Math Chaos Patterns 2.8.1

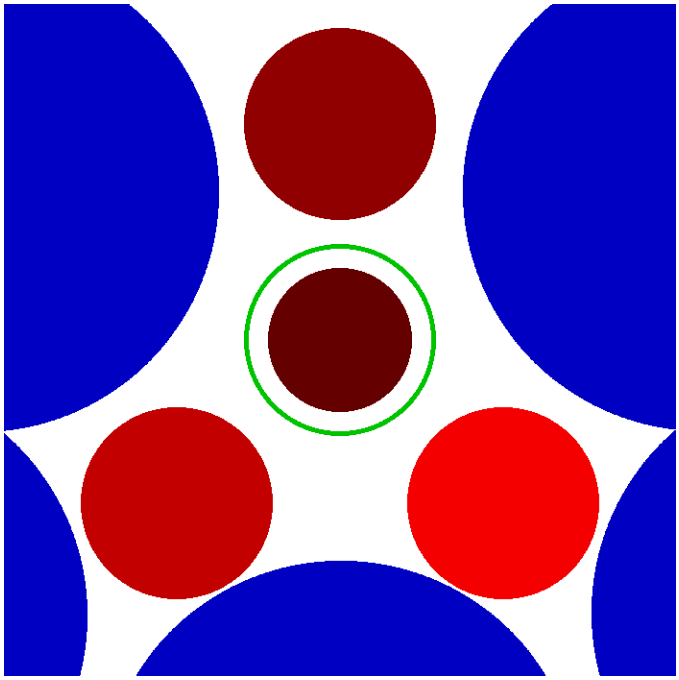
D1B LLC

With Math Chaos Patterns (MCP), you can control thousands of brushes on a digital canvas. It provides a visual programming environment for painting that requires no coding. Brushes are managed through the user interface (UI) and through user-defined images with color-coded areas, where each pixel can trigger instructions that change the direction, speed, transparency, and other properties of passing brushes.

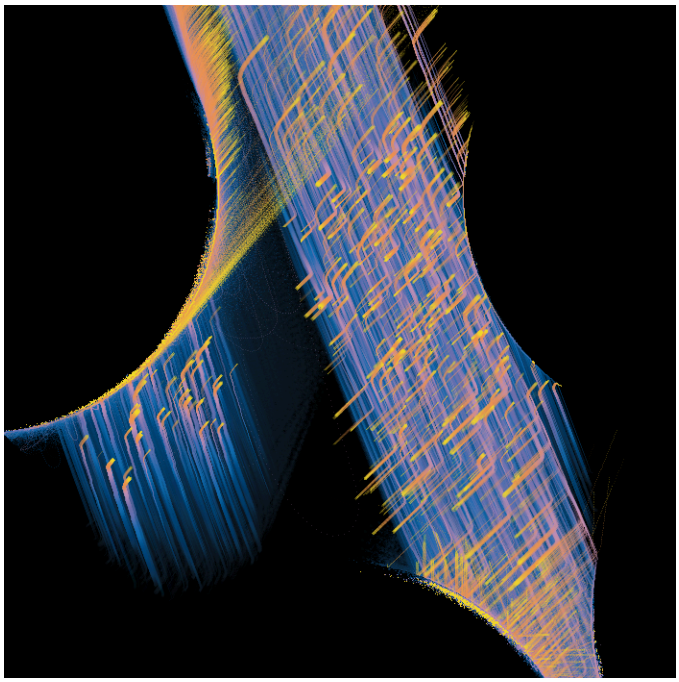
To master MCP, you need to understand four core concepts: brushes, the canvas, vector fields, and interactions between brushes. These concepts are unique to MCP and are unlike anything found in conventional digital art software. The following section provides a brief introduction to these concepts. More detailed explanations can be found throughout this manual.

Each brush is circular and consists of circular bristles. You can control the brush density as well as the size of each bristle tip. Brushes and their bristles can also interact with one another as they move across the canvas. Each brush moves across the canvas step by step. You can observe the output after every step or after several steps to reduce rendering time. The total number of steps taken by brushes can be configured. Brush speed is measured in pixels per step. The default speed is 1 pixel per step, but it can also be a fractional value, such as 0.5 pixels per step, or a larger value, such as 5 or 10 pixels per step.

The MCP canvas is covered by areas dedicated to different actions, some of which may overlap. Only one area is mandatory: the area where brushes can originate. All other areas are optional. As a result, the MCP canvas can be viewed as a special type of digital image. Unlike a conventional image, where each pixel has only one property, namely its color, each element of the MCP canvas has multiple properties and functions. These properties allow canvas elements to interact with passing brushes. For example, an element can change a brush's speed or direction, reflect the brush, absorb it entirely, or trigger other actions. The image Canvas.png (shown below) provides an example of how MCP loads and interprets different canvas areas. Red pixels indicate areas where brushes can originate. Notice the four different shades of red, which MCP uses to define four brush groups. This grouping enables special interactions between brushes belonging to different groups. Green pixels are used to define the initial direction of brushes, while blue pixels indicate areas that reflect brushes.



The resulting output shown below does not look much like Canvas.png, because Canvas.png is only one of many inputs used by MCP to generate the final image. This example highlights an important aspect of MCP: to master it, you need to spend some time experimenting and exploring how different parameters interact with one another.



The vector field adds another property to each canvas element, called a preferred direction. Each brush has a parameter that determines the extent to which it is affected by the vector field. For example, if a brush passes through an element whose preferred direction is Up, the element may force the brush to abandon its current direction and move upward. Alternatively, the brush may completely ignore the preferred direction and continue along its current path, or it may adjust its direction only slightly toward Up. As a result, the vector field can have a strong influence on brush movement, a subtle influence, or no influence at all, depending on the brush's [Impact of Vector Field] parameter (see below).

Another important property of each canvas element is memory. Every element can remember the last brush that passed through it. Similar to human memory, this memory gradually fades over time, but its behavior can be controlled in MCP through the brush fading parameter. There are three important aspects of this property:

1. Every element remembers the parameters of the last brush that passed through it.
2. The number of steps for which an element remembers a brush is controlled by that brush's fading value: the lower the fading value, the longer the brush will be remembered.
3. With each step, the memory gradually fades. Each passing brush overwrites the previously stored memory.

Brush interactions are directly related to this memory mechanism. Each brush may interact with other brushes within its neighborhood on the canvas. The lower the fading value, the longer brushes remain stored in the memory of canvas elements. Consequently, more brushes may be remembered within a neighborhood and can therefore interact with the current brush. In other words, lower fading values effectively increase the number of brushes available for interaction.

This manual is organized as follows. Part One introduces the major elements of UI and explains each parameter in detail. Part Two shows how these parameters interact to help you create impressive digital paintings. Part Three has some additional useful information.

We have invested a great deal of time in creating video lessons, demos, and this manual to help you learn MCP. If you still have questions or get stuck, please email us at D1BLLC@MathChaosPatterns.com. We will be happy to help.

Table of Contents

Part One **5**
MCP screen 5
MCP menu 6
Work folder 7
MCP parameters 8
Parameters to control the canvas 10
Parameters to control brushes 14
Parameters to control interactions between brushes 18
Parameters to control the rendering process 20
Part Two **24**
Tutorials 24
Deep In Matrix 25
Flower 31
Illusion Of Depth 35
Two Groups 38
Jellyfishes 43
Cells Experiment 47
Entrance 51
Seaweed 56
Flow 59
Part Three **61**
Important updates, hints, and tips 61
How to quickly create a video from MCP output? 63
Want to know more? 64

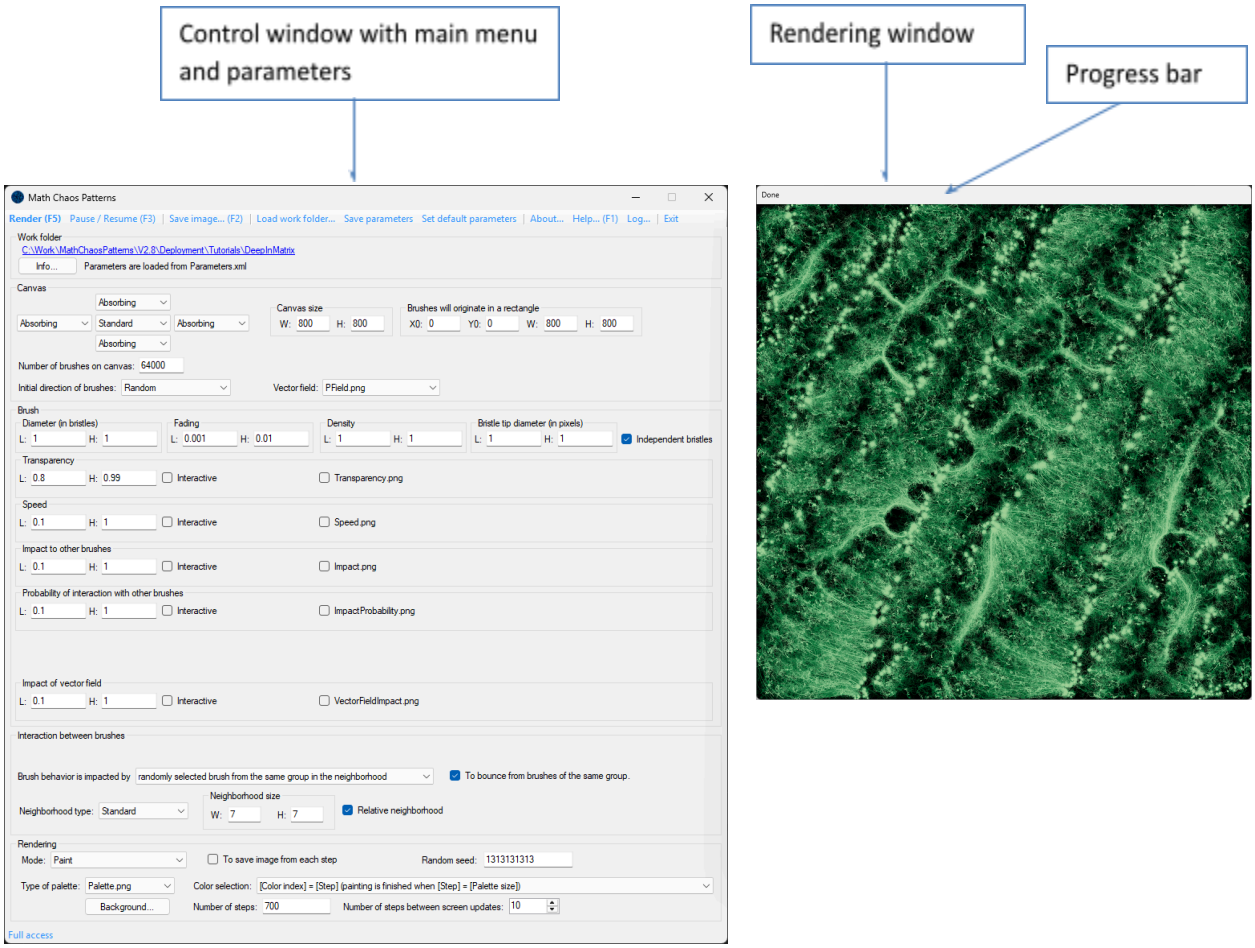
Part One

MCP screen

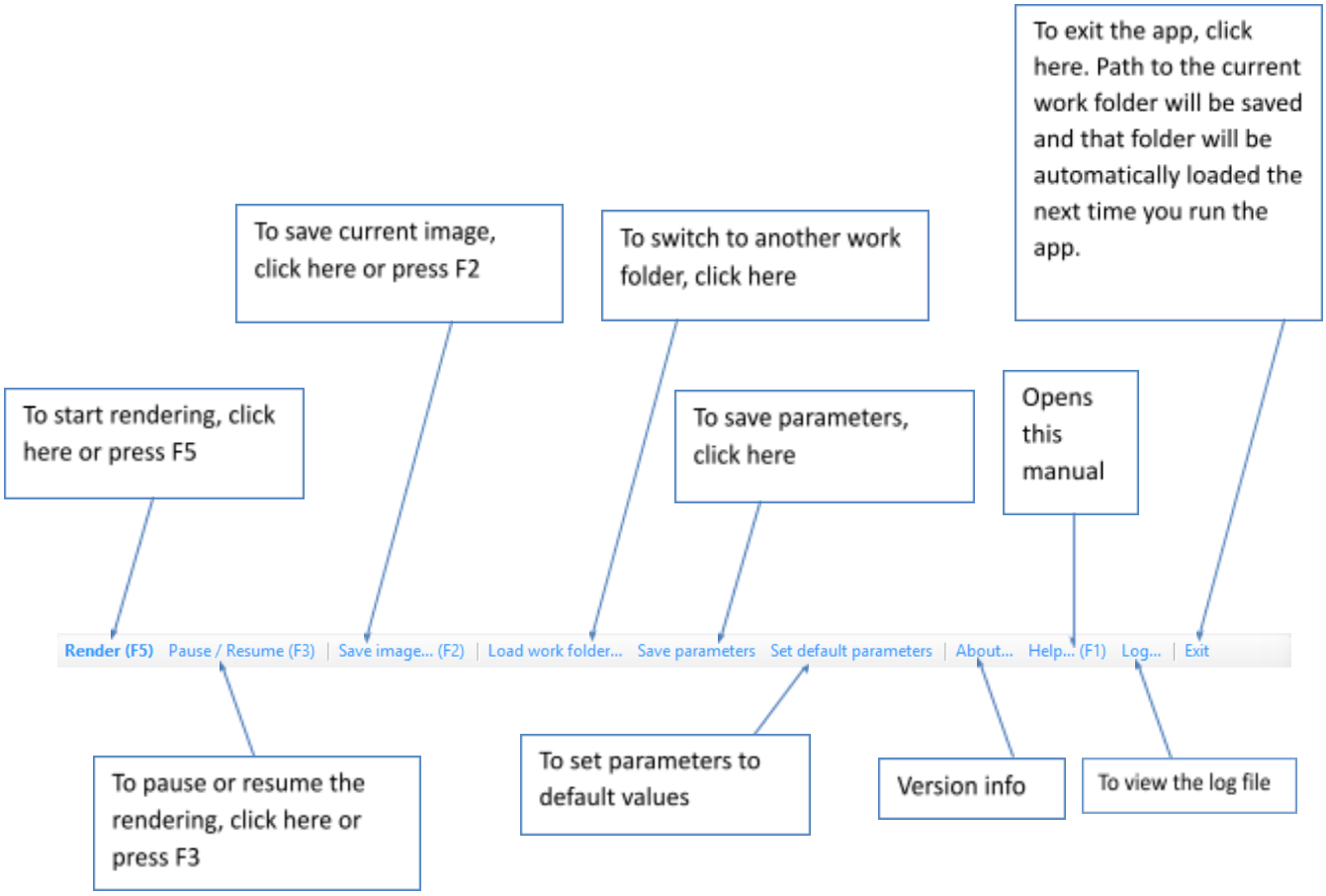
MCP supports dual monitors with up to 8K resolution. For example, the control window can be displayed on the first monitor, while the rendering window appears on the second (see the figure below).

In the control window, you configure all parameters and start/pause/resume the rendering process.

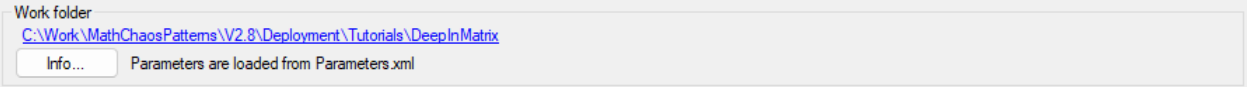
The rendering window displays both intermediate and final output, while the progress bar shows the current color index and/or step being processed, along with other processing details.



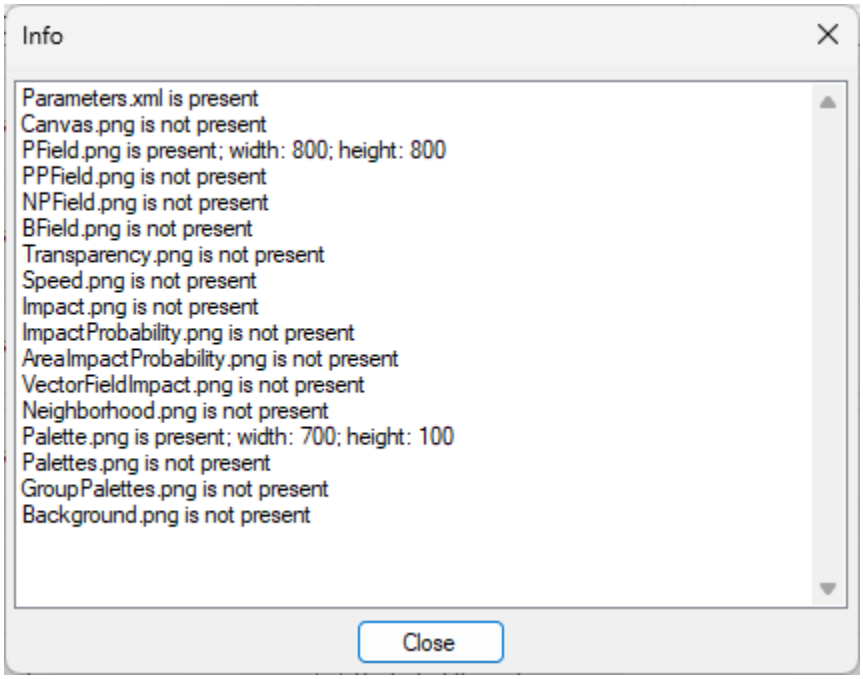
MCP menu


To exit the app, click here. Path to the current work folder will be saved and that folder will be automatically loaded the next time you run the app.
To save current image, click here or press F2
To switch to another work folder, click here
To start rendering, click here or press F5
To save parameters, click here
Opens this manual
Render (F5) Pause / Resume (F3) | Save image... (F2) | Load work folder... Save parameters Set default parameters | About... Help... (F1) Log... | Exit
To pause or resume the rendering, click here or press F3
To set parameters to default values
Version info
To view the log file

Work folder



Having a dedicated work folder simplifies the process of configuring MCP parameters. Files that control the brushes on the canvas use standardized names and are stored in this folder. To view the control files currently present in the work folder, along with their characteristics (where applicable), click [Info...].



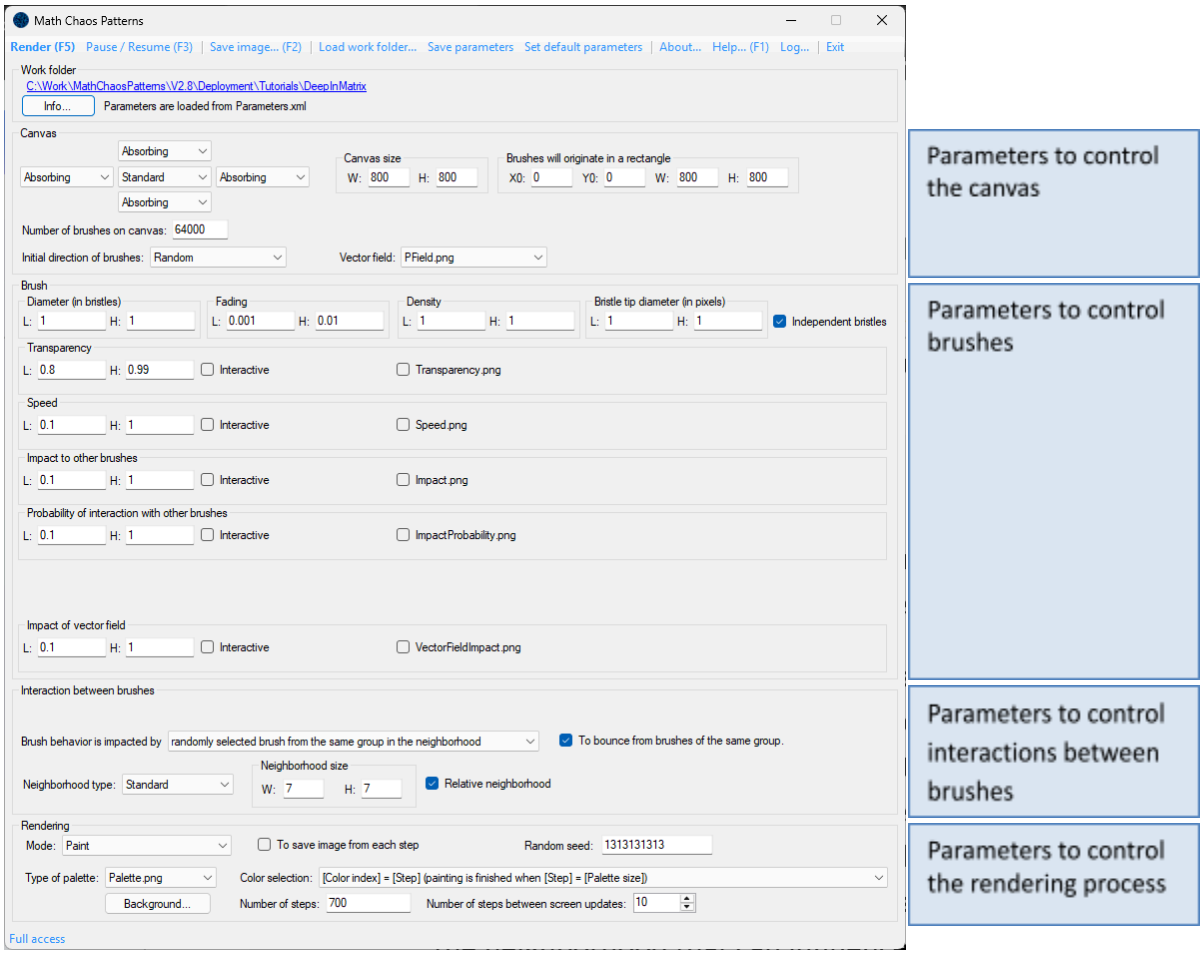
A control file is an image containing color-coded areas. After a control file is loaded onto the canvas, these areas can, depending on the context, trigger instructions that modify the direction, speed, transparency, and other properties of passing brushes. For example, the control file Speed.png can be used to speed up or slow down brushes at locations of the canvas chosen by you. The purpose of each control file is explained in this manual. Control files can be created using any image-editing software, such as Windows Paint, provided they are saved in PNG format.

