A Flickernoise handbook

VJing with Milkymist

October 25, 2011

1 Getting started

2 Interfaces and controls

2.1 Keyboard controls

2.2 Infrared remote control

2.3 MIDI controls

MIDI messages received through OpenSoundControl are merged with messages coming from the MIDI interface.

2.4 Using DMX512

2.5 Using OpenSoundControl

3 Authoring patches

This chapter is inspired by the MilkDrop guides from Geiss, Rovastar and Krash.

3.1 About patches

A "patch" is a collection of parameters that tell Flickernoise how to draw the wave, how to warp the image around, and so on. Flickernoise ships with dozens of patches, each one having a distinct look and feel to it.

Each patch is saved as a file with the ".fnp" extension, so you can easily send them to your friends or post them on the web. You can also go to `http://www.milkymist.org` to see what other people have come up with, or post your own cool, new patches.

The Flickernoise rendering engine largely draws ideas from MilkDrop, an audio visualization plug-in for the Winamp music player. In fact, many MilkDrop effects (called "presets") can be made to work with Flickernoise, sometimes with minor modifications only.

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

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 patches with very little mathematics knowledge. You will need to know basic operations and what the sine (sin) and cosine (cos) equations roughly do. These are used loads in the Flickernoise patches.

But to be fair the more you know the greater your potential of writing a better patch. 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.

3.2 Drawing a wave

One of the basic elements in a patch is the *wave*, which is a graphical representation of the audio signal much like that of an oscilloscope.

To draw a wave, all that one has to do is select one of the eight possible wave modes, numbered from 0 to 7. This is done by setting the value of the wave_mode parameter.

Open the patch editor, and enter the text:

```
wave_mode=6
```

Flickernoise also recognizes nWaveMode instead of wave_mode in order to make the porting of MilkDrop presets easier.

Then, start the rendering by clicking the Run button or pressing the F8 key of the keyboard. Put on some music in order to get a signal to trace other than a straight line. Feel free to experiment with the other wave modes. Use the Enter key or a left mouse click to terminate the renderer and get back to the patch editor.

3.3 Decay

In the previous example, the wave is repeatedly drawn over itself at each frame, without the screen being cleared. This does not look very nice, and after a while nothing happens anymore as the whole wave drawing area becomes all white.

A solution to this problem comes with the "decay" feature of the renderer. What it does is it fades at each frame all the colors to black. It is controlled by a variable names "decay". If this variable is 1 (default), no decay is applied (100% brightness). If it is 0 (0% brightness), the screen is completely filled with black before drawing the next frame. Intermediate values cover the whole range from 0 to 1. For example a value of 0.99 will darken the image by 1% at each frame:

```
wave_mode=6
decay=0.99
```

Flickernoise also groks fDecay from MilkDrop.

This gives a nice effect, with the previous traces remaining visible for a while and slowly decaying.

Even though the decay variable is theoretically in the range 0 to 1 it realistically is a range of 0.9 to 1 in most cases. But beware, if you make the decay value at 1 (or very close to), it may eventually turn the entire screen completely white. This is generally to be avoided; always run the patch for a reasonable length of time so ensure that the decay is not too high.

3.4 Motion

3.4.1 Zoom

A popular effect is to make the picture zooming. This is achieved using a variable adequately named “zoom”, which you can access like any other variable by adding a line such as:

```
zoom=1.05
```

If this value is 1.0 (default), 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.

Increase and decrease the value of the zoom variable 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 (you can simply delete the line that affects a value to the "zoom" variable).

3.4.2 Displacement

It is also possible to translate the image at every frame. This is done with the dx and dy variables, which affect horizontal and vertical motion respectively.

These variables are in the range -1 to 1, and have the default value 0. They are expressed in the fraction of the screen to move at each frame. For example, setting the variable dy to 0.01 will move the screen by 1% at each frame. Since there are typically 24 frames per second, these variables are very sensitive and you'll often want to keep their values small and close to 0.

