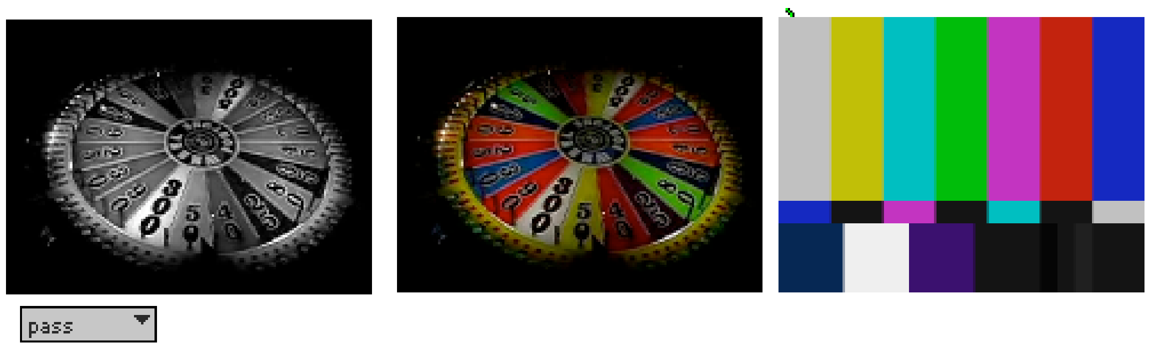
Jit.op Spotter

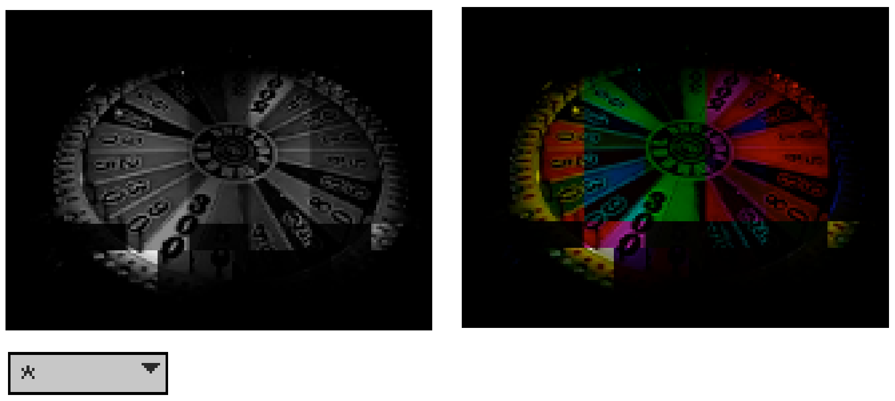
Jit.op is more intimidating than it needs to be, probably because it raises specters from high school algebra class. That's too bad, because jit.op is the most useful jit object. You can often use one to replace a more complex effect, greatly improving the frame rate.

One potentially confusing point is the way matrices of chars are treated. In many cases, chars with the range 0-255 are converted to floats in the range 0.0 – 1.0. This adds to the usefulness of the object (multiplying images by 0.5 cuts the level in half) but will give some surprising results. Constants are modified as necessary to be consistent with the effects of images.

This document runs through the ops and shows the image effects easily available.

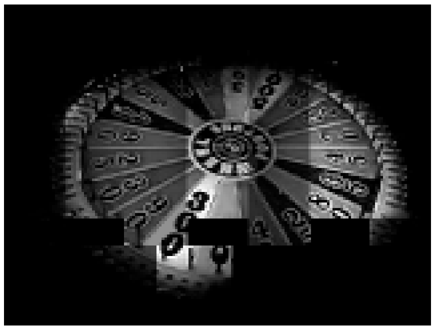
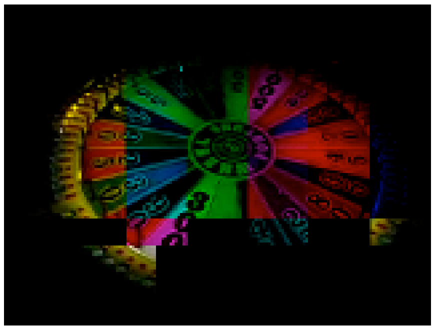


The original images – Some of these processes show better with grayscale images, so I'm showing a B&W and color version- otherwise it's just the help file. The B image is on the right. That (or the gray equivalent) is applied to the right input of jit.op.



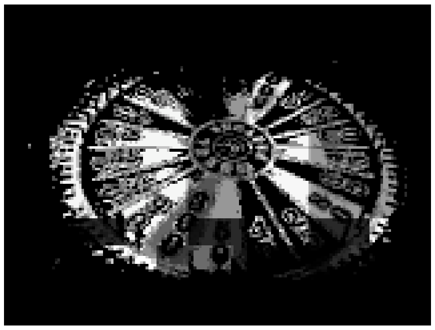
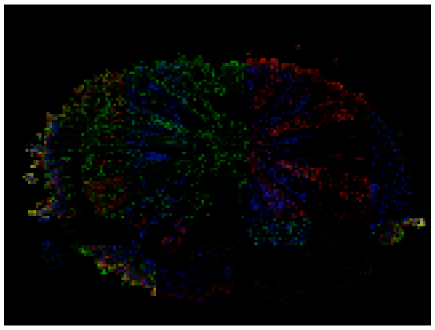
Multiply. The images are usually dim, because jit.op converts the rgb chars 0-255 to floats 0.0-1.0. The product of two numbers that are less than 1.0 is smaller than either. It will often be useful to preprocess one image by adding an offset (using another jit.op, of course).

