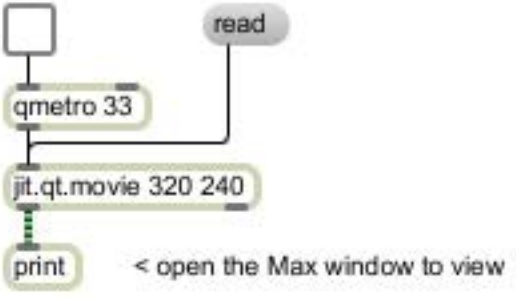
start.stop read a movie file



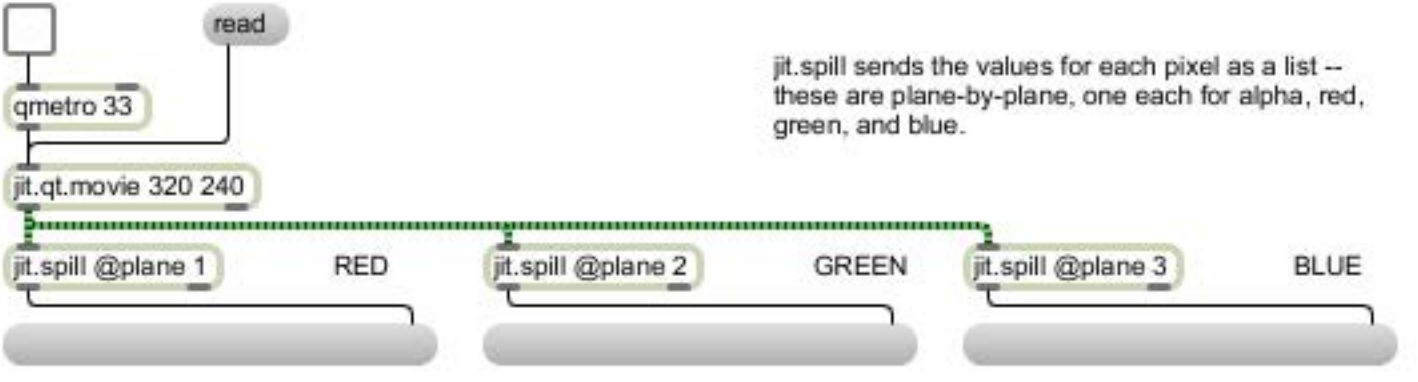
WHAT IS A DIGITAL VIDEO?

When you load and play a movie file, you will notice that it just sends a weird mash of a letter and numbers.

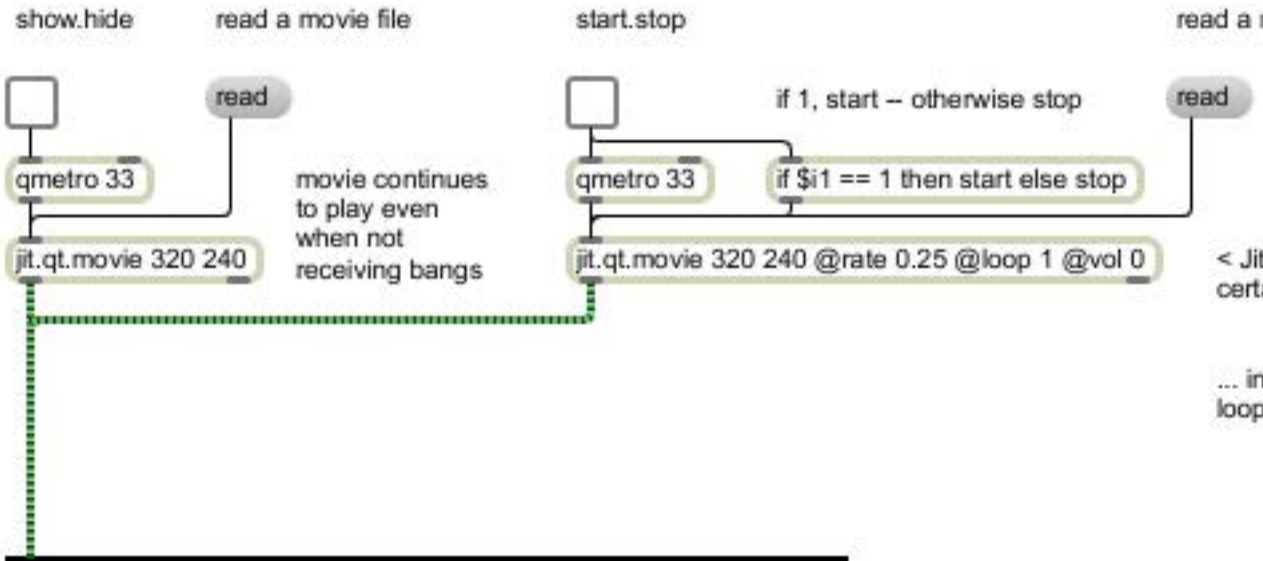
This is because Jitter doesn't actually send the video file itself through the patch-cords. Rather, it sends a symbol that allows Jitter to find that stream of numbers elsewhere on the computer, thereby speeding up the processing.

In a few examples, we'll take a look inside those numbers to alter and process them.

start.stop read a movie file



jit.spill sends the values for each pixel as a list -- these are plane-by-plane, one each for alpha, red, green, and blue.



PLAYING VIDEO FILES

Jitter uses Quicktime for most video playback, allowing files in the .mov format to be played, manipulated, and output.

The "engine" of this system is the qmetro object, which is like the regular metro, except that it defers to other processes, meaning it floats up and down depending on processing in the computer.

The playback itself is done with the jit.qt.movie object. This object takes at least one argument - the dimensions of the video "matrix" in pixels. For most real-time projects, anything above 320x240px will slow even RAM-powerful computers considerably.

start.stop

read a movie file

resize the video

width

height

read

qmetro 33

0

0

jit.qt.movie 320 240 @vol 0.

pak dim 0 0

s newDims

r newDims

jit.matrix 4 char 16 16 16 @dim 16 12

jit.matrix 4 char 16 16 16 @dim 1 240

< the matrix object is where the transformation occurs

0.00000

fps

< jit.fpsgui allows you to see current framerate depending on your processor

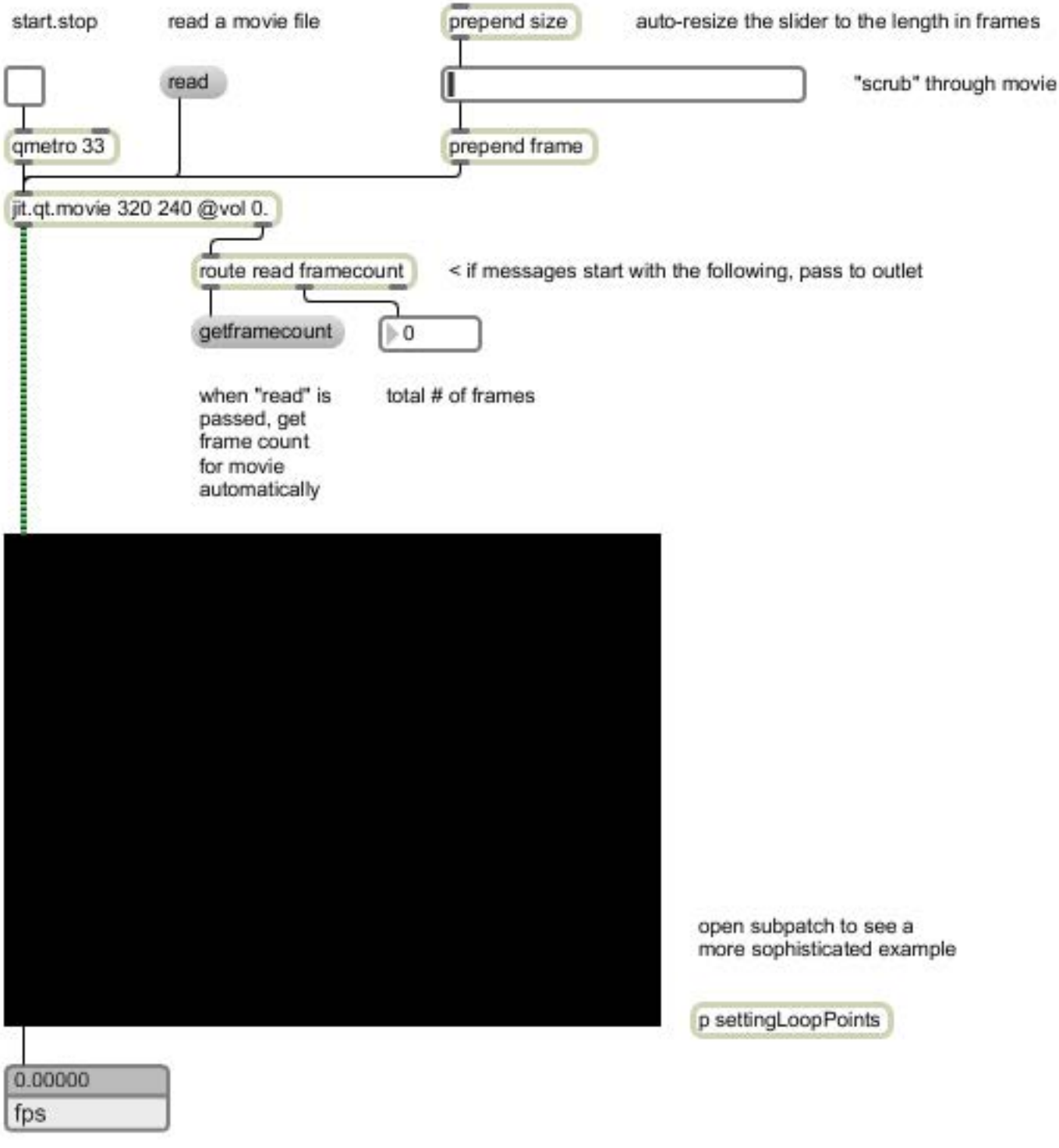
TRANSFORMING THE MATRIX
(lowering resolution)

