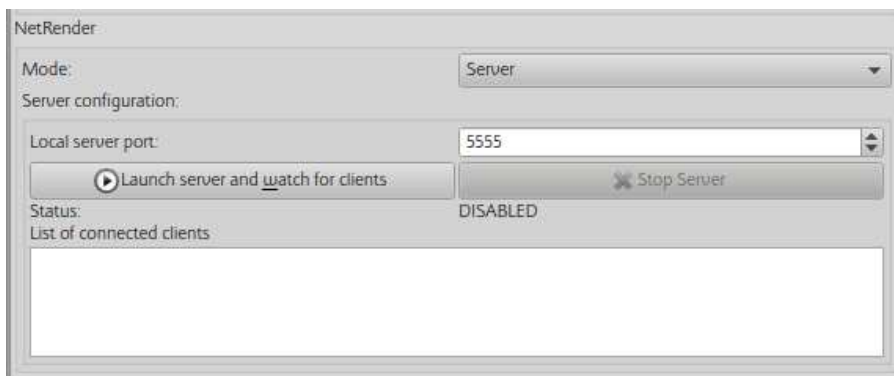
The Server is also the computer which renders the combined image .

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

### 8.1 Starting NetRender

#### 8.1.1 Server configuration

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



*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

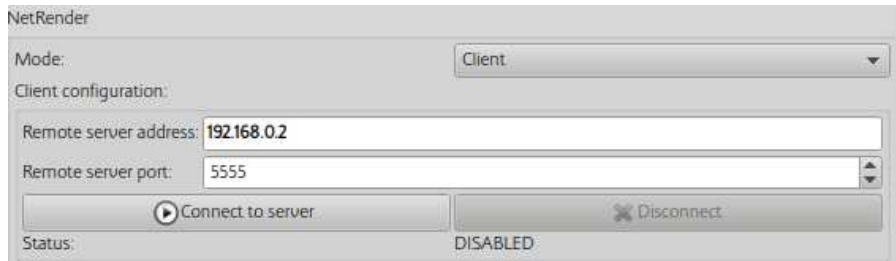
Alternative way to launch the server is to use command line. Example:

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

#### 8.1.2 Configuring the clients

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



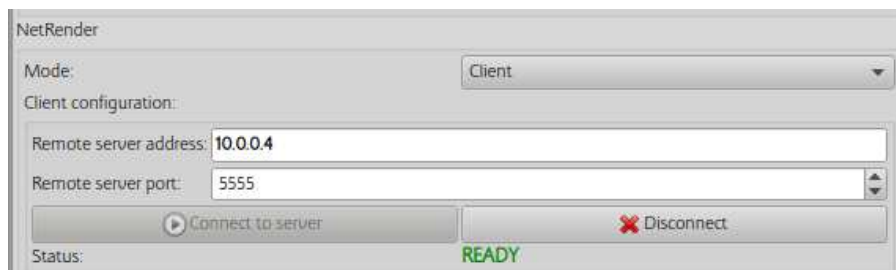


The remote server address must be set to the same as the Server computer which is running Mandelbulber in Server mode. The address can be given as an IP address or a computer name.

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*



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

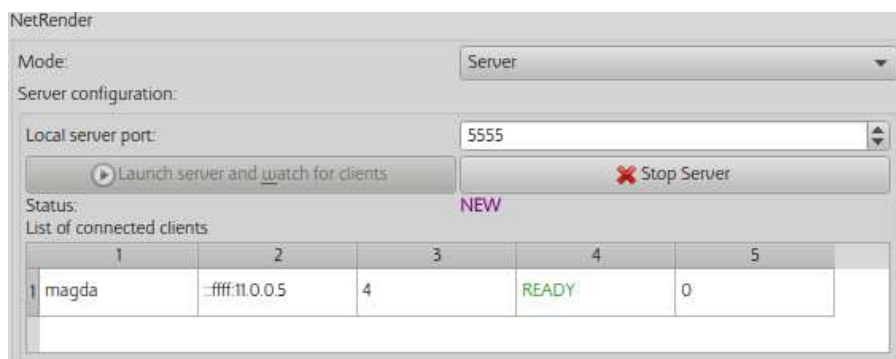
```
$ mandelbulber2 --nogui --host 10.0.0.4 --port 5555
```

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 “magda” computer has 4 cores and is *READY*.