MCP parameters

There are four sets of parameters:

1. Parameters to control the canvas
2. Parameters to control brushes
3. Parameters to control interactions between brushes
4. Parameters to control the rendering process

All parameters can be changed by editing file Parameters.xml in the current work folder (for example, using Notepad). After editing the file, you need to reload the work folder (by selecting [Load work folder…] from MCP menu). Frequently used parameters can be changed in the control window:

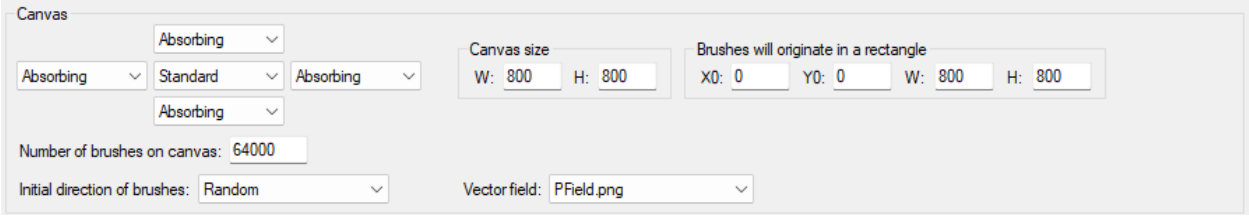


The following parameters can be changed only by editing file Parameters.xml since they are advanced parameters and should be changed rarely:

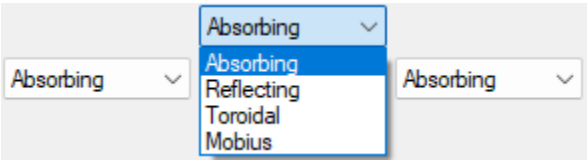
- The parameter **maxNumberOfPointsForVectorFields** controls how detailed the vector field is (details about vector fields are below). A larger value produces a more detailed field but increases the computation time of the vector field. You can change it only by editing the Parameters.xml file (for example, in Notepad). After editing the file, you need to reload the corresponding work folder.
- The parameter **maxNumberOfPointsInNeighborhood** limits the number of brushes from the neighborhood that can influence the current brush. You can change it only by editing the Parameters.xml file (for example, in Notepad). After editing the file, you need to reload the corresponding work folder.

Parameters to control the canvas

The MCP canvas (or simply the canvas) is covered by areas dedicated to different actions, some of which may overlap. Only one area is mandatory: the area where brushes can originate. All other areas are optional.

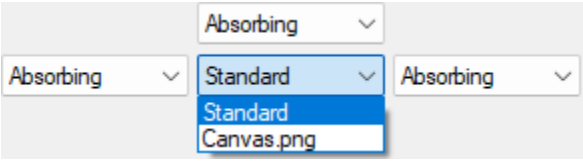


Each boundary of the canvas can be configured to absorb, reflect, or transfer brushes to the opposite side of the canvas by interpreting the canvas surface as either a 2D torus or a Möbius band:

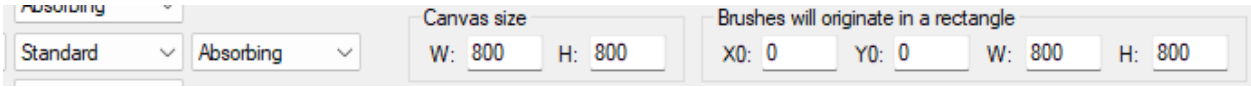


For example, if all boundaries are toroidal, the canvas is represented in MCP as the surface of a 2D torus.

There are two types of canvas: standard and user defined



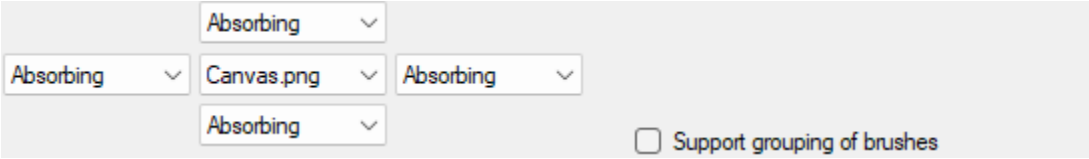
Parameters for standard canvas:



Parameters in group [Canvas size] define the size of the rendering area in pixels.

Parameters [X0], [Y0], [W]idth, and [H]eight define a rectangle (in pixels) within the rendering area where brushes originate (i.e., where they start their walk around the canvas).

Parameters for user defined canvas:



Canvas.png (if available in the current work folder) controls the size of the canvas and loads special areas onto the canvas with different functions. In particular, the area where brushes can originate, the area that defines their initial direction, the area for absorbing brushes, and the area for reflecting brushes are loaded from Canvas.png (see Part Two for examples):

1. Brushes can originate only in red areas (color R = 255, G = B = 0, or any color for which R > G and R > B). Here and throughout this manual, we assume the RGB (Red, Green, Blue) color model.

2. Each brush may begin its walk toward the nearest pixel in a green area (color G = 255, R = B = 0, or any color for which G > R and G > B). See parameter [Initial Direction] below for more details.

3. Brushes are absorbed in black areas (color R = G = B = 0).

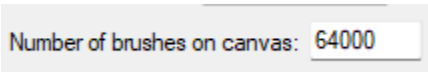
4. Brushes are reflected by blue areas (color B = 255, R = G = 0, or any color for which B > R and B > G).

5. Brushes can be grouped based on the R values of pixels in red areas.

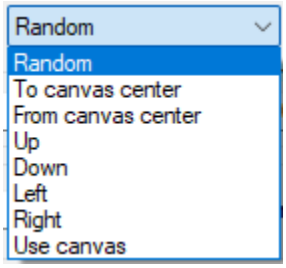


For example, if Canvas.png contains two red areas—area #1 with pixels at R=150 and area #2 with pixels at R=200—MCP will split brushes into two groups: brushes originating in area #1 and brushes originating in area #2. MCP supports brush grouping only if [Support grouping of brushes] is checked. The [Info…] button displays the number of groups identified in Canvas.png, and additional details are reported in the log (see MCP menu). See Part Two for examples demonstrating the use of brush grouping.

[Number of Brushes] sets the initial number of brushes on the canvas.

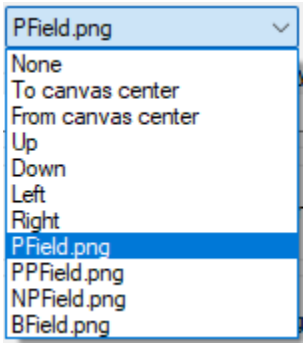


[Initial direction of brushes] sets one of eight possible initial directions for each brush:



If the initial direction is set to “Use canvas” and either there are no green areas in Canvas.png or the standard canvas is used, the initial direction is chosen randomly.

[Vector field] sets one of the eleven types of vector fields:



As mentioned above, the canvas can be viewed as a special type of image. In a conventional image, the basic elements are pixels, and each pixel has only one property: its color. In the MCP canvas, each element has multiple properties and functions that allow it to interact with passing brushes. For example, a canvas element can change the speed of a passing brush or absorb it entirely.

The vector field adds another property to each canvas element, called a preferred direction. Each brush has a parameter that determines the extent to which it is affected by the vector field. For example, if a brush passes through an element whose preferred direction is Up, the element may force the brush to abandon its current direction and move upward. Alternatively, the brush may completely ignore the preferred direction and continue along its current path, or it may adjust its direction only slightly toward Up.

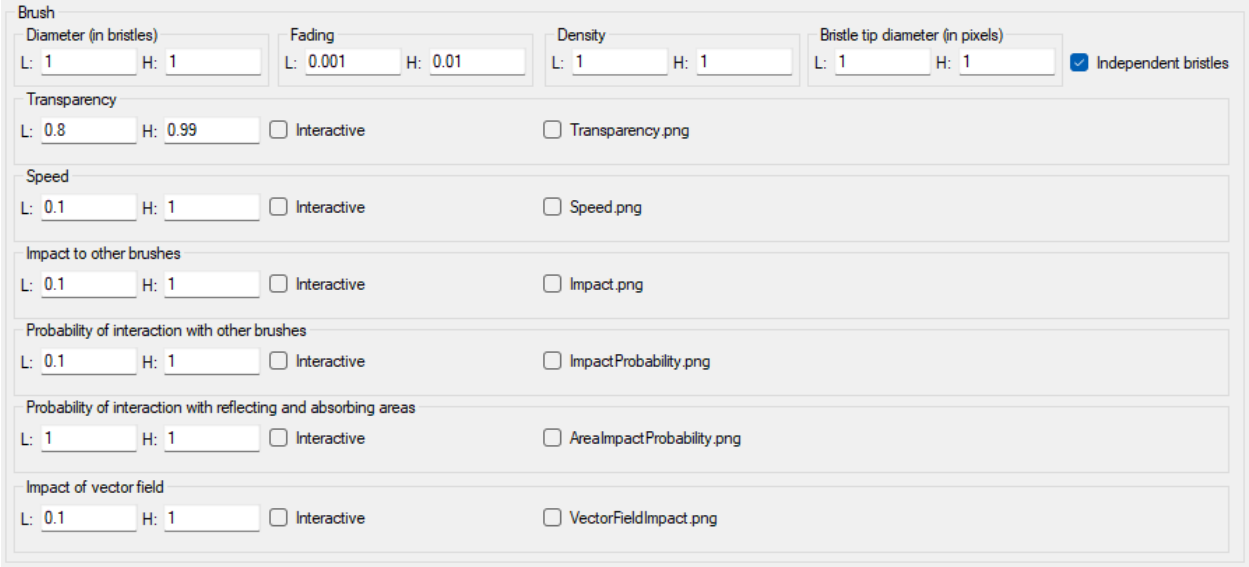
See [Impact of vector field] below for more details about the interaction between brushes and the vector field.

The standard vector fields are self-explanatory and require no further description.

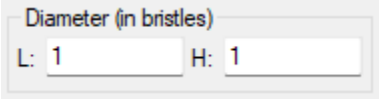
There are two major types of user defined vector fields:

1. A potential vector field is created from an image (PField.png, PPField.png, or NPField.png) located in the current work folder. Red pixels act as sources that push brushes outward, while green pixels act as sources that pull brushes inward. PField.png supports both pushing and pulling, PPField.png supports only pulling (similar to positive gravity), and NPField.png supports only pushing (similar to negative gravity). The strength of a pushing or pulling force at any point is inversely proportional to the squared distance from the corresponding field source to that point. The resulting direction is then calculated as the normalized sum of all push and pull directions, with weights inversely proportional to the squared distances to their corresponding sources (see Part Two for examples). Here, normalization of a vector means dividing the vector by its length, so a normalized vector always has length 1.
2. A basis vector field is created from an image (BField.png) located in the current work folder. Each basis vector is defined by a pair of nearest pixels: the red pixel marks the beginning of the vector, and the green pixel marks its end. Together, these vectors form a collection called the basis, which gives the field its name. For each canvas element, the resulting direction is calculated as the normalized sum of the normalized basis vectors, with weights inversely proportional to the squared distances to those vectors (see Part Two for examples).

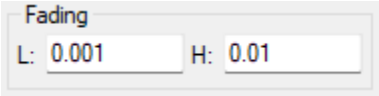
Parameters to control brushes



[Diameter] controls the range of brush diameters, measured in bristles, where each bristle corresponds to a single pixel. Brushes are circular.



[Fading] controls the range of brush fading and has a significant effect on the interaction between brushes (see below).

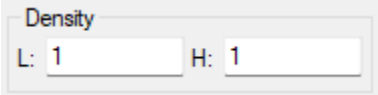


As mentioned above, each element of the MCP canvas has multiple properties and functions that allow it to interact with passing brushes. For example, a canvas element can change the speed or direction of a passing brush, or absorb the brush entirely. Another important property of each canvas element is memory. Every element can remember the last brush that passed through it. Similar to human memory, this memory gradually fades over time, but its behavior can be controlled in MCP. There are three important aspects of this property:

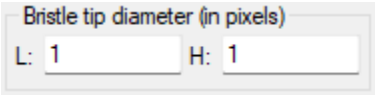
1. Every element remembers the parameters of the last brush that passed through it.
2. The number of steps for which an element remembers a brush is controlled by that brush's fading value: the lower the fading value, the longer the brush will be remembered.

3. With each step (see below for information about the number of steps taken by brushes), the memory gradually fades. Each passing brush overwrites the previously stored memory.

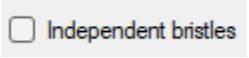
[Density] controls the range of brush densities. A brush can be fully dense (1, the maximum number of bristles) or empty (0, no bristles).



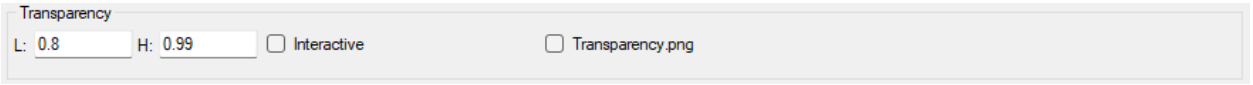
[Bristle tip diameter] controls the range of bristle tip sizes among brushes. The base of each bristle has a size of one pixel; however, its tip may be larger. Bristle tips are circular, so their size is defined by their diameter. Because of the brush-to-bristle hierarchy, thick brushstrokes can be achieved in different ways. If you want a uniformly thick brushstroke, you can set the brush diameter to 1 and then set the bristle tip diameter to the desired larger value. However, if you want a non-uniform brushstroke, you can set the brush diameter to a larger value and draw the bristle tip diameter from a range of values. Additional variation can be introduced by specifying a range for the brush density.



When [Independent bristles] is checked, the bristles of a brush act as independent brushes and can interact with one another. When [Independent bristles] is not checked, the bristles remain dependent and cannot interact with other bristles of the same brush. In both cases, they can interact with other bristles and their bristles.



[Transparency] controls how output colors are blended during rendering. For greater color blending at lower brush speeds, use higher transparency values.



[L]ow sets the minimum possible initial value of the parameter.

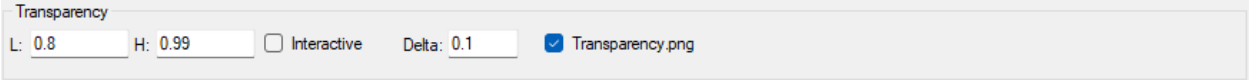
[H]igh sets the maximum possible initial value of the parameter.

If [Interactive] is checked, brushes may exchange parameter values during their interactions. The amount of exchange depends on the brush impact value.

Parameters can also be controlled during brush movement using user-defined images that contain two types of areas:

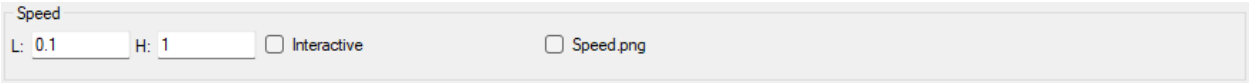
1. Green pixels (color G = 255, R = B = 0, or any color for which G > R and G > B) increase the parameter value of a passing brush.
2. Red pixels (color R = 255, G = B = 0, or any color for which R > G and R > B) decrease the parameter value of a passing brush.

The magnitude of the increase or decrease is determined by the [Delta] parameter. In the screenshot below, brush transparency is controlled by decreasing or increasing the transparency value by 0.1 (see [Delta]) when brushes pass through red or green areas, respectively. These areas are loaded onto the MCP canvas from Transparency.png.

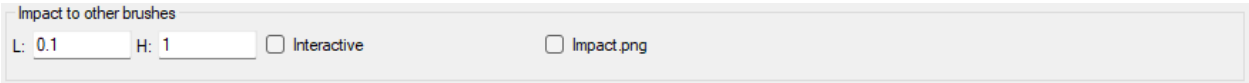


The values of [Transparency], [Speed], [Impact], [Probability of interaction with other brushes], [Probability of interaction with reflecting and absorbing areas], and [Impact of vector field] can be controlled in a similar manner. See Part Two for examples.

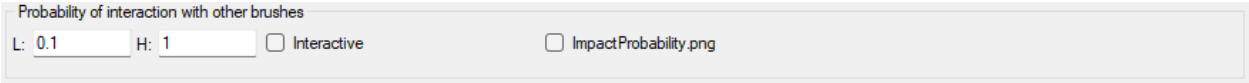
[Speed] controls how fast brushes move across the canvas, it is measured as pixels by step.



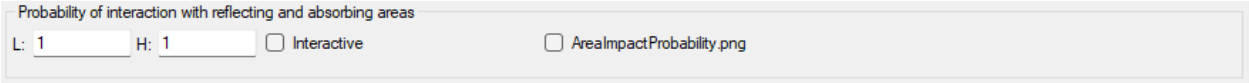
[Impact to other brushes] controls the degree to which one brush influences another in terms of direction, speed, transparency, and other properties.



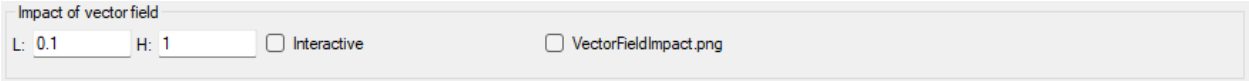
[Probability of interaction with other brushes] controls how frequently each brush interacts with other brushes. Less value causes less interactions.



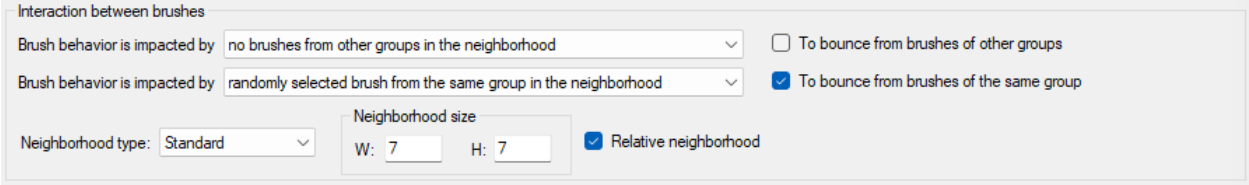
[Probability of interaction with reflecting and absorbing areas] controls how often brushes interact with reflecting and absorbing areas (if such areas exist on the canvas). Less value causes less interactions.



[Impact of vector field] controls the influence of the vector field on passing brushes. As noted earlier, the vector field affects only brush directions. Less value causes less influence.

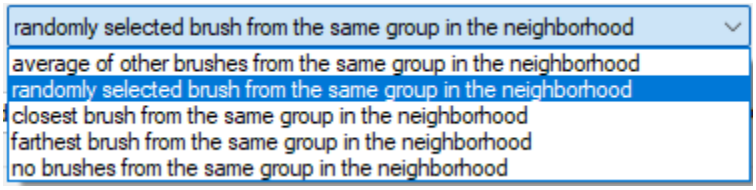


Parameters to control interactions between brushes



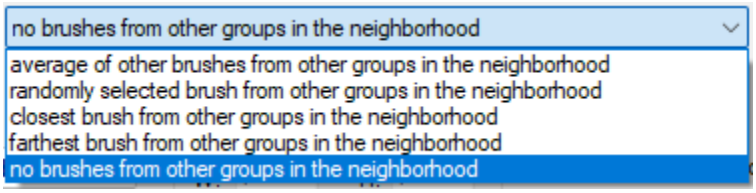
Each brush may interact with other brushes within its neighborhood on the canvas. This is directly related to the fading parameter. The lower the fading value, the longer brushes remain stored in the memory of canvas elements. Consequently, more brushes may be remembered within a neighborhood and can therefore interact with the current brush.

Here you can configure both the interaction parameters and the neighborhood definition. Four basic types of interactions are available, depending on which brush or brushes are selected from the neighborhood of the current brush:

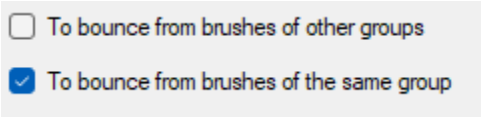


For example, a randomly selected brush from the neighborhood of the current brush may affect its direction, speed, transparency, impact, and probability of interaction, depending on the parameters of both brushes (see Brush Parameters above).

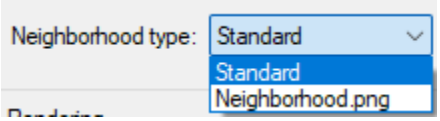
Brushes belonging to different groups may interact differently than brushes belonging to the same group:



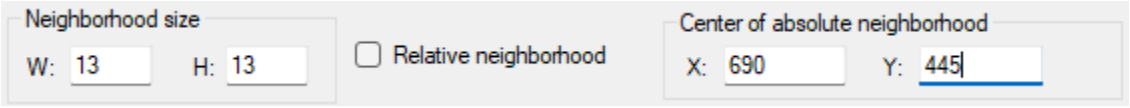
If [To bounce from brushes…] is checked, the direction of the current brush is influenced by the opposite direction(s) of the selected brush or brushes in its neighborhood. This option doubles the number of basic interaction types, increasing them from four to eight.



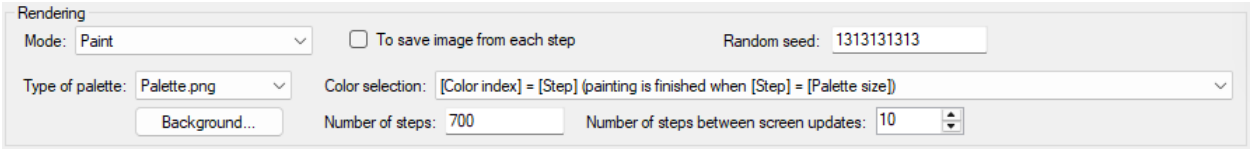
The neighborhood can be either standard or loaded from an image (Neighborhood.png in the current work folder). The size of the image defines the neighborhood dimensions, where both width and height must be odd numbers. Brushes passing through black areas (R=G=B=0) in the neighborhood are ignored.



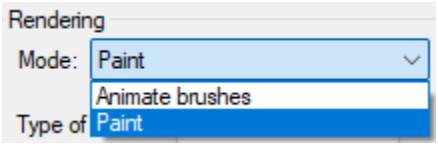
The neighborhood can also be absolute. In this mode, only brushes from a fixed area of the canvas can affect the current brush, with [X] and [Y] specifying the center of that area. In the following example, any brush on the canvas can be influenced only by brushes passing through a 13 × 13 square centered at pixel coordinates (690, 445):



Parameters to control the rendering process

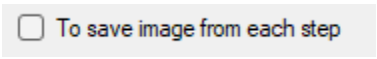


Rendering [Mode]



[Animate Brushes] — This mode uses only the grayscale gradient palette and a black background. It allows you to observe the trajectories of brushes as they move across the canvas. This mode is particularly useful for planning artwork and studying vector fields. It allows you to see how brushes move and interact with one another and with different areas of the canvas before performing the final render. When working with vector fields, or whenever it is unclear why brushes are behaving in a certain way, this rendering mode can be especially helpful for understanding and debugging brush behavior.

[Paint] — Brushes paint the canvas using the current color selected from the color palette (see [Color selection] below). The color palette itself is determined by the [Type of palette] parameter (see below).



