# Title: Beginners guide to MilkDrop preset writing

## Version: 1.7

**Last Updated: 4th April 2002**

**Authors: Rovastar & Krash**

**Valued Contributors: Unchained, Geiss**

http://www.milkdrop.co.uk/

*Table of Contents*

## *Introduction*

This guide is been written by some of the more established and experienced preset authors out there (honest!!) to attempt to produce a step-by-step guide/ tutorial to writing MilkDrop presets. It isn't in any real order and is added to all the time.

A useful place to learn more about how existing presets work is the new walkthroughs section. Krash wrote all of these so and are excellent guides for explaining what is going on. They build up slowly and invaluable for explaining what is going off in detail.

It is a guide for the potential budding new preset authors out there who see MilkDrop and think "this is cool I want to get involved but where on earth do I start!" as well as those who have started writing and want to improve their skills.

First off giving a black and white, step-by-step guide is unbelievably difficult because Milkdrop can do so, so much. And all the different variables have to be just right for it to look good. A bit like art in a way as what someone thinks is really good another may think is truly awful.

That said it is not that difficult to create effect that you should be pleased with.

Remember to get the latest versions of MilkDrop from:
http://www.nullsoft.com/free/milkdrop/

And when you have written your brilliant new preset post it to:
http://forums.winamp.com/forumdisplay.php?forumid=84

Also you can post here if you want to know any more information about this guide, suggestions, mistakes, thank you notes, etc. Or email us authors (links in the title page). Postal addresses for the authors available on request for donations and/or gifts. ☺

Also post there for additional assistance with any problems that you might encounter or for suggestions of what we could explain in more detail.

## *Getting Started*

Four things are useful for writing presets: Mathematics knowledge, artistic flare, persistence and luck. If you have any of these you will be able to create a decent preset and the more of each of these you have the better presets will become.

And before you go any further please read Ryan's "MilkDrop Preset Authoring Guide" guide found in the help. There is a lot of useful stuff in there. Print out a copy, if you can, along with this guide it will help when editing your presets on the fly.

If you have not got a clue after reading this then maybe a mathematics textbook will be handy. If you are thinking about maths "Arrrah, let me out!!" then don't worry you can create decent presets with very little mathematics knowledge. You will need to know what the sine (sin), cosine (cos), and (to a lesser extent) tangent (tan) and logarithms (log and log10) equations roughly do. These are used loads in the MilkDrop presets.

But to be fair the more you know the greater your potential of writing a better preset. You can just randomly put things in and possibly get a decent result, but if you actually understand what you're doing with the mathematics, you'll be able to get specific effects with relative ease. A background in programming doesn't go astray either.

## C*olour Cycling*

To start with one of the first things people want to do is to have different colours on the screen. The simplest way to do this is add come simple formulas to the per_frame section. 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);
```

Note: The colour cycling code needs to be in the per_frame section.

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).

Remember that the sine (and cosine) waves have a range of -1 to 1 so if you use these waves with the range 0 to 1 for these effects you will have to modify them.

0.5+ 0.5*sin(time) ≡ 0.5 + (-0.5 to 0.5) ≡ 0 to 1

This will generate the range 0 to 1 (and then back again from 1 to 0, etc) over a period of 6.28 seconds (the value of π*2) for a complete cycle. If you want to speed up the time period (i.e. make the colour changes quicker) then multiple the time variable. E.g. 0.5*sin(2*time). The time period is now 6.28/2 = 3.14 seconds. And to slow it down divide.

If you want the colour variable to be focused on a stricter range than you should alter the equation. For example, to generate a more ‘redder’ image you may want to have the range 0.5 to 1 for the wave_r. Which would require the following equation 0.75 + 0.25*sin(time) ≡ 0.75 + (-0.25 to 0.25) ≡ 0.5 to 1.

Also you can base the colour output on the sound variable (bass, treb_att, etc) to create colours based upon what the music is doing.

*What the variables do – A basic tutorial*

As you will have seen there are hundreds upon hundreds of different effects that can be achieved with MilkDrop. You have probably changed existing presets a bit adding random values here and there to see what happens. What you will find as most beneficial is to start from scratch and create your own new preset. Hopefully this will show you how.

So the initial advice is simple create a blank preset with no zoom, no rot, no warp, no colours, etc. And here is one created for you.

```
[preset00]
fRating=3.000000
fGammaAdj=1.000000
fDecay=0.925000
fVideoEchoZoom=1.006596
fVideoEchoAlpha=0.000000
nVideoEchoOrientation=3
nWaveMode=7
bAdditiveWaves=1
bWaveDots=0
bModWaveAlphaByVolume=1
bMaximizeWaveColor=0
bTexWrap=0
bDarkenCenter=0
bMotionVectorsOn=0
bRedBlueStereo=0
nMotionVectorsX=12
nMotionVectorsY=9
bBrighten=0
bDarken=0
bSolarize=0
bInvert=0
fWaveAlpha=4.099998
fWaveScale=1.285751
fWaveSmoothing=0.630000
fWaveParam=0.000000
fModWaveAlphaStart=0.710000
fModWaveAlphaEnd=1.300000
fWarpAnimSpeed=1.000000
fWarpScale=1.331000
fZoomExponent=1.000000
fShader=0.000000
zoom=0.999514
rot=0.000000
cx=0.500000
cy=0.500000
dx=0.000000
dy=0.000000
warp=0.010000
sx=1.000000
sy=1.000000
wave_r=0.650000
wave_g=0.650000
wave_b=0.650000
wave_x=0.500000
wave_y=0.500000
ob_size=0.500000
ob_r=0.010000
ob_g=0.000000
ob_b=0.000000
ob_a=0.000000
ib_size=0.260000
```

```
ib_r=0.250000
ib_g=0.250000
ib_b=0.250000
ib_a=0.000000
```

Copy this into notepad and save it with a .milk extension in it own directory/folder.

Now run MilkDrop and press F8 to change to this directory.

Press M for the Menu and use keys to navigate.

Using the zoom variable as an example increase and decrease the values to see what the effects are like. As you can see small changes in the zoom equate to quite noticeable effects. The zooms default normal value is 1 and small amounts of zoom are say 0.9 to 1.1 these look reasonable – not too drastic. If you make the zoom 20 it makes little or no difference as making it say 5 as it is out of a sensible range. So it important to realise what the “sensible” ranges for the variables are. Change the zoom yourself to find a nice range and revert to zoom value back to 1.

Now it is time to start to edit the code of the preset. There are 2 sections of this per_frame and per_pixel explanations of these in the Ryan’s guide.

Focus on the per_pixel equation and add the following line.

```
zoom = 1 + 0.1*sin(ang)
```

Remember sine waves have a range of –1 to 1 (ANY value put into a sin equation will return a value from –1 to 1) therefore the range of this equation is 1 + (0.1 to –0.1) which equates to 0.9 to 1.1. The values mentioned previously.

Many authors tend to use multiplication by the reciprocal rather than division when they can as in:

```
Zoom = 1 + 0.1*sin(ang)
```

Rather than

```
Zoom = 1 + sin(ang)/10
```

As it tends to help to clarify intuitively that you're changing the value by a maximum of 0.1. Initially I though that it is down to personal preference but Ryan has let me know that the reason is that actually, division on a PC is very slow (27 clock cycles, vs. 3 for addition/subtraction/multiplication). So now you know the correct reason for avoiding division.

Now you have seen what those equations do you can edit this line to create different effects. Edit the line so it looks like the following examples.