The jit.matrix object does nothing on its own, but allows changes to be made to the matrix passing through it.

jit.matrix has a few required arguments which are a bit hard to understand. They are:

1. "4" this is the "plane count"; in most cases: Alpha, Red, Green, Blue
2. "char" this is what kind of value stores the video data; likely you will not change this
3. "16..." this is the color mode for the video, also not changed; also likely not to be changed

The object can receive a lot of different arguments, but in our case we send a "dim x x" message that resizes the video dynamically. While it may be obvious, scaling higher than the incoming resolution will do nothing except slow your computer.


start.stop
read a movie file
prepend size
auto-resize the slider to the length in frames
read
"scrub" through movie
qmetro 33
prepend frame
jit.qt.movie 320 240 @vol 0.
route read framecount
< if messages start with the following, pass to outlet
getframecount
0
when "read" is passed, get frame count for movie automatically
total # of frames
open subpatch to see a more sophisticated example
p settingLoopPoints
0.00000
fps

read

qmetro 33

t gettime b

r looppoints

jit.qt.movie 320 240 @vol 0.

route read time duration timescale fps looppoints

prepend size

set the size of the slider

getfps,
gettimescale,
getduration,
getrate

send message
to get info
automatically

timescale

0

0.

fps

/ 1.

timescale / fps

set looppoints for movie

0

current time

duration

0

0

time units per frame

pak looppoints 0 0

combine into a list

/

/

total time / units per frame

s looppoints

0

current frame

0

total frames

0.00000

fps

read

qmetro 33

if $i1==1 then start else stop

jit.qt.movie 320 240 @vol 0.

jit.matrix 4 char 16 16 16 @dim 320 240

p delColor-Green

p delColor-Blue

jit.fill delColor 1

jit.fill delColor 2

jit.fill delColor 3

jit.matrix delColor 4 char 16 16 16

jit.spill @plane 1

jit.spill @plane 2

jit.spill @plane 3

zl sort

zl sort

zl sort

jit.fill sorted 1

jit.fill sorted 2

jit.fill sorted 3

jit.matrix sorted 4 char 16 16 16

all pixels sorted numerically

zl rev

zl rev

zl rev

jit.fill random 1

jit.fill random 2

jit.fill random 3

jit.matrix random 4 char 16 16 16

all pixels randomly shuffled around

jit.unpack

unpack colors...

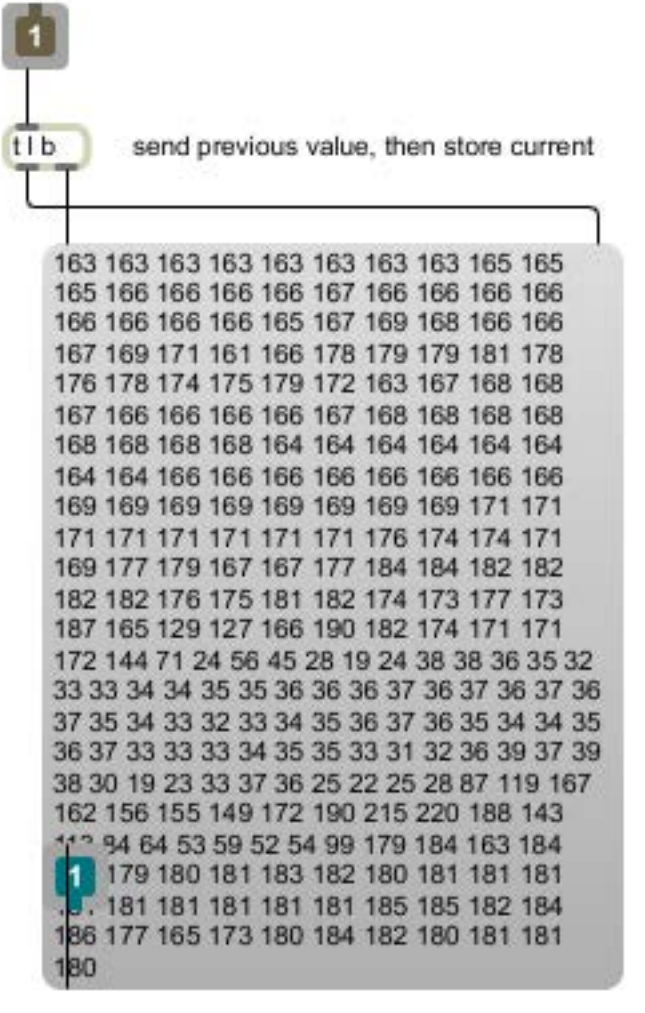
jit.pack

... and send to
other color inputs

shift colors

0.00000

fps


1
t l b
send previous value, then store current
163 163 163 163 163 163 163 163 165 165
165 166 166 166 166 167 166 166 166 166
166 166 166 166 165 167 169 168 166 166
167 169 171 161 166 178 179 179 181 178
176 178 174 175 179 172 163 167 168 168
167 166 166 166 166 167 168 168 168 168
168 168 168 168 164 164 164 164 164 164
164 164 166 166 166 166 166 166 166 166
169 169 169 169 169 169 169 169 171 171
171 171 171 171 171 171 176 174 174 171
169 177 179 167 167 177 184 184 182 182
182 182 176 175 181 182 174 173 177 173
187 165 129 127 166 190 182 174 171 171
172 144 71 24 56 45 28 19 24 38 38 36 35 32
33 33 34 34 35 35 36 36 36 37 36 37 36 37 36
37 35 34 33 32 33 34 35 36 37 36 35 34 34 35
36 37 33 33 33 34 35 35 33 31 32 36 39 37 39
38 30 19 23 33 37 36 25 22 25 28 87 119 167
162 156 155 149 172 190 215 220 188 143
94 64 53 59 52 54 99 179 184 163 184
1
179 180 181 183 182 180 181 181 181
181 181 181 181 181 185 185 182 184
186 177 165 173 180 184 182 180 181 181
180

qmetro 33

read

stop

jit.qt.movie 320 240 @vol 0.

original video

r reset

crossfade between
two videos

0.85

prepend xfade

jit.xfade @xfade 0.85

neat feedback-reset trick

reset

t b 0.

first set to 0, then reset

del 100

sel 0.

if it's not 0, then store

float

store previous value

s reset

other fun stuff to look at:

jit.cycle

jit.scanwrap

jit.scalebias

jit.chromakey

direct out
or further
process

+ 1

speedlim 50

gate 2

video with feedback/processing

1.