[To save image from each step] — If checked, rendered images are saved in the Video folder within your work folder. These images can later be merged into a video using any video editor.

⚠️ Important: Ensure the Video folder does not contain files you wish to keep, as all files in that folder will be deleted before rendering begins.

After the images are generated you can use **MCP Video Glue** app to “glue” them into a video.

The [Random seed] parameter controls reproducibility. When all other parameters are fixed, the output depends only on the current seed value. To reproduce the same output, set the random seed to any fixed positive value. To obtain a different output, choose a new value.

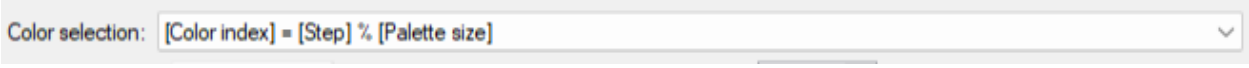


To generate new outputs automatically each time you render, set the seed to 0, then MCP will select the random seed for you. In case you like the output and want to reproduce it, you can find the last random seed in the log (click “Log…” in the main menu and go to the bottom of the log file).

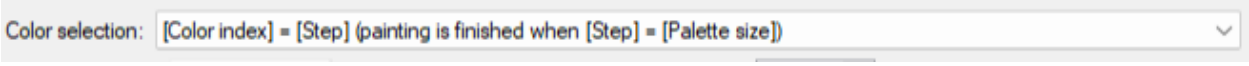
Color selection

Each brush is assigned a color palette, which is an ordered sequence of colors that you can customize by changing the [Type of palette] parameter (see next). To determine the current color used by the brushes, MCP uses the current [Step], where each step corresponds to one move of all brushes. There are three ways to compute the color index within the palette:

With the first type of color selection, each step uses the next color in the palette. After the last color is used, MCP returns to the first color and repeats the cycle. In this mode, the total number of steps is determined solely by [Number of steps].



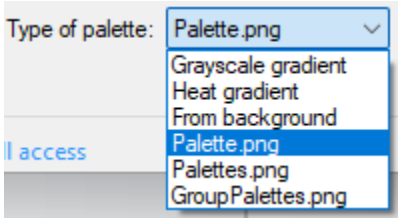
With the second type of color selection, each step uses the next color in the palette. After the last color has been used, MCP stops rendering. Therefore, the total number of steps is limited to the smaller of [Number of Steps] and [Palette Size].



With the third type of color selection, MCP does not return to the first color after reaching the end of the palette. Instead, it selects the color preceding the last one, then the color before that, and so on until it reaches the beginning of the palette. It then reverses direction and proceeds through the palette again. This back-and-forth process continues until the specified number of steps is reached. In this mode, the total number of steps is determined solely by [Number of steps].



[Type of palette] sets the palette used by brushes. There are six palette types:



Grayscale gradient. Here, R, G, and B range from 0 to 255, producing a palette of 256 grayscale colors ranging from black (R = G = B = 0) to white (R = G = B = 255).

Heat gradient. A popular color palette used in scientific visualization, consisting of 1280 colors.

From background. The color of a brush is determined by the background color at its point of origin and remains fixed throughout its movement. This feature allows MCP to be used as an image filter, with the source image supplied in Background.png located in the current work folder. See the experiments that use vector fields in the work folders StarryNight and VanGogh.

Palette.png. The color palette is loaded from Palette.png in the current work folder. MCP then assigns the middle row of Palette.png as the color palette for each brush.

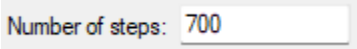
Palettes.png. The color palettes are loaded from Palettes.png in the current work folder. Each brush is assigned a randomly selected row from Palettes.png as its color palette.

GroupPalettes.png. The color palettes are loaded from GroupPalettes.png in the current work folder. This file allows you to assign a dedicated color palette to each brush group. Each palette is stored in a separate set of rows of GroupPalettes.png. The first pixel of the set must have exactly the same shade of red as the shade used in Canvas.png to define the origin area of the corresponding brush group. MCP uses this color to match brush groups with their assigned palettes.

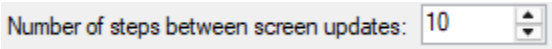


If the file Background.png is present in the current work folder then it is used as background and also can be used as source of color for brushes, see above [Type of palette]; otherwise, you can select a solid color by clicking the button [Background…].

[Number of steps] sets an upper bound on the number of steps that brushes can take. It is also used in color selection (see above).



A larger [number of steps between screen updates] allows to get to the final image faster since the rendering window will be refreshed less frequently.



Part Two

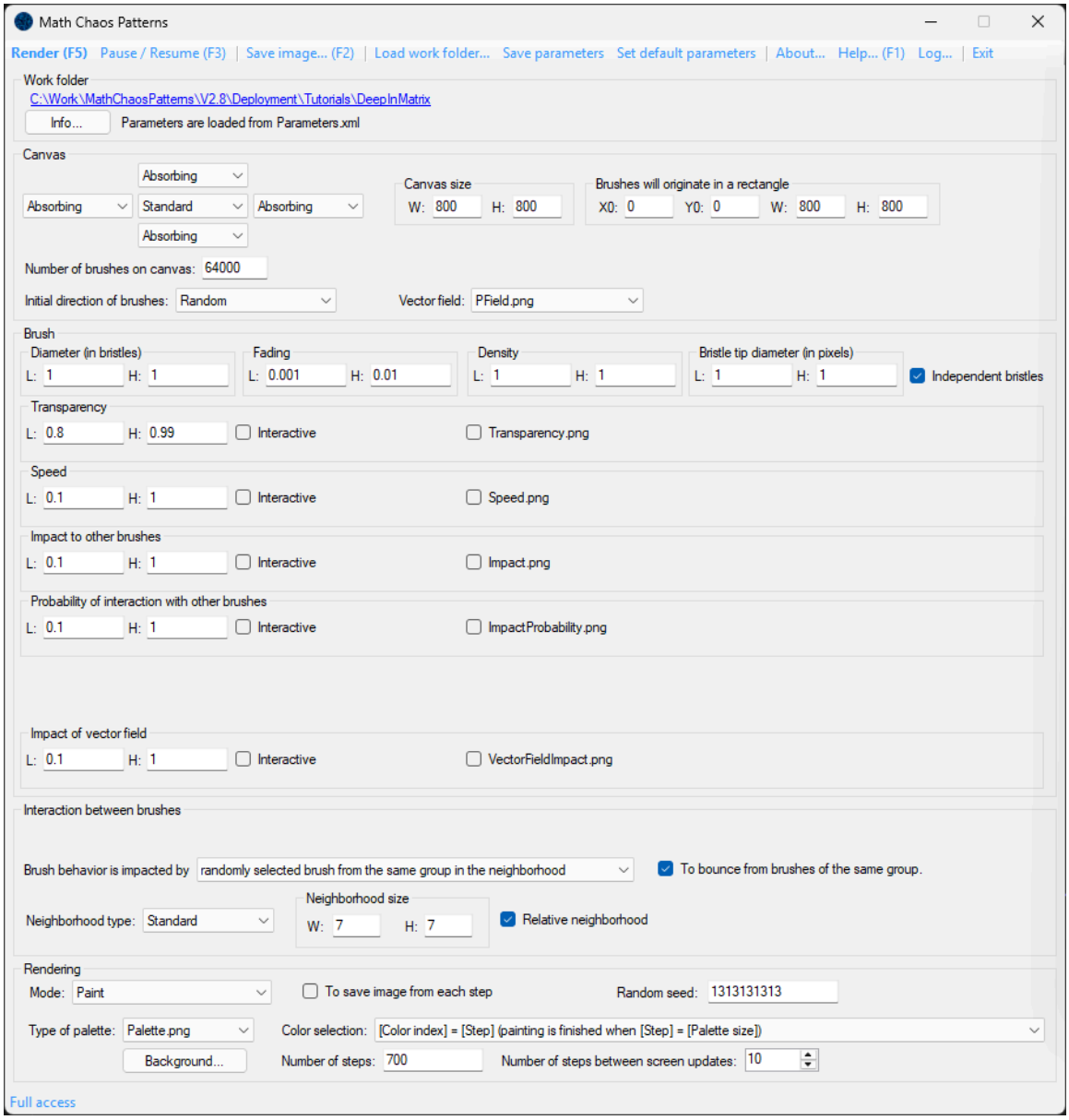
Tutorials

This section illustrates how the parameters detailed in Part One interact to produce complex digital paintings. The tutorials described here, along with many additional examples, are located in the folder **Documents\D1B LLC\MCP\Tutorials**. It is a good idea to make a backup of that folder so you can restore the original parameters if, after multiple experiments, you want to return to the initial settings.

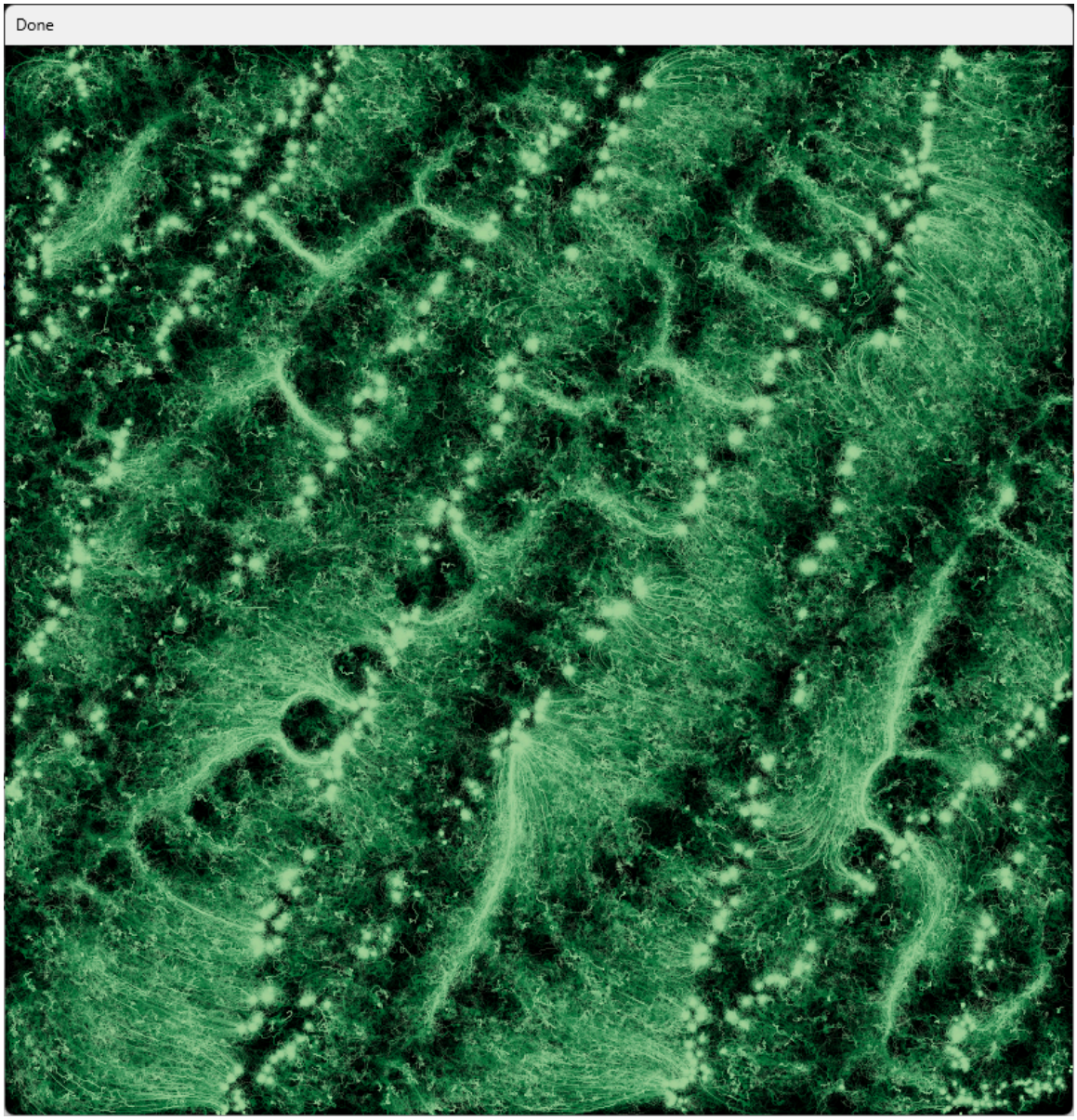
The tutorials below can be executed by MCP if you are subscribed to the full-access plan, which allows rendering images at resolutions up to 8K. If you are using the basic version of MCP, you can access a selection of tutorials located in the **LowResolution** subfolder.

Deep In Matrix

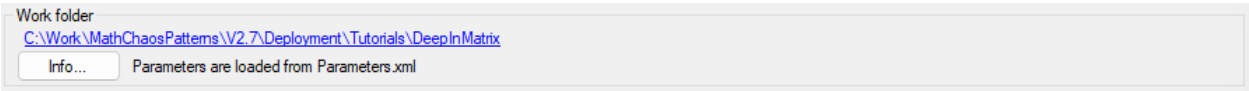
This tutorial demonstrates how to use a potential vector field and to set interactions between brushes and the vector field. In the MCP menu select [Load work folder…], then select folder DeepInMatrix from folder Tutorials. Then the control window appears:



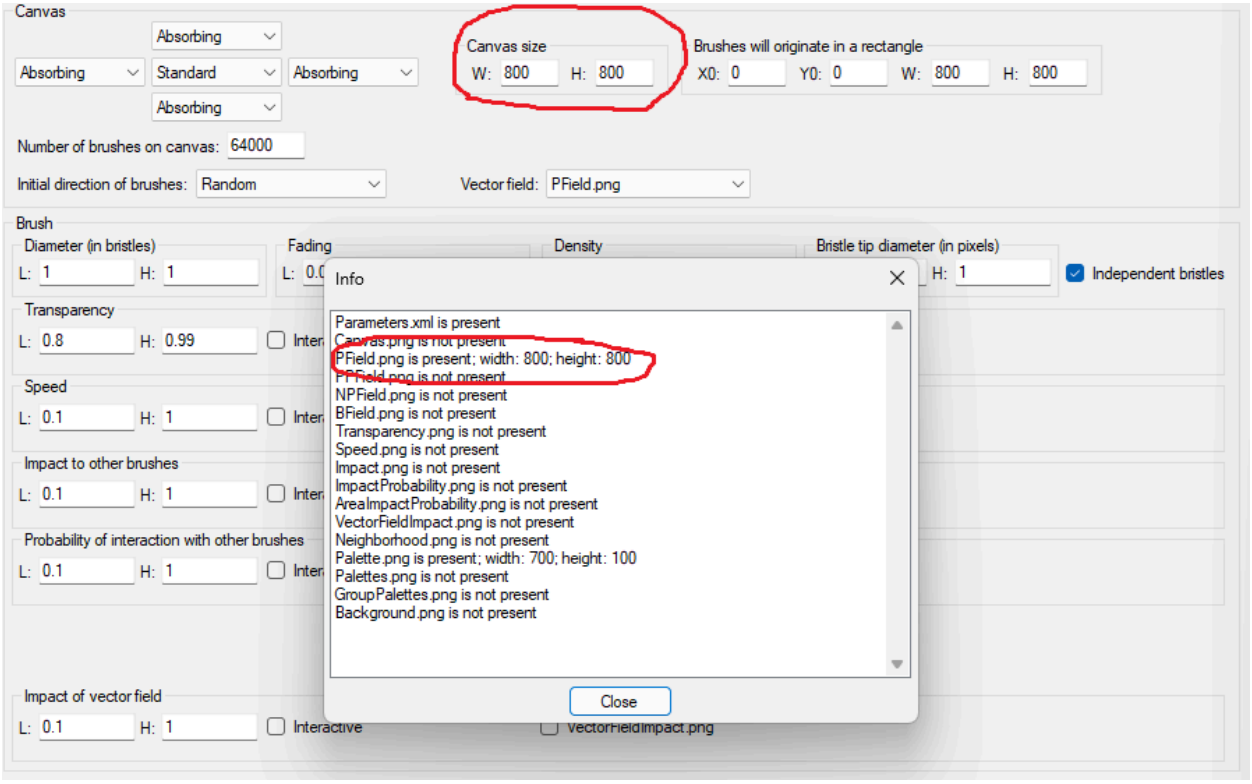
And the rendering window after 700 steps for 64000 brushes should look like this:



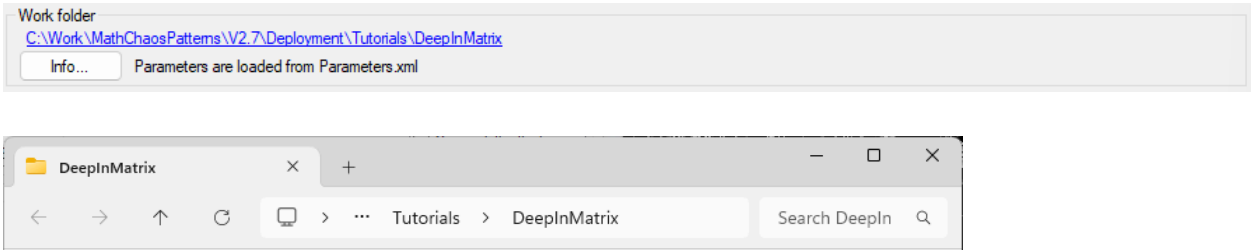
In the control window note two parameters that refer to files: PField.png and Palette.png. They are both located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*:

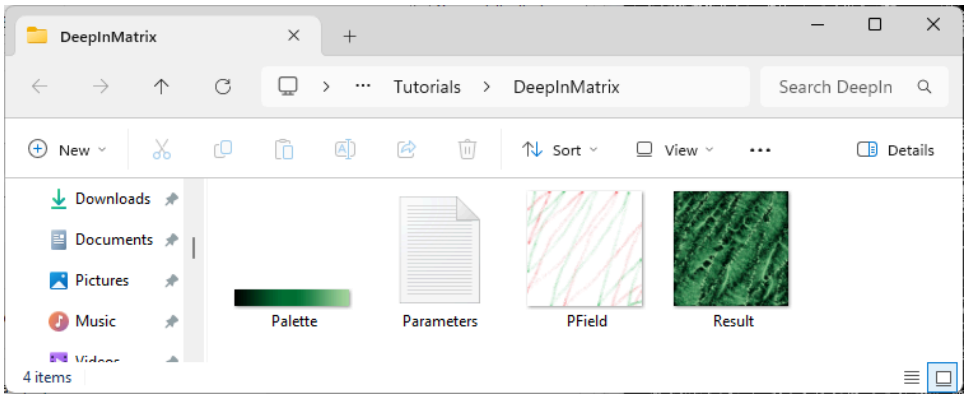


Note that the size of canvas is exactly the same size as the file PField.png (800 x 800):



If you want to examine those files closer then click on the link in the group *Work folder* to open that work folder in Windows File Explorer:

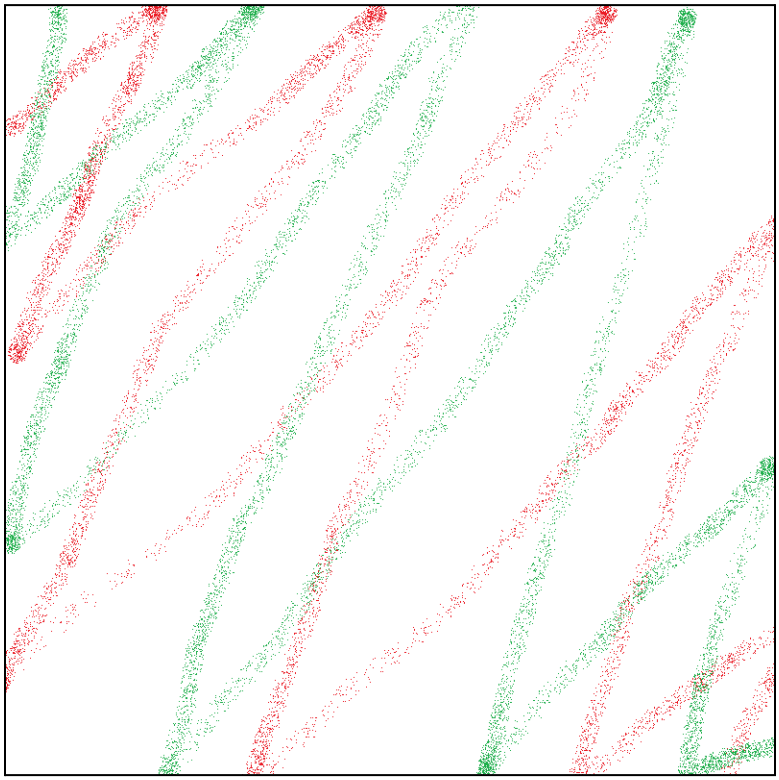




File Palette.png stores the palette used by the brushes. As you could see, during the rendering, colors of the palette are used sequentially from the dark green (on the left) to the bright green (on the right):



File PField.png stores information about the potential field:



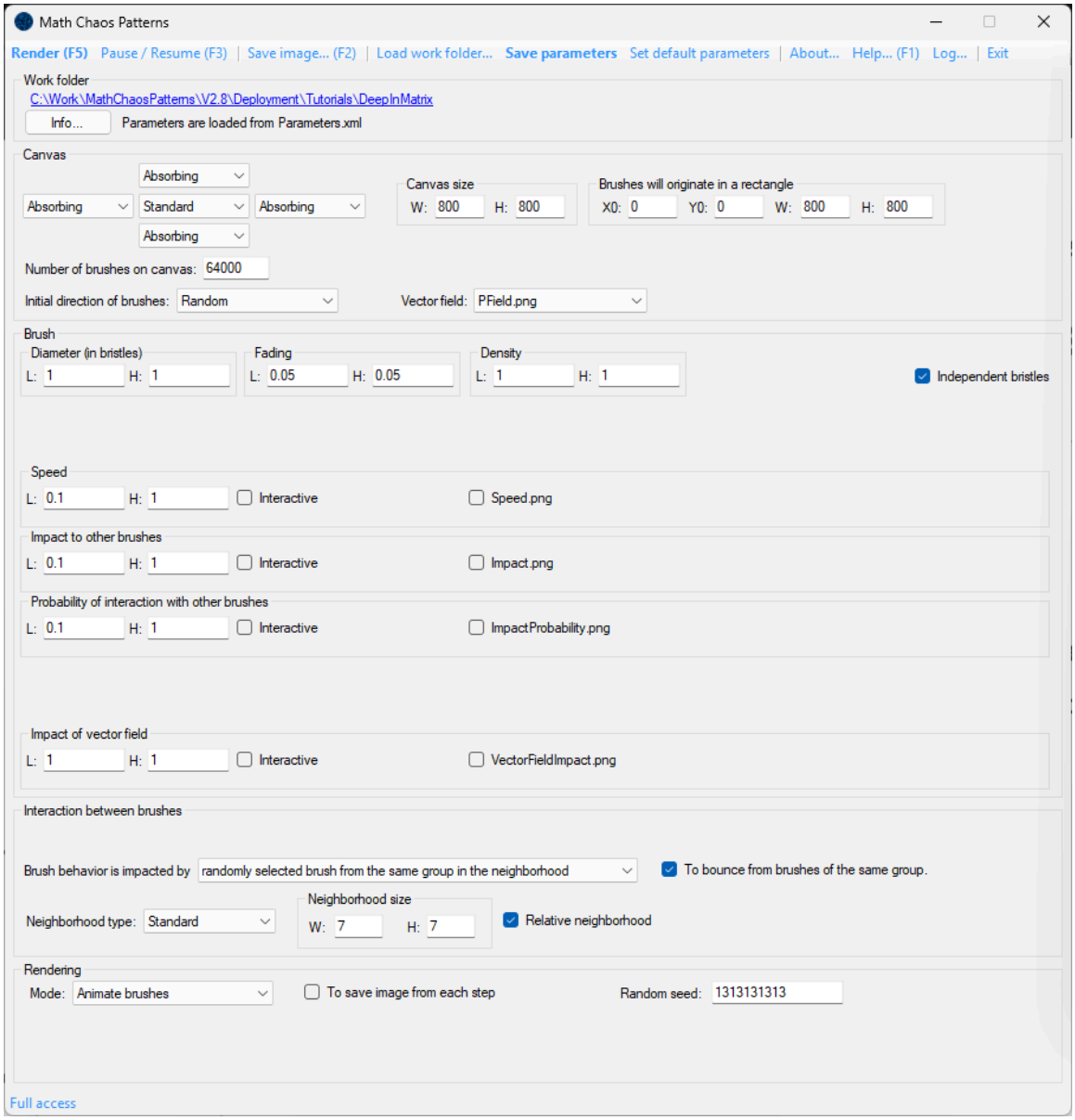
Here red pixels push brushes outward and green pixels pull brushes inward. You can edit it in Windows Paint to see possible variations or come up with your own potential field (as you can see it is simple). Not all of those pixels are actually used, because MCP samples them such that the resulting vector field is still correctly represented but the computational time is not too long, see parameter maxNumberOfPointsForVectorFields.