```
zoom = 1 + 0.1*sin(-ang)
zoom = 1 + 0.1*sin(1/ang)
zoom = 1 + 0.1*sin(2*ang)
zoom = 1 + 0.1*sin(3*ang)
zoom = 1 + 0.1*sin(-2*ang)
zoom = 1 + 0.1*sin(ang/2)
```

etc, etc (hopefully you get the picture)

Now change the waveform for each of these to create different effects.

Then try the same equation with rad, x and y instead or ang. And mix these with time.

```
Like 1 + 0.1*sin(time+rad)
```

This will create an effect based upon the sin of the radius, which changes over time, like a ripple spreading inward/outward.

Also try cos instead of sin. Maybe if brave try the tan command.
Multiply by a different amount than 0.1 say by 0.2 or 0.05. Or change the 1 at the start to 0.9 or 1.1 etc.
Try the ‘abs’ command it generates only positive values.
e.g.

```
zoom = 1.1 + abs(0.1*sin(ang))
```

Also try – instead of + in the equations.

Then if you want to go crazy do something like:

```
zoom = 1.12 + sin(-2*ang)/9 - cos(rad + (2*x-y))/11
```

or

```
zoom = 1 + sin(sin(time*rad))/10 - cos(x/y)/30
```

(Try using with waveform 1 for a nice effect.)

See it isn't really that difficult to get a nice effect. Just keep playing at this stage until you feel comfortable with them.

## *Additional per_pixel effects*

Now you have seen the range of things that just the zoom command can do on it own. Now let's move the focus from zoom to other variables. Another main one is 'rot' for the rotation.

Again follow the same steps again for the zoom variable (remove the zoom equation first) but simply change zoom for rot. Again we start with basics from the menus.

You will see that rot is based around 0 for the default value rather than 1 for zoom. So the equations will have to be changed. This is achieved by simply removing the "1 +" at the start of the equations. The reasonable looking range is about –0.1 to 0.1. So the equation will look like:

```
Rot = 0.1*sin(ang)
```

Dx and dy effects are some of the most powerful effects in the MilkDrop but the range of these is very delicate instead of 0.1 as the multiplier use say 0.01 or a range that you are comfortable with. Dx and Dy are so powerful in fact you can use these values alone to create effects like zoom and rot.

Now work out ranges for the other variables MilkDrop provides like sx, sy, etc.

Warp is another effect that can be very tempting to use in the presets but this is strongly discouraged as it is such a powerful effect. It takes away a lot from the intentional effects. So try and avoid.

Now it is time to combine a few of these effects together and add some basic colour cycling back again, as in Ryan's guide, and you should be able create a simple Geiss-like effect now.

## *Brightness control*

The brightness of the waveform is controlled in part by the variable opacity. This is available from the menus and cannot be changed in per_pixel or per_frame sections.

There are 2 states for determining opacity by the volume of the music playing or not by the volume of the music playing.

If the opacity is not generated from the volume then the waveform will always be visible. If it is generated from the volume then if there is no music there will be no waveform displayed.

If you find that the image on the screen appears too dark there are 2 different variables you can change that to achieve greater brightness. The first to look at is decay/sustain.

Decay is sometimes referred to as sustain. In the menu options it is called the sustain value but when referred to in the equation is called decay. But they are the same variable. Decay controls the eventual fade to black.

Although Ryan's documentation states that decay is in a range 0 to 1 it realistically is a range of 0.9 to 1. But beware if you make the decay value at 1 and just below say 0.998 it may eventually turn the entire screen completely white. So this is generally to be avoided always run the preset for a reasonable length of time so ensure that the decay is not too high.

Note: Decay cannot be used in a per_pixel capacity it is only used in the per_frame section.

A high decay effect is not always ideal as it has a tendency to create very 'grey' effects on the screen. To stop this there is another variable that you can edit this is the Gamma.

The gamma is the brightness of the colours on the screen. You cannot change the gamma in either per_frame or per_pixel. Simply change it from the menu options or alter the '.milk' text file.

To be honest adjusting the gamma will make a dramatic change to your preset and is probably the nice and colourful effect that you are looking for but there is a drawback and a major one at that, speed. When you change the gamma from say 1 to 2 is will unfortunately reduce the speed substantially, how much depends on your system and the setting within Milkdrop but it could be in the region of 50%. Sadly it is not something of nothing in this world but that's life.

## *Advanced Effects*

- Video Echo

Video echo is an interesting effect that creates a mirror image of the natural screen display and superimposes it onto the screen. There are 3 factors to generating this effect: opacity of the mirror image (alpha), the size of the 2nd graphics (scale or zoom) and the flip of the mirror image or its orientation.

To get to these effects go to:

Post Processing, Global effects→ Video Echo

Alpha:

This controls the opacity of the 2nd graphics layer. 0 means the video echo effect is off. 1 means the only the mirror image version is displayed. More than likely you will want this at 0.5 - a half and half mix.

Scale (called zoom in the actual preset code):

This controls the size of the second graphics layer. Where 1 is the identical size of the default image. The identical size you might find easiest to work with. 2 is double the size, 0.5 is half the size, etc.

Orientation:

There are 4 different states (well 3 actually) that the video echo can be in:

0 = Normal
1 = Flip the image on the x axis
2 = Flip the image on the y axis
3 = Flip the image on both x and y axis's

Sadly this effect does have a downside in this case it is speed adding video echo may make the preset look "twice as good" but you will lose those valuable frames per second. Experiment with the previous tutorial by adding video echo to witness its effects.

- Borders

Borders were introduced to MilkDrop in version 0.99f.

There are 2 parts to the border effect – the inner border and the outer border. They each have individual variables of size, opacity and red, green & blue colour effects.

Augmentations → Outer border and Inner borders

Size (ob_size/ib_size):

This is the thickness of the outer border drawn at the edges of the screen every frame. The size of both these (in Ryan's documentation states) ranges from 0 to 0.5, making a total of 1 for the entire screen. In fact this is inaccurate as each border has a range from 0 to 1 but you cannot access this from the menus you have to input the values yourself in the per_frame code.

Opacity (ob_a/ib_a):

The opacity of the border where 0=transparent and 1=opaque.

Colour Effects (ob_r, ob_b, ob_g/ ib_r, ib_b, ib_g):

Borders have the same colour control as the waveform. Each made up of the red, blue and green components. There is one difference though, when value of these get above 1 it acts like 0 again so 1.1 is equivalent of 0.1 and 2.232 is the equivalent to 0.232, etc. With the wave waveform colour variables all values above 1 act as 1.

- Motion Vectors

Dynamic Motion Vectors (MV) are the newest feature for MilkDrop and came out in version 1.02. They we used on the previous versions but this is the first version where they have been configurable.

Augmentations → Motion Vectors

Placement (mv_x,mv_y)

This the used to control the amount of motion vectors on the screen. You can have a maximum of 64 vectors on the x direction or horizontally and 48 on the y direction or vertically. You can also use fraction of these values to generate the different placement on the x and y axis.

Opacity (ob_a/ib_a):

The opacity of the motion vectors where 0=transparent and 1=opaque.

Colour Effects (mv_r, mv_b, mv_g):

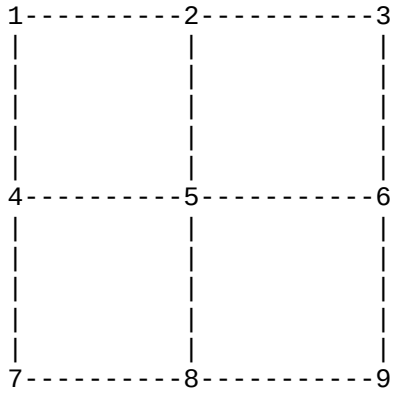
Motion Vectors have the same colour control as the waveform. Each made up of the red, blue and green components. There is one difference though, when value of these get above 1 it acts like 0 again so 1.1 is equivalent of 0.1 and 2.232 is the equivalent to 0.232, etc. (This is the same methods as the borders). With the wave waveform colour variables all values above 1 act as 1.

Length (mv_l)

You can denote the length of these vectors with this. The size of these (in Ryan's documentation states) ranges from 0 to 5. In fact this is inaccurate as motion vectors have an unlimited range (well many thousands) both positive and negative but you cannot access this from the menus you have to input the value yourself in the per_frame code.

## *Per_pixel values explained?*

Here is a (very!) basic diagram showing the positions on the screen and their corresponding X, Y, Ang & Rad values.




| Value | X | Y | Ang | Rad |
|---|---|---|---|---|
| 1 | 0 | 0 | 3π/4 (or 2.356) | 1 |
| 2 | 0.5 | 0 | π/2 (or 1.57) | √2/2 |
| 3 | 1 | 0 | π/4 (or 0.785) | 1 |
| 4 | 0 | 0.5 | π (or 3.1415) | √2/2 |
| 5 | 0.5 | 0.5 | 0 | 0 |
| 6 | 1 | 0.5 | 0 | √2/2 |
| 7 | 0 | 1 | 5π/4 or -3π/4 | 1 |
| 8 | 0.5 | 1 | 3π/2 or -π/2 | √2/2 |
| 9 | 1 | 1 | 7π/4 or -π/4 | 1 |

Where π = 3.1415… and √2/2 = 0.707

Sine and cosine waves are used a lot in MilkDrop in conjunction with x, y, ang and rad. It is important to learn exactly what a sine curve is, and how changes to various parts of the equation will affect that curve. The relationship between sin and cos should be understood - you have to realise that they are essentially the same, but at the same time you need to see that they are out of phase, and you need to know what this means.

The more you get to the limits of what you can do with the sine and cosine, the more you will need understand the behaviour of other curves – tangent (tan) curves, logarithms (log and log10), squares, cubes, exponentials, etc. You need to learn what arcsin, arccos and arctan do too.

All of this will appear in a decent maths textbook. Obviously, you don't need to know all of this for most cases - it's a very rare preset that uses more than one or two of these curves. But you should be able to use them if you need/want to.

In short, you need to have a firm grounding in the basics before you can go getting complicated.

## *Preset Walkthroughs: Introduction*

There was a need for further explanation of practical examples of existing MilkDrop presets. How do you get the screen to do that? And what that ‘that’ mean in the code?

These are a collection of tutorial which gives step by step descriptions various different presets. Hopefully teaching about the various variables and how they are used without making you, the reader, overwhelmed by the complexity.

Starting with the easy ones and will hopefully this section will grow to give documentation of the more complicated examples.

## *Preset Walkthroughs: Tutorial 1 – Approach*

Okay, fire up MilkDrop and load one of the Geiss' original presets called 'Approach' (hit 'L' to bring up the load preset menu). Now turn on scroll lock (on your keyboard) to prevent the preset from changing. You have probably seen this preset before, and you may be disappointed that we're going to be working with something so boring. But this preset utilises some of the simple features that many new authors are unaware of, or confused by, and so a simple example is best. Besides, we have to start somewhere, right?

The first thing we will be doing is checking the static values of the variables. This being the first tutorial, I will walk you through this. Hit 'M' to bring up the preset editing Menu. You can navigate this menu using the arrow keys. Enter (or the right-arrow) will go into the various menu options, and the left-arrow will back you out to earlier menus.

**Static Values** (feel free to skip this section if you know it already)

The first item on the list is '--waveform'. Highlight this option and hit enter (or the right-arrow). You will be shown a new menu. All of the changes you can make to the actual waveform on the screen are done from here.

- *Waveform*