### 8.1.3 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.

## 9 Case study

### 9.1 Examples

#### 9.1.1 Example of MandelboxMenger UI

Example settings

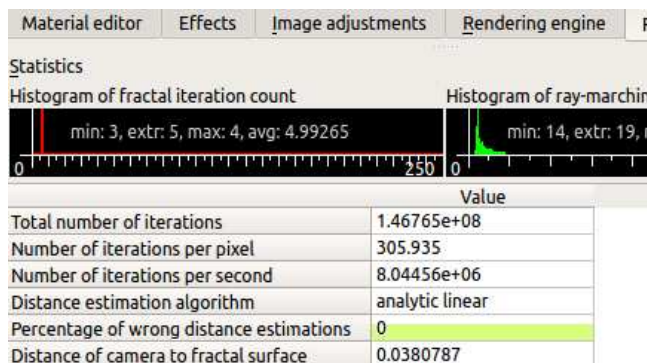
(copy to clipboard, then load in Mandelbulber using : *File – Load settings from clipboard*):

```
# 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) 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.



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, and 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 =  $54.7356^\circ$ .

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  $35.2644^\circ$  ( $= 90^\circ - 54.7356^\circ$ ). The fractal will then render parallel to the x-y plane.

### 9.1.2 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.

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

This is how I have been doing it.

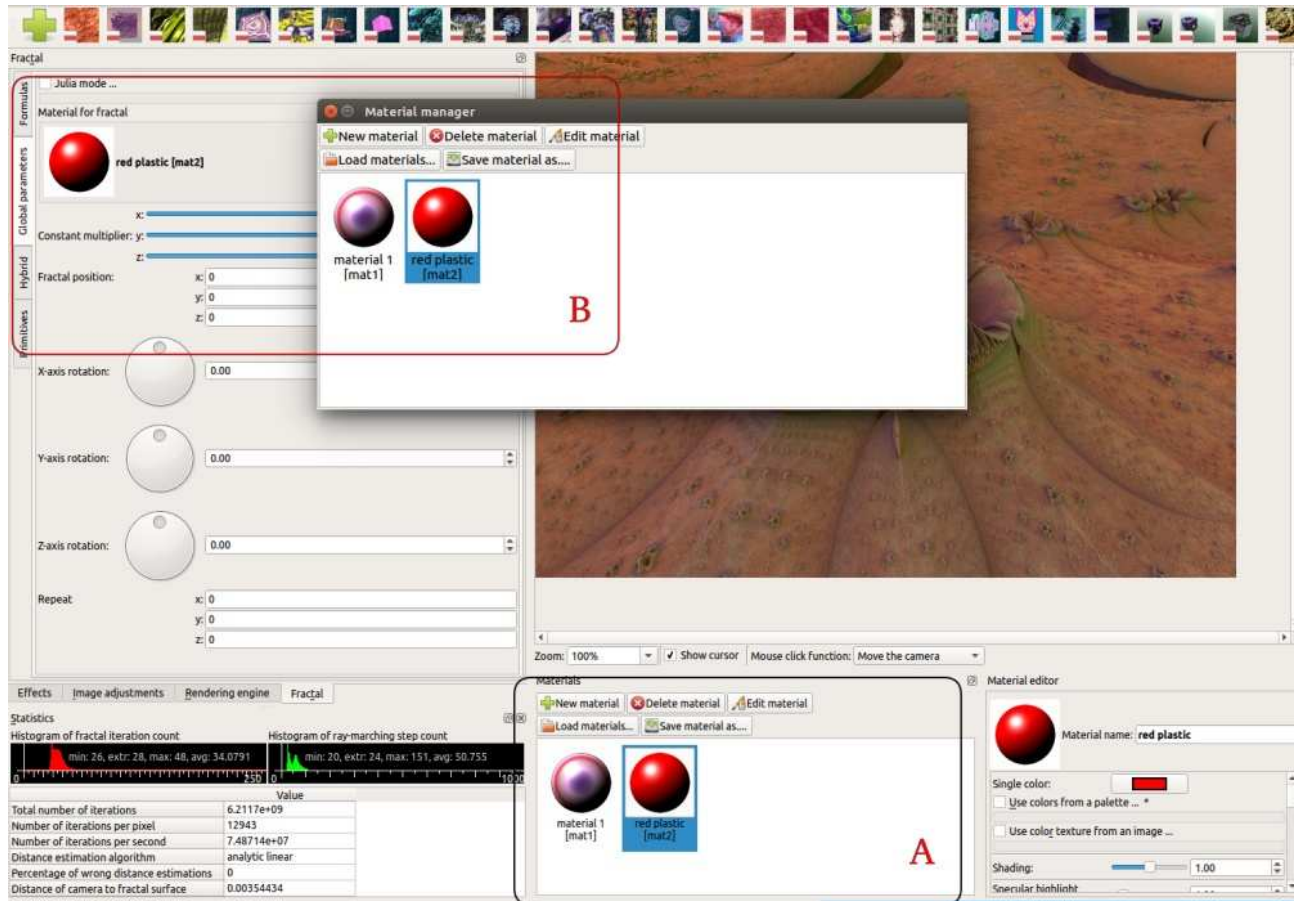
*Rectangle at the bottom marked A.*

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.*

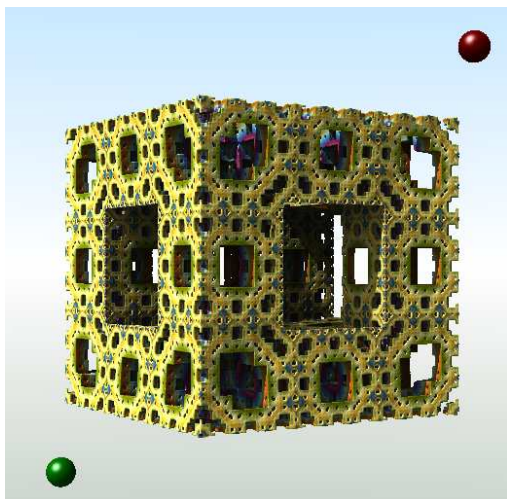
