MilkDrop preset authoring guide

Author: Ryan Geiss

Converted to word: Rovastar

This is the original authoring guide written by Ryan Geiss and is correct as of version 1.02 of MilkDrop. It has been converted to Word for ease of use. Also added a few missing functions.

1. About Presets

A 'preset' is a collection of parameters that tell MilkDrop how to draw the wave, how to warp the image around, and so on. MilkDrop ships with around ~100 built-in presets, each one having a distinct look and feel to it.

Using MilkDrop's built-in “preset-editing menu”, you can edit presets on the fly, from within the program. You can make slight adjustments to existing presets, and then save over them; or you can change lots of things, so the preset doesn't look anything like the original, and then save it under a new name. You can even write insane new mathematical equations, of your own imagination, into your preset files and come up with things that MilkDrop has never done before!

Each preset is saved as a file with the “.milk” extension, so you can easily send them to your friends or post them on the web. You can also go to http://www.nullsoft.com/free/milkdrop and then jump to the “preset sharing forum” to see what other people have come up with, or post your own cool, new presets.

2. Preset Authoring - Basic

You can edit the properties of the current preset by hitting 'M', which brings up the “preset-editing menu”. From this menu you can use the up and down arrow keys to select an item. Press the RIGHT arrow key to move forward through the menu and select the item (note: you can also hit SPACE or RETURN to do this); press the LEFT arrow key to go back to the previous menu.

Pressing 'M' while the menu is already showing will hide the menu; pressing ESCAPE will do the same thing. Press 'M' again to bring the menu back.

Once you've reached an item on the menu whose value can be edited, use the UP and DOWN arrow keys to increase or decrease its value, respectively. Changes will register immediately. Use PAGE UP and PAGE DOWN to increase the value more quickly. Hold down SHIFT and use the UP/DOWN arrow keys to change the value very slowly. Hit RETURN to keep the new value, or ESC to abort the change.

If the item you're editing is a text string, you can use the arrow keys to move around. The Insert key can be used to toggle between insert and overtype modes. You can hold shift and use the arrow keys (home, end, left, right) to make a selection, which will be identified by brackets []. You can then use CTRL-C or CTRL-X to copy or cut text. CTRL-P pastes. When finished editing, hit RETURN to keep the new string, or ESC to abort the change.

You'll want to get into the habit of using SCROLL LOCK whenever you're making changes to a preset that you intend to save; otherwise, the preset is sure to keep randomly changing on you. You might ask me, “Why don't you just automatically lock the preset while the menu is up?” I will answer you, “because in the instant between exiting the menu and going to save the preset, the preset might then switch, and you'd lose your changes.” Then you might ask me, “then why don't you just leave it locked whenever the user makes a change from the menu?” and I will say to you, “because I hate it when programs do things like that. You have an idea in your brain about the state of the Scroll Lock key, because you're the one setting that state. I want your mind to always be up-to-date.” And you might then ask me: “how large is large?” And I will tell you: “thirty.”

There are also some hotkeys that will allow you to change certain common parameters to the current preset. These are listed below.

- Motion

  i/I - zoom in/out
  y/Y - decrease, increase zoom exponent (perspective)
  [/] - push motion to the left/right (dx)
  { / } - push motion up/down (dy)
  < / > - rotate left/right (rot)
  o/O - shrink/grow the amplitude of the warp effect

- Waveform

  W - cycle through waveforms
  j/J - scale waveform down/up
  k/K - smooth the waveform less/more
  e/E - make the waveform more transparent/more solid

- Brightness

  d/D - decrease, increase decay (fades image to black over time)
  g/G - decrease, increase gamma (brightness)

- Video Echo effect

  q/Q - scale 2nd graphics layer down/up
  a/A - decrease/increase alpha of 2nd graphics layer
  F - flip 2nd graphics layer (cycles through 4 fixed orientations)

3. Preset Authoring - Advanced

This section describes how to use the 'per-frame' and 'per-pixel' equations to develop unique new presets.

a. Per_frame Equations

When you hit 'm' to show the preset-editing menu, several items show up. If you explore the sub-menus, you'll see that all of the properties that make up the preset you're currently viewing are there. The values you can specify here (such as zoom amount, rotation amount, wave colour, etc.) are all static values, meaning that they don't change in time. For example, take the 'zoom amount' option under the 'motion' submenu. If this value is 1.0, there is no zoom. If the value is 1.01, the image zooms in 1% every frame. If the value is 1.10, the image zooms in 10% every frame. If the value is 0.9, the image zooms out 10% every frame; and so on.

However, presets get far more interesting if you can take these parameters (such as the zoom amount) and animate them (make them change over time). For example, if you could take the 'zoom amount' parameter and make it oscillate (vary) between 0.9 and 1.1 over time, the image would cyclically zoom in and out, in time.