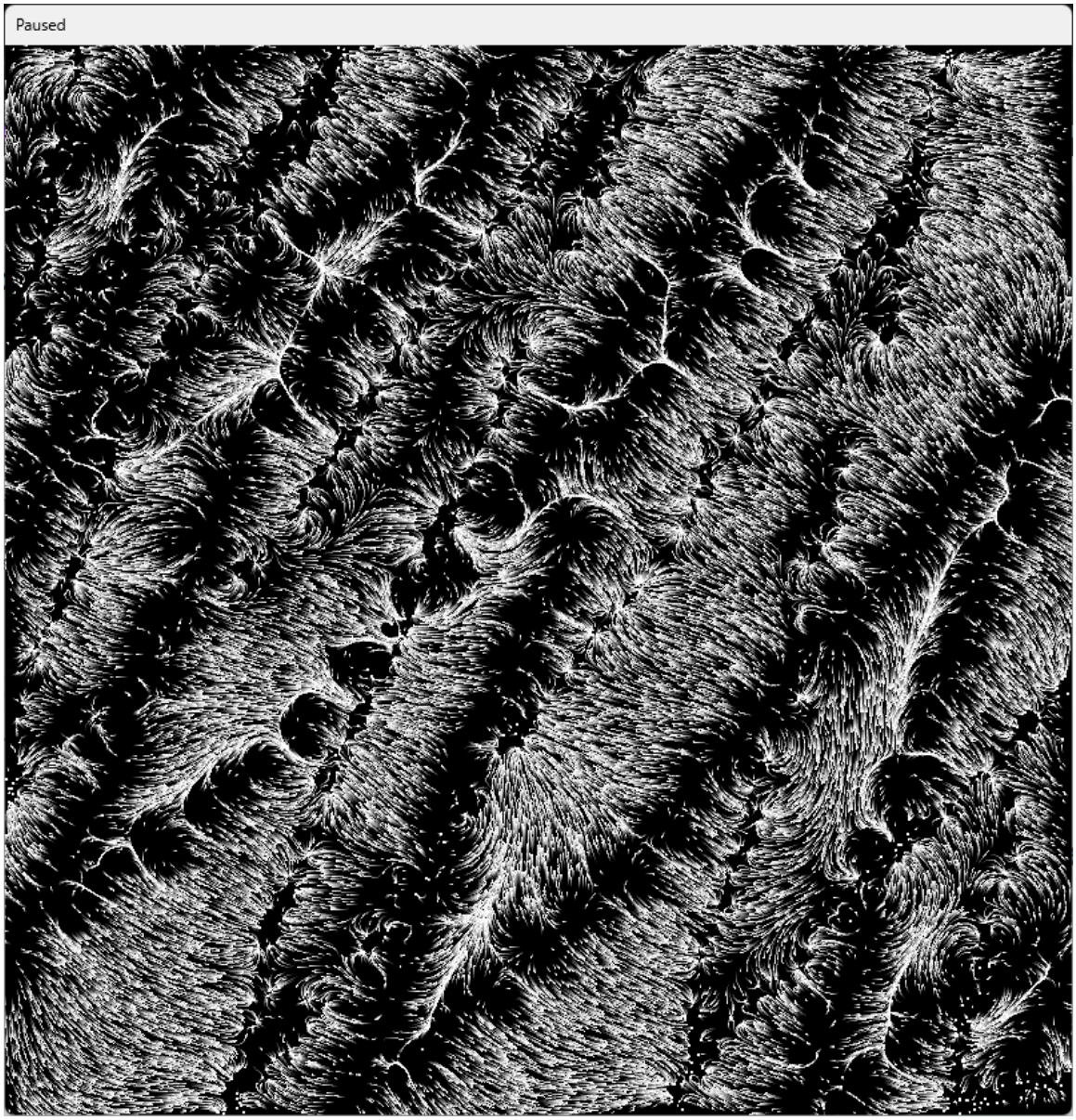
Note in the control window that many parameters have ranges. For example, speed has range [0.1, 1] which means that when a brush is created (there are 64000 brushes, see the control window) it is assigned a random speed from [0.1, 1] (pixels per step). Such randomization allows creating complex digital paintings.

Note that [Impact of vector field] also has range [0.1, 1] which means that some brushes are influenced by the vector field more than others.

If you want to see how brushes are influenced by the vector field, switch rendering [Mode] to *Animate brushes*, set [Impact of vector field] to [1, 1], change L and H of [Brush fading] to 0.05,

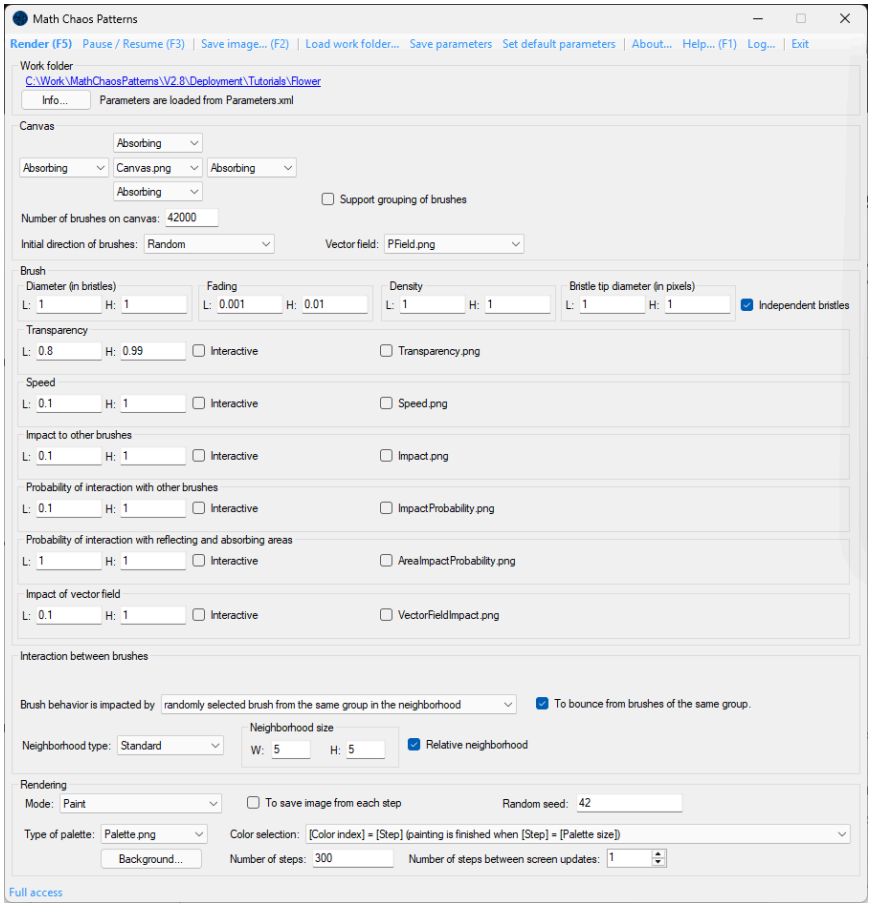
and hit F5 (or click Render in the MCP menu). Then you will see how the vector field influences brushes. Here is the snap of control window and rendering window:



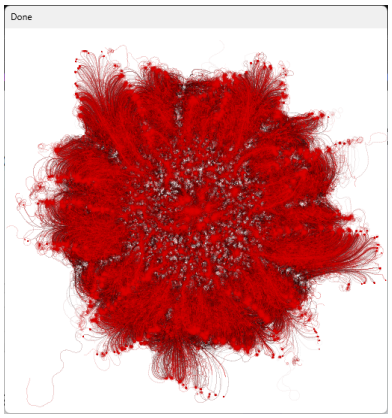

Paused

Flower

This tutorial demonstrates how to use user defined canvas and a potential vector field. In the MCP menu select [Load work folder…], then select folder Flower from folder Tutorials. Then the control window appears:

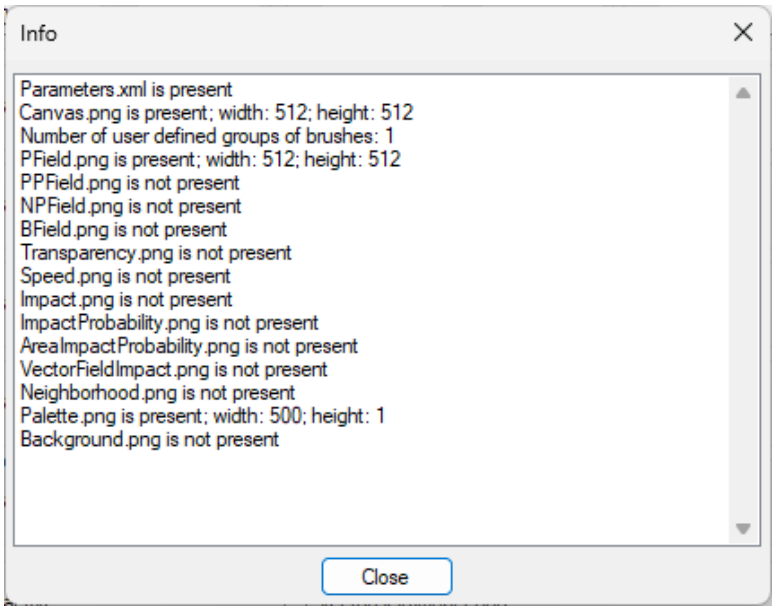


And the rendering window after 300 steps for 42000 brushes should look like this:



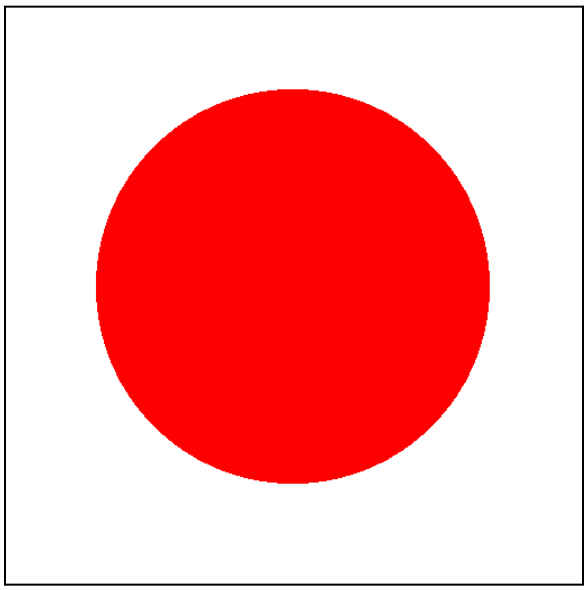
In the control window note three parameters referring to files: Canvas.png, PField.png, and Palette.png. They are located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*.

Note that the size of canvas is exactly the same size as the file PField.png:



If you want to examine those files closer then click on the link in the group *Work folder* to open that work folder in Windows File Explorer.

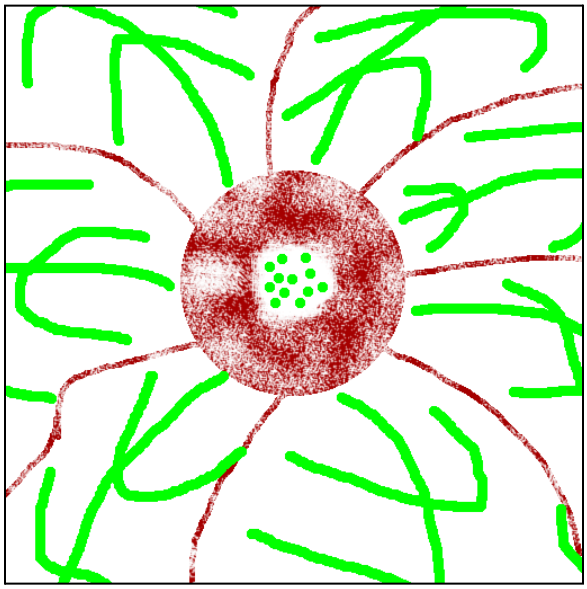
File Canvas.png sets an initial location for all brushes, which is indicated by the red circle:



File Palette.png stores the palette used by the brushes. Since parameter [Number of steps] is 300 then only the first 300 colors (out of 500) are used from the palette:

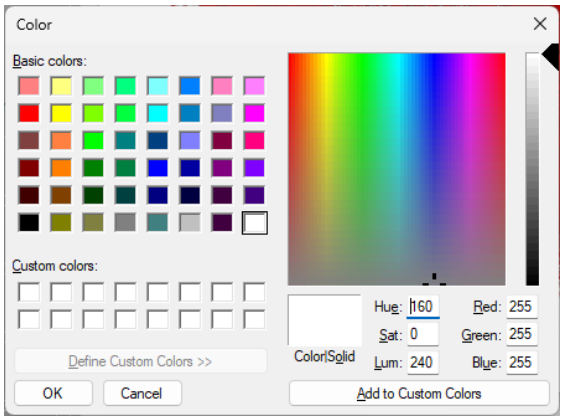


File PField.png stores information about the potential field:

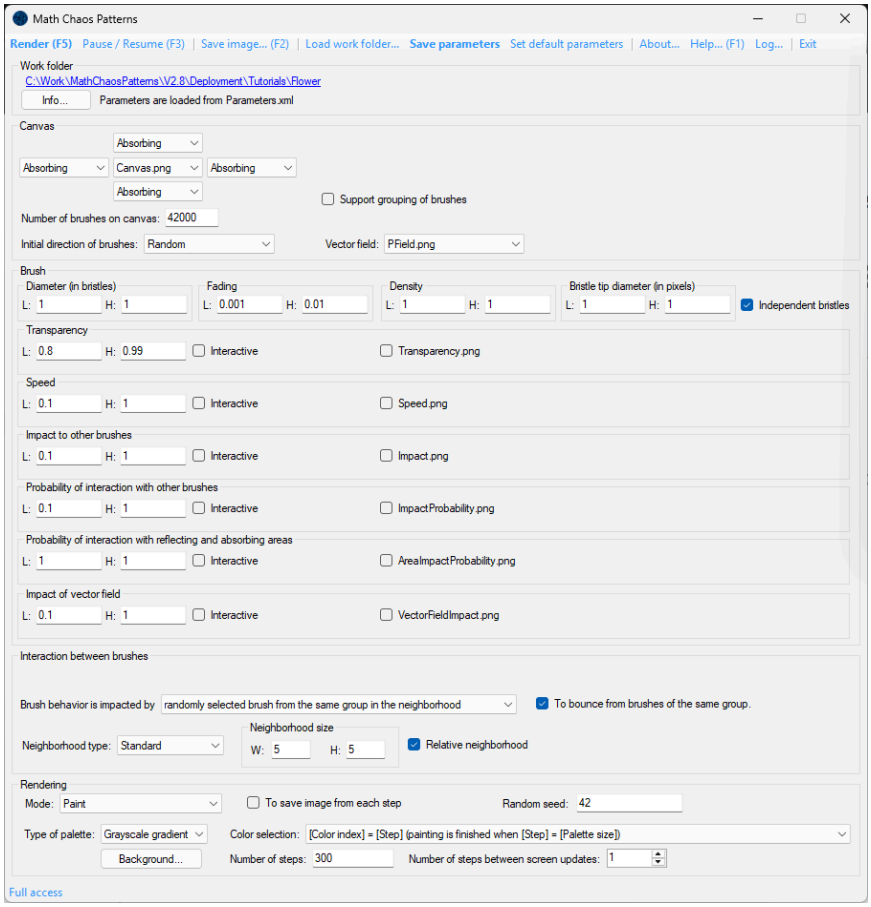


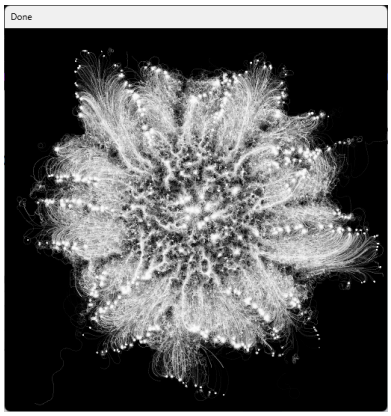
Here red pixels push brushes outward and green pixels pull brushes inward. You can edit it in Windows Paint to see possible variations or come up with your own potential field (as you can see it is simple).

If you do not like white background you can change it by clicking button [Background…] and selecting color that you would like:



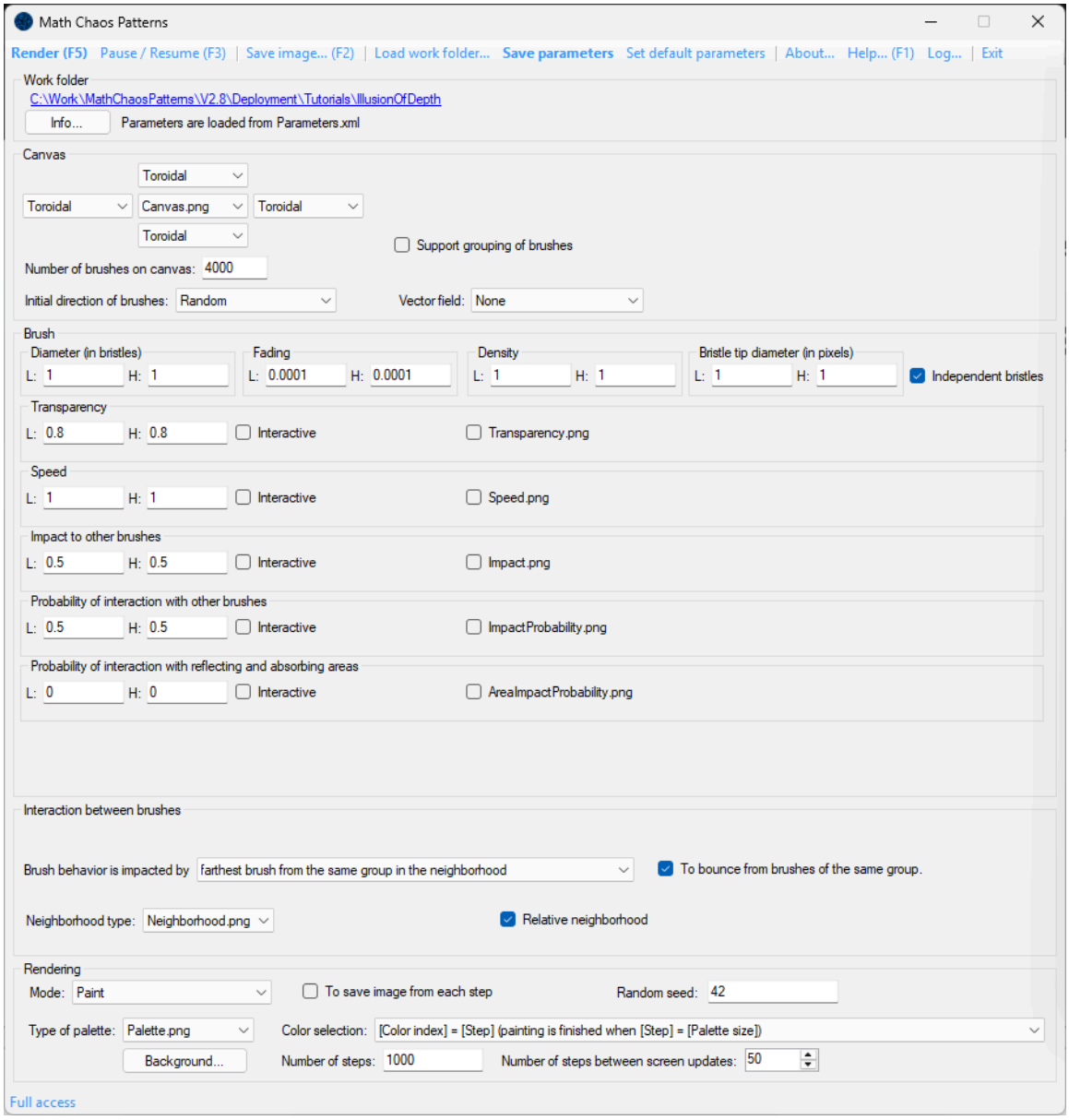
Let us select black background and the grayscale palette, here is controls window followed by the result:



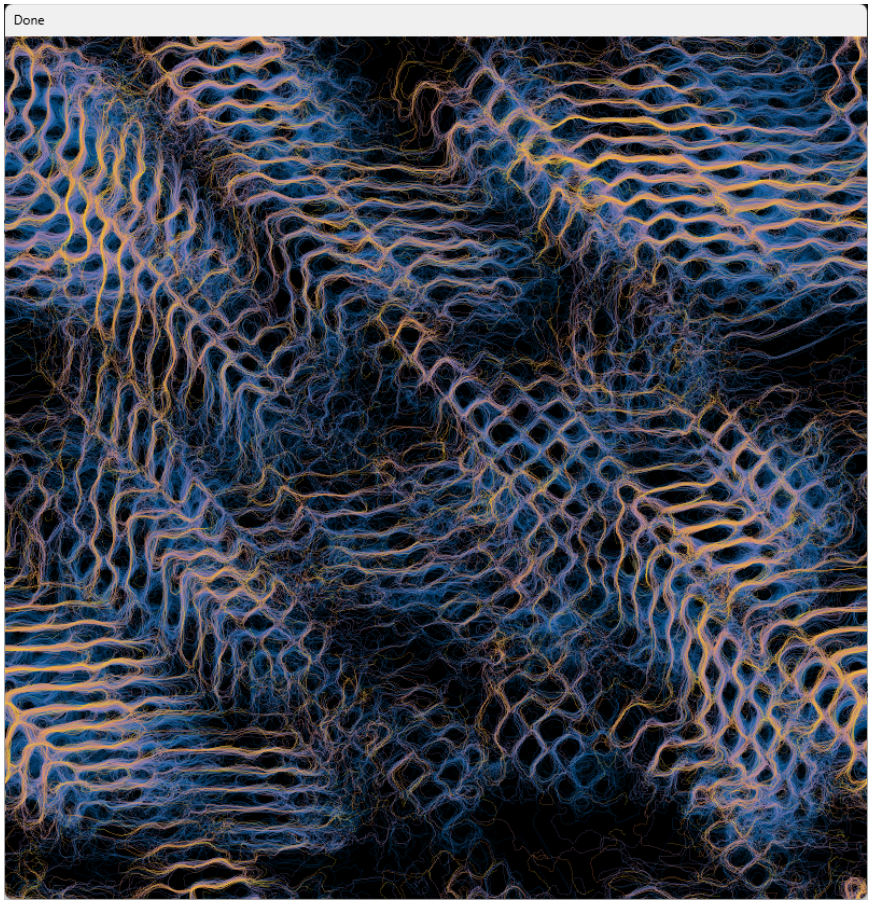


Illusion Of Depth

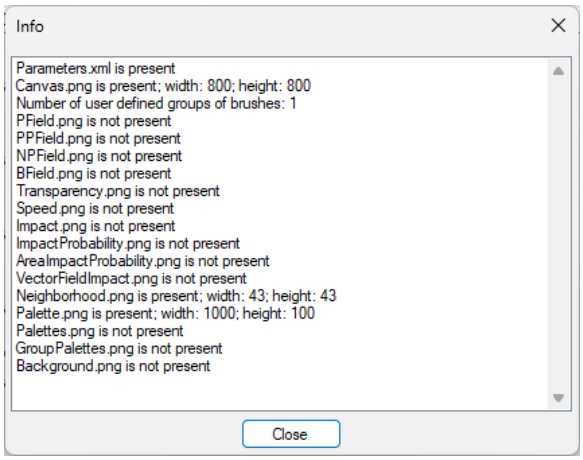
This tutorial demonstrates how to use a user defined canvas and a user defined neighborhood for interacting brushes. In the MCP menu select [Load work folder…], then select folder IllusionOfDepth from folder Tutorials. Then the control window appears:



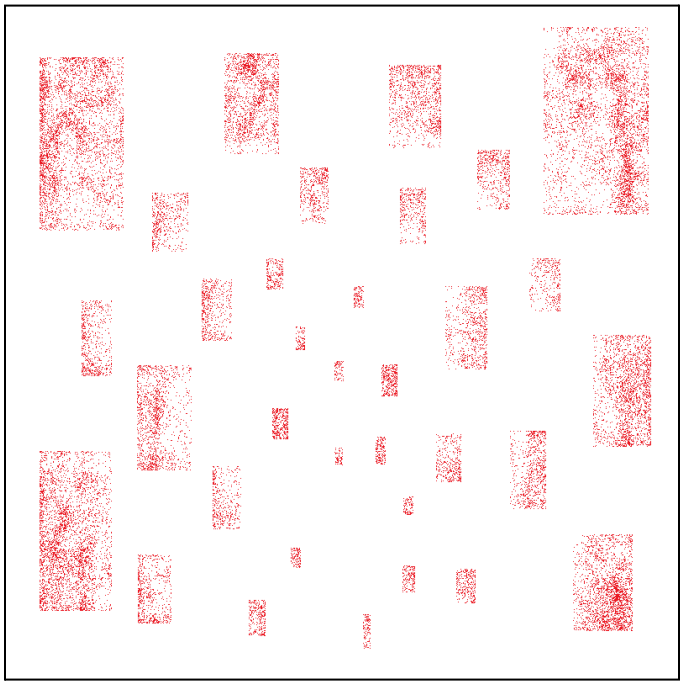
And the rendering window after 1000 steps for 4000 brushes should look like this:



In the control window note three parameters referring to files: Canvas.png, Neighbourhood.png and Palette.png. They are located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*:



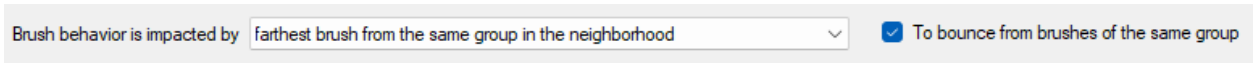
File Canvas.png sets an initial location for all brushes:



File Neighbourhood.png stores the neighbourhood of the brush. The size of the image defines the neighborhood dimensions (in this tutorial it is 43x43). Brushes located at the black areas (R=G=B=0) in the neighborhood are ignored. In this tutorial brushes that are not inside the red circle are ignored:



Because brush behavior is impacted by the farthest brush in the neighbourhood and brushes bounce from each other (see control window above) then you can observe patterns similar to fish net. If you like this effect (you can apply it as an image filter) use the following parameters:

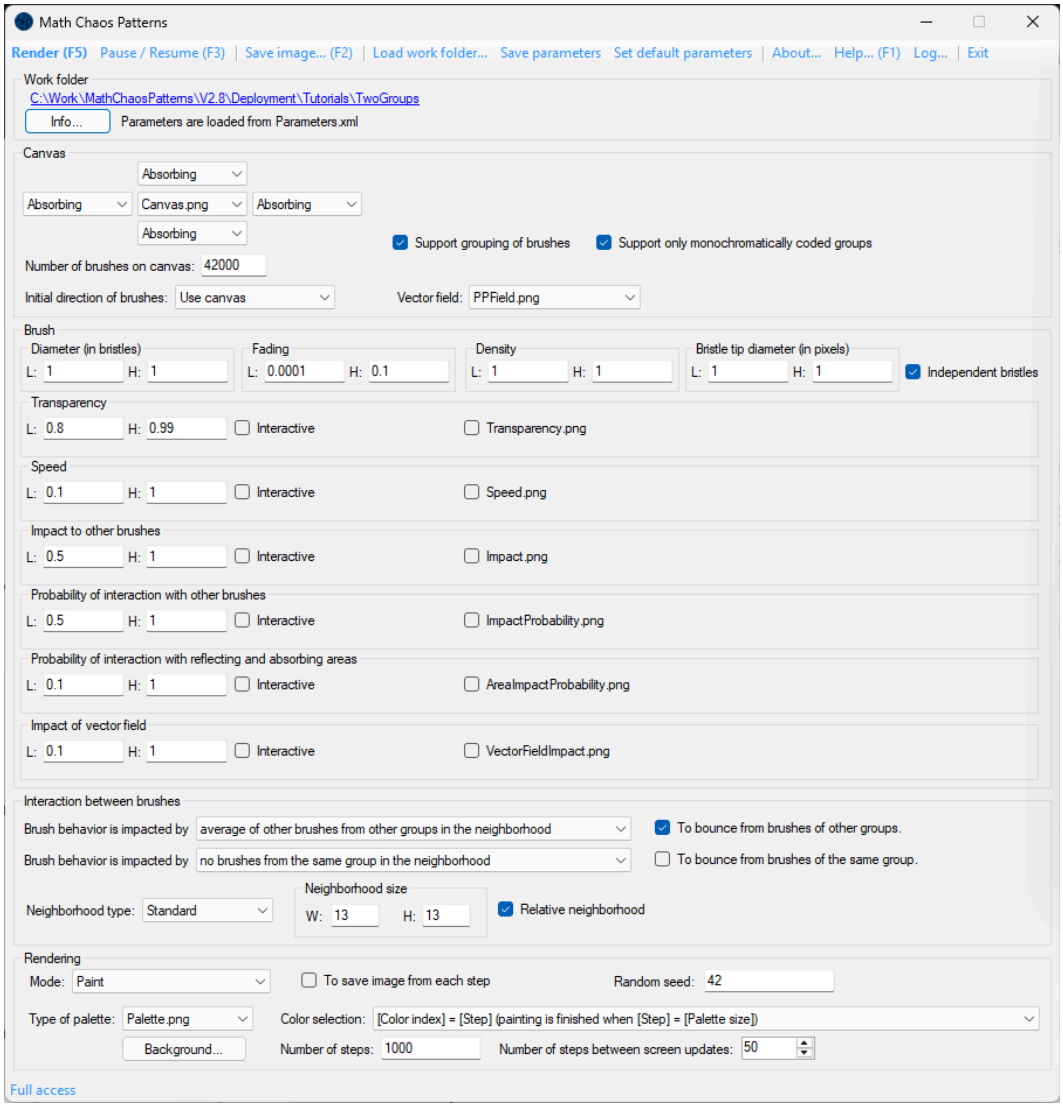


File Palette.png stores the palette used by the brushes:

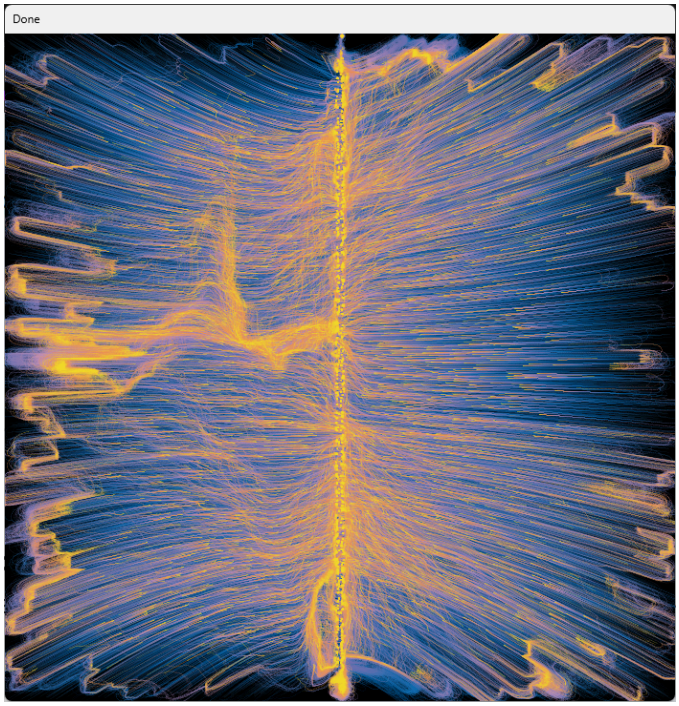


Two Groups

This tutorial demonstrates how to work with user defined canvas, groups of brushes, and the vector field. In the MCP menu select [Load work folder…], then select folder TwoGroups from folder Tutorials. Then the control window appears:

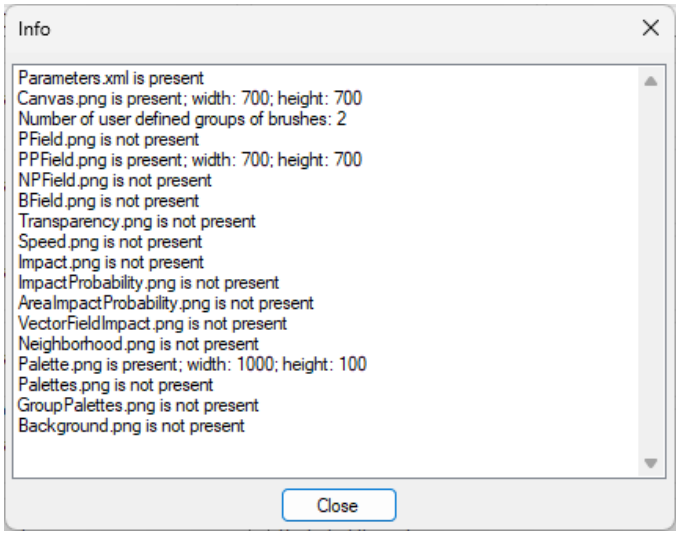


And the rendering window after 1000 steps for 42000 brushes should look like this:



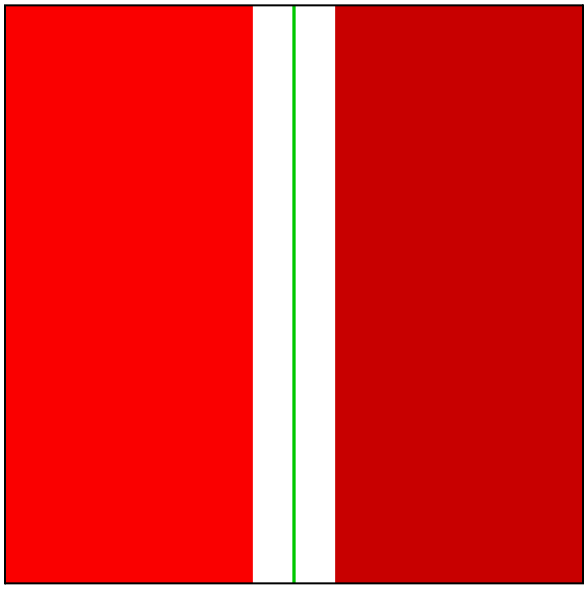
In the control window note three parameters referring to files: Canvas.png, PPField.png, and Palette.png. They are located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*.

Note that the size of canvas is exactly the same size as the file PPField.png and there are two groups of brushes:

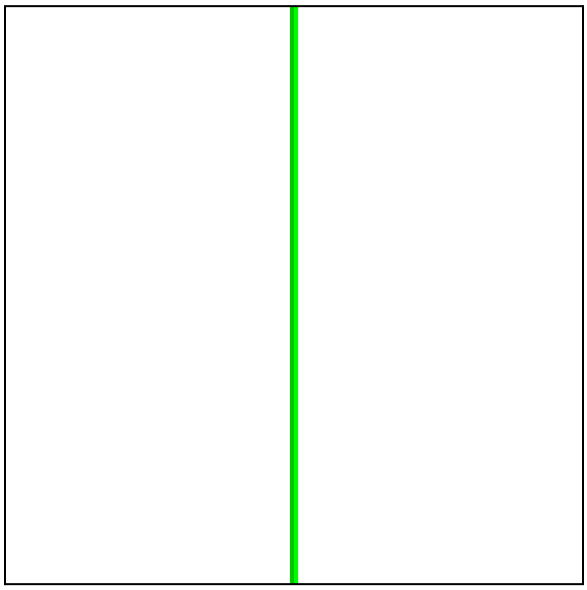


If you want to examine those files closer then click on the link in the group *Work folder* to open that work folder in Windows File Explorer.

File Canvas.png sets the initial location for all brushes, initial direction for brushes, and two groups of brushes. In the control window the parameter [Initial direction] is set to *Use canvas*. In that case each brush starts its walk from red areas towards the closest pixel from green areas. Directed by Canvas.png, brushes will move from two red blocks to the center green line:

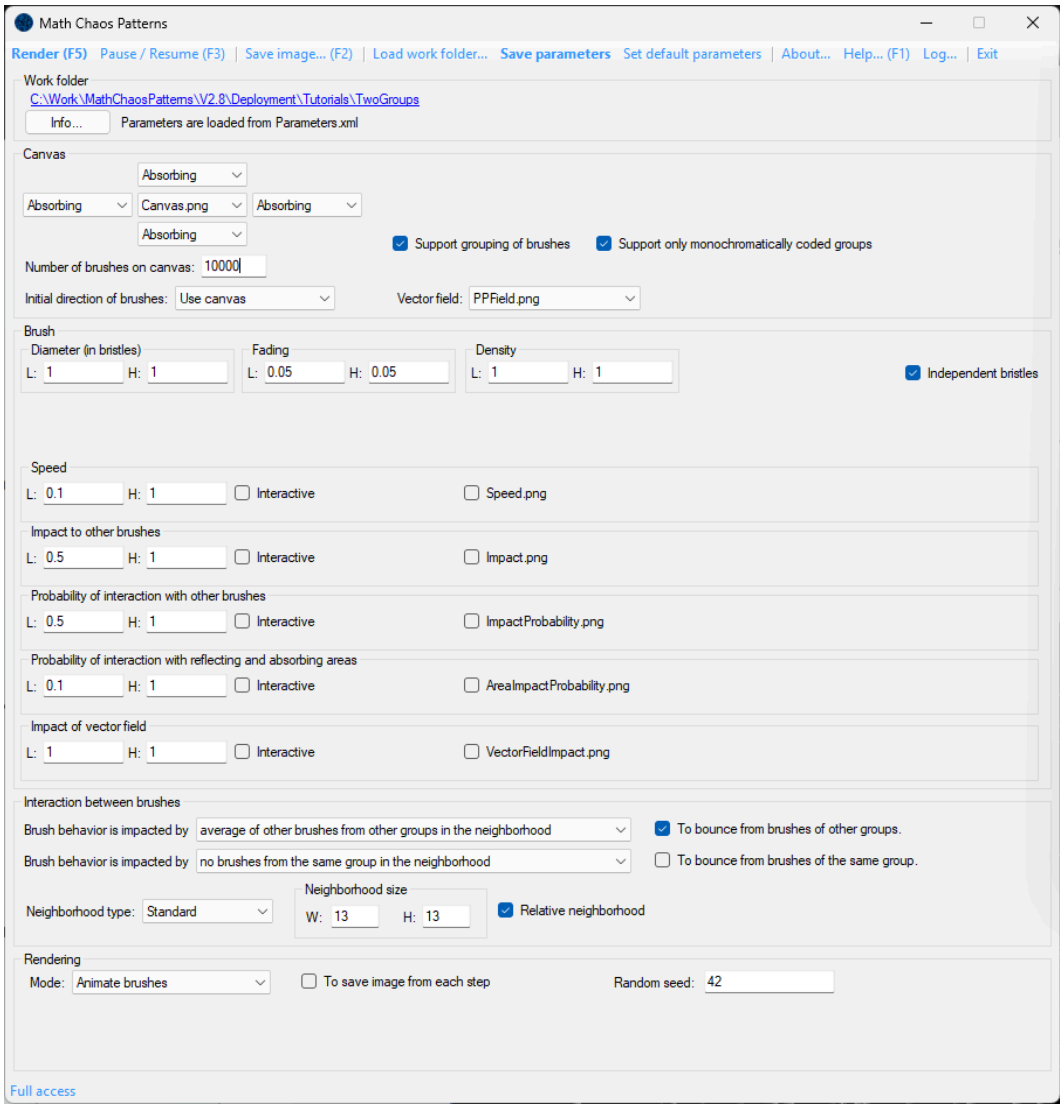


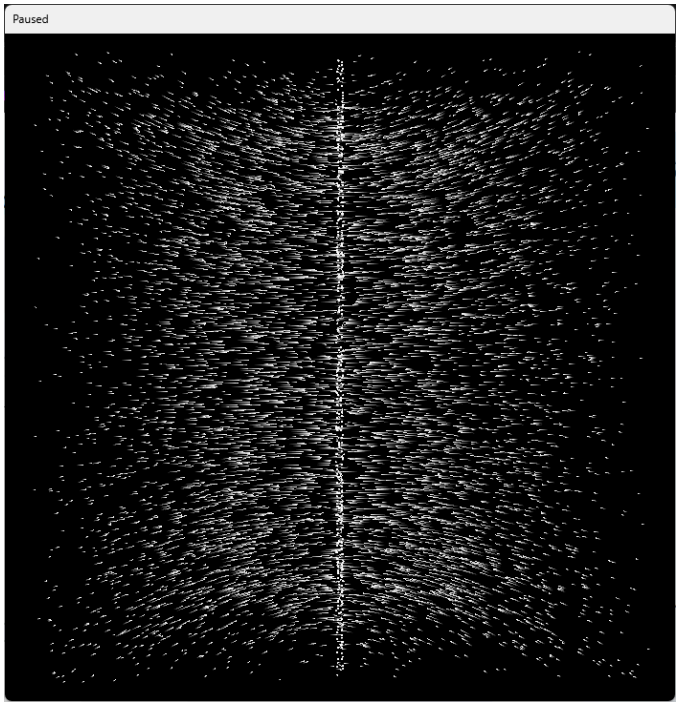
As mentioned before, brushes can form groups based on the R values in red areas. In this tutorial, the canvas contains two red areas— left red block with pixels at R=250 and right red block with pixels at R=200—MCP will split brushes into two groups: brushes originating in the left block and brushes originating in the right block. Now let's look at the file PPField.png controlling the vector field:



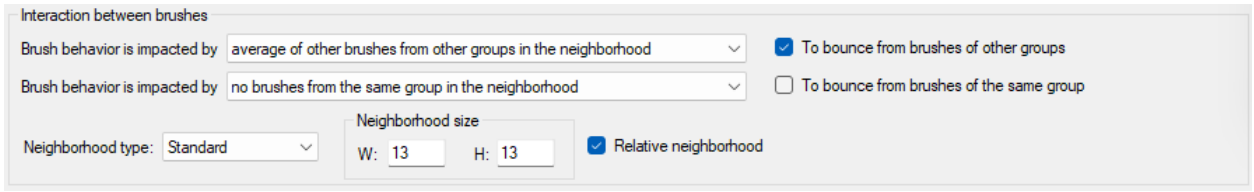
It has sources that pull brushes inward (green pixels). Those sources are also split in two groups: one for the left block on the canvas and another for the right block on the canvas based on their amount of green G. Left green line has pixels with G=200 (these affect the second group of brushes) and the right green line has pixels with G=250 (these affect the first group of brushes). Note that the amount of R in Canvas.png for the group is equal to the amount of G in PPField.png for the corresponding sources of potential field, this allows MCP to compute different vector fields for different groups. MCP checks this correspondence and if there is a mismatch then it is reported in the log, click [Log…] in the MCP menu to check it out.