prepend brightness

1.1

prepend contrast

1.1

prepend saturation

jit.brcosa

operator

value

%

0.25

prepend op

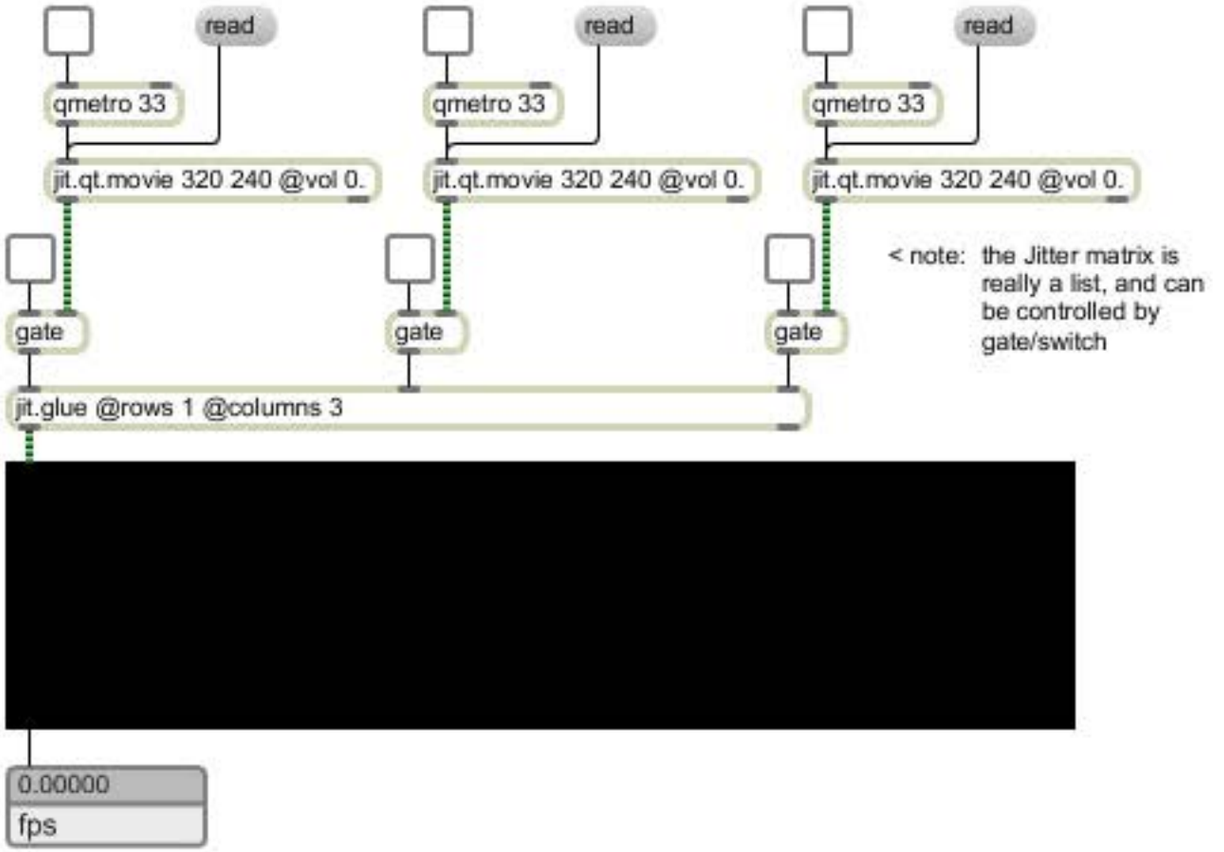
jit.op

0.00000

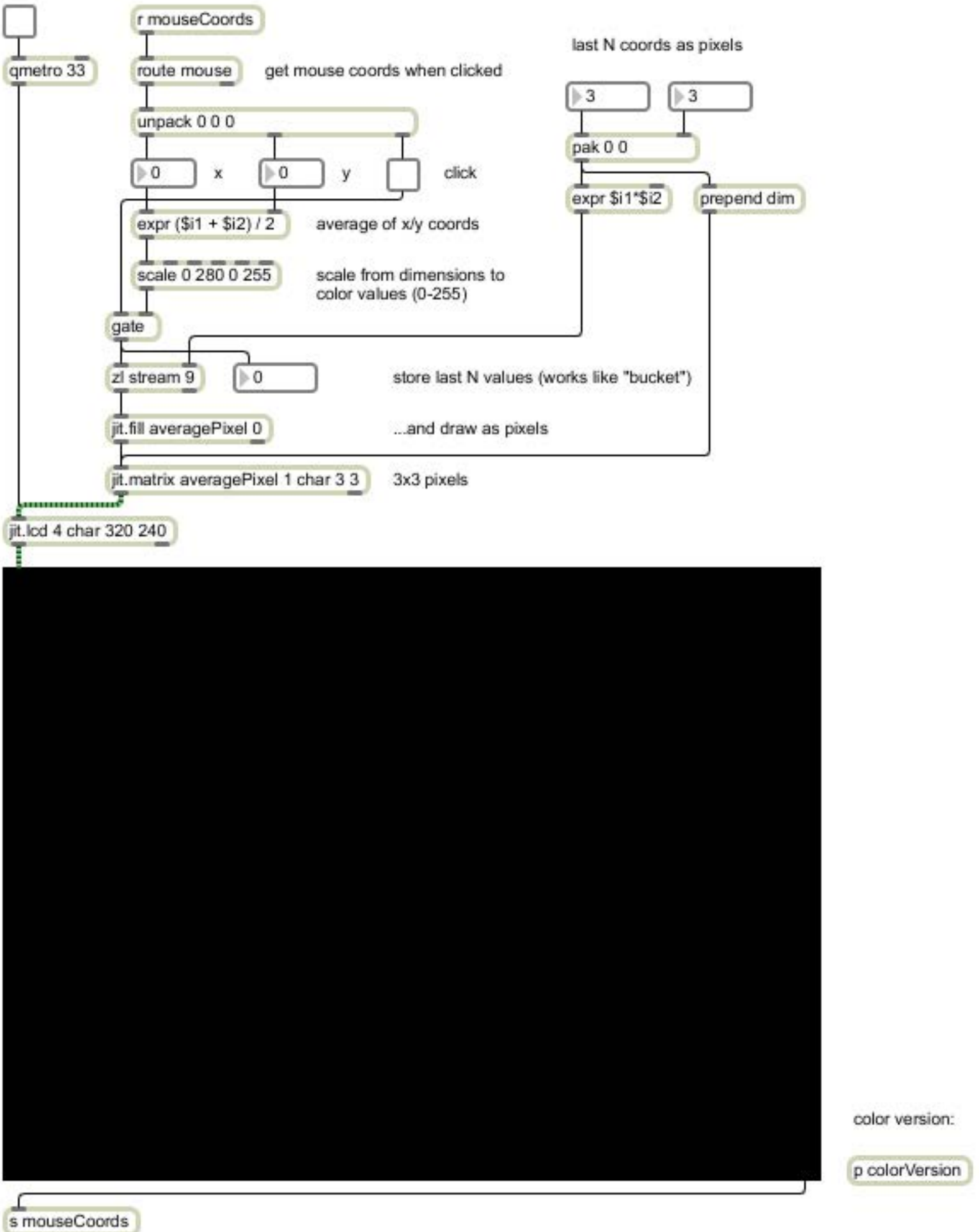
fps

CUTTING and BUILDING VIDEO

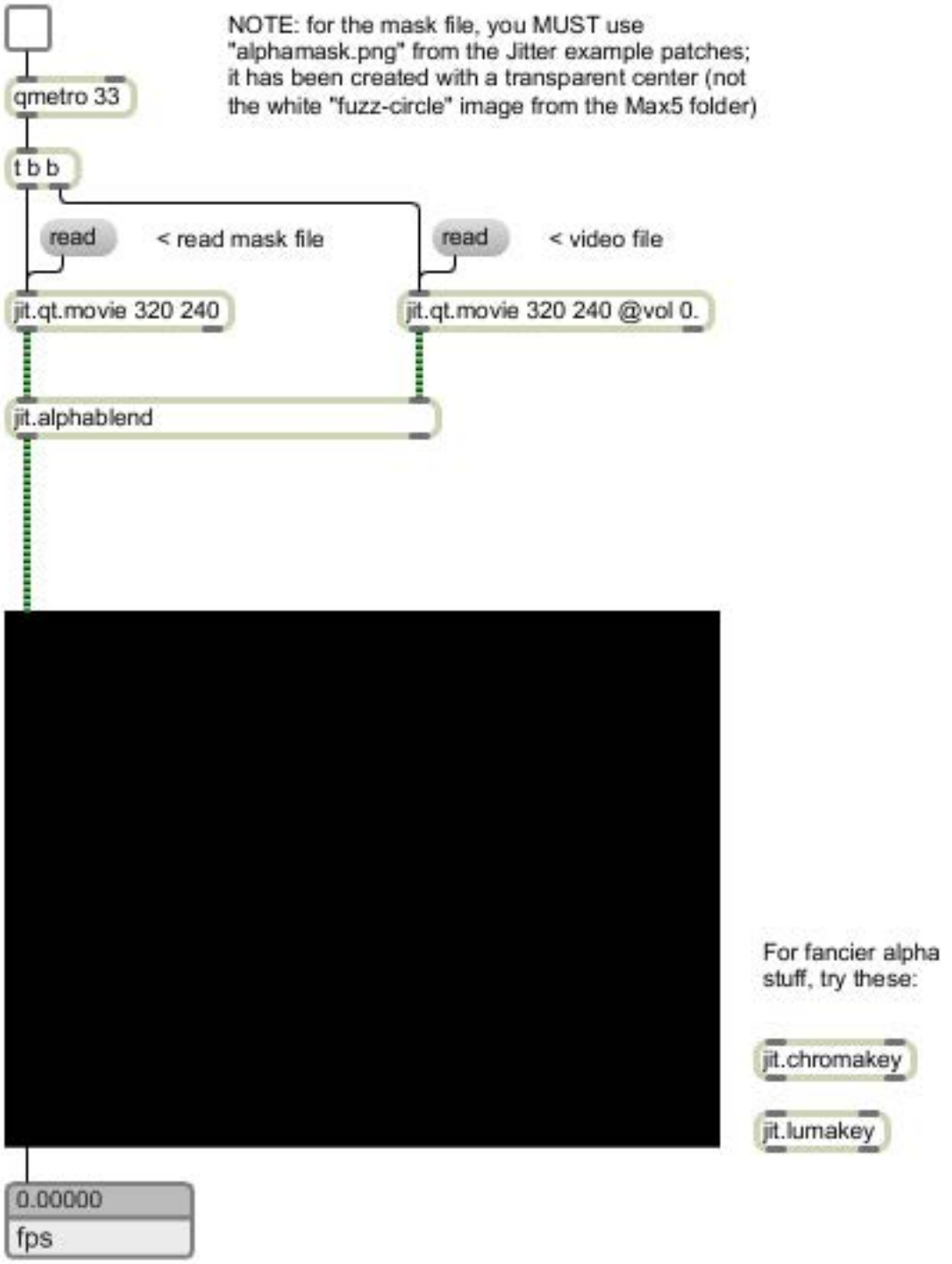
You can cut videos into multiple parts (jit.scissors)
and/or put separate videos together (jit.glue).