The first item is 'wave type'. Highlight it and press enter. You will see that the current wave type is number 2. MilkDrop has 8 different wave types (all of which are oscilloscopes, as opposed to spectrum analysers), numbered from 0 - 7. Feel free to change the wave type (using the up and down arrow keys) to see what the other types look like. You can save your changes by hitting ctrl-enter, or cancel them by hitting esc (this is true for every setting). When you're done looking at the wave types, change it back to wave type 2, and hit ctrl-enter. You should be brought back to the previous menu.

The next item is 'size'. For most of the wave types, this is fairly self-explanatory. You can change the value in the same way as the wave type, or you can change it in larger increments using Page Up and Page Down. Up will obviously make the wave bigger, and down will make it smaller. You should remember, though, that this value controls the height of the peaks of the wave, not the overall size of the wave. Also, in wave type 0 (the circle), this value DOES NOT make the circle bigger and smaller. It makes the peaks bigger and smaller. This is something to remember.

Smoothing is controlled in the same way - less smoothing means you get sharp angles, more smoothing and you get smooth curves.

Mystery Parameter is, as the name suggests, one of the mysteries of MilkDrop. What this value actually does depends on the currently selected wave type. For wave type 0, this value controls the size of the circle. For the rotating wave type, it controls the radius of the wave's movement. For other types, it controls the angle of the wave. For our currently selected wave type (number 2), it performs no discernible function (if someone does find out what it does for this wave type, let me know).

Opacity is controlled the same way, and changes the transparency of the waveform. Most presets have this set to 1, which is fully opaque.

The Position (X) and Position (Y) settings control the position of the waveform on the screen, as you would expect. X controls its horizontal location; Y controls its vertical location. For both these settings, 0.5 is the centre of the screen. You will notice that in Approach, the wave form is moving around the screen, and yet these values are not changing. This will be explained later, but remember that what we are currently looking at are the STATIC values of these variables.

The colour (R, G, and B) settings control the colour of the waveform, obviously using values of red, blue, and green. These values are normalised, meaning that a value of 0 represents none of that colour, while a value of 1 represents as much of that colour as you can have.

Now we move on to some of the On/Off variables. These can be swapped between their two states either by pressing enter or the right arrow.
Use dots, as it suggests, toggle between the waveform being drawn as lines or dots.

Modulate opacity by volume will turn on the use of the following two variables.

Modulation transparent/opaque volume. These two settings (when the above setting is turned on) will change the opacity of the waveform on a frame-by-frame basis, based on the current volume level. When the volume is below the value of the transparent value, it will be completely invisible. When the volume is between the two settings, the opacity will change to somewhere between 0 and 1 based on a linear relationship between the transparent and opaque volumes and the current volume. When the volume is above the opaque volume, it will be completely opaque.
The values here are representatives of the volume as MilkDrop interprets it. A value of approximately 1 is considered 'normal' volume. Less than 0.8 is 'quiet', and greater than 1.2 is 'loud'. If you wish to experiment with these values, go right ahead. Make sure that 'modulate opacity by volume' is set to ON, or you won't see your changes.

The final two settings will change the appearance of the colours of the waveform. 'Additive drawing' will cause the colour of the waveform to be added to the colour of the pixels over which it is being drawn. What this basically means, is that the centre of the waveform is likely to be very bright white. The second value is 'colour brightening'. When this is set to ON, MilkDrop forces at least one of the colour values to be set to 1, no matter what you do. It means that you can never get any dark colours.

Right, that's it for that menu. Back out of it by hitting the left arrow, and select the next menu option, which is:

❖ *Augmentations*

This is where the border effects can be set, as well as a nifty little tool called motion vectors. I will deal with border effects in a later tutorial, but the motion vectors we will discuss now.

Scroll down and set motion vectors opacity to 1.

You will notice the appearance of a bunch of white dots around the screen with little tails coming off of them. These are designed as a tool to show you exactly what is happening to a pixel at any given point on the screen. You will notice that not all the tails are pointing in the same direction, and also that they move around a bit. The cause for this will be discussed shortly. You can also increase or decrease the density of the motion vectors in both the X and Y directions, if you want to. Turn off the motion vectors when you are done.

You can also alter colour and length of these motion vector for some interesting effects.

That's all for this section just now. Back out and choose the next section, which is:

- *Motion*

This is where things start to get interesting. These settings control what happens to the waveform AFTER it has been drawn. In other words, it's the blending and motion effects that are present in all but a few presets.

The first two settings control zoom. The zoom amount controls how fast things zoom. A value of 1 is no zoom at all. Values higher than 1 are zoom in, while values below 1 zoom out. These values are very sensitive, and a small change can yield a big difference in what you see on screen. The zoom exponent is used to add perspective. This changes the zoom proportional to the distance from the centre of the screen. A value of 1 yields a flat zoom, where all points on the screened are zoomed equally. Higher than one and the outside of the screens zooms faster than the inside. This gives the impression of things racing past you, and gives the impression of depth. Taking the exponent the other way, below 1, yields the opposite effect. When the inside of the screen zooms more than the outside, you end up with effects that resemble balls in the centre of the screen.

Back to the motion menu, the next three settings control warp. Warp is an effect which swirls the contents of the screen around in seemingly random patterns, and is a very overpowering effect. It should be noted that this preset contains no warp, and so you can skip this part if you like.

Increasing the warp amount now will show you exactly what I mean. It doesn't take long before the effect becomes overpowering, and removes any of the zoom which is present. Put the warp back down as far as it will go now.

If you put the warp amount up high enough, you would have noticed the appearance of angled squares with curly corners. These are the functional warp units, and altering the warp scale value will change their size. Warp speed will alter how fast these units morph into each other.

The next three settings control the rotation of the screen. Rotation amount should by now be self-explanatory. A value of 0 is no rotation, positive values are anticlockwise, and negative values are clockwise. Like zoom, this value is very sensitive.

The rotation doesn't have to be centred on the middle of the screen - by changing the rot., centre of (X, and Y) values, you can make the centre of rotation be any point on the screen.

The last four variables in the motion section are actually sub-variables: The zoom, warp, and rotation settings all modify these four variables to different amounts at different points on the screen.

The translation values will shift the screen left, right, up, or down, depending on the values you give them. Try now, if you like, and remember that these variables are probably the most sensitive of all.
The scaling variables cause the screen to be stretched or compressed in the x and y directions. Fiddle with the values, if you're interested.

Right, we're done with motion. The last of the static value menus is:

- *Post Processing, Global Effects*

Okay, the explanation of all these variables is starting to get a bit long-winded, so I'll just say what each setting does, and you can experiment if you like.

Sustain level changes how long it takes for things to fade to black. If the setting is too high, the screen never fades to true black, and stops at some value of grey.

Darken centre draws a small black dot in the centre of the screen, which can help overcome over-saturation in presets with a high sustain value.

Gamma Adjustment changes the overall brightness of the screen. Be careful with this - due to the way it's implemented, the higher the gamma is, the slower MilkDrop presets go.

When the Hue Shader is on, MilkDrop applies a shifting colour filter over the top of everything, which yields subtle variations in colour across the screen.

Video Echo is one of the most commonly used features in MilkDrop. When the Video Echo: Alpha value is set to anything but 0, MilkDrop will draw everything on the screen twice, except that you can modify the second render somewhat. The Alpha setting is essentially the brightness setting of the second render. When it is set to 0.5, you get a half-half mix of the original and second renders, and it looks like it's all the same thing. A value of 1 gives you only the second render. The scale setting changes the relative size of the second render - 2 will be twice the size of the original, 0.5 will be half, etc. The orientation setting changes the way the second render is drawn relative to the original. A value of 0 means the second render is drawn the same way as the original, a value of 1, will flip the second render horizontally, 2 will flip it vertically, and 3 will flip horizontally and vertically.

Texture wrap will cause things going off one side of the screen to reappear on the other - provided the motion effects allow it.

Stereo 3D will turn on stereoscopic mode.

The filters will alter the colours on the screen in some way. These aren't used very often.

Okay, we're finally at the end of the static values section. I won't be repeating this in future tutorials unless the preset specifically uses them.

**Per-Frame Equations**

Now we're getting into the interesting part of preset editing. Everything we've looked at so far is persistent, unchanging, and generally pretty boring to watch. Here, we add the good bits.

Go back to the main menu (left arrow again), and select 'edit per-frame equations'. You should be greeted with a couple line of mathematical equations, as shown:

```
wave_x = wave_x + 0.150*( 0.60*sin(2.121*time) + 0.40*sin(1.621*time));
wave_y = wave_y + 0.150*( 0.60*sin(1.742*time) + 0.40*sin(2.322*time));
wave_r = wave_r + 0.200*( 0.60*sin(0.823*time) + 0.40*sin(0.916*time));
wave_g = wave_g + 0.500*( 0.60*sin(0.900*time) + 0.40*sin(1.023*time));
wave_b = wave_b + 0.500*( 0.60*sin(0.808*time) + 0.40*sin(0.949*time));
rot = rot + 0.002*sin(time+0.073);
decay = decay - 0.03*equal(frame%30,0);
```

I'm going to take you through these equations one by one, and explain what they are doing.

- *Moving the Waveform* - wave_x and wave_y

```
wave_x = wave_x + 0.150*( 0.60*sin(2.121*time) + 0.40*sin(1.621*time) );
wave_y = wave_y + 0.150*( 0.60*sin(1.742*time) + 0.40*sin(2.322*time) );
```

These are the first two line of per-frame code in the Approach preset. You can see at a glance that they are altering the x and y position of the waveform, because the lines begin with 'wave_x =' and 'wave_y =' respectively. Wave_x and wave_y are the same as the Position (X) and (Y) settings back in the waveform menu. From here, I'll just talk about the first line, but the same applies for the second.

The first part means that we are going to be changing the x co-ordinate of the waveform that will be drawn during the current frame.
The next part of the equation is wave_x again. When wave_x is written anywhere to the right of the equals sign, it is referring to the current value of the variable. At the beginning of each frame, MilkDrop takes this value from the one in the menus. If you remember, this is set to 0.5. So what 'wave_x = wave_x + ...' actually means is 'wave_x = 0.5 + ...'.

The next part of the equation is '0.150*(some stuff in the brackets)'. Inside the brackets, you will see two sin functions (sin is short for sine). If you don't know, or don't remember from school, sine of a number X can be anywhere from -1 to 1, and as

X increases, the sine of X will smoothly oscillate between those two extremes. In MilkDrop, sine of X is written as sin(X).

You will notice that inside the sin functions, we have a number multiplied by time. Time is a variable within MilkDrop which changes with time. Every millisecond, 0.001 is added to the time variable, meaning that time goes up by 1 every second. This is the changing value of X from the previous example, and so sin(2.121*time) will oscillate between -1 and 1. The 2.121 is there simply to keep things seemingly irregular. You will notice that the number multiplying time is different in every instance.

Now, both of the sin functions will oscillate between -1 and 1. You will notice that one of theses is multiplied by 0.6, and the other by 0.4. Therefore, they will now be oscillating between -0.6 and 0.6, and -0.4 and 0.4 respectively. When these two functions are added together, we're back to oscillating between -1 and 1 again (as it is conceivable that both sin function reach their maximum at the same time). However, because we now have two sin function being added together, and these functions are out of phase (different numbers multiplying time), Our result will not go straight from -1 to 1 and back again, instead it will wobble around a bit in the middle, first.

When we multiply all this by 0.150, it means that our end result will oscillate from -0.15 to 0.15.

So if we look at the full equation again:

'wave_x = wave_x + 0.150*(...);'
becomes:
'wave_x = wave_x + ;'
but wave_x on the right hand side is actually 0.5, so we get:
'wave_x = 0.5 + ;'
therefore wave_x equals a number between 0.35 (0.5 - 0.15) and 0.65 (0.5 + 0.15), which changes over time. If you look at the waveform moving around, you will see that that is exactly what it is doing.

If you like, change some of the numbers in these two lines, hit ctrl-enter to make your changes take effect, and observe the results.

- *Changing Colours* - wave_r, _g, and _b

You will notice that the equations for wave_r, wave_g, and wave_b look very similar to the wave_x and wave_y equations. That's because, in this instance, they are.

Let me take the wave_r equation as an example. The others work the same way.

```
wave_r = wave_r + 0.200*( 0.60*sin(0.823*time) + 0.40*sin(0.916*time) );'
```

it works the same way as the x and y equations did. The value for red in the menus is set to 0.7, so this equation can be simplified to:

'wave_r = 0.7 + ;'

Remember that the whole section in the brackets there is oscillating between -1 and 1, and now it's being multiplied by 0.2. The end result is that wave_r is changing to values somewhere between 0.5 and 0.9.

The other two colours are set to 0.3 in the menus, and the equations here have a multiplier of 0.5. This means that their values are changing from -0.2 to 0.2. But what about the time when the value is below 0? Simply put, nothing happens. MilkDrop treat a colour value of less than 0 as simply 0, meaning no colour.

All the colour equations have different numbers multiplying the time variable, which makes them cycle their values at different rates, which is why you don't notice any repetition in the colour cycling.

If you ever want to make a preset where the waveform cycles colours, it's by far easiest to simply copy some equations from another preset, and change the numbers as you see fit.

Right, now we're on to the next equation.

- *Rotation* - using the rot variable

If you haven't noticed by now, the Approach preset rotates very slightly backwards and forwards. This rotation is controlled by the following equation:

```
rot = rot + 0.002*sin(time+0.073);
```

You should have guessed by now that the variable in these equations for the rotation is rot, and that this variable is the same as the 'rotation amount' setting in the motion menu.

Breaking this equation apart, we have two parts, the rot variable, and the sin function. As before, the rot variable is being sourced from the menus, which you may remember is set to 0. This means that the inclusion of this part of the equation is essentially useless.

The sin function (with the multiplier) yields an oscillation between -0.002 and 0.002. You will recall I said that rotation was very sensitive. The inside part of the sin function also has a pointless part. That +0.073 will make absolutely no difference to the end product of the equation. The time variable is increasing all the time, and if you've had MilkDrop running the whole time you've been following this tutorial, the time variable will be at several thousand by now. An addition of 0.073 will make absolutely no difference at all. take it out and see for yourself, if you like.

The end result of the rot equation is therefore cycling the rotation from -0.002 to 0.002, and back again. If you like, change the 0.002 to a larger number, and you will notice the rotation much more. If you get too big, though, you won't notice.

That leaves us with one more equation to cover.

- *Decay* - also known as sustain

The line beginning with 'decay =' is actually altering the ‘sustain’ level variable we found back in post processing. It's the same variable; it's just called decay here. Don't ask why, I don't know.

Here is the equation:

```
decay = decay - 0.03*equal(frame%30,0);
```

The decay to the right of the equals sign is, like the other equations, referencing the value in the menus, which is 0.98. The second part contains something interesting. The equal function is something you can use to determine whether two numbers are equal. In the brackets, there are two numbers of functions separated by a comma. If they are equal, the result of the equal function is 1. If they aren't, the result is 0.
So, you can read the above equation as: 'when the two numbers inside the brackets are equal, subtract 0.03 (0.03 * 1) from the decay, otherwise, subtract 0 (0.03 * 0).

But now, you will look inside the brackets, and wonder what is going on. Firstly, frame is a variable like time, which increases all the time. The difference is that frame is increased every drawn frame, rather than after a certain amount of time. The %30 part is sort of like dividing by 30, except it only takes the remainder. So, for example, if frame was 35, frame%30 would be 5.
So what the equal function is actually doing, is checking to see if the frame number is a multiple of 30. If it is, then there will be no remainder when it is divided by 30, and so the two numbers in the brackets will be equal, and the function will equal 1.

So we can simplify the whole equation by saying: 'Every 30 frames, make decay 0.03 less than it is normally.'

In real terms, you can't actually notice this difference. But if you change the 0.03 to a much larger number, like 0.5, you will see that every 30 frames, pretty much everything on the screen disappears.

**Conclusion**

Okay, that covers everything in the Approach preset very thoroughly, and hopefully, you've come out of the tutorial understanding what's going on. At the very least, you should now be capable of creating a preset in which the waveform moves around and changes colours smoothly.

If you're creative, you'll figure out that many of the other variables (zoom, warp, etc) can be treated in the exact same way. You won't be generating any award-winning presets this way, but you should be on your way to understanding them.

I apologise that this was such a long-winded tutorial, but it being the first, I thought it somewhat necessary to make sure I covered everything from scratch. The next tutorials should be less lengthy, and as I go on, they will probably get shorter and shorter, until it's a quick explanation of a couple of important lines of equations.

## *Preset Walkthroughs: Tutorial 2 – Tornado*

Okay, get MilkDrop up and running, and load up the preset 'Tornado', again one of Geiss' own presets. Looks pretty cool, doesn't it? The screen twisting around like that is a very simple effect to produce, and yet can make quite an impact. But we're not quite up to that, yet - we'll start at the beginning.

**Static Values**

If you scan through the settings of all the static values, you will notice that they are very similar to the settings for 'Approach'. The wave type is the same, although the size and smoothing have been altered slightly. Colour brightening has been set to ON, and the only other major difference is the static settings of the waveform colour - which is set to 0.6 for all three components.

Under motion, there are a few minor changes. Zoom has been set to 1.031 - this value is greater than 1, meaning that we get a zooming in effect. The zoom exponent is set to 2.1 - a number which makes the outside of the screen zoom faster than the inside, giving the preset some perspective.
You will also observe that we have some warping effect in this preset - it's not significant, with the warp amount being only 0.309, but it adds a subtle effect, when the screen isn't twisting. The reasonably large warp scale makes the effect harder to notice.
You will also notice that despite the fact that the screen is rotating around, the rotation is set to 0 - this is because the rotation is handled in the per-* sections, which we'll talk about now.

**Per-Frame Equations**

Here is the list of per-frame equations for 'Tornado':

```
wave_r = wave_r + 0.400*( 0.60*sin(0.933*time) + 0.40*sin(1.045*time) );
wave_g = wave_g + 0.400*( 0.60*sin(0.900*time) + 0.40*sin(0.956*time) );
wave_b = wave_b + 0.400*( 0.60*sin(0.910*time) + 0.40*sin(0.920*time) );
zoom = zoom + 0.023*( 0.60*sin(0.339*time) + 0.40*sin(0.276*time) );
rot = rot + 0.030*( 0.60*sin(0.381*time) + 0.40*sin(0.579*time) );
decay = decay - 0.01* equal(frame%6,0);
```

If you've read tutorial #1, these should look familiar, because they have a lot in common with those equations.

This time, we don't have any equations controlling the position of the waveform (these would begin with wave_x or wave_y). We do have equations controlling the colour though. Here is one of them:

```
wave_r = wave_r + 0.400*( 0.60*sin(0.933*time) + 0.40*sin(1.045*time) );
```

You should remember that the 'wave_r' to the right of the equals sign refers to the value of wave_r in the menus. You should also remember that the part inside the brackets basically means 'a number somewhere between -1 and 1'. When this is multiplied by the 0.4 outside the brackets, the equation can be summarised like so:

'make the red component of the waveform equal to the red value in the menus plus some number between -0.4 and 0.4'. A quick calculation tells us that the wave_r value is therefore moving around somewhere between 0.2 and 1. The same goes for all the other colour equations.

The zoom and rot equations have the same structure. The only difference is the value they are referring to from the menus, and the range of values that they add this to.

For example, the zoom equation there is taking the zoom amount of 1.031 from the menus, and adding some number between -0.023 and 0.023. This means that the zoom value is anywhere from 1.008 to 1.054. We can see from this that the zoom is always zooming in, but sometimes more, and sometimes less.

The rot equation is the same, and the decay equation is similar to the one in 'Approach'

But now, we're on to the best, and most versatile part of all preset editing:

**Per-Pixel Equations**

Per-pixel equations are a series of equations which are calculated once for every pixel on the screen (they aren't really, but you can treat them like they are). If you write the equations correctly, you can get different effects in different areas of the screen. In 'Tornado', you can see that on beats, the inside of the screen rotates clockwise, while the outside rotates anti-clockwise. This is done through Per-Pixel Equations.

❖ *Pixel Reference Variables*

Each pixel on the screen has four variables that we can use to reference it, and it's necessary for you to at least be aware of them.

X: This variable refers to a pixel's distance horizontally across the screen. A value of 0 means all the way on the left hand side, and a value of 1 is all the way to the right.

Y: Same as X, except this one is vertical. The top is 0, the bottom is 1.

rad: This variable is a pixel's distance, in a straight line, from the centre of the screen. A value of 1 is all the way to the corners. The edges of the screen are a bit below 1 (if you're running MilkDrop windowed, this depends on the size and shape of the window).

ang: This one confuses a lot of people. Here is the simplest way to think about it: Imagine a line from the centre of the screen, out to the right. Now imagine another, drawn from the pixel in question to the centre of the screen. The ang variable is the angle between these two lines. There's a catch, though. Ang is expressed in radians, not degrees. 180 degrees equals pi radians. If your pixel is below the horizontal halfway mark, it will have an ang value somewhere between -pi and 0. Above halfway, and ang will be between 0 and pi. If you don't follow that explanation, I suggest you consult a decent maths textbook - you will find a far more in-depth explanation than I am able to give here.

Right, now that that's sorted, we can get to the per-pixel equations in 'Tornado'.

❖ *Breaking Down the Equation*

Go to the per-pixel equations section, and you will notice that there is only a single equation, though it might seem imposing. I'm going to explain it piece by piece.