If you want to see how brushes are affected by the vector field, switch Rendering [Mode] to *Animate brushes*, set [Impact of vector field] to [1, 1], and change L and H of [Brush fading] to 0.05. Reduce the number of brushes to 10000 and Hit F5. Then you will see how the vector field affects brushes. Here is the snap of control window and rendering window:





Also check out parameters for interaction between brushes in the control window. As you can see only brushes from different group can interact, meanwhile brushes from the same groups cannot interact:

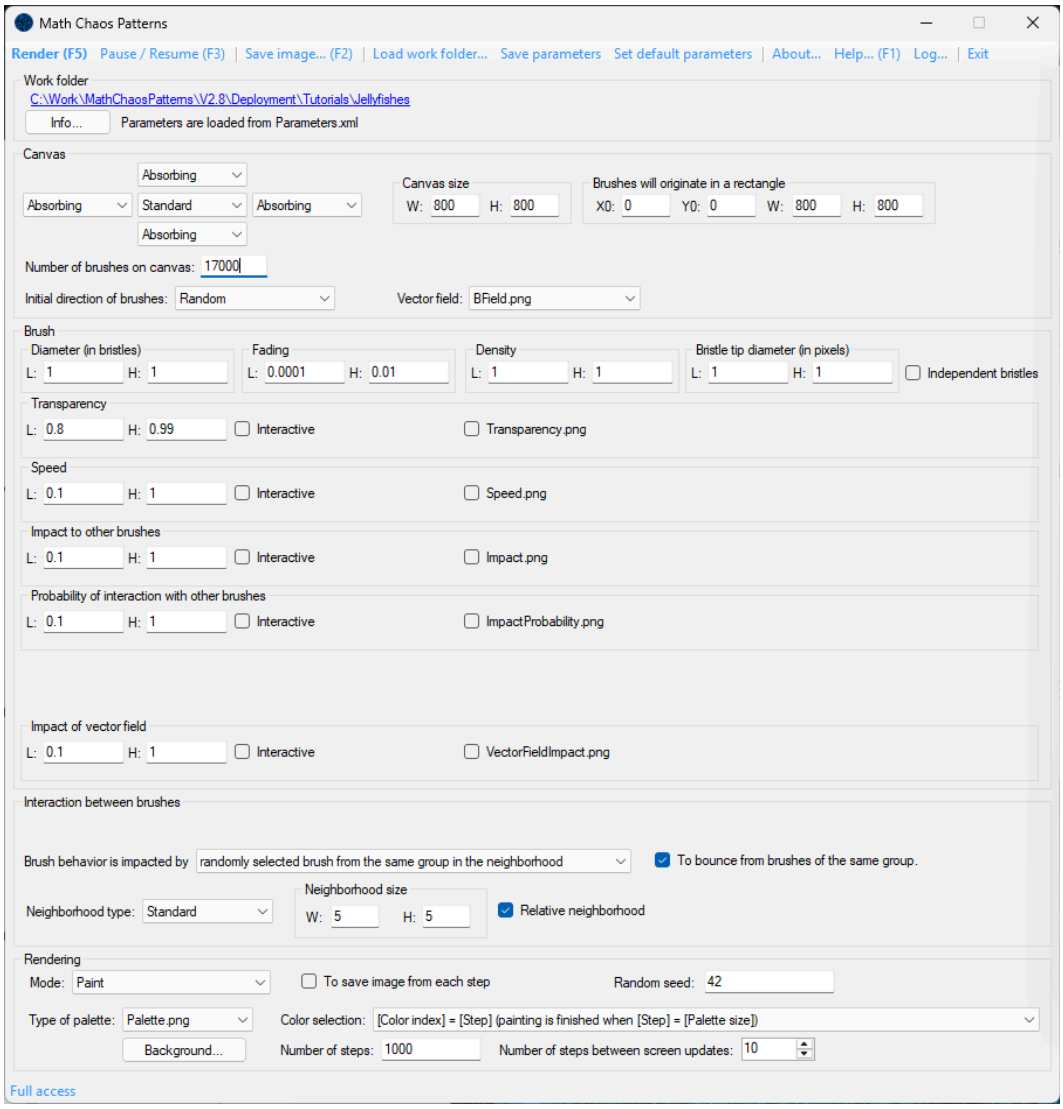


File Palette.png stores the palette used by the brushes:

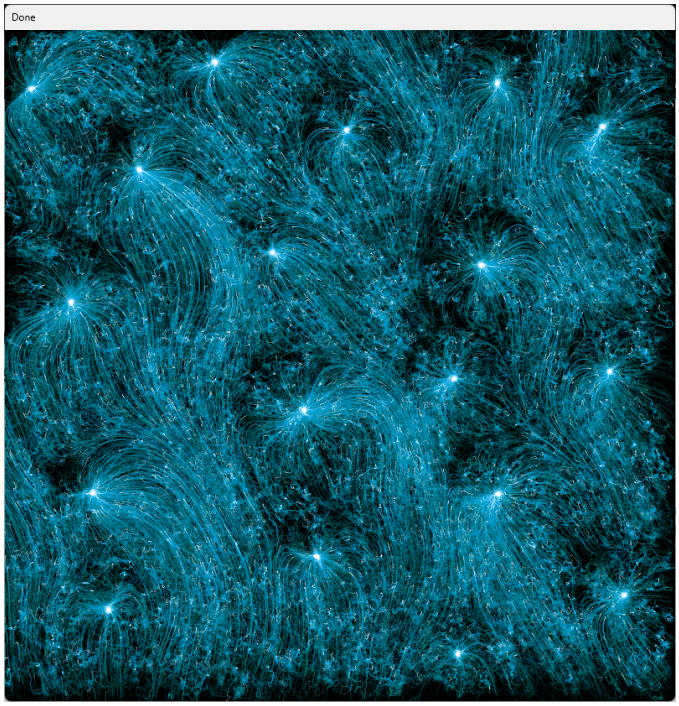


Jellyfishes

This tutorial demonstrates how to use the *basis* vector field (this field is different from potential vector fields controlled by PField.png, PPField.png, and NPField.png). In the MCP menu select [Load work folder…], then select folder Jellyfishes from folder Tutorials. Then the control window appears:

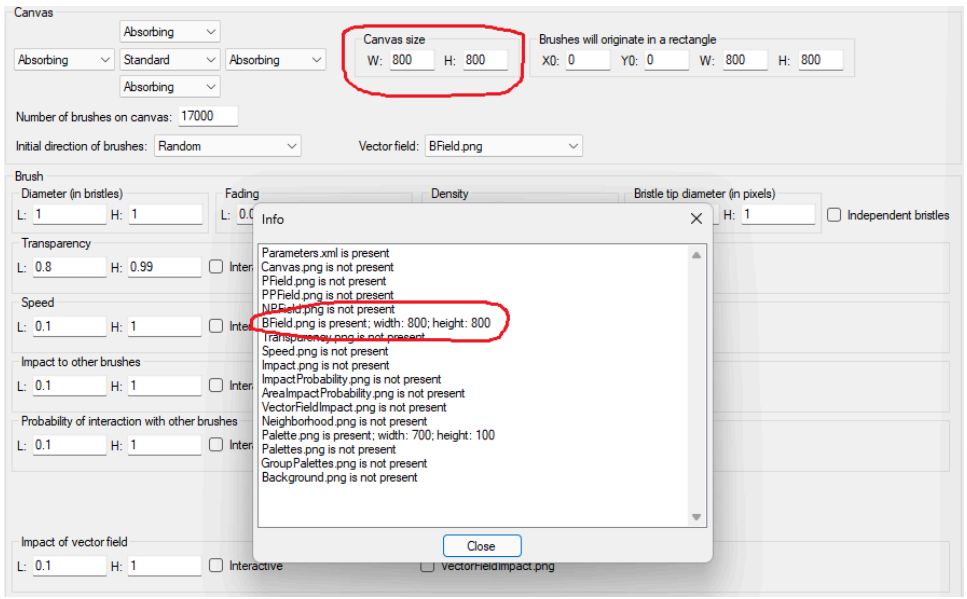


And the rendering window after 1000 steps for 17000 brushes should look like this:



In the control window note two parameters referring to files: BField.png and Palette.png. They are located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*.

Note that the size of canvas is exactly the same size as the file BField.png:

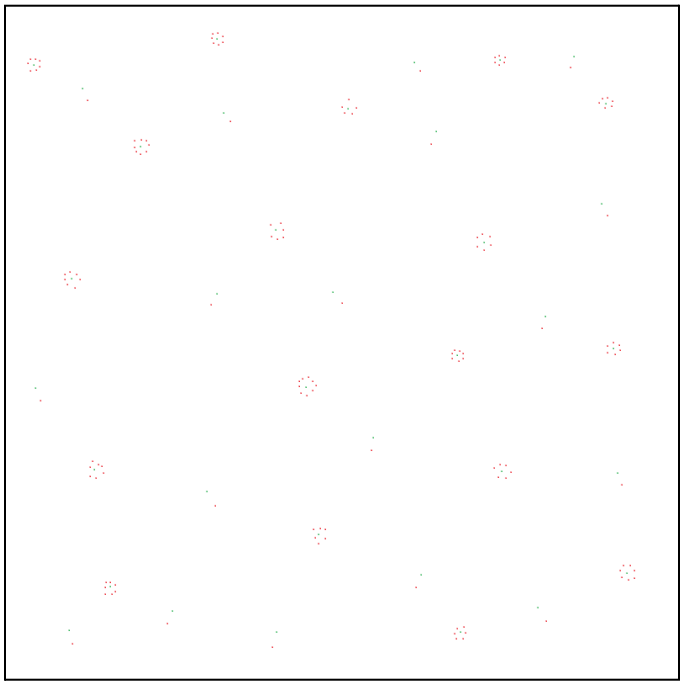


If you want to examine those files closer then click on the link in the group *Work folder* to open that work folder in Windows File Explorer.

File Palette.png stores the palette used by the brushes:

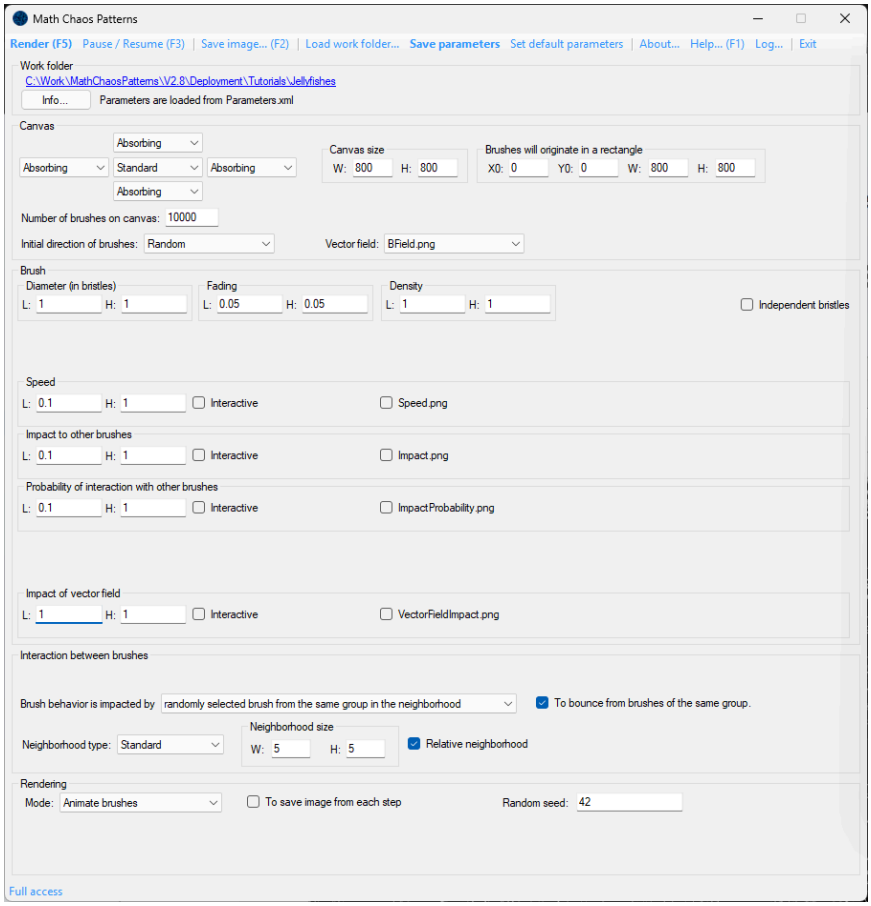


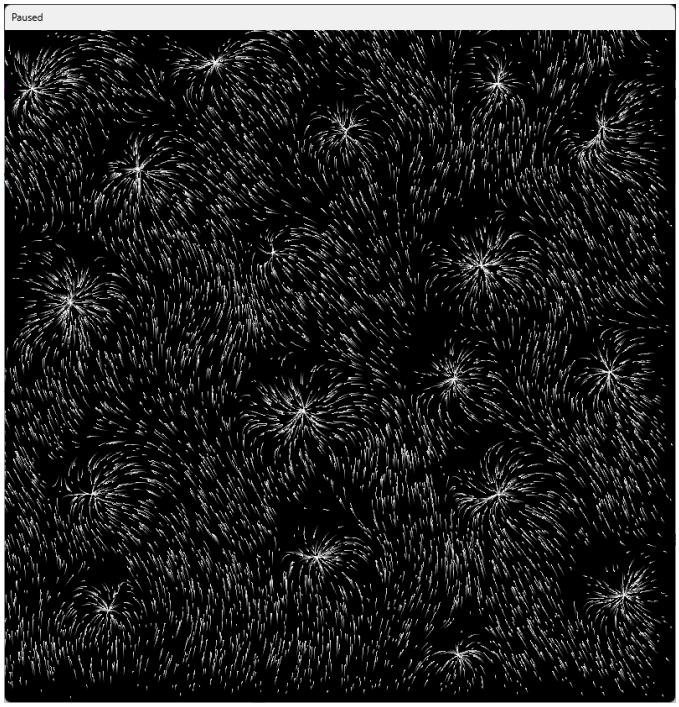
File BField.png stores information about the basis vector field:



A basis field is created using image BField.png from the current work folder, where each preferred direction is defined by a pair of closest pixels: the red pixel marks the beginning of the vector and the green pixel marks its end. All such pairs form a collection of vectors called the *basis*. For each pixel on the canvas, the resulting direction is calculated as the normalized sum of normalized basis vectors, weighted by the squared distances to those vectors. You can edit BField.png in Windows Paint to see possible variations or come up with your own basis field (as you can see it is simple).

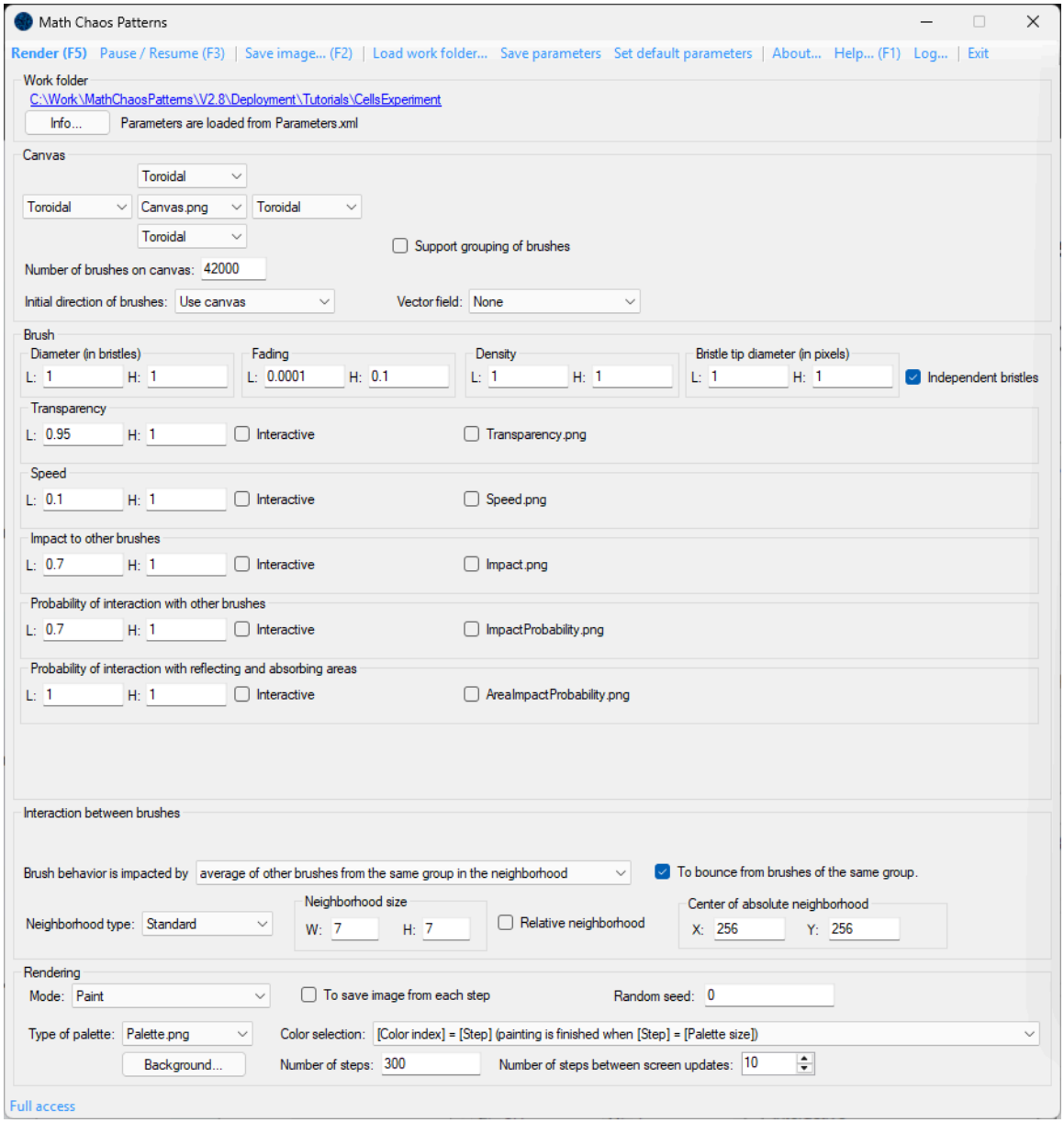
If you want to see how brushes are affected by the vector field, switch Rendering [Mode] to *Animate brushes*, set [Impact of vector field] to [1, 1], and change L and H of [Brush fading] to 0.05. Reduce the number of brushes to 10000 and Hit F5. Then you will see how the vector field affects brushes. Here is the snap of control window and rendering window:


Math Chaos Patterns
Render (F5) Pause / Resume (F3) | Save image... (F2) | Load work folder... Save parameters Set default parameters | About... Help... (F1) Log... | Exit
Work folder
C:\Work\MathChaosPatterns\V2.8\Deployment\Tutorials\Jellyfishes
Info...
Parameters are loaded from Parameters.xml
Canvas
Absorbing
Absorbing
Standard
Absorbing
Absorbing
Canvas size
W: 800 H: 800
Brushes will originate in a rectangle
X0: 0 Y0: 0 W: 800 H: 800
Number of brushes on canvas: 10000
Initial direction of brushes: Random
Vector field: BField.png
Brush
Diameter (in bristles)
L: 1 H: 1
Fading
L: 0.05 H: 0.05
Density
L: 1 H: 1
Independent bristles
Speed
L: 0.1 H: 1
Interactive
Speed.png
Impact to other brushes
L: 0.1 H: 1
Interactive
Impact.png
Probability of interaction with other brushes
L: 0.1 H: 1
Interactive
ImpactProbability.png
Impact of vector field
L: 1 H: 1
Interactive
VectorFieldImpact.png
Interaction between brushes
Brush behavior is impacted by randomly selected brush from the same group in the neighborhood
To bounce from brushes of the same group.
Neighborhood type: Standard
Neighborhood size
W: 5 H: 5
Relative neighborhood
Rendering
Mode: Animate brushes
To save image from each step
Random seed: 42
Full access

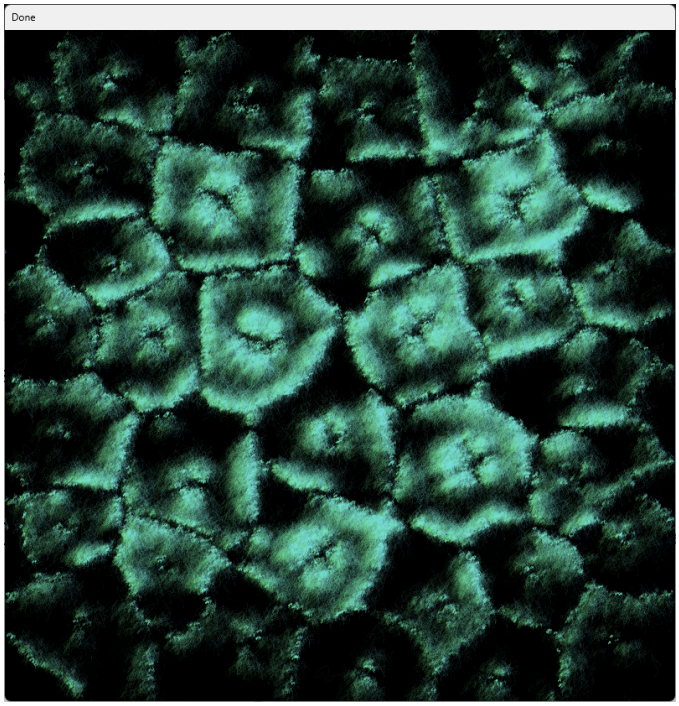

Paused

Cells Experiment

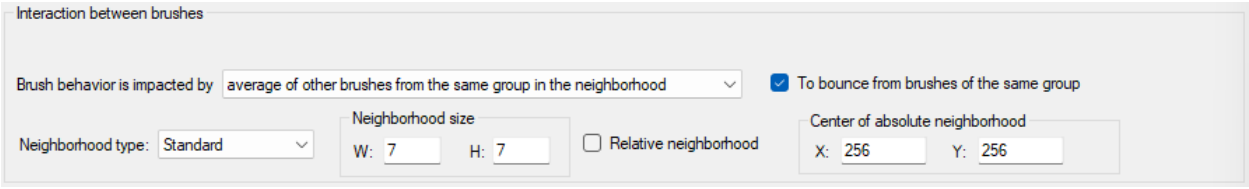
This tutorial demonstrates how to use a user defined canvas and an absolute neighborhood for interacting brushes. In the MCP menu select [Load work folder…], then select folder CellsExperiment from folder Tutorials. Then the control window appears:



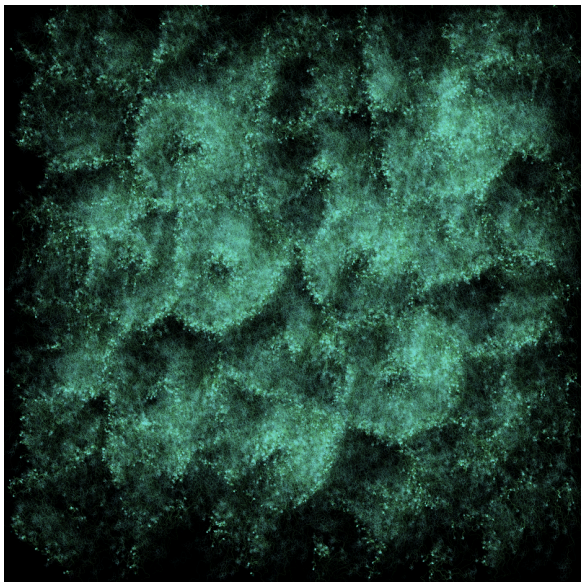
And the rendering window after 300 steps for 42000 brushes should look like this:



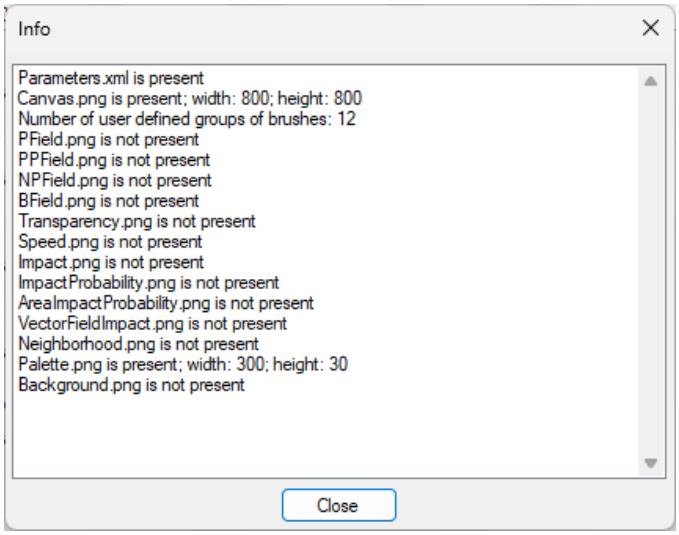
When the neighborhood is absolute, only brushes from a fixed area on canvas can impact the current brush, with parameters [X] and [Y] specifying the center of that area. In this tutorial, any brush on the canvas may be impacted only by brushes passing through a 7×7 square centered at pixel coordinates (256, 256):



Note that using absolute neighborhoods brings the most uncertainty in the output. This is due to an unknown number of brushes with unknown characteristics that may end up in that neighborhood. On the other hand this allows you to produce truly amazing brush paintings, for example, like this (an absolute neighbourhood 13x13 with a different center was used):

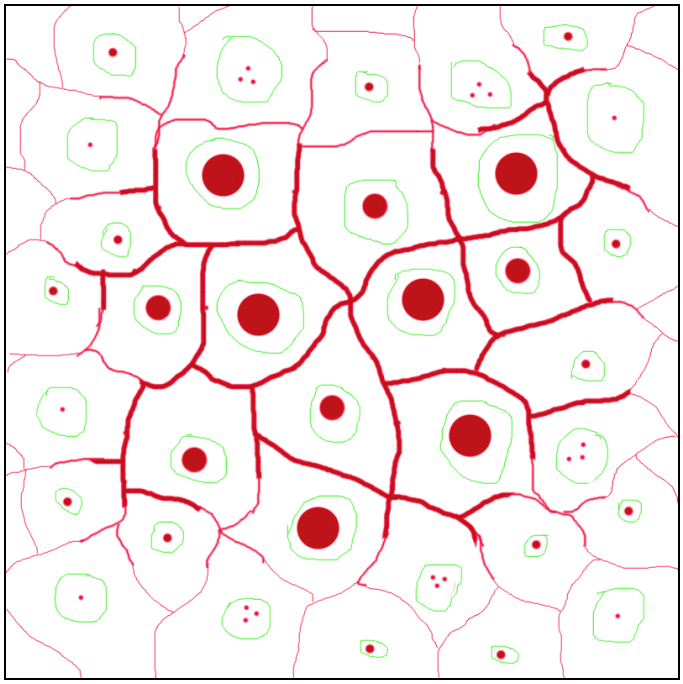


In the control window note two parameters that are files: Canvas.png and Palette.png. They are located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*:



If you want to examine those files closer then click on the link in the group *Work folder* to open that work folder in Windows File Explorer.

File Canvas.png sets an initial location for all brushes and initial direction. In the control window the parameter [Initial direction] is set to *Use canvas*. In that case each brush starts its walk from red areas towards the closest pixel from green areas:

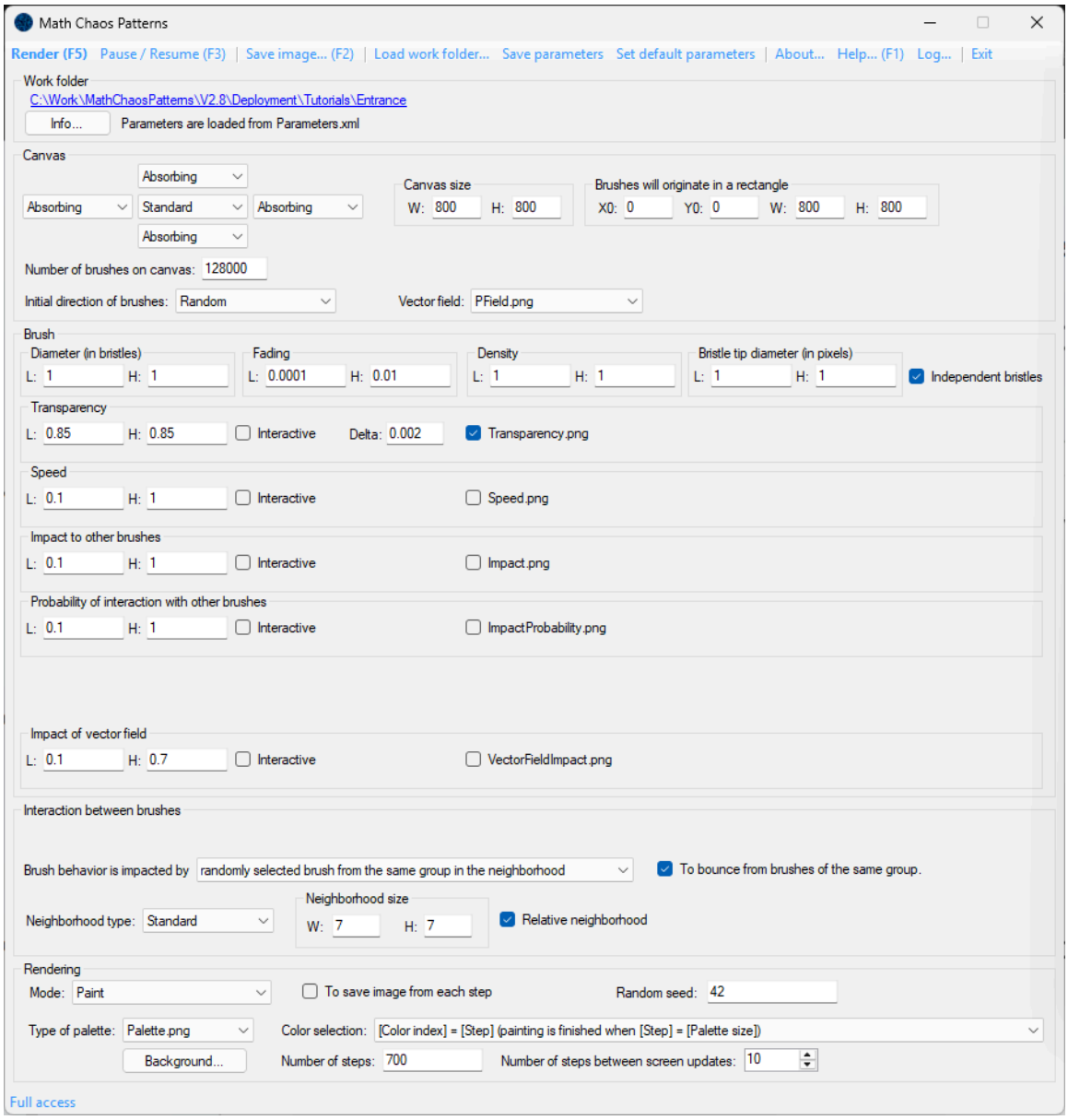


File Palette.png stores the palette used by the brushes:

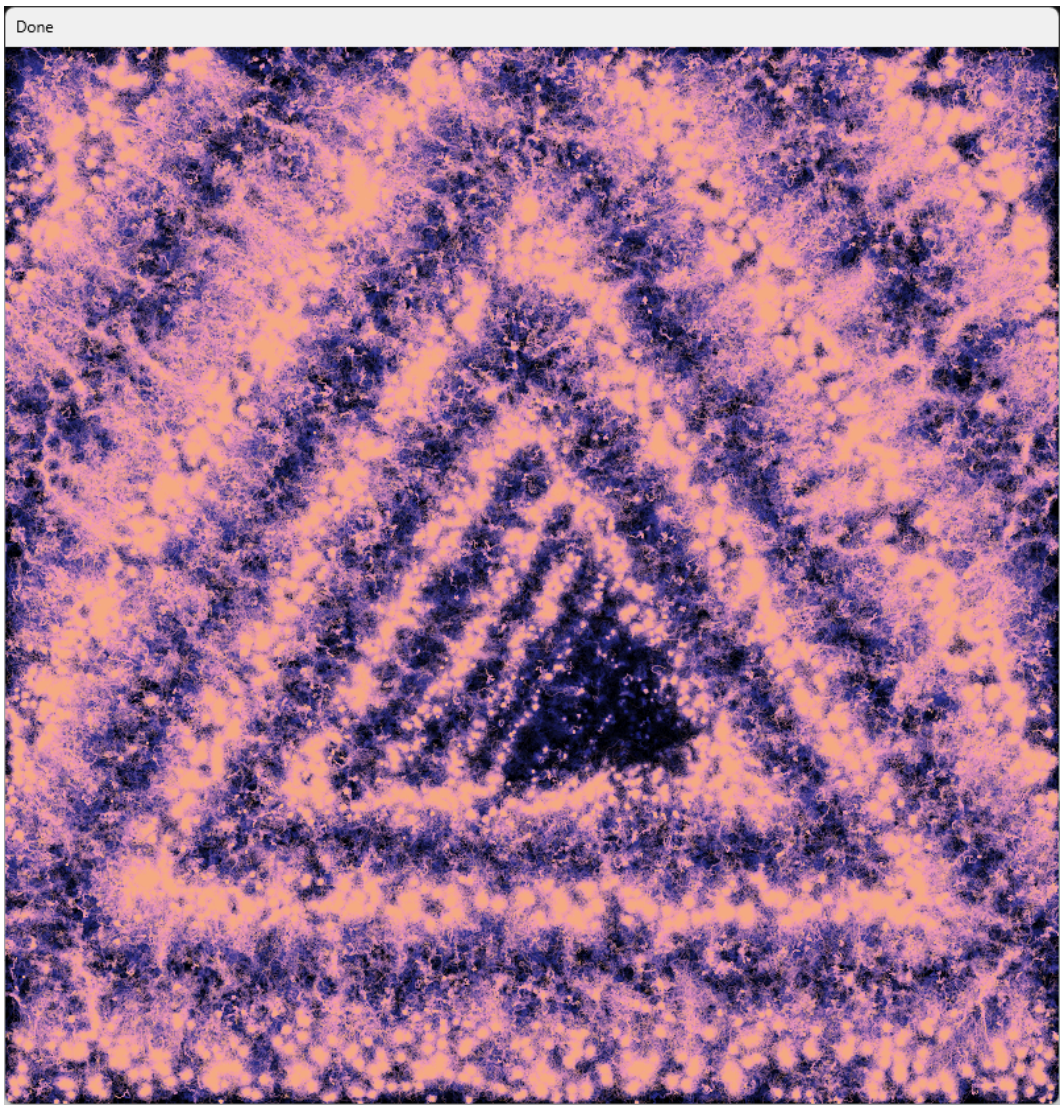


Entrance

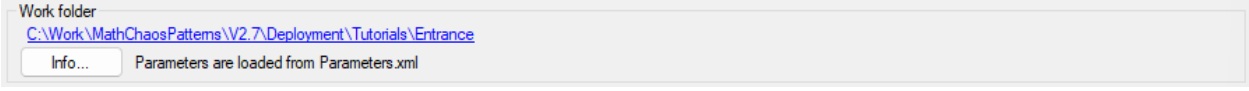
This tutorial explains how to apply a potential vector field to influence brush trajectories and how to use Transparency.png to control per-step transparency, enabling depth effects in the final composition. In the MCP menu select [Load work folder…], then select folder Entrance from folder Tutorials. Then the control window appears:



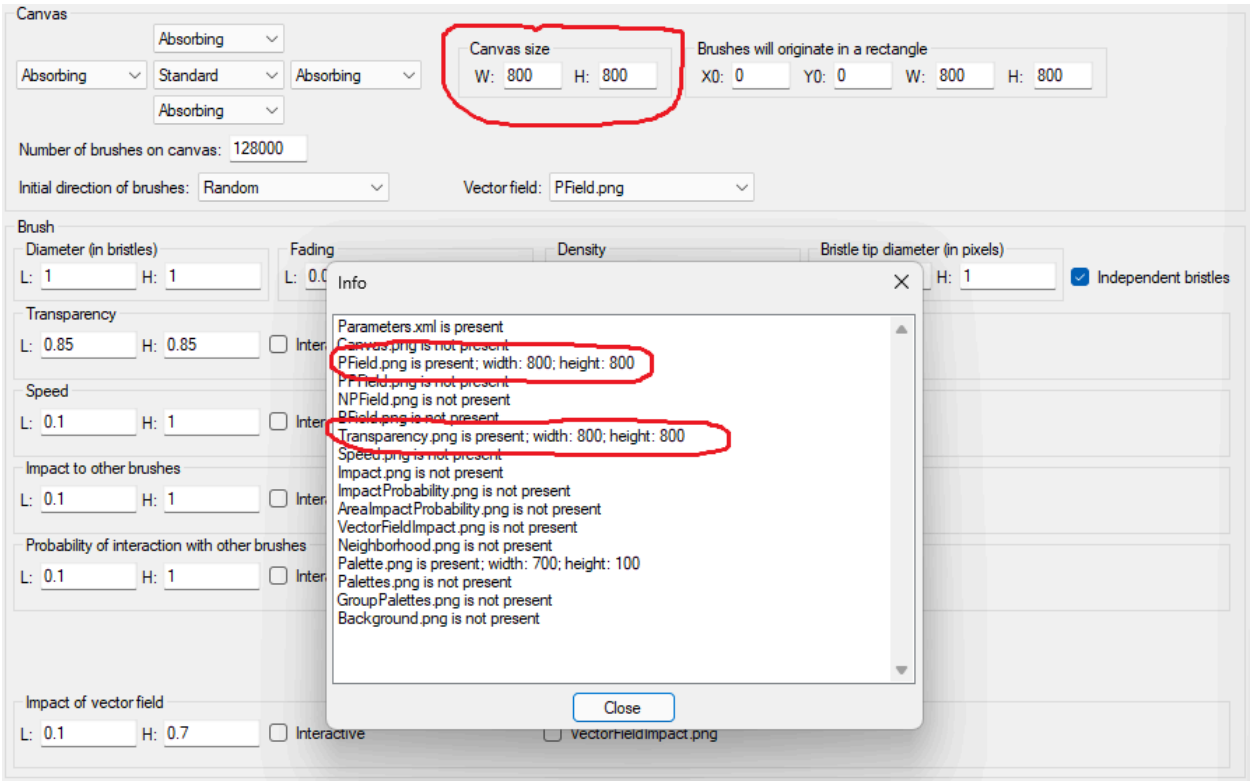
And the rendering window after 700 steps for 128000 brushes should look like this:



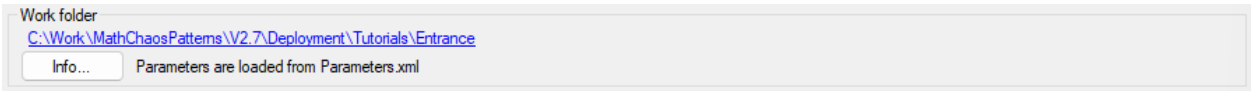
In the control window note parameters that refer to files: PField.png, Transparency.png, and Palette.png. They are all located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*:

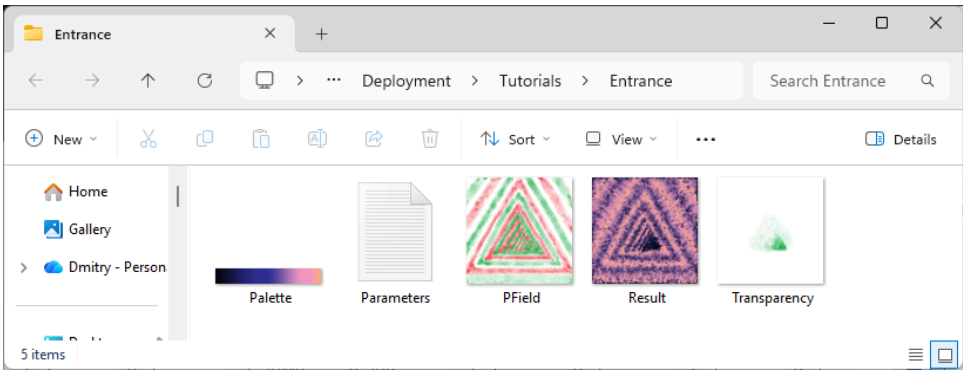


Note that the size of canvas is exactly the same size as files PField.png and Transparency.png (800 x 800):



If you want to examine those files closer then click on the link in the group *Work folder* to open that work folder in Windows File Explorer:

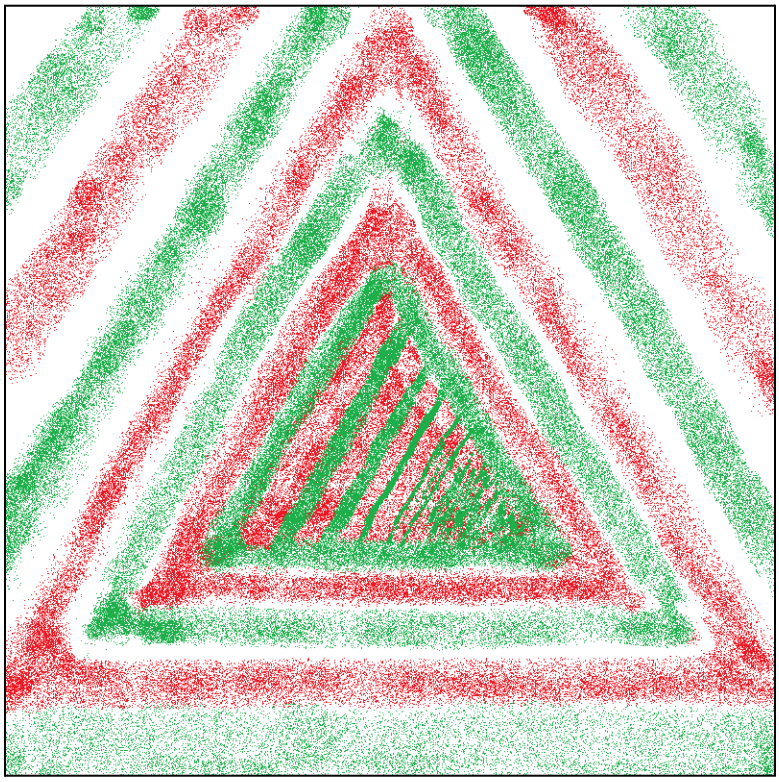




File Palette.png stores the palette used by the brushes.

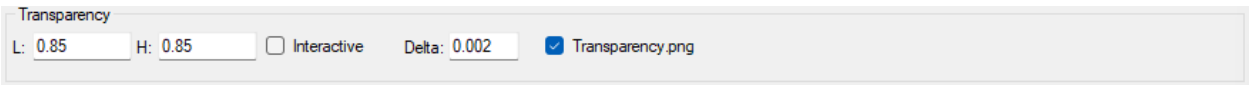


File PField.png stores information about the potential field:

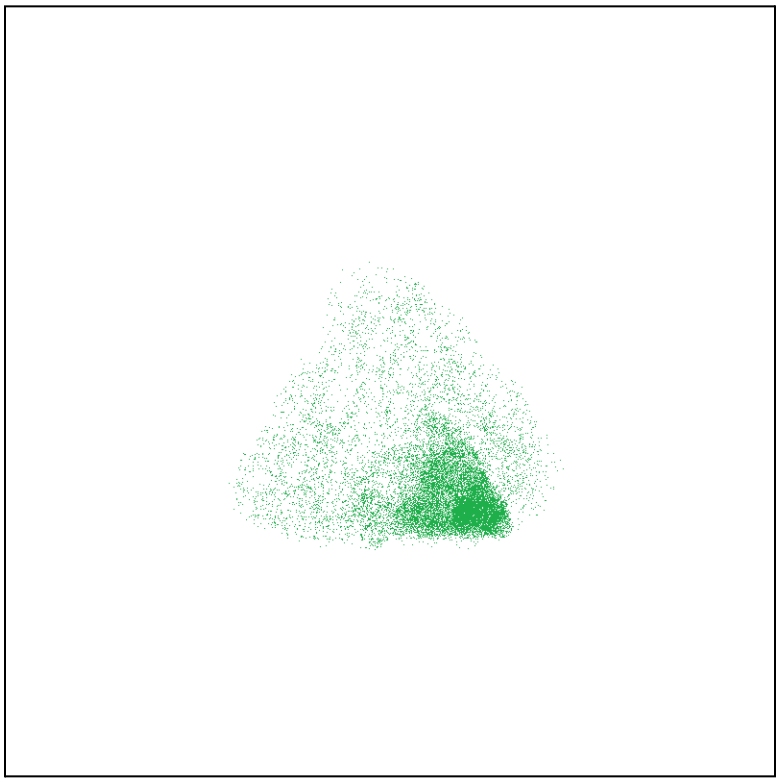


Here red pixels push brushes outward and green pixels pull brushes inward. You can edit it in Windows Paint to see possible variations or come up with your own potential field (as you can see it is simple). Not all of those pixels are actually used, because MCP samples them such that the resulting vector field is still correctly represented but the computational time is not too long, see parameter maxNumberOfPointsForVectorFields.