For example, try and experiment with the following patch (the wave_x variable controls the position of the wave on the

screen):

```
wave_mode=6
dy=0.01
wave_x=0.9
```

3.4.3 Rotation

Similarly, you can make the picture rotate using a variable called "rot". It is expressed in radians per frame (1 radian $\approx$ 57.3 degrees). The value can be positive or negative, and the sign defines the direction of rotation. Since the rotation is applied at each frame, any non-zero value in "rot" will make the picture continuously rotate.

The center of rotation is defined by the "cx" and "cy" variables, whose values are 0.5 by default which corresponds to the center of the screen.

3.4.4 Scaling

Scaling works like the zoom effect, but can be controlled independently in horizontal and vertical directions. This is acheived with the "sx" and "sy" variables (respectively). The meaning of their values is the same as for the zoom effect.

Another difference between scaling and zoom is that the zoom effect will always zoom the center of the picture, while scaling will do the same using the point defined by the "cx" and "cy" variables.

Scaling and zoom (as well as the other effects) can be used in the same patch.

3.4.5 Warping

Warping is a complex built-in effect that produces a moving distortion of the picture. You activate it by setting a non-zero positive value to the "warp" variable. The higher the value, the more intense the effect. A value of 2 already produces a major modification of the image.

Now, you have quite a few parameter to play with already. Experiment with them in order to get a feeling of what the sensible values for them are and how the system reacts. You can of course combine all the effects in the same patch.

3.5 Other objects

3.5.1 Motion vectors

3.5.2 Borders

3.5.3 Video echo

3.5.4 User pictures

3.5.5 Live video

3.6 Interacting: per-frame equations

3.6.1 Introduction

Patches get far more interesting if you can take the variables (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-vertex* equations. Let's start with per-frame equations. These are executed once per frame. So, if you were to type the following equation in:

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

To ensure compatibility with MilkDrop, Flickernoise also accepts per_frame_xx where xx can be any number.

...then the zoom amount would oscillate between 0.9 and 1.1 over time.[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 approximately 6.28 ($\approx 2 \cdot \pi$) seconds.

The "time" parameter is a read-only variable that retrieves the amount of time, in seconds, since Flickernoise started generating the video effects.

If you wanted it to make the zoom cycle every 2 seconds, you could use:

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

Now, let's say you wanted to make the color of the waveform (sound wave) that gets plotted on the screen vary through time. The color is defined by three values, one for each of the main color 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:

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

[1] Recall from your geometry classes that sin() returns a value between -1 and 1

It's nice to stagger the frequencies (1.13, 1.23, and 1.33) of the sine functions for the red, green, and blue color 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, and the wave_{r,g,b} parameters take values between 0 and 1.

$0.5 + 0.5 \cdot sin(time) \equiv 0.5 + (-0.5 \text{ to } 0.5) \equiv 0 \text{ to } 1$

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 approximate value of $2 \cdot \pi$) for a complete cycle. If you want to speed up the time period (i.e. make the color 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, multiply with a number between 0 and 1.

If you want the color variable to be focused on a stricter range than you should alter the equation. For example, to generate a "redder" image you may want to have the range 0.5 to 1 for the wave_r. Which would require the following equation:

$0.75 + 0.25 \cdot sin(time) \equiv 0.75 + (-0.25 \text{ to } 0.25) \equiv 0.5 \text{ to } 1$

3.6.2 Reacting to sound

3.6.3 DMX and OSC controls

3.6.4 The variable monitor

3.7 Fine-tuned motion: per-vertex equations

3.7.1 About per-vertex equations

If the built-in motion effects configured by variables such as "zoom", "rot" and "warp" are not enough for you, you can define your own motion equations.

3.7.2 Q variables

To ensure compatibility with MilkDrop, Flickernoise also accepts per_vertex_xx and per_pixel_xx where xx can be any number.