You can do this - by writing 'per-frame' and 'per-pixel' equations. Let's start with 'per-frame' equations. These are executed once per frame. So, if you were to type the following equation in:

```
zoom = zoom + 0.1*sin(time);
```

...then the zoom amount would oscillate between 0.9 and 1.1 over time. (Recall from your geometry classes that sin() returns a value between -1 and 1.) The equation says: "take the static value of 'zoom', then replace it with that value, plus some variation." This particular equation would oscillate (cycle) every 6.28 seconds, since the sin() function's period is 6.28 (PI*2) seconds. If you wanted it to make it cycle every 2 seconds, you could use:

```
zoom = zoom + 0.1*sin(time*3.14);
```

Now, let's say you wanted to make the colour of the waveform (sound wave) that gets plotted on the screen vary through time. The colour is defined by three values, one for each of the main colour components (red, green, and blue), each in the range 0 to 1 (0 is dark, 1 is full intensity). You could use something like this:

```
wave_r = wave_r + 0.5*sin(time*1.13);
wave_g = wave_g + 0.5*sin(time*1.23);
wave_b = wave_b + 0.5*sin(time*1.33);
```

It's nice to stagger the frequencies (1.13, 1.23, and 1.33) of the sine functions for the red, green, and blue colour components of the wave so that they cycle at different rates, to avoid them always being all the same (which would create a greyscale wave).

Here is a full list of the variables available for writing per-frame equations:

Name	Writable?	Range	Description
zoom	yes	>0	Controls inward/outward motion. 0.9=zoom out 10% per frame, 1.0=no zoom, 1.1=zoom in 10%
zoomexp	yes	>0	Controls the curvature of the zoom; 1=normal
rot	yes		Controls the amount of rotation. 0=none, 0.1=slightly right,-0.1=slightly clockwise, 0.1=CCW
warp	yes	>0	Controls the magnitude of the warping; 0=none, 1=normal, 2=major warping...
cx	yes	0..1	Controls where the centre of rotation and stretching is, horizontally. 0=left, 0.5=centre, 1=right
cy	yes	0..1	Controls where the centre of rotation and stretching is, vertically. 0=top, 0.5=centre, 1=bottom
dx	yes		controls amount of constant horizontal motion; -0.01 = move left 1% per frame, 0=none, 0.01 = move right 1%
dy	yes		Controls amount of constant vertical motion; -0.01 = move up 1% per frame, 0=none, 0.01 = move down 1%
sx	yes	>0	Controls amount of constant horizontal stretching; 0.99=shrink 1%, 1=normal, 1.01=stretch 1%
sy	yes	>0	Controls amount of constant vertical stretching; 0.99=shrink 1%, 1=normal, 1.01=stretch 1%
decay	yes	0..1	Controls the eventual fade to black; 1=no fade, 0.9=strong fade
wave_r	yes	0..1	Amount of red colour in the wave (0..1),
wave_g	yes	0..1	Amount of green colour in the wave (0..1)
wave_b	yes	0..1	Amount of blue colour in the wave (0..1)
wave_x	yes	0..1	Position of the waveform: 0 = far left edge of screen, 0.5 = centre, 1 = far right
wave_y	yes	0..1	Position of the waveform: 0 = very bottom of screen, 0.5 = centre, 1 = top
wave_mystery	yes	-1..1	What this parameter does is a mystery. (honestly, though, this value does different things for each waveform; for example, it could control angle at which the waveform was drawn.)
ob_size	yes	0..0.5	Thickness of the outer border drawn at the edges of the screen every frame
ob_r	yes	0..1	Amount of red colour in the outer border
ob_g	yes	0..1	Amount of green colour in the outer border
ob_b	yes	0..1	Amount of blue colour in the outer border
ob_a	yes	0..1	Opacity of the outer border (0=transparent, 1=opaque)
ib_size	yes	0..0.5	Thickness of the inner border drawn at the edges of the screen every frame
ib_r	yes	0..1	Amount of red colour in the inner border
ib_g	yes	0..1	Amount of green colour in the inner border
ib_b	yes	0..1	Amount of blue colour in the inner border
ib_a	yes	0..1	Opacity of the inner border (0=transparent, 1=opaque)