In the screenshot below, the Transparency.png file is selected, causing each brush to gain an additional 0.002 transparency (see [Delta]) whenever its trajectory passes a canvas element corresponding to a green pixel in Transparency.png. This lets you locally modulate opacity based on the brush's position.

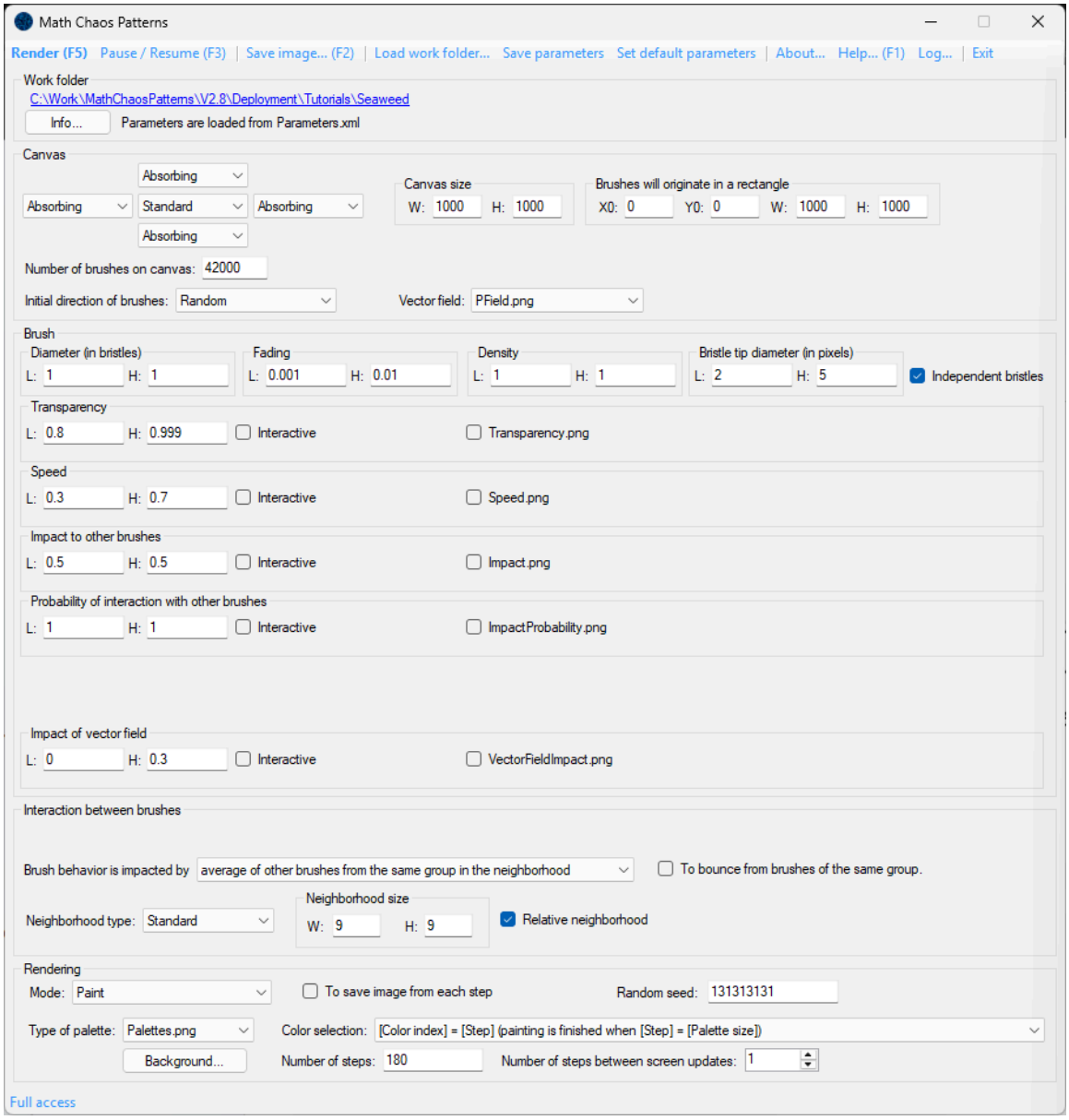


The Transparency.png file enables you to define regions of the canvas where passing brushes become more or less transparent. Brushes that move near the area corresponding to the green triangle in Transparency.png (see below) receive higher transparency values. By varying opacity based on position, the system creates an illusion of depth.

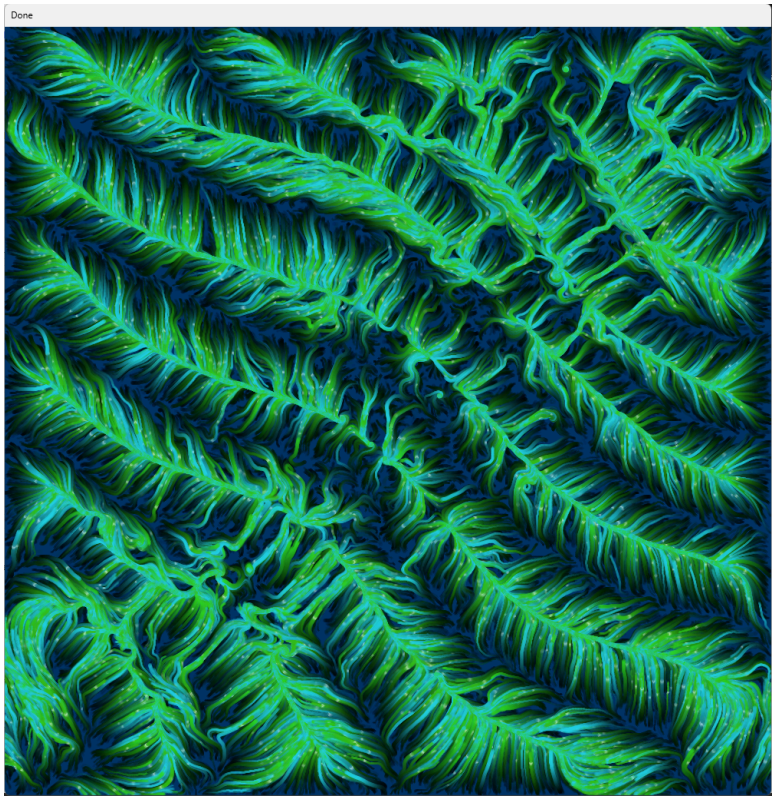


Seaweed

This tutorial demonstrates how to use the *potential* vector field and assign different color palettes to brushes. In the MCP menu select [Load work folder…], then select folder Seaweed from folder Tutorials. Then the control window appears:



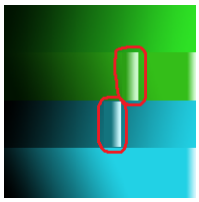
And the rendering window after 180 steps for 42000 brushes should look like this:



This is the first tutorial in which brushes use different color palettes. As mentioned above, when [Type of palette] is set to Palettes.png, MCP assigns a randomly selected row from Palettes.png to each brush as its color palette:

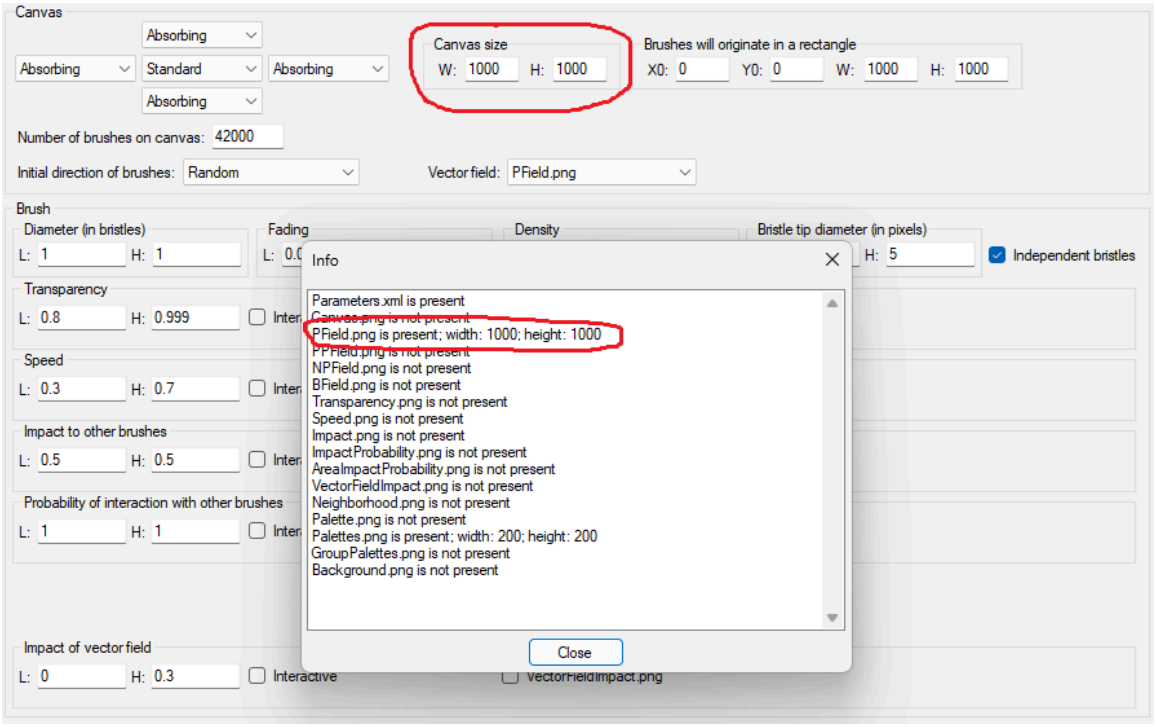


Different shades can therefore be used for the seaweed. The seaweed seeds can be rendered with their own distinct color variations. These variations originate from the two regions of Palettes.png enclosed by the red outlines:

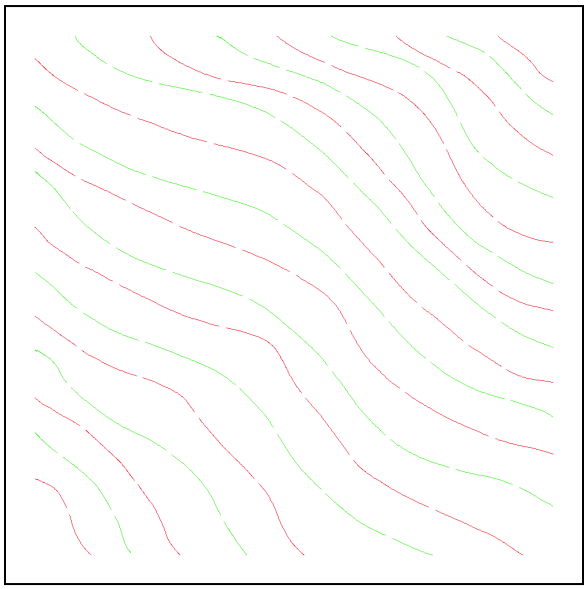


In the control window note two parameters referring to files: PField.png and Palettes.png. They are located in the current work folder. You can see additional info about them if you click the button [Info…] in the group *Work folder*.

Note that the size of canvas is exactly the same size as the file PField.png:

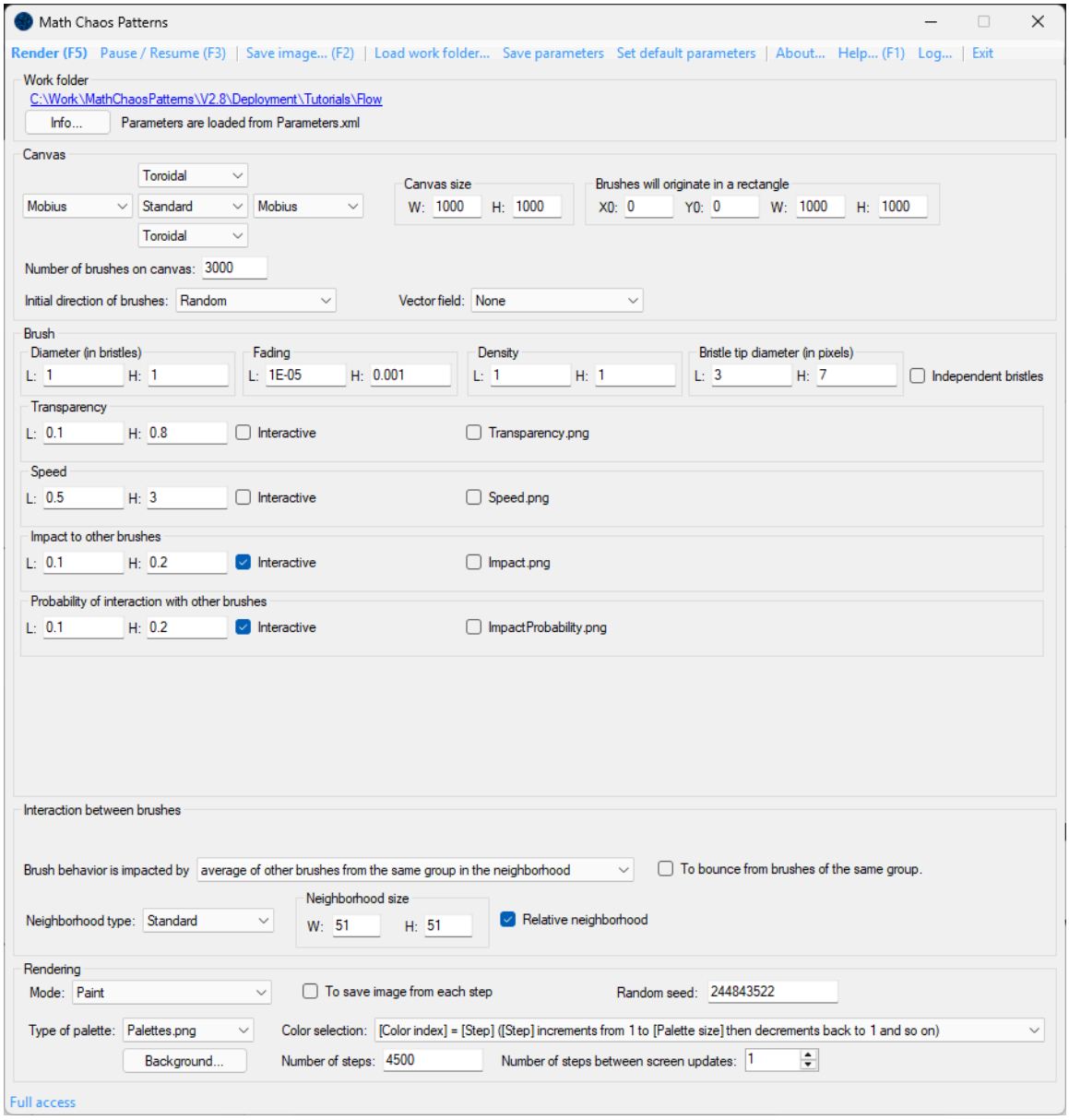


PField.png stores information about the potential field. Green pixels act as sources that pull brushes inward, while red pixels act as sources that push brushes outward:



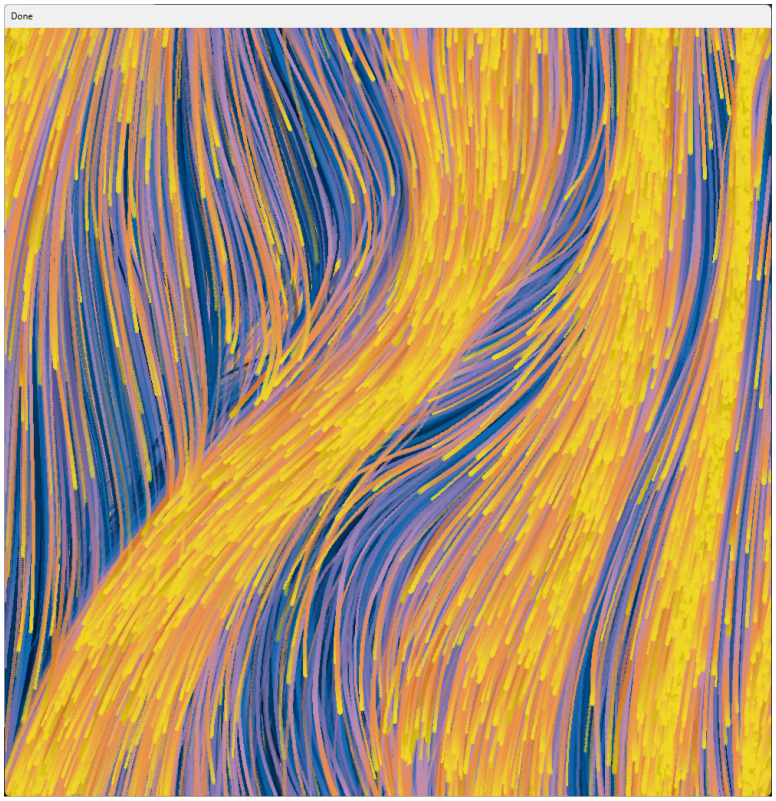
Flow

This tutorial demonstrates how to use different types of canvas boundaries and assign different color palettes to brushes. In the MCP menu select [Load work folder…], then select folder Flow from folder Tutorials. Then the control window appears:



As you can see, the left and right boundaries of the canvas are set to Möbius, while the top and bottom boundaries are set to Toroidal. This means that brushes crossing a boundary are transferred to the opposite side of the canvas. Vertically, the canvas surface is interpreted as a 2D torus, while horizontally it is interpreted as a Möbius band. As a result, brushes wrapping around the left or right boundary reappear on the opposite side with their orientation reversed,

whereas brushes crossing the top or bottom boundary wrap around normally. After 4500 steps the output is as follows:



This is another tutorial in which brushes use different color palettes. As a result, different parts of the flow can be rendered using different shades. This effect is made possible by the following palette file, Palettes.png:



As you can see, there are essentially two color palettes, one smaller than the other. The smaller palette is simply a dimmer version of the larger one. This creates additional visual dynamics within the flow.

If [Save image from each step] is enabled, MCP will save all 4500 images to the Video folder in the Flow work folder (you can save rendering time if you set [Number of steps between screen updates] to a larger value, like 50 or 100). You can then run MCP Video Glue to convert those images into an MP4 video. Here is a link to the resulting video:

https://www.reddit.com/r/generative/s/UyjbNYeXFy

Part Three

Important updates, hints, and tips

1. Updating your MCP 2.8.0 project for MCP 2.8.1. If you created a project using MCP 2.8.0, you will need to make a few small updates to your Parameters.xml file so the project loads correctly in MCP 2.8.1. Required changes:
    a. If in your project boundaries of the canvas absorb brushes then change lines:
        <isReflectedFromTop>false</isReflectedFromTop>
        <isReflectedFromBottom>false</isReflectedFromBottom>
        <isReflectedFromLeft>false</isReflectedFromLeft>
        <isReflectedFromRight>false</isReflectedFromRight>
    to the following lines:
        <btTop>0</btTop>
        <btBottom>0</btBottom>
        <btLeft>0</btLeft>
        <btRight>0</btRight>
    b. If in your project boundaries of the canvas reflect brushes then change lines:
        <isReflectedFromTop>true</isReflectedFromTop>
        <isReflectedFromBottom>true</isReflectedFromBottom>
        <isReflectedFromLeft>true</isReflectedFromLeft>
        <isReflectedFromRight>true</isReflectedFromRight>
    to the following lines:
        <btTop>1</btTop>
        <btBottom>1</btBottom>
        <btLeft>1</btLeft>
        <btRight>1</btRight>
    c. If your project still does not load correctly in MCP 2.8.1, contact us at D1BLLC@MathChaosPatterns.com
2. Updating your MCP 2.7.* project for MCP 2.8.1. If you created a project using MCP 2.7.*, you will need to make a few additional (to the previous updates) small updates to your Parameters.xml file so the project loads correctly in MCP 2.8.1. Required changes:
    a. Anywhere before </ProcessParameters> add the line:
    <randomSeed>0</randomSeed>
    Then MCP will select the random seed for you. In case you like the output and want to reproduce it, you can find the last random seed in the log (click “Log…” in the main menu and go to the bottom of the log file).

   b. Change every instance of:
      <deltaFileName />
      to:
      <isFileUsed>false</isFileUsed>
   c. If your project still does not load correctly in MCP 2.8.1, contact us at D1BLLC@MathChaosPatterns.com
3. For each project—large or small—it is practical to create a separate work folder, since multiple parameters are represented as images (canvas, vector fields, control files such as Speed.png and Transparency.png, other files like brush neighborhood, palette, and background).
4. When rendering, you can save time by increasing the number of steps between screen updates. You can also minimize MCP and check its progress by hovering your mouse over its icon on the Windows taskbar.
5. Ensure that the Video folder inside your work folder does not contain any files you may regret losing; all files in the Video folder will be deleted before rendering.
6. Before rendering with a large number of brushes, it is recommended to test your parameters with a smaller number of brushes to save time.
7. If you encounter unexpected results that you would like to debug, the rendering mode [Animate brushes] often helps identify the issue.
8. It is also a good idea to make a copy of the Parameters.xml after achieving an interesting result. You may want to recreate or modify the result later.
9. To view MCP performance statistics and other process information, click [Log…] in the MCP menu. Information is always appended to the end of the log file log.txt. If you want to start a new log, rename or remove log.txt and MCP will create a new file automatically. When log.txt grows larger than 10 MB, MCP deletes it for performance reasons and starts a new log. The log file is located in Documents\D1B LLC\MCP.
10. In case of any issues with MCP (bugs, crashes, etc.), please, email the relevant portion of the log along with screenshots (if available) to: D1BLLC@MathChaosPatterns.com. Please do not attach screenshots as separate files. Instead, insert them directly into the body of your email.

How to quickly create a video from MCP output?

If you have a set of images in various sizes and want to turn them into a video, the **MCP Video Glue** utility makes the process effortless. Its clean, minimal interface includes only the essential controls — video width and height, frame rate, bit rate, looping options, and more. It is especially useful when working with generative art software such as MCP. The **MCP Video Glue** is available on the Microsoft Store.

Want to know more?

For the best learning experience, we recommend using this manual together with the accompanying video lessons available here:

https://www.youtube.com/@MathChaosPatterns/videos

To see more examples and tutorials visit:

https://www.facebook.com/MathChaosPatterns/

https://www.facebook.com/groups/MathChaosArt/

SIGGRAPH 2020 presentation:

https://youtu.be/JH71tUzIO0I

SIGGRAPH 2020 paper:

https://dl.acm.org/doi/fullHtml/10.1145/3388529.3407308