3.8 Variable index

Here is the complete list and description of the variables that can be used in patches. Some are read-only, and some only

make sense in the context of per-frame or per-vertex equations.

The sometimes inconsistent naming of the variables is legacy from MilkDrop.

3.8.1 zoom

Range: ≥ 0

Controls inward/outward motion; 0.9=zoom out 10% per frame, 1.0=no zoom, 1.1=zoom in 10%.

3.8.2 rot

Range: N/A

Controls the amount of rotation; 0=none, 0.1=slightly counter-clockwise, -0.1=slightly clockwise.

3.8.3 warp

Range: ≥ 0

Controls the magnitude of the warping; 0=none, 1=normal, 2=major warping...

3.8.4 fWarpAnimSpeed

Range: ≥ 0

Controls the frequency of the warp effect oscillation.

3.8.5 fWarpScale

Range: ≥ 0

Controls the amplitude of the warp effect oscillation.

3.8.6 cx, cy

Range: $0..1$

Controls where the centre of rotation and stretching is.

3.8.7 dx, dy

Range: $-1..1$

Controls amount of constant displacement; -0.01=move left (or up) 1% per frame, 0=none, 0.01=move right (or down) 1%.

3.8.8 sx, sy

Range: ≥ 0

Controls amount of constant stretching; 0.99=shrink 1%, 1=normal, 1.01=stretch 1%

3.8.9 decay

Range: 0..1
Alias: fDecay

Controls the eventual fade to black; 1=no fade, 0.9=strong fade.

3.8.10 bTexWrap

Range: 0..1 (integer)

Controls whether pixels that are pushed into a border of the screen (because of the motion) appear at the opposite corner.

3.8.11 wave_mode

Range: 0..7 (integer)
Alias: nWaveMode

Selects one of the eight wave drawing modes.

3.8.12 wave_additive

Range: 0..1 (integer)
Alias: bAdditiveWaves

If this parameter is 1, the waves are drawn by adding to the existing colors instead of replacing the pixels.

3.8.13 wave_brighten

Range: 0..1 (integer)
Alias: bMaximizeWaveColor

If this parameter is 1, the color components of the wave are scaled until the largest one reaches 1.0.

3.8.14 wave_scale

Range: ≥ 0
Alias: fWaveScale

Controls the amplitude of the drawn wave.

3.8.15 wave_usedots

Range: 0..1 (integer)
Alias: bWaveDots

If this parameter is 1, the waves are drawn with dots (yielding a “stippled” effect).

3.8.16 wave_thick

Range: 0..1 (integer)
Alias: bWaveThick

If this parameter is 1, the waves are drawn thicker.

3.8.17 wave_r, wave_b, wave_g

Range: 0..1

Amount of red, green and blue color in the wave.

3.8.18 wave_a

Range: 0..1

Opacity (alpha channel) of the wave. 0=transparent, 1=opaque.

3.8.19 wave_x, wave_y

Range: 0..1

Controls where the wave is drawn on the screen.

3.8.20 ob_size

Range: 0..0.5

Thickness of the outer border drawn at the edges of the screen every frame.

3.8.21 ob_r, ob_b, ob_g

Range: 0..1

Amount of red, green and blue color in the outer border.

3.8.22 ob_a

Range: 0..1

Opacity (alpha channel) of the outer border. 0=transparent, 1=opaque.

3.8.23 ib_size

Range: 0..1

Thickness of the inner border drawn at the edges of the screen every frame.

3.8.24 ib_r, ib_b, ib_g

Range: 0..0.5

Amount of red, green and blue color in the inner border.

3.8.25 ib_a

Range: 0..1

Opacity (alpha channel) of the inner border. 0=transparent, 1=opaque.

3.8.26 mv_r, mv_b, mv_g

Range: 0..1

Amount of red, green and blue color in the motion vectors.

3.8.27 mv_a

Range: 0..1

Opacity (alpha channel) of the motion vectors. 0=transparent, 1=opaque.