One way to use a material is to go to Global Parameters, click on the material preview image, and the **Material Manager** UI will appear with the materials you have loaded or created. Click on the one you want to use, then close that UI.

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.



## 10 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 or 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.

$\text{animVal} = \text{paraVal} + \text{soundVal}$ .

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)

<http://cdn.mandelbulber.org/doc/audio/9%20tut.zip>



unzip and place “9 tut” folder in home/mandelbulber/animations/

## 10.1 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

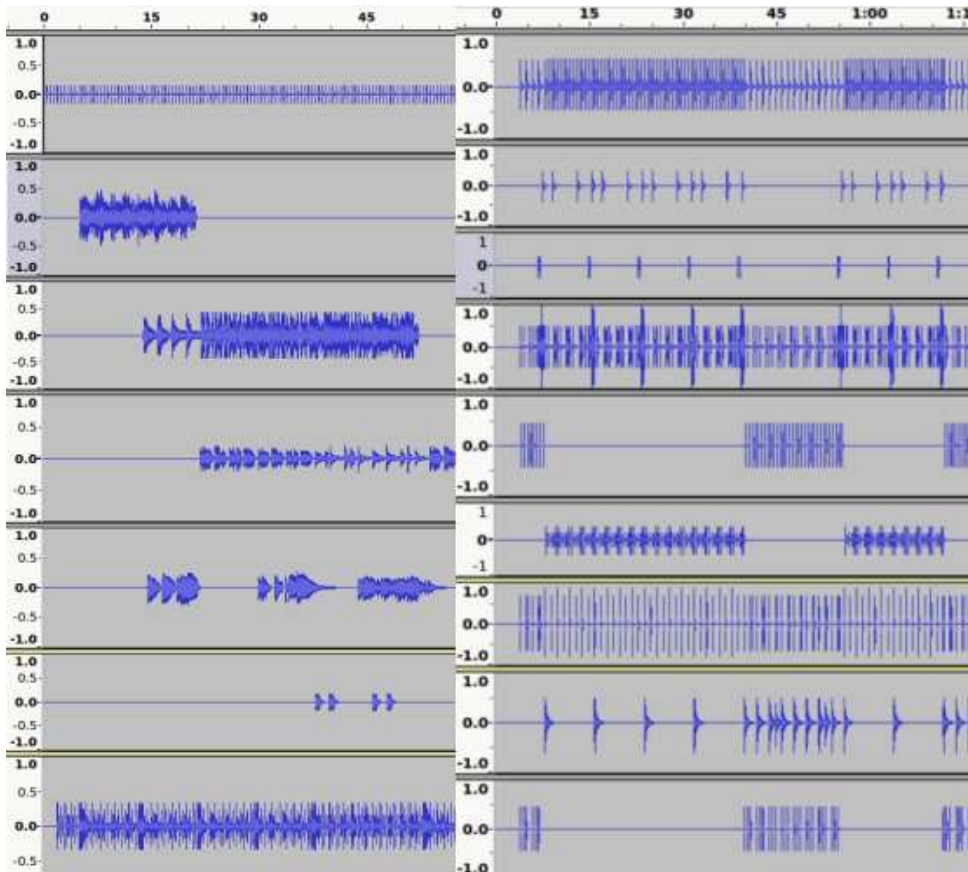
I used Hydrogen Drum Kit emulator to make all the individual drum track files. These are .wav files but as I am using Linux I converted them to .mp3 to use in Mandelbulber. These are mono working files, when I make the video in VirtualDub I then use the final song mix .wav file.