mv_a	yes	0..1	Opacity of the motion vectors (0=transparent, 1=opaque)
mv_r	yes	0..1	Amount of red colour in the motion vectors
mv_g	yes	0..1	Amount of green colour in the motion vectors
mv_b	yes	0..1	Amount of blue colour in the motion vectors
mv_x	yes	0..64	The number of motion vectors in the X direction
mv_y	yes	0..48	The number of motion vectors in the Y direction
mv_l	yes	0..5	The length of the motion vectors (0=no trail, 1=normal, 2=double...)
time	NO	>0	retrieves the current time, in seconds, since MilkDrop started running
bass	NO	>0	Retrieves the current amount of bass. 1 is normal; below ~0.7 is quiet; above ~1.3 is loud bass
mid	NO	>0	-same, but for mids (middle frequencies)
treb	NO	>0	-same, but for treble (high) frequencies
bass_att	NO	>0	Retrieves an attenuated reading on the bass, meaning that it is damped in time and doesn't change so rapidly.
mid_att	NO	>0	-same, but for mids (middle frequencies)
treb_att	NO	>0	-same, but for treble (high) frequencies
frame	NO		Retrieves the number of frames of animation elapsed since the program started
progress	NO	0..1	Progress through the current preset; if preset was just loaded, this is closer to 0; if preset is about to end, this is closer to 1.Note: If Scroll Lock is on, 'progress' will freeze!

Some of the variables are read-only, meaning that you shouldn't change their values them through the equations. You can; it won’t stop you; but the results are unpredictable.

You can also make your own variables. For example:

```
my_volume = (bass + mid + treb)/3;
zoom = zoom + 0.1*(my_volume - 1);
```

This would make the zoom amount increase when the music is loud, and decrease when the music is quiet. *However*, if you used the variable 'my_volume' in the per-frame equations, you can *not* read that value in the per-pixel equations (see next section)! If you need to precompute some custom values in the per-frame equations for later use in the per-pixel equations, use the following variables:

```
q1
q2
q3
q4
q5
```

For a good example of this, see the 'dynamic swirls' preset.

b. Per_Pixel Equations

So far we've discussed only how to change parameters based on time. What if you wanted to also vary a parameter, such as the zoom amount, in different ways, for different locations on the screen? For example, normally, the result of the 'zoom' parameter is to just do a flat zoom. This doesn't look very realistic, because you don't see any perspective in the zoom. It would be better if we could give a unique zoom amount to each pixel on the screen; we could make the pixels far away from the centre zoom more, and this would give it more perspective. In order to do this, we use "per-pixel" equations, instead of per-frame equations.

The code for this per-pixel equation is simple:

```
zoom = zoom + rad*0.1;
```

Where 'rad' is the radius of the pixel if it were cast into polar coordinates; from another perspective, 'rad' is the distance of the pixel from the centre of the screen. 'rad' is zero at the centre, and 1 at the corners. So if we run the above code, the image will be zoomed into 10% more at the edges of the screen than at the centre.

The per-pixel equations are really just like the per-frame equations, except for current variables. The following variables are available exclusively to per-pixel equations (and not to per-frame equations):

Name	Writeable?	Range	Description
`x`	NO	0..1	Retrieves the x-position of the current pixel. At the very left edge of the screen this would be 0; in the middle, 0.5; and at the right, 1.
`y`	NO	0..1	Retrieves the y-position of the current pixel. At the very top edge of the screen this would be 0; in the middle, 0.5; and at the bottom, 1.
`rad`	NO	0..1	Retrieves the distance of the pixel from the centre of the screen. At the centre of the screen this will be zero, and at the corners, 1. (The middle of the edges will be 0.707 (half of the square root of 2).
`ang`	NO	0..6.28	Retrieves the angle of the current pixel, with respect to the centre of the screen. If the point is to the right of the centre, this is zero; above it, it is PI/2 (1.57); to the left, it is PI (3.14); and below, it is 4.71 (PI*3/2). If it is just a dab below being directly to the right of the centre of the screen, the value will approach 6.28 (PI*2). (note: this is simply the arctangent of y over x, precomputed for you.)

There are also a several variables that are missing for per-pixel equations.

They are:

```
decay,
wave_r,
wave_g,
wave_b,
wave_x,
wave_y,
wave_mystery,
ob_size,
ob_r,
ob_g,
ob_b,
ob_a,
ib_size,
ib_r,
ib_g,
ib_b,
ib_a,
mv_a,
mv_r,
mv_g,
mv_b,
mv_x,
mv_y,
mv_l
```

The main reason for distinction between per-frame and per-pixel equations is simple: SPEED. If you have a per-pixel equation that doesn't make use of the x, y, rad, or ang variables, then there's no reason for it to be executed per-pixel; it could be executed once per frame, and the result would be the same. So, here's a maxim to write on the wall:

“If a per-pixel equation doesn't use at least one of the variables {x, y, rad, ang}, then it should be actually be a per-frame equation.”