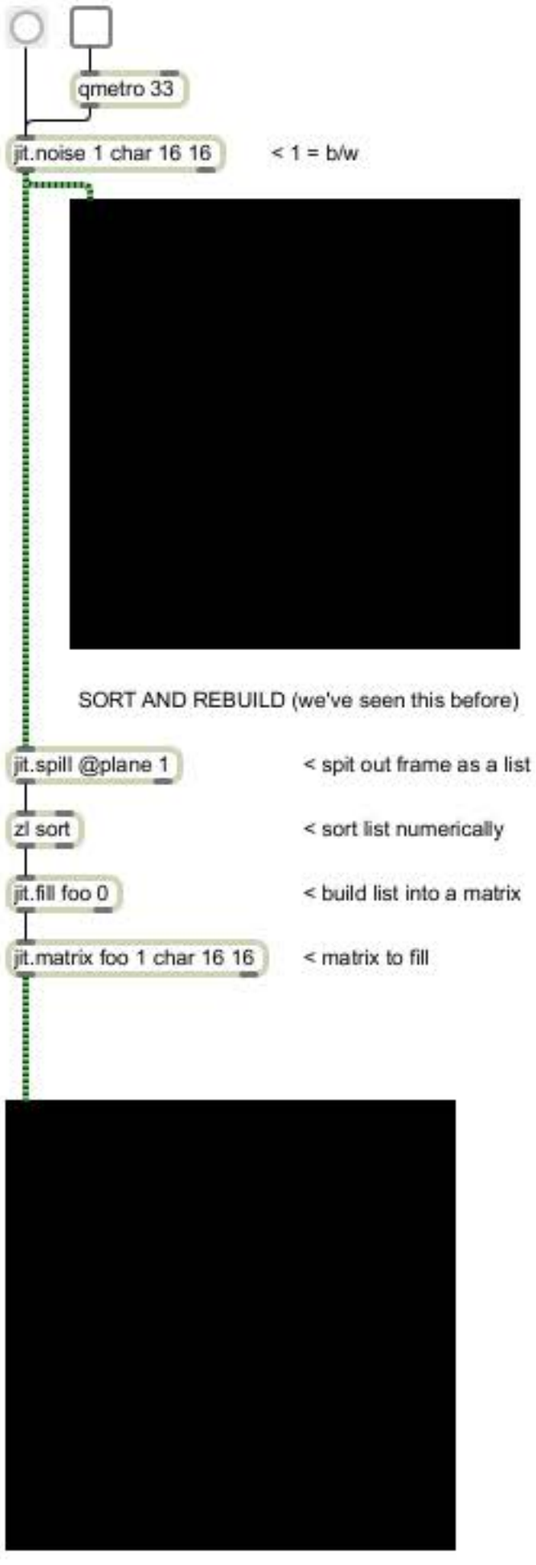
USE jit.lcd TO TRACK MOUSE ACROSS VIDEO



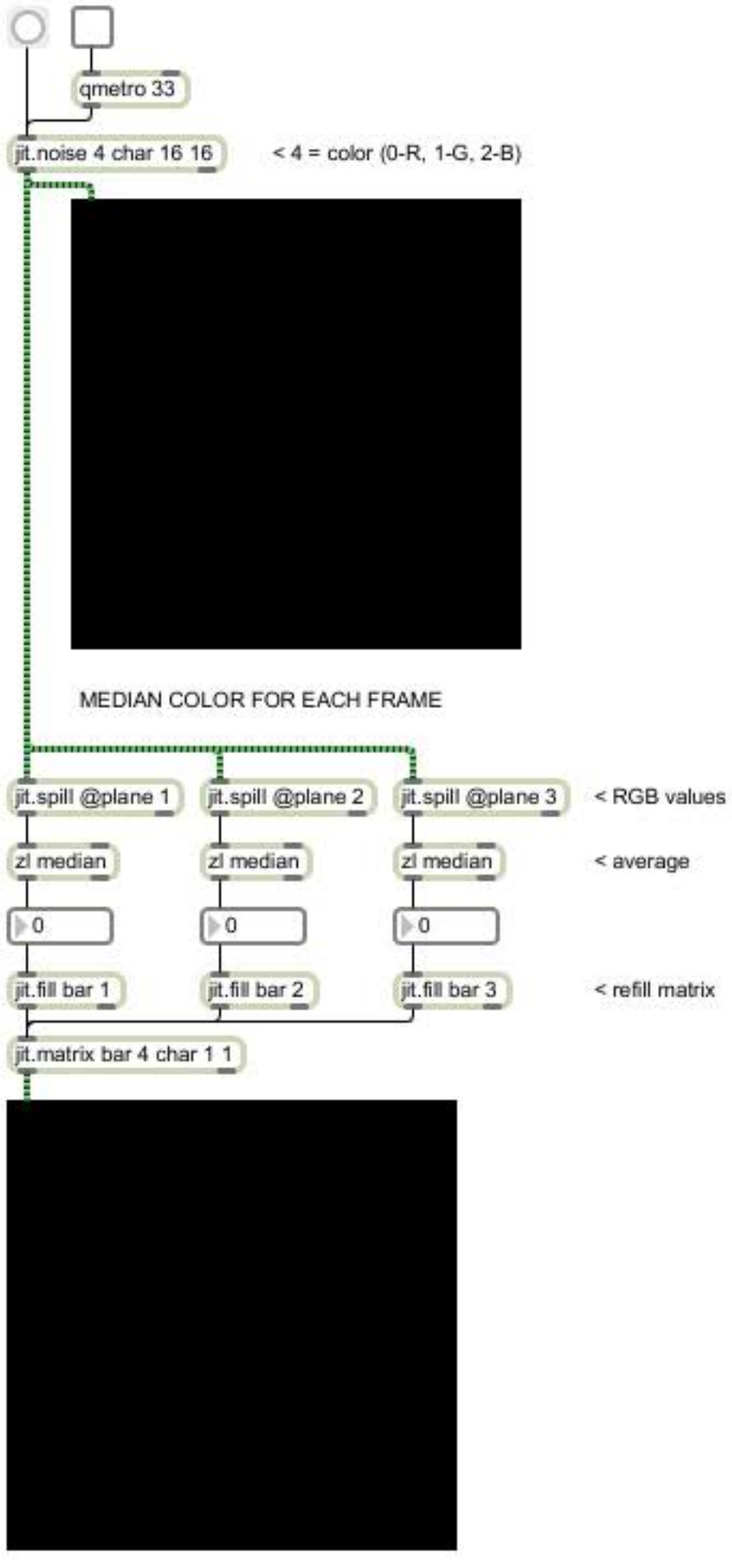
PNG FILE as MASK



SYNTHESIZE RANDOM B/W VIDEO



SYNTHESIZE RANDOM COLORED VIDEO

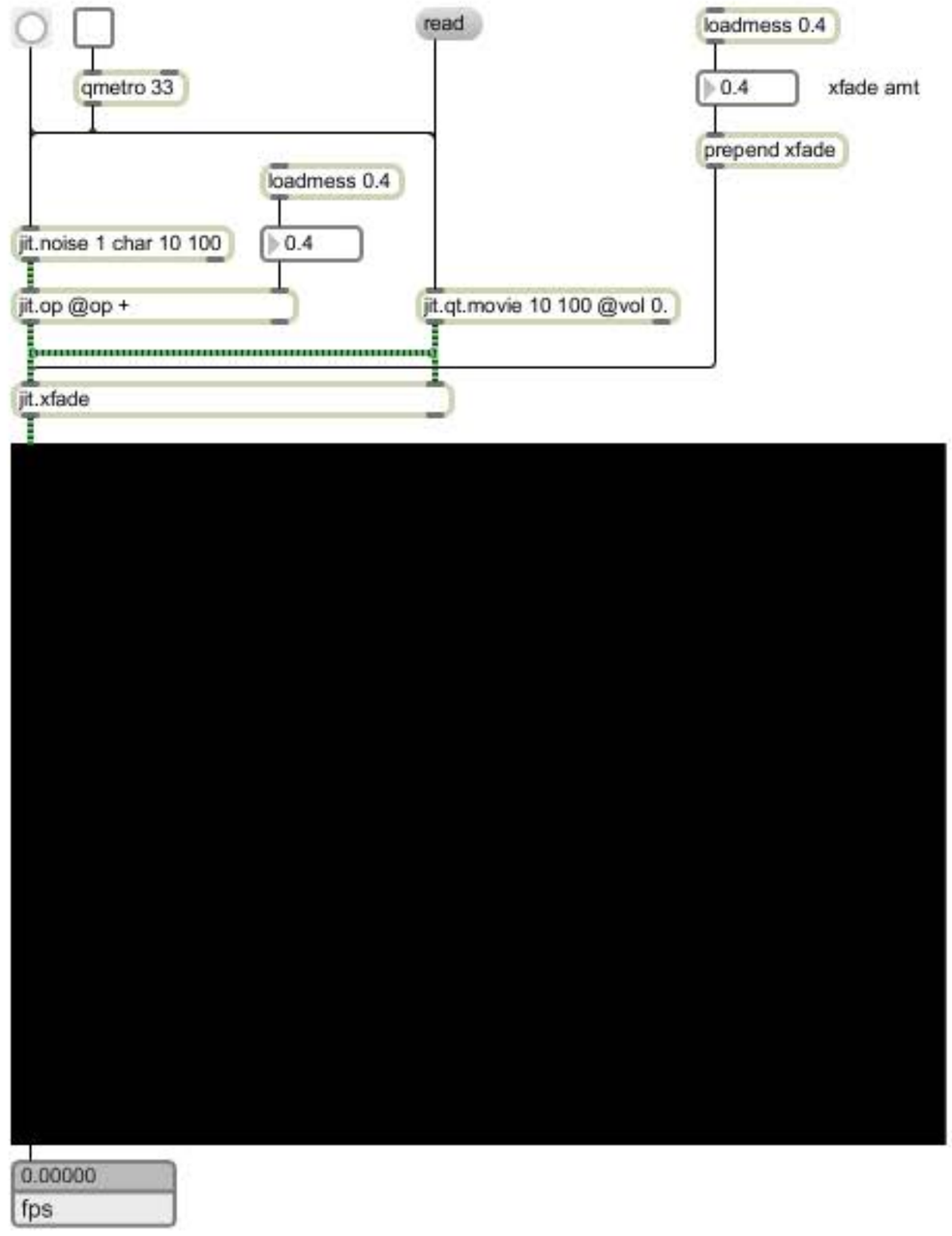