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.



## 10.2 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.

**Animation**

Flight animation    Keyframe animation

Render animation    Validate    Refresh table

Insert keyframe    Add keyframe    Delete keyframe

Modify keyframe

Show Animation    Delete all images    Export to Flight

frames per keyframe: 7920

frames per second: 30.0000

collision distance: 1e-06

path for images: ilber/animation/tut/    Select folder

	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
<b>fractal0_transf_addition_constant_y</b>	<b>Audio loaded</b>
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*.

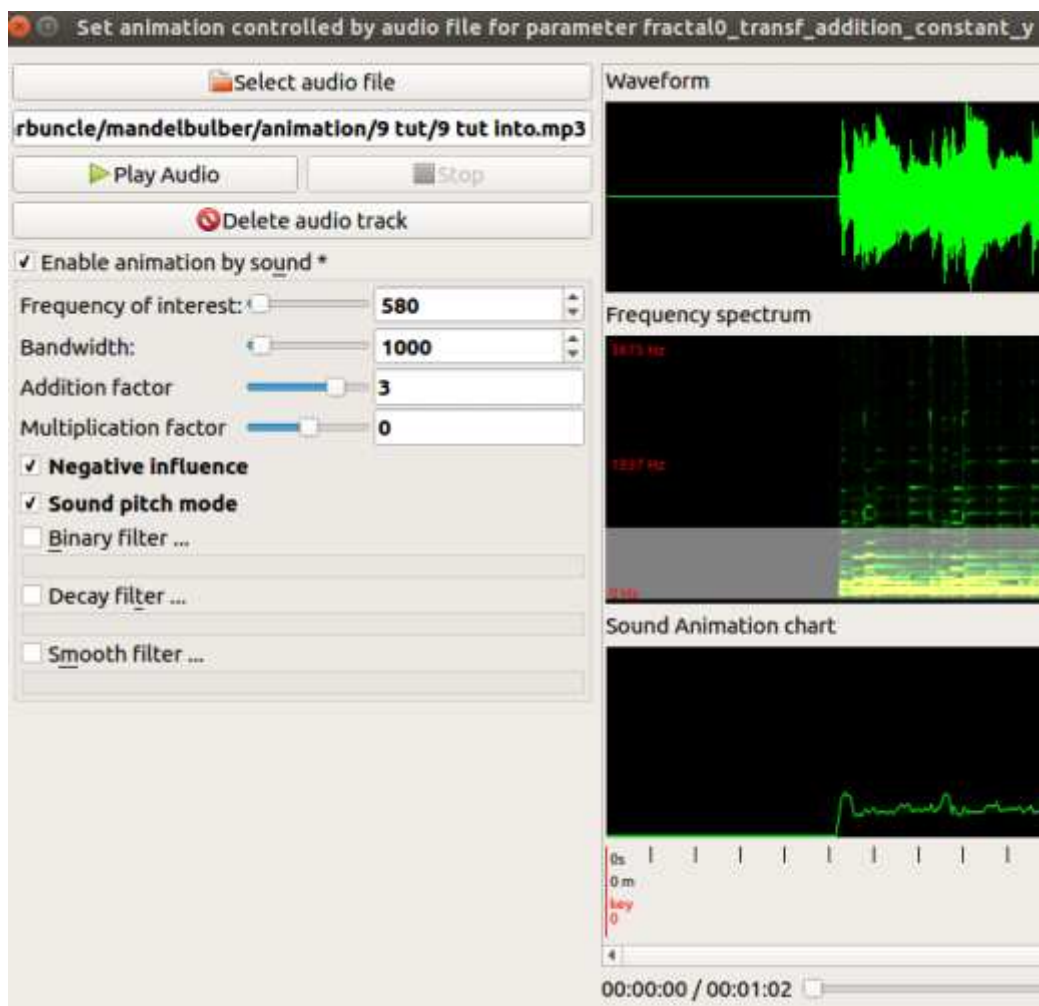
Parameters x & z are also added because this parameter is part of a vector3.

### 10.3 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.



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

$\text{animVal} = \text{paraVal} + \text{soundVal}$ .

$\text{soundVal} = (\text{paraVal} * \text{multiplication-factor} * \text{sound}) + (\text{addition-factor} * \text{sound})$