```
rot = rot + (rad-0.4)*1.7*max(0,min((bass_att-1.1)*1.5,5));
```

Okay the first part: 'rot =...'
First off, we can see that the equation will be altering the rotation effect. And because this is a per-pixel equation, it's going to be altered differently for every pixel.

Next: '= rot +...'
Normally, this 'rot' would refer to the rot value back in the menus. But rot has already been altered back in the per-frame equations, and so this rot is the result of that per-frame equation.

'(rad-0.4)*...'
This is our multiplier for what is to follow. You can see that it will be a different number for different values of rad (which are owned by different pixels). When rad is above 0.4, we will have a positive value, and otherwise, the value will be negative.

'1.7*max(0,min((bass_att-1.1)*1.5,5));'
Whoa! That's a mouthful. It needs to be broken up a bit further. Let's look at the bit inside all those brackets first.

'bass_att-1.1'
Bass_att is one of the six variables we can use to detect what's happening in the music. It is the percentage difference between the amount of bass in this frame and the amount of bass last frame. A loud bass tone will often cause the value of bass to be 150% or more what it was previously. This would be a bass_att value of 1.5.
The above statement will obviously be above 0 if the bass_att is higher than 1.1, and negative if it is lower. This is the core of the music response you see in this preset.
The other five variables that we can use to get information about the music are mid_att, treb_att (which both work the same as bass_att), and bass, mid, and treb (which are the instantaneous values of bass, middle, and treble sounds)

'min((bass_att-1.1)*1.5,5)'
The minimum function selects whichever of the two numbers in the brackets is smallest. So, in this case, if (bass_att-1.1)*1.5 is less than 5, this value will be kept (to do this, bass_att would have to be less than 4.4). Otherwise, the value kept will be 5.

'1.7*max(0,*everything above*);'
The max function is the same as the min function, except it retains the larger number. Here, we get a value of zero if the result of the previously discussed part is negative. Otherwise, we get the previous result. This is then multiplied by 1.7.

By looking at the past couple paragraphs, you should be able to follow through the

fact that if bass_att is below 1.1, the value we are adding to rot (at any pixel) is 0 i.e. - no change. When bass_att is above this value, the '1.7*max(...);' part will equal a number greater than 0, and so the rotation will be changed. The amount it is changed will differ at various parts of the screen, due to the 'rad-0.4' component.

**Conclusion**

Although that's a lot of different variables and concepts to introduce in a single equation, this guide should have explained to you how the preset has been constructed, in particular, the per-pixel equation. If you like, change some of the numbers in the equation, or swap the 'rad' for a 'x', 'y', or 'ang'. You could also exchange the 'bass_att' for one of the other music response variables.

Hopefully, you now understand what is happening behind the scenes in 'Tornado'. With some experimentation, you should now be capable of putting together some presets that are on par with those that come with MilkDrop. Just try stuff, and see what happens. Learning is best accomplished through experimentation.

## *Preset Walkthroughs: Tutorial 3 – Cruizin'*

As I get further into these tutorials, I will be going into less and less detail. I'm only going to be covering things that we haven't seen before, in order to reduce the length of the tutorials. So if you feel lost at any point, go back and read the previous tutorials - hopefully they will fill you in.

This time, I will be covering the preset 'Cruizin'. It's not the most interesting of presets to watch, but it's a simple example of a couple concepts that I need to introduce. The mathematics involved in these presets is something which you should be able to follow. If you can't it's not my job to teach and I suggest you pay more attention in school. (EDIT: Or '*should*' have paid more attention in school for those of us over 18's - Rovastar)

**Static Values**

By this stage, you should be perfectly capable of going through the menus on your own. Most presets don't have anything particularly special set in the menus, but a setting of the mystery parameter or position variables can often be the reason your preset doesn't look quite how you intended. I'll leave you to investigate the static values in this preset. I will point out one thing, though - despite the appearance of this preset, there is NO zoom effect involved at all. The zooming motion is performed in the per-frame and per-pixel components.

**Per-Frame Equations**

You should be familiar with most of the per-frame equations in 'Cruizin'. You will notice that there is a line involving rot:

```
rot = rot + 0.004*( 0.60*sin(0.381*time) + 0.40*sin(0.579*time) );
```

The fact that the multiplier in this line is so small (-0.004), shows us that it is a very subtle effect indeed. In fact, it is hardly noticeable. After watching the preset for a while, you may perceive that the movement is not always straight down, but sometimes swings to the left or right. This effect, believe it or not, is a result of the rotation equation above, as well as some per-pixel code I will discuss later. The power behind this shifting actually comes from the two lines below:

```
cx = cx + 0.110*( 0.60*sin(0.374*time) + 0.40*sin(0.294*time) );
cy = cy + 0.110*( 0.60*sin(0.393*time) + 0.40*sin(0.223*time) );
```

Now, 'cx' and 'cy' are two variables we have not seen before. The 'c' stands for centre, and these variables control the centre of rotation, that can be found in the 'motion' menu. By checking the values in the menu, we can see that the cx value is changing between 0.39 and 0.61. The cy value, on the other hand, is changing between 0 and 0.22. Remember back to tutorial #2, where I explained about the x and y references of the screen. We therefore understand that the centre of rotation is hovering around the middle of the top quarter of the screen.
When the values are only changing by 0.11, it can be difficult to notice the difference - 'Cruizin' is a very subtle preset. Try putting the multiplier of the cx line up to 0.5 - the shifting centre of rotation is now much more noticeable.

**Per-Pixel Equations**

Now we're on to the Per-Pixel equations. In this preset, there are two parts of the per-pixel equations which I need to cover, and I'll cover these as separately as possible.

❖ *Shifting the Movement* - dx and dy

First, take a look at the LAST two lines of the per-pixel equations.

```
dx = q*du;
dy = q*dv;
```

The variables 'dx' and 'dy' correspond to the values in the motion menu for 'translation'. The physics term for 'translation' is 'displacement', and that is what the 'd' stands for in these variables.
I have mentioned previously that the warp, rot, and zoom effects are actually just combinations of shifting and stretching. This is true, but you must be aware that 'dx' does NOT equal the total x shift of the other three motions combined. The 'dx' variable is a completely separate value, which is added to the other motions at a later stage.

That said, you will notice other things in the above equations that I haven't mentioned before - the variables q, du, and dv.

❖ *User-Defined Variables*

So, what are these three variables? What is their corresponding entry in the menus? What do they do?

The answers to those questions are: Just numbers, they don't have one, and whatever you want them to do.

These variables were created to make the code simpler to work with, and make the preset easier to understand. They were created in the following lines of equations:

```
du = (x-cx)*2;
dv = (y-cy)*2;
q = 0.01*pow(du*du + dv*dv,1.5);
```

So how do you define your own variable? Simple - just write an equation for it. If you write an equation that begins with 'temp =', then temp will equal whatever the result of the equation is, and you can refer to temp in other equations. Your variable can be anything you like, as long as it starts with a letter, and it isn't one of MilkDrop's 'reserved' variable names. The only reserved variables that you might use by accident are 'q1' through 'q5'. I will explain how these are used in a later tutorial.

Back to the preset in question.
You can see, first of all, that du is being used to refer to things to do with x, and dv is used for y. Specifically, they are being set to twice the difference between the centre of rotation and the current pixel (remember that we're talking about per-pixel equations here). The range of possible x values is known (0..1), as is the range of cx

values (remember they were calculated in the per-frame equations). We can therefore find the range of values for du (-1.22 to 1.22), and dv (-0.44 to 2).

These values are then put into our equation for q (another User-Defined variable)