SYNTHESIZING VIDEO

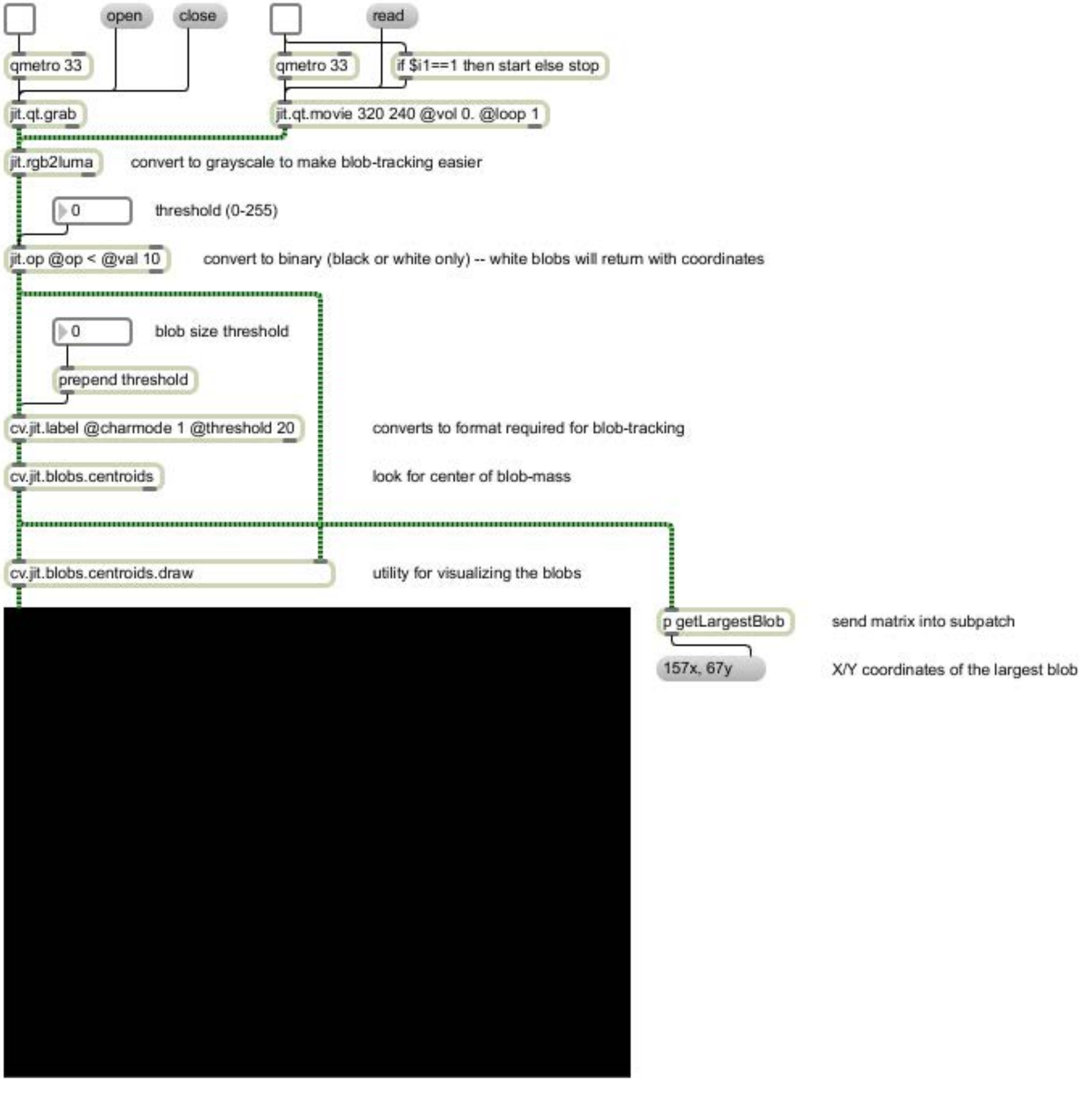
Video can be made without cameras! Since the Jitter maxtrix is really just a list of numbers, those numbers can be manipulated or created from scratch.