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

## 10.4 Using Sound Pitch mode.

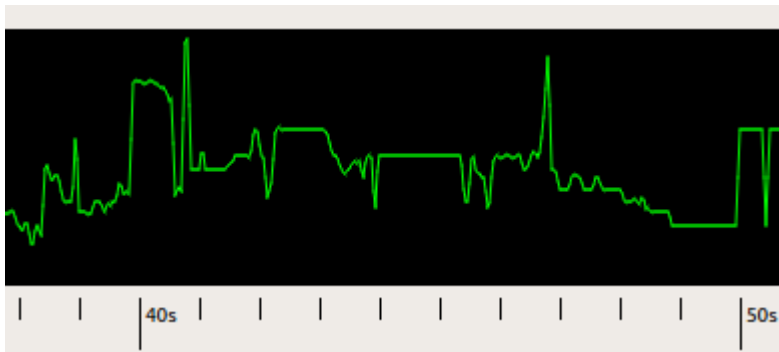
I set frequency at 580Hz and bandwidth to 1000Hz, this covers the range of the fundamental frequencies of my lead guitar notes ( I am removing higher harmonic frequencies, although this may not be necessary).

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.

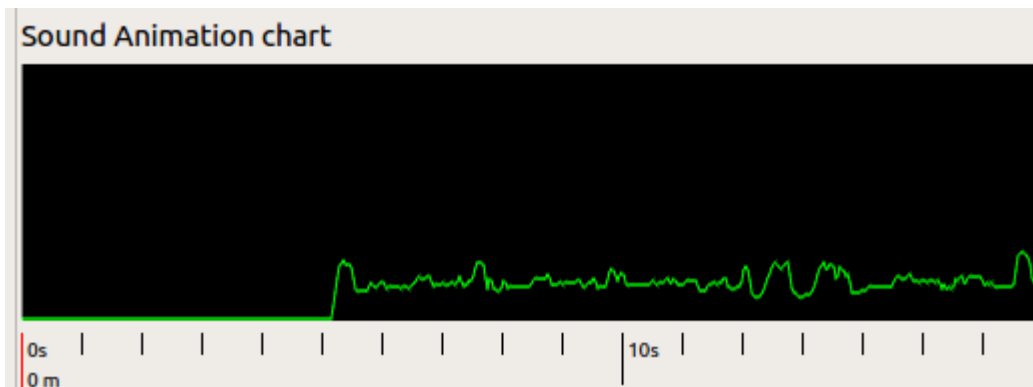
The main point is not to limit the Pitch by having a small band width that does not cover the full spectrum of the fundamental notes used in the audio track.

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.

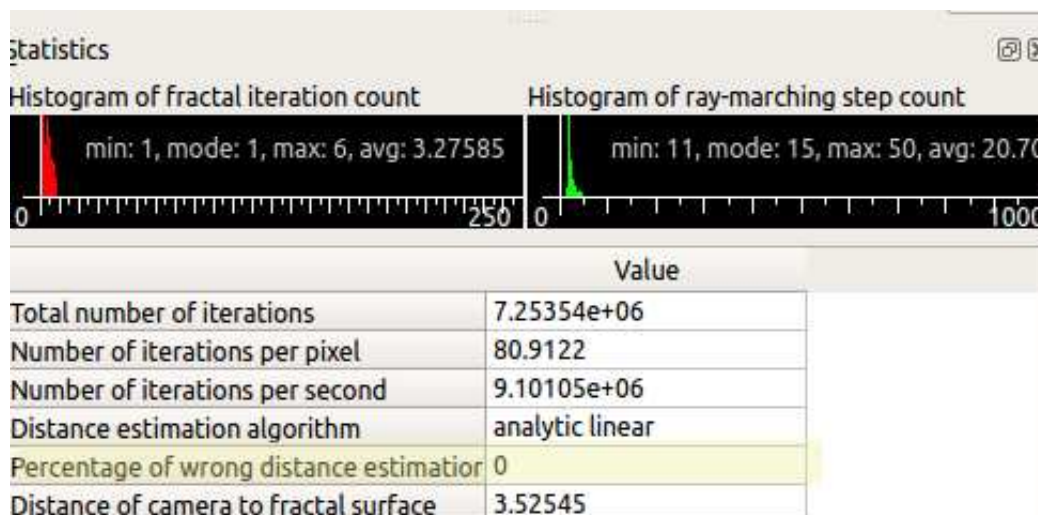


## 10.5 Testing the parameter

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.*



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.  $0.0 = 0.0 * -3.0 \text{ max}$ ,  $-0.6 = 0.2 * -3.0 \text{ max}$ .

*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.