Op Spotting

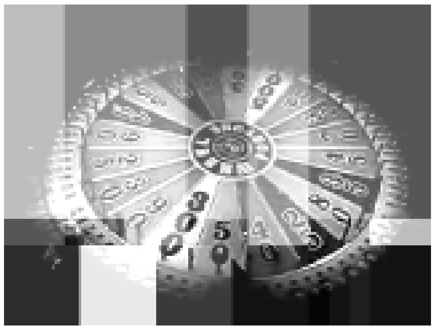
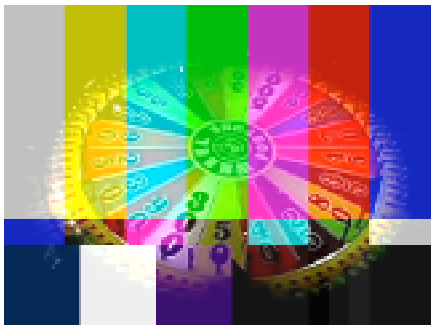
/

Dividing usually gives black unless the B image is very dim. To get this, I multiplied the bars by 0.01.

/

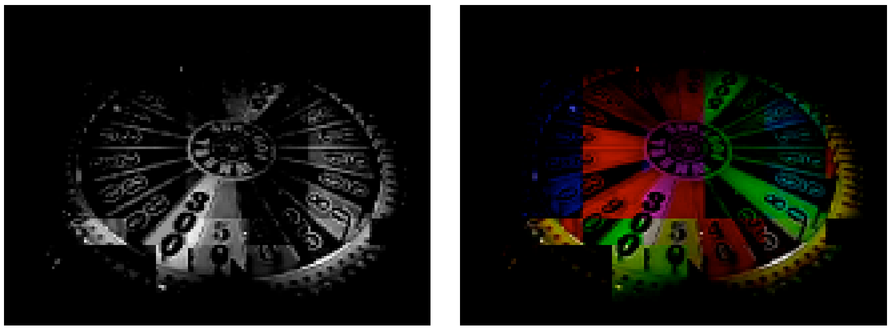
Exchanging the inputs gave me this.

+

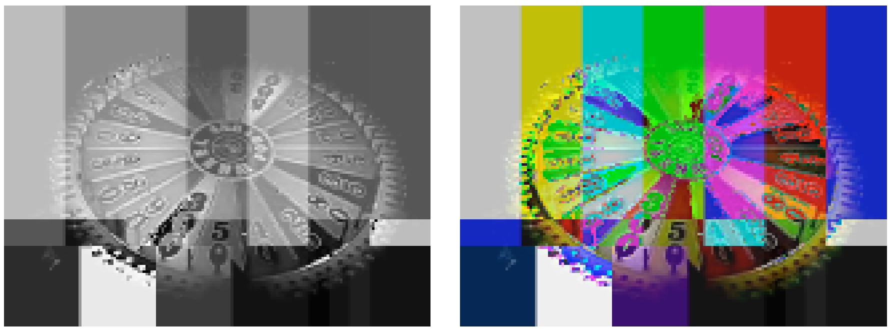
Addition gives a mix. This works as well as keying if your images contain a lot of black. Points that add up to more than 255 will give white or saturated colors.

Op Spotting



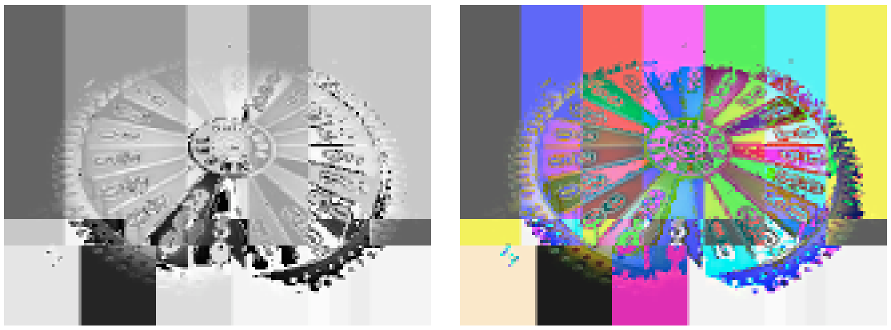
-

Subtraction will tend to give black results. Exchanging the images can be quite dramatic. (See !-)



+m

In modulo addition, results that exceed char 255 are wrapped back around 0. What was white becomes black.



-m

In modulo subtraction results below 0. are wrapped to be below 255. Note how the color bars change where (for instance) blue is subtracted from black.

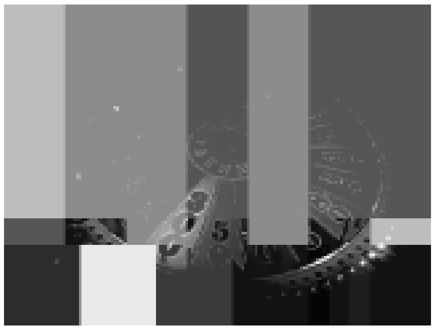
Op Spotting

%

The modulus operation divides the left by the right, but outputs the remainder. 53 % 50 gives you 3. So