Note: we're using 16x16 matrices since the zl objects are limited to 256 numbers

MIX RANDOM and RECORDED VIDEO

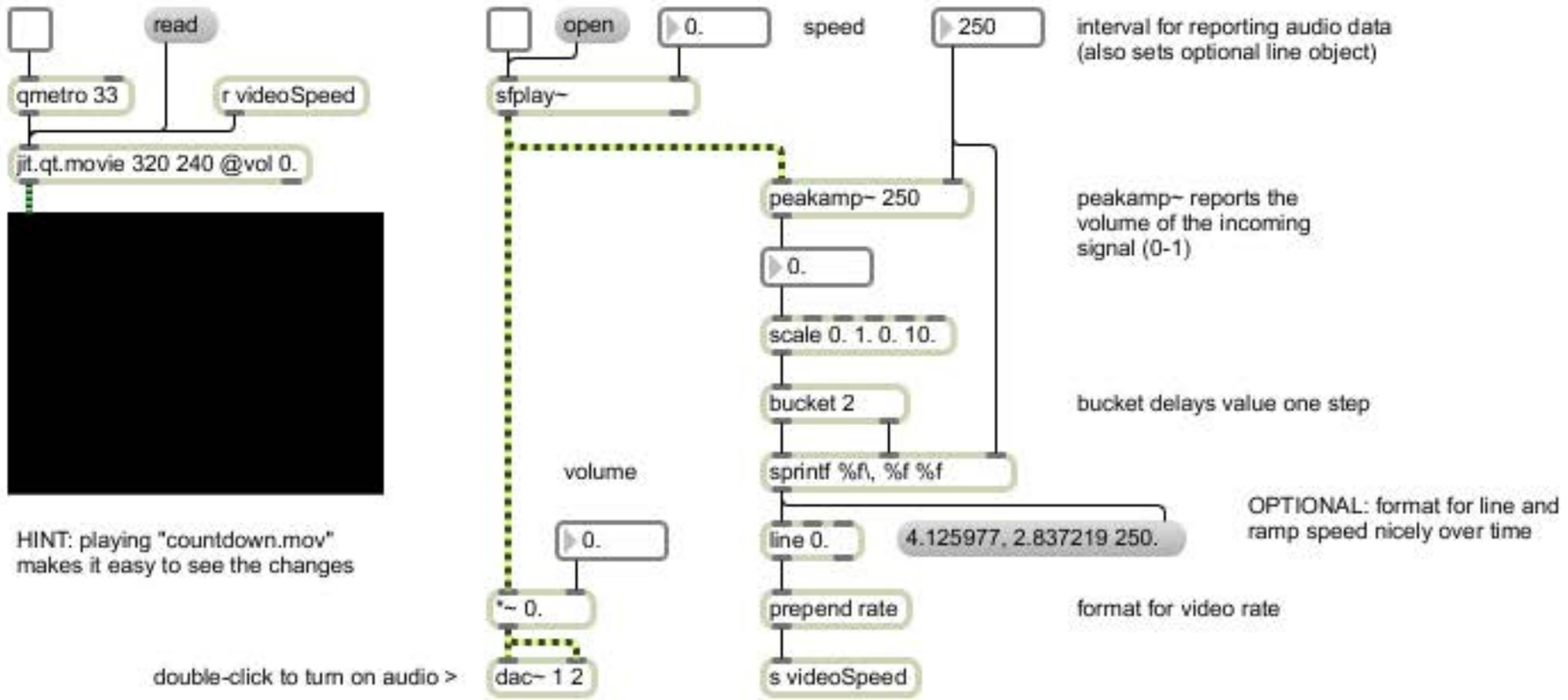



open
close
qmetro 33
jit.qt.grab
read
qmetro 33
if $i1==1 then start else stop
0.
prepend rate
jit.qt.movie 320 240 @vol 0.
jit.rgb2luma
convert to grayscale to make blob-tracking easier
0
threshold (0-255)
jit.op @op < @val 10
convert to binary (black or white only) -- white blobs will return with coordinates
blob size threshold
0
prepend threshold
p numBlobsTo_sfplay
cv.jit.label @charmode 1 @threshold 20
converts to format required for blob-tracking
cv.jit.blobs.centroids
s numBlobs
look for center of blob-mass
s xyCoords
send to subpatch
cv.jit.blobs.centroids.draw
utility for visualizing the blobs
sonification of the blobs
p xyCoords


open
close
read
qmetro 33
qmetro 33
if $i1==1 then start else stop
jit.qt.grab
jit.qt.movie 320 240 @vol 0. @loop 1
jit.rgb2luma
convert to grayscale to make blob-tracking easier
0
threshold (0-255)
jit.op @op < @val 10
convert to binary (black or white only) -- white blobs will return with coordinates
0
blob size threshold
prepend threshold
cv.jit.label @charmode 1 @threshold 20
converts to format required for blob-tracking
cv.jit.blobs.centroids
look for center of blob-mass
cv.jit.blobs.centroids.draw
utility for visualizing the blobs
p getLargestBlob
send matrix into subpatch
157x, 67y
X/Y coordinates of the largest blob

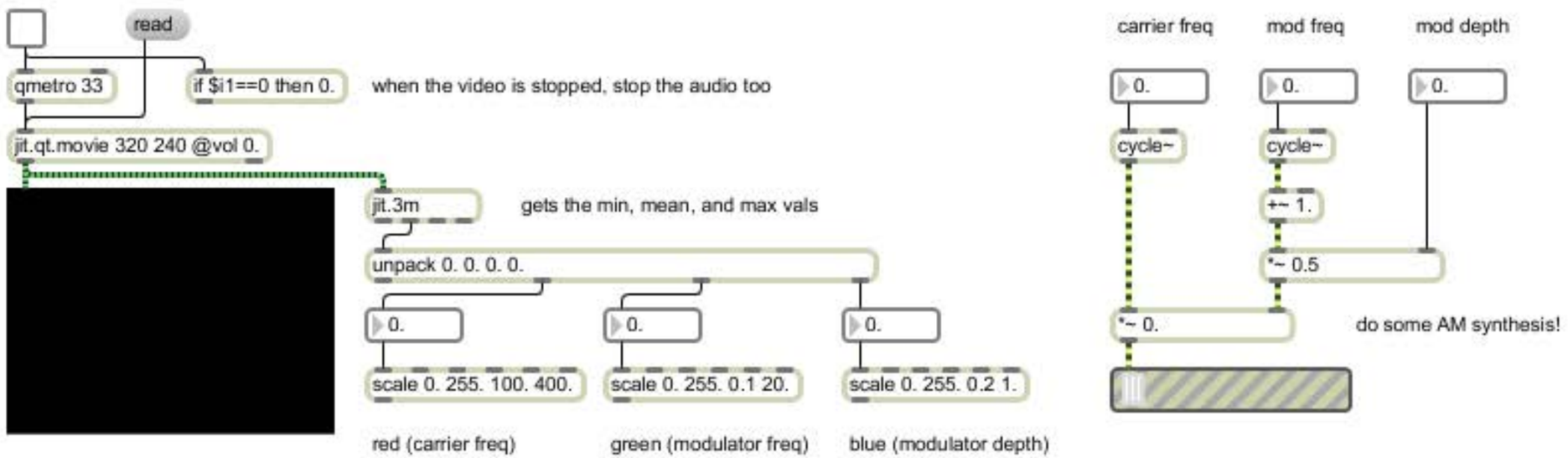
USING AUDIO TO CONTROL VIDEO

This patch uses the volume of an audio file to control the speed of a video being played. Using the peakamp~ object, we retrieve the current amplitude (volume) of the audio source at 1/4-second intervals and change the rate of the video programatically.



... AND THE OPPOSITE (VIDEO > AUDIO)

The reverse of the above patch, this path takes a video file and reads the mean (average) R, G, and B values for each frame. Those values are scaled into more useable ranges and are mapped to the carrier and modulation frequency and modulation depth of an AM synthesis module.



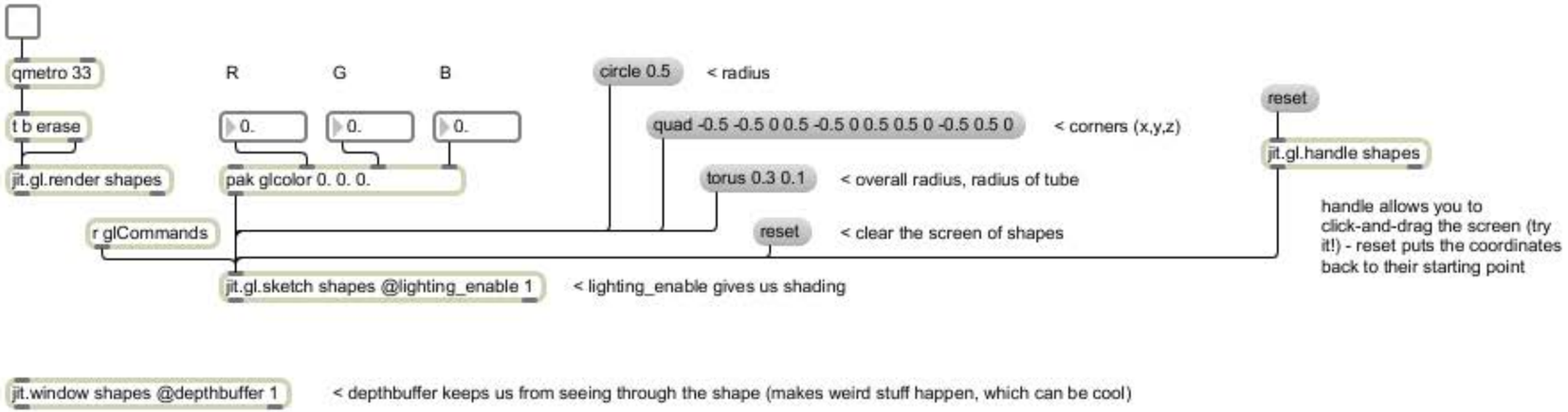
BASIC GL PATCH

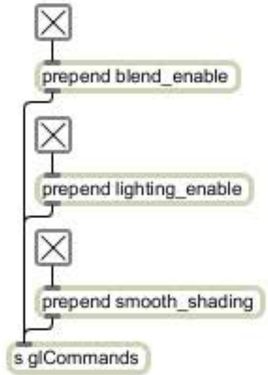
Using OpenGL is a bit more complicated than Jitter video and timing in your patch will be critical.