## 10.6 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.

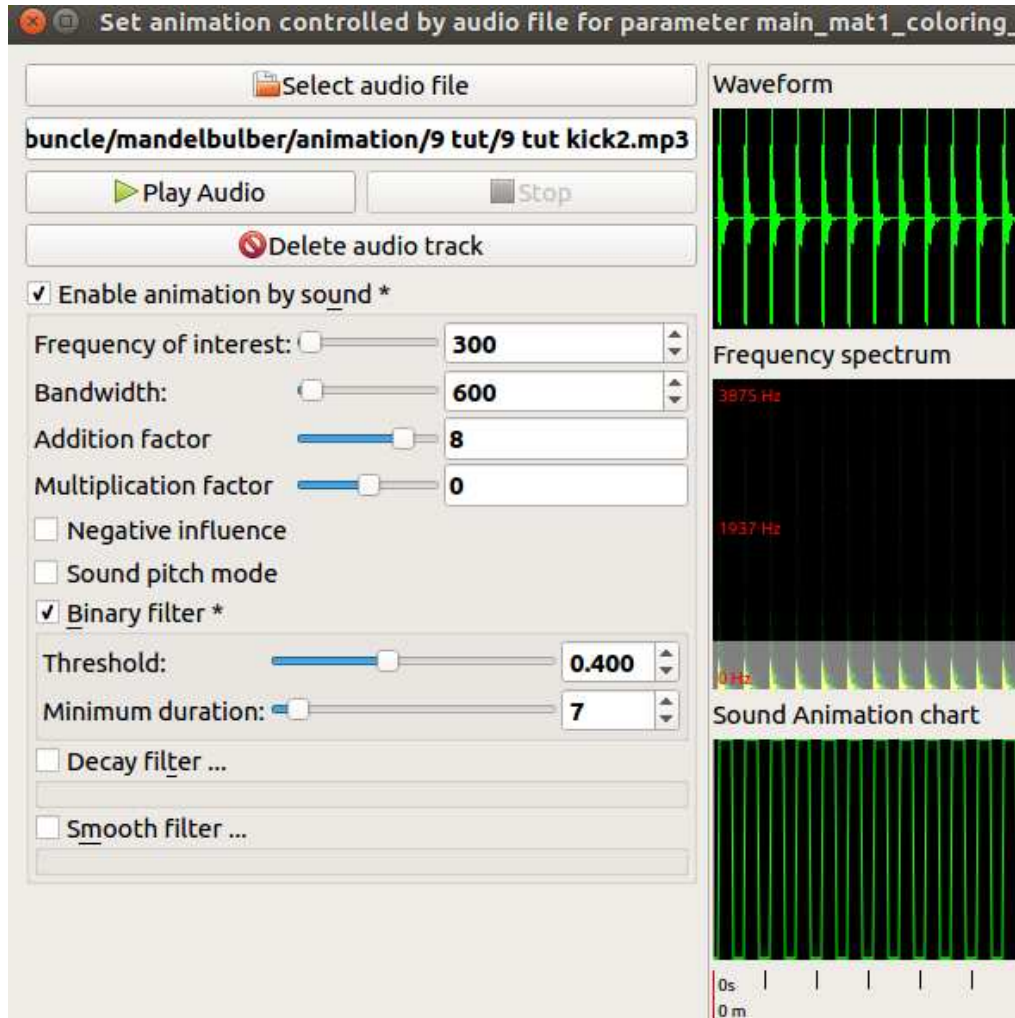
Open Anim By Sound, load audio file and Enable Animation by sound.

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.





## 10.7 Rendering the animation.

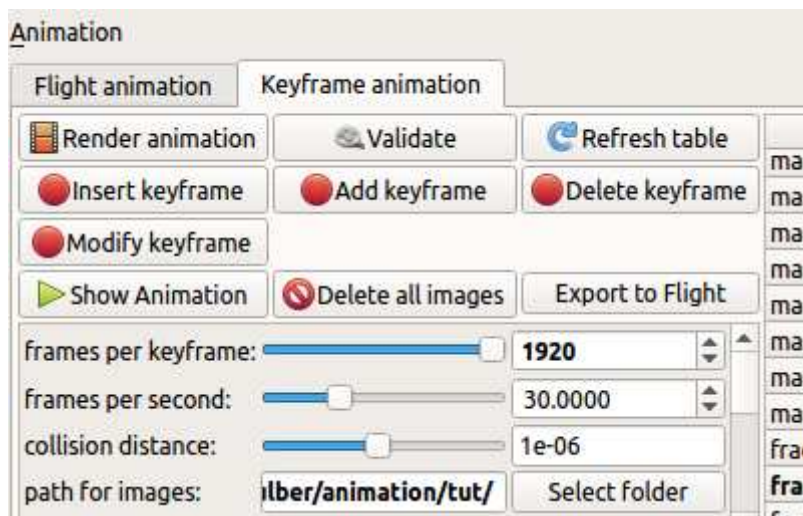
First add two identical keyframes to the Keyframe Animation table,

Refresh table				
Delete keyframe				
Export to Flight				
1920				
30.0000				
1e-06				
Select folder				
		Audio	0 (0:00)	1 (1:04)
main_camera_z	Anim By Sound	0	0	0
main_target_x	Anim By Sound	0	0	0
main_target_y	Anim By Sound	0	0	0
main_target_z	Anim By Sound	0	0	0
main_camera_top_x	Anim By Sound	0	0	0
main_camera_top_y	Anim By Sound	0	0	0
main_camera_top_z	Anim By Sound	1	1	1
fractal0_transf_addition_constant_x	Anim By Sound	0	0	0
fractal0_transf_addition_constant_y	Audio loaded	0	0	0
fractal0_transf_addition_constant_z	Anim By Sound	0	0	0
main_mat1_coloring_palette_offset	Audio loaded	0	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:

$64\text{sec} * 30\text{fps} = 1920$  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.



## 10.8 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.

## 11 Thanks

Thanks to the fractal community for your ongoing support!

Sincerely,

Mandelbulber Team



# Index

- animation
  - rotation, [18](#)
- distance estimation, [8](#)
  - analytical, [8](#), [9](#)
  - delta DE, [8](#), [9](#)
- fractal, [5](#)
  - hybrid, [30](#), [31](#)
- IFS, [7](#)
- interpolation, [20](#)
  - Akima, [22](#)
  - Catmul-Rom, [22](#)
  - linear, [21](#)
  - none, [21](#)
- Mandelbrot Set, [5](#)
- Mandelbulb, [6](#)
- Menger Sponge, [7](#)
- navigation
  - absolute step, [14](#)
  - camera, [14](#), [15](#)
  - relative step, [14](#)
  - reset view, [17](#)
  - rotate, [17](#)
  - target, [14](#), [16](#)
- NetRender, [26](#)
- ray marching, [8](#)
  - distance threshold, [8](#), [11](#)
  - maximum view distance, [30](#)
  - step multiplier, [8](#), [30](#)
- statistics
  - wrong distance estimations, [10](#), [29](#)
- termination condition
  - bailout, [6](#), [11](#)
  - maxiter, [6](#), [12](#)