3.8.28 mv_x, mv_y

Range: 0..48
Aliases: nMotionVectorsX, nMotionVectorsY

Number of motion vectors in each direction. This number can be non-integer in order to fine-tune the distance between the motion vectors.

3.8.29 mv_dx, mv_dy

Range: 0..1

Controls the position of the origin of the motion vector grid.

3.8.30 mv_l

Range: 0..5

Size of the dots used for the motion vectors.

3.8.31 fVideoEchoAlpha

Range: 0..5

Opacity (alpha channel) of the second graphics layer. 0=transparent, 1=opaque.

3.8.32 fVideoEchoZoom

Range: ≥ 0

Zooming of the second graphics layer. 0.9=zoom out 10%, 1.0=no zoom, 1.1=zoom in 10%.

3.8.33 nVideoEchoOrientation

Range: $0..3$ (integer)

Orientation of the second graphics layer:

- 0: Do not flip the second layer
- 1: Flip the second layer on X axis
- 2: Flip the second layer on Y axis
- 3: Flip the second layer on both axes

3.8.34 time

Range: ≥ 0

Retrieves the time, in seconds, since Flickernoise started rendering.

This variable is read only, and can be used in the variable monitor.

3.8.35 frame

Range: ≥ 0 (integer)

Retrieves the number of displayed frames since Flickernoise started rendering. The nominal frame rate is 24 frames per second.

This variable is read only, and can be used in the variable monitor.

3.8.36 bass, mid, treb

Range: ≥ 0

Retrieves the current amount of bass (respectively, middle and high frequencies). 1 is normal; below 0.7 is quiet; above 1.3 is loud bass (respectively, middle and high frequencies).

These variables are read only, and can be used in the variable monitor.

3.8.37 bass_att, mid_att, treb_att

Range: ≥ 0

Retrieves an "attenuated" reading of the bass, mid and treb variables, meaning that it is damped in time and does not change so rapidly.

These variables are read only, and can be used in the variable

monitor.

3.8.38 idmx1..idmx4

Range: 0..1

Retrieves the current values of the DMX input channels.

These variables are read only, and can be used in the variable monitor.

3.8.39 dmx1..dmx4

Range: 0..1

Sets the values of the DMX output channels.

3.8.40 osc1..osc4

Range: N/A

Retrieves the current values of the OSC input channels.

These variables are read only, and can be used in the variable monitor.

3.8.41 x, y

Range: 0..1

In per-vertex equations, retrieves the coordinates of the current point.

3.8.42 rad

Range: ≥ 0

In per-vertex equations, retrieves the distance of the current point to the center of the screen.

3.8.43 q1..q8

Range: N/A

Carry values from the per-frame to the per-vertex equations.

3.9 Operator and function index

Here is the list of the operators and functions that you can use to operate on variables in per-frame or per-vertex equations.

Function	Description
$+$	Addition
$-$	Subtraction
$*$	Multiplication
$/$	Division
%	Convert to integer and take remainder
$int(x)$	Return the integer value of x (floored)
$abs(x)$	Return the absolute value of x
$sign(x)$	Return the sign of x (-1, 0 or 1)
$min(x, y)$	Return the smallest value
$max(x, y)$	Return the greatest value
$sqr(x)$	Return the square of x (i.e. $x \cdot x$)
$sqrt(x)$	Return the square root of x (i.e. $\sqrt{x}$)
$sin(x)$	Return the sine of x (expressed in radians)
$cos(x)$	Return the cosine of x (expressed in radians)
$if(c, x, y)$	If c is different than 0, return x, otherwise return y
$equal(x, y)$	Return 1 if $x = y$ and 0 otherwise
$above(x, y)$	Return 1 if $x > y$ and 0 otherwise
$below(x, y)$	Return 1 if $x < y$ and 0 otherwise

3.10 Tips and tricks

4 Open source vision