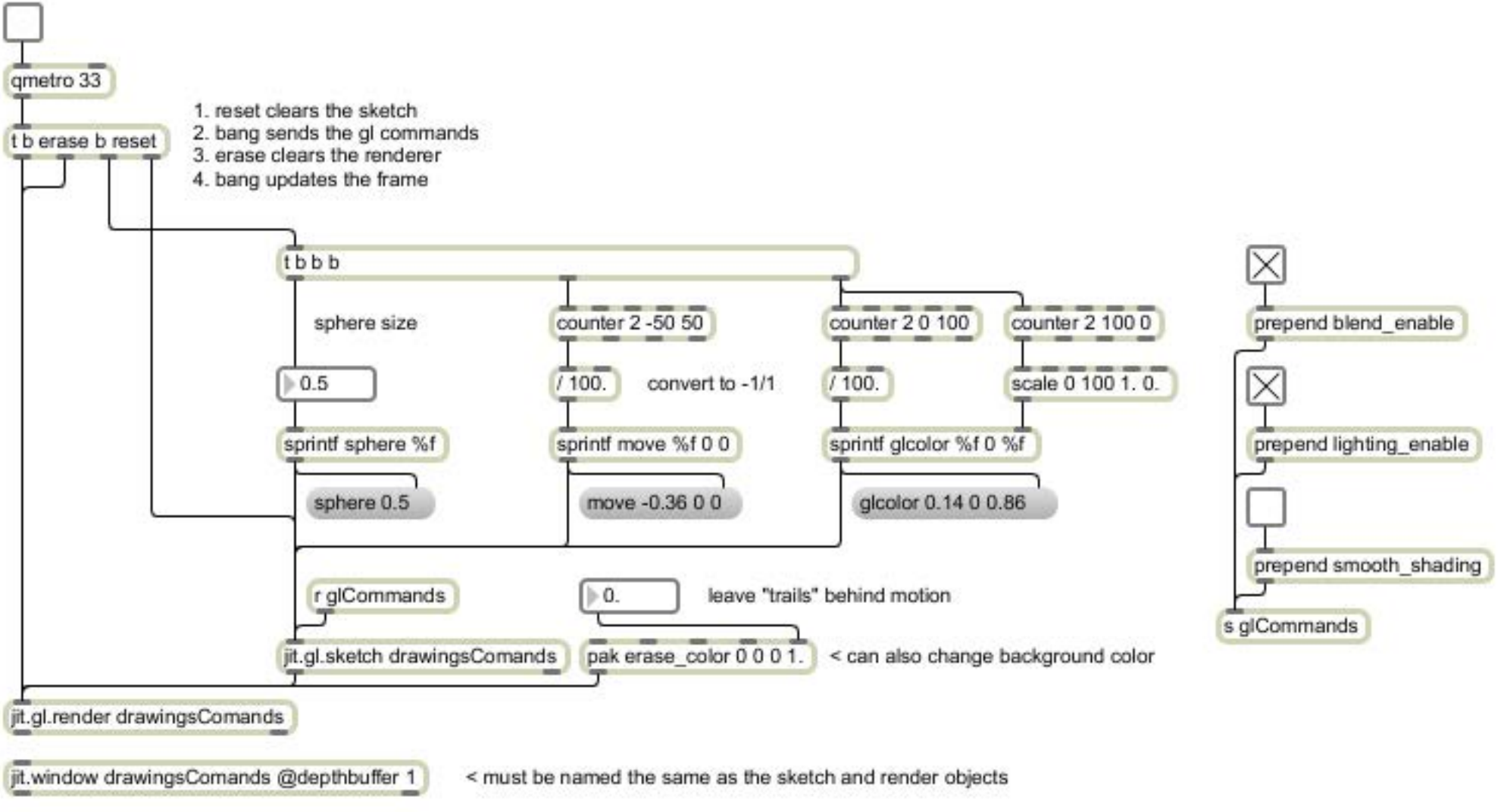
We use jit.gl.sketch to create simple shapes (2d and 3d), jit.gl.render converts it to a useable format, and jit.window to display.

DRIVING GL PATCHES
+ qmetro drives the patch, just like with Jitter video
+ the trigger (t) object ensures proper timing

DRAWING COMMANDS
1. erase clears the renderer
2. finally, a bang updates the renderer and sends the data to the window



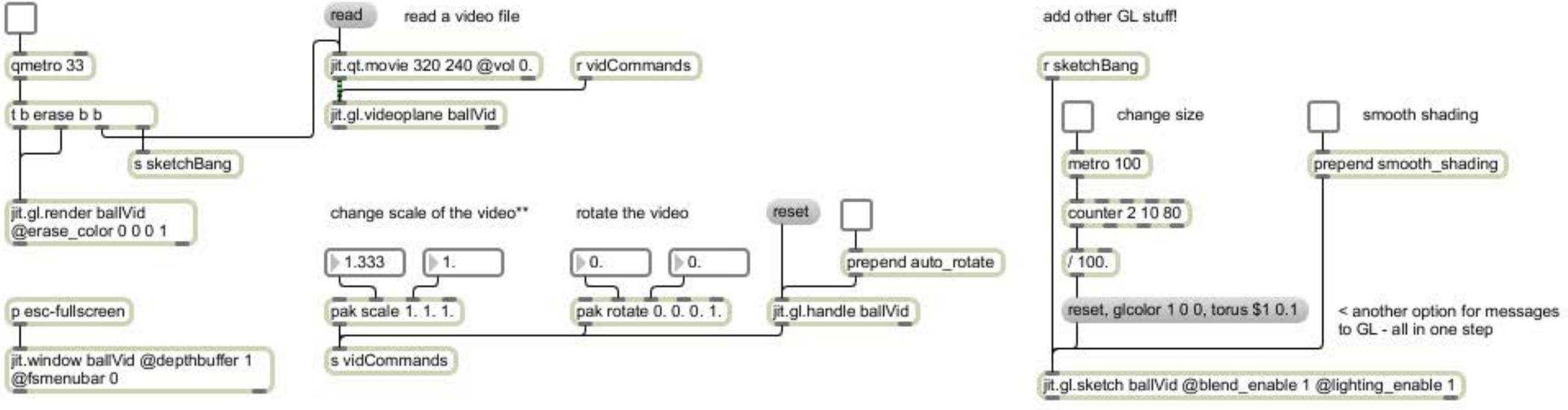



qmetro 33
1. reset clears the sketch
2. bang sends the gl commands
3. erase clears the renderer
4. bang updates the frame
t b erase b reset
t b b b
sphere size
counter 2 -50 50
counter 2 0 100
counter 2 100 0
0.5
/ 100.
convert to -1/1
/ 100.
scale 0 100 1. 0.
sprintf sphere %f
sprintf move %f 0 0
sprintf glcolor %f 0 %f
sphere 0.5
move -0.36 0 0
glcolor 0.14 0 0.86
r glCommands
0.
leave "trails" behind motion
jit.gl.sketch drawingsComands
pak erase_color 0 0 0 1.
< can also change background color
jit.gl.render drawingsComands
jit.window drawingsComands @depthbuffer 1
< must be named the same as the sketch and render objects
prepend blend_enable
prepend lighting_enable
prepend smooth_shading
s glCommands

GL VIDEOPLANE

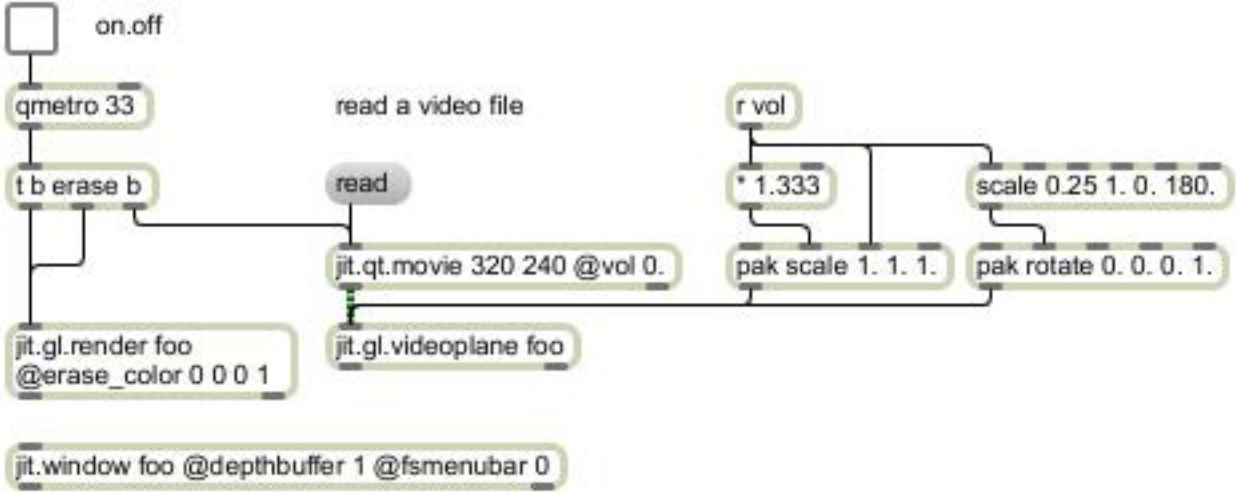
Bringing video into the GL realm can improve your computer's performance (rendering on the graphics processor, rather than the CPU) and allow you to do really cool stuff like rotate a video in 3d space.

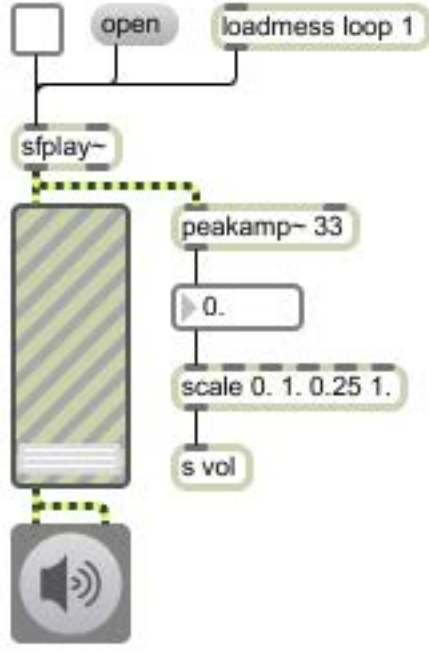
** note we've set the scale to 1.33 since this is a 4:3 video, which fills the screen

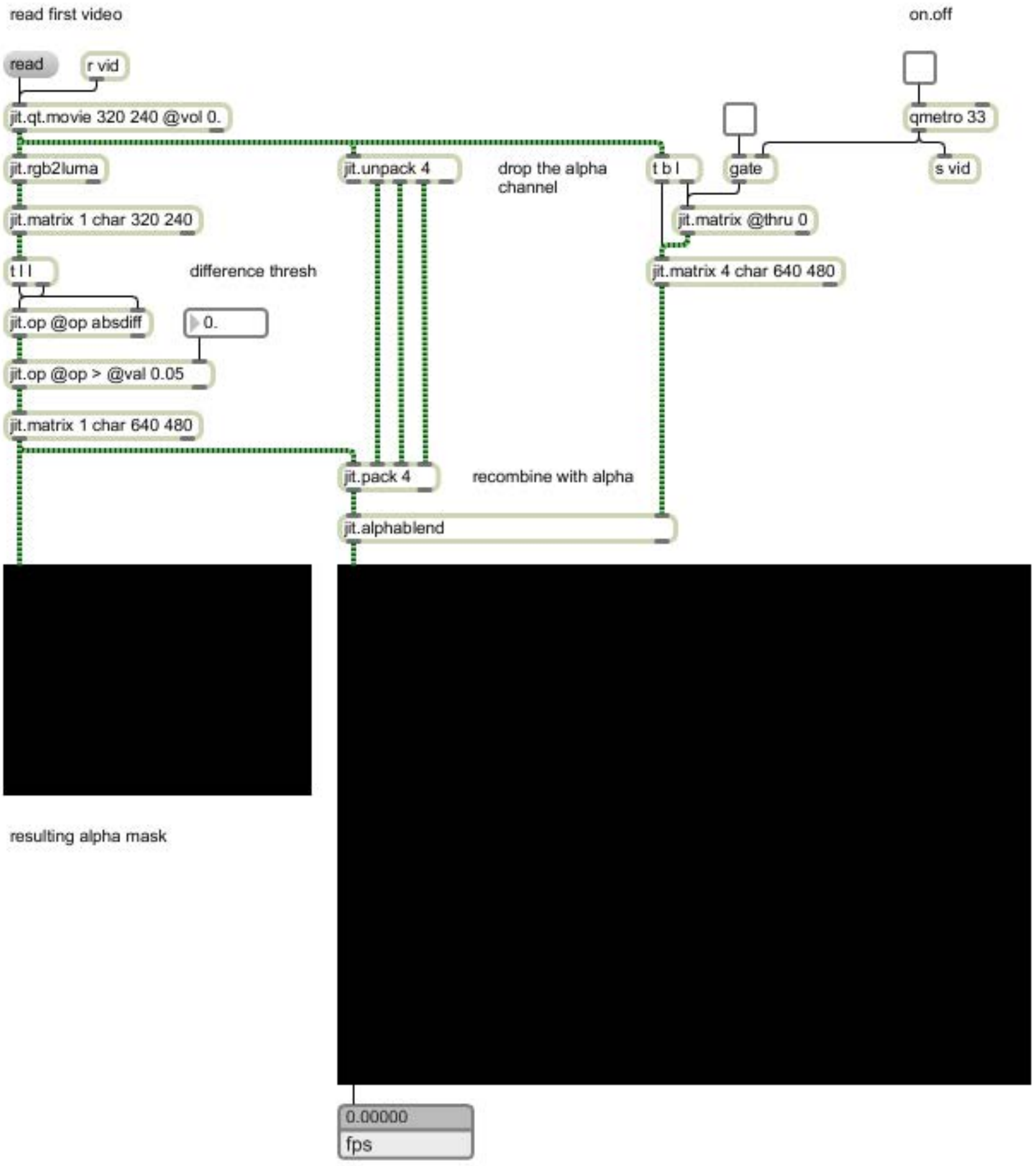


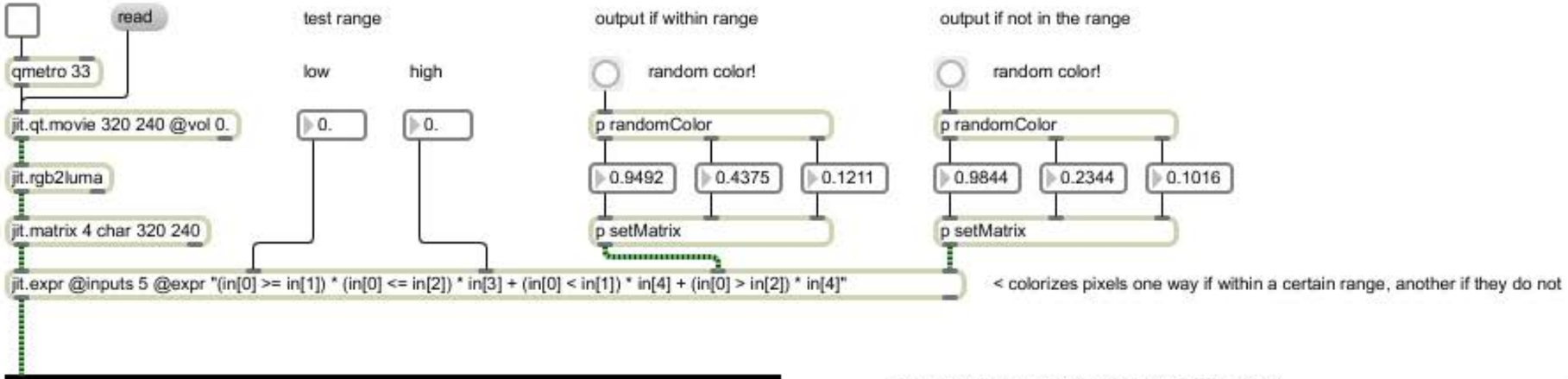
USING AUDIO DATA TO CONTROL VIDEO

Using the peakamp~ object to retrieve the current volume of a sound recording, those values are mapped to scale and rotation of a video.






read first video
on.off
read
r vid
jit.qt.movie 320 240 @vol 0.
qmetro 33
jit.rgb2luma
jit.unpack 4
drop the alpha channel
t b l
gate
s vid
jit.matrix 1 char 320 240
jit.matrix @thru 0
t l l
difference thresh
jit.matrix 4 char 640 480
jit.op @op absdiff
0.
jit.op @op > @val 0.05
jit.matrix 1 char 640 480
jit.pack 4
recombine with alpha
jit.alphablend
resulting alpha mask
0.00000
fps



THE BASICS OF USING JITTER EXPRESSIONS

Ok, so this is REALLY confusing, but quite powerful. Using jit.expr you can format if/then statements for entire matrices - something that would otherwise take lots of jit.op objects and as a result slowing down the frame rate.

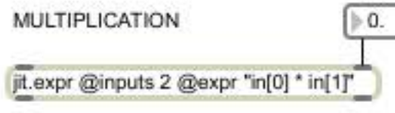
SPECIFYING THE NUMBER OF INPUTS
The attribute @inputs sets the number of inputs

BASIC EXPRESSIONS
Using the in[0], in[1], etc we can build basic expressions. For example:

ADDITION

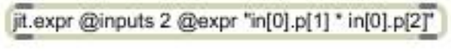
value or another matrix

MULTIPLICATION



value or another matrix

You can also apply expressions to single planes from an incoming matrix:



< multiply plane 0 by plane 1 of the same matrix!

SO HOW DO WE MAKE AN IF/THEN STATEMENT?

Above, we compare each pixel of the frame to a low and high range; if the pixel is within that range, it outputs one color, if not then another.

A basic if statement:
(in[0] >= in[1]) - if true this then returns 1, else it returns 0

Using this, we can create binary images of true and false; this can be easily acheived with jit.op.

However, turning it into an if/then statement:
(in[0] >= in[1]) * in[3]

This returns 0/1 just like above and multiplies the result by the third inlet; since anything times 0 = 0, we only get in[3] if the expression is true.

So what are the +'s in there for? Consider some possible outcomes for the three if/then statements in the expressions:

true false false
in[3] + 0 + 0 = in[3]

false true false
0 + in[4] + 0 = in[4]

... it is really confusing to think of it as a string of additions. In building these statements, consider how multiplying by 0 and adding the results can be used to set conditions for testing.