You might be wondering how on earth all these formulas could be computed for every pixel on the screen, every frame, and still yield a high frame rate. Well, that's the magic of the hamster. And the fact that it really does the processing only at certain points on the screen, then interpolates the results across the space between the points. In the config panel, the "mesh size" option defines how many points (in X and Y) there are at which the per-pixel equations are actually computed. When you crank this option up, you start eating up CPU cycles rather quickly.

c. Quality Assurance

In order to make sure the presets you create work well on other systems, keep the following in mind:

1. Design your presets using the default mesh size (24x18) option from the config panel, or at least check, before you distribute them, to make sure they look correct at the default mesh size. If your mesh is too coarse (small), then a viewer with the default mesh size might see unexpected "bonus" effects that you might not have intended, and might mess up your preset. If your mesh is too fine, then a viewer with the default might not see all the detail you intended, and it might look bad.

2. Try to design your presets in a 32-bit video mode, so that its brightness levels are standard. The thing to really watch out for is designing your presets in 16-bit colour when the "fix pink/ white colour saturation artefact" checkbox is checked. This checkbox keeps the image extra dark to avoid colour saturation, which is only necessary on some cards, in 16-bit colour. If this is the case for you, and you write a preset, then when you run it on another machine, it might appear insanely bright.

3. Don't underestimate the power of the 'dx' and 'dy' parameters. Some of the best presets a based on using these. If you strip everything out of a preset so that there's no motion at all, then you can use the dx and dy parameters to have precise manual control over the motion. Basically, all the other effects (zoom, warp, rot, etc.) are just complicated abstractions; they could all be simulated by using only {x, y, rad, ang} and {dx, dy}.

4. If you use the 'progress' variable in a preset, make sure you try the preset out with several values for 'Time Between Auto Preset Changes'. The biggest thing to avoid is using something like sin(progress), since the rate at which 'progress' increases can vary drastically from system to system, depending on the user's setting for 'Time Between Auto Preset Changes'.

d. Debugging

One feature that preset authors should definitely be aware of is the variable monitoring feature, which lets you monitor (watch) the value of any per-frame variable you like. First, hit the 'N' key to show the monitor value, which will probably display zero. Then all you have to do is add a line like this to the per-frame equations:

```
monitor = x;
```

where 'x' is the variable or expression you want to monitor. Once you hit CTRL+ENTER to accept the changes, you should see the value of the per-frame variable or expression in the upper-right corner of the screen!

Once again, note that it only works for **per_frame** equations, and NOT for per-pixel equations.

e. Function Reference

Following is a list of the functions supported by the expression evaluator. The list was blatantly ripped from the help box of Justin Frankels' AVS plug-in, since MilkDrop uses the expression evaluator that he wrote.

Format your expressions using a semicolon (;) to delimit between statements. Use parenthesis ['(' and ')'] to denote precedence if you are unsure. The following operators are available:

Operator	Description
`=`	Assign
`+,-,/,*`	Plus, minus, divide, multiply
`|`	Convert to integer, and do bitwise or
`&`	Convert to integer, and do bitwise and
`%`	Convert to integer, and get remainder

The following functions are available:

Function	Description
`int(var)`	Returns the integer value of 'var' (rounds toward zero
`abs(var)`	Returns the absolute value of var
`sin(var)`	Returns the sine of the angle var (expressed in radians)
`cos(var)`	Returns the cosine of the angle var
`tan(var)`	Returns the tangent of the angle var
`asin(var)`	Returns the arcsine of var
`acos(var)`	Returns the arccosine of var
`atan(var)`	Returns the arctangent of var
`sqr(var)`	Returns the square of var
`sqrt(var)`	Returns the square root of var
`pow(var,var2)`	Returns var to the power of var2
`exp(var)`	The same as pow(e, var), where e is Euler's constant (2.7182..., the base of natural logarithms).
`log(var)`	Returns the log base e of var
`log10(var)`	Returns the log base 10 of var
`sign(var)`	Returns the sign of var or 0
`min(var,var2)`	Returns the smallest value
`max(var,var2)`	Returns the greatest value
`sigmoid(var,var2)`	Returns sigmoid function value of x=var (var2=constraint)
`atan2(var,var2)`	Calculates the arctangent of (var/var2)
`rand(var)`	Returns a random integer modulo 'var'; e.g. rand(4) will return 0, 1, 2, or 3.
`band(var,var2)`	Boolean and, returns 1 if var or var2 is != 0
`bor(var,var2)`	Boolean or, returns 1 if var or var2 is != 0
`bnot(var)`	Boolean not, returns 1 if var = = 0 or 0 if var != 0
`if(cond,vartrue,varfalse)`	If condition is nonzero, returns valtrue, otherwise returns valfalse
`equal(var,var2)`	Returns 1 if var = var2, else 0
`above(var,var2)`	Returns 1 if var > var2, else 0
`below(var,var2)`	Returns 1 if var < var2, else 0