```
q = 0.01*pow(du*du+dv*dv,1.5);
```

The pow function is one that we haven't seen yet, but it is a function to find a power of a number (like square, or cubed, or whatever). In this case, the square of du (du*du) is added to the square of dv, and this is raised to a power of 1.5. You can see that the power is the number within the brackets, after the comma. The result is then multiplied by 0.01.

As an example of how this equation works, I'll take the maximum values of du and dv when x is 0 and y is 1 (the bottom-right corner).
The value for du will be -0.78 ((0 - 0.39)*2), and for dv it will be 2 ((1 - 0)*2). When these values are substituted into the equation for q, we get:

q = 0.01*pow(-0.78*-0.78 + 2*2, 1.5);
Which is the same as
q = 0.01*pow(0.6084 + 4, 1.5);
Simplified to
q = 0.01 * 9.8929

I will point out that taking a number to the power 1.5 is the same as adding that number to its square root. Which means the above equation bears a resemblance to Pythagoras' Theorem of right-angled triangles (square root of the sum of squares), and therefore has something to do with the straight-line distance from our pixel in question to cx and cy (remember that du and dv are dependant on cx and cy).

So at this point, at this particular time, q is approximately 0.1.
When this value (as well as the values for du and dv) is substituted into the equations for dx and dy, we get:

dx = 0.1 * -0.78 = -0.078
dy = 0.1 * 2 = 0.2

So the pixels at that point are moving the left a little bit (-0.078), and down reasonably fast (0.2). This is only the movement at the point I have used for my example. If you have the patience to work things out, you will see that these values will get smaller as the x and y co-ordinates you test get closer to cx and cy. This means that things move slowly when they are close to cx and cy, and fast when they are further away. This is a similar effect to simply using the zoom exponent in the menus, except we now have a greater degree of control as to exactly how things work.

**Conclusion**

You should have been able to follow how the shifting movement in 'Cruizin' has been achieved. The mathematics involved in this process is slightly more complicated than what we've seen in previous tutorials again, and I make no apologies for it. The equations we have used here are probably very similar to the internal equations MilkDrop performs when you ask it to do a standard zoom.

More important than simply understanding this preset, you should also understand how user-defined variables work, and how you can use them. One particularly useful aspect of these variables is that they are persistent from one frame to the next (as opposed to the menu-based variables, which are reset every frame. I'll leave you to discover ways in which you can utilise this feature.

## *Preset Walkthroughs: Tutorial 4 – Shift*

As per the last tutorials, I'm going into less and less detail - by the time you've read and understood the last three tutorials, you should be able to follow this one no problem.

The fourth tutorial covers the preset "Shift". This preset introduces a commonly used method of beat detection, which can be easily transported across into other presets.

This time, I'm going straight into the interesting and complicated stuff. A vast majority of code in this preset has been covered before.

**Per-Frame Equations**

```
dx = dx + dx_residual;
dy = dy + dy_residual;
bass_thresh = above(bass_att,bass_thresh)*2 + (1-
above(bass_att,bass_thresh))*((bass_thresh-1.3)*0.96+1.3);
dx_residual = equal(bass_thresh,2)*0.016*sin(time*7) + (1-
equal(bass_thresh,2))*dx_residual;
dy_residual = equal(bass_thresh,2)*0.012*sin(time*9) + (1-
equal(bass_thresh,2))*dy_residual;
```

These are the parts of the preset which we're concerned with in this tutorial. The rest of the preset is simple colour cycling or movement code that's similar to what we've seen before.

From these 5 lines, a couple things should jump into view. Firstly, you can see that we're declaring three user-variables (as described in tutorial #3). These variables are bass_thresh, dx_residual, and dy_residual. The second thing you should notice is that we're using dx_residual and dy_residual in equations *before* we actually declare the variables. From previous experience, you would know that variables are restored to their menu-determined values at the beginning of every frame. So what happens with user variables? Wouldn't they get set to zero?

No, actually. While MilkDrop's internal variables (dx, dy, zoom, etc) are reset every frame, user variables are not. User defined variables retain the value that they had at the end of the previous frame. So when we have a line

```
dx = dx + dx_residual;
```

we're actually talking about the value of dx_residual that we got the last time we calculated it.

On to explaining the code!

```
bass_thresh = above(bass_att,bass_thresh)*2 + (1-
above(bass_att,bass_thresh))*((bass_thresh-1.3)*0.96+1.3);
```

You'd look at this line of code, and think it is exceedingly complicated. I wouldn't blame you. But like all things, breaking it up into its component parts can make the complex become simple.

The first way to simplify the equation is like this:

```
bass_thresh = above(bass_att,bass_thresh)*2 + (1-
above(bass_att,bass_thresh))* (stuff);
```

The first term is determining whether or not the current bass_att is greater than the bass_thresh value that was calculated the previous frame. If this is the case, then the "above()" statement equals one, which is multiplied by two to give two. If bass_att is equal to or below bass_thresh, the result is zero, and so this term is nullified.

By looking at the second term, you should see that it will yield an opposite result - the "above()" statement is the same, and will yield the same result, but the result is subtracted from one. So if the bass_att is above bass_thresh, then the result will be zero, multiplied by (stuff), to give 0. When the opposite is true, we will end up with 1*(stuff).

If you think that this sounds like a complicated way of doing an if statment, you'd be partly right. This is essentially an if statement, but is more simple for the computer to interpret and execute, producing an increase in speed. If I were to write it as an if statement, it would look like this:

IF bass_att is greater than bass_thresh
THEN set bass_thresh to 2
ELSE do (stuff).
ENDIF

Now, we need to focus on what that (stuff) part does. I've reproduced it here:

(bass_thresh - 1.3)*0.96 + 1.3

The easiest way to start thinking about this, is to imagine that bass_thresh is currently 2 (i.e, bass_att was greater than bass_thresh on the previous frame).

Then we get this:

(2 - 1.3)*0.96 + 1.3 which becomes
0.7*0.96 + 1.3

The result of which is 1.972. So after passing through the equation once, bass_thresh has been lowered a little. If you go through again and again, bass_thresh will continue to get lower, but it will never drop below 1.3. Why? because the 1.3's in the above equation define the lower limit for the variable. A plain english version of the equation is:

"Take 96% of the difference between bass_thresh and 1.3, and add it to 1.3"

This generates an exponential/log curve, for those mathematically minded among you. This means that the bass_thresh value will start at a point (in this case, 2), and drop. The value will drop relatively quickly at first, but will slow down; eventually getting

so close to 1.3 that it IS 1.3.

So what happens then? Well, at some point, the bass_att (which is the percentage of the bass in this frame relative to the last frame) will be above our threshold, and the threshold will be set to 2 again. Incidentally, this is the same way that MilkDrop detects beats for its "Hard Cuts".

So all this is well and good, but what does this actually do for us?

Well, now we have a variable that is equal to a known value (2) whenever there is a significant increase in the level of bass (commonly a beat in the music). This is used to calculate the dx_ and dy_residual values.

```
dx_residual = equal(bass_thresh,2)*0.016*sin(time*7) + (1-
equal(bass_thresh,2))*dx_residual
```

Once again, you can see that this is like an if statement as used above, and writing it as an if statement is the bestway to explain it

IF bass_thresh = 2 (i.e. we have just detected a beat)
THEN set dx_residual to 0.016*sin(time*7) (which is essentially a random number between -0.016 and 0.016)
ELSE keep dx_residual the same (multiplying itself by 1)
ENDIF

The dy_residual equation works in the same way. The three lines of code are therefore generating two random numbers between -0.016 and 0.016 every time we detect a beat.

Going back to the actual declaration of dx and dy, we can now see that all we're doing is adding the calculated values of dx_ and dy_residual to the values of dx and dy set in the menus. In Shift, these values are zero, and so the line could simply read "dx = dx_residual", and it would make no difference. Left as it is, you can easily change the values in the menus to change the effect slightly.

**Conclusion**

Hopefully, you've been able to follow through the processes involved in Shift. This method of beat detection is quite adequate for most uses, and is very easy to port over to other presets for use there. The _thresh method of detection has found a wide range of uses, from intelligent colour cycling, right up to modulated beat detection. It's an extremely versatile tool, and understanding how it operates, and how changes will affect the outcome, is key to developing music-reactive presets with the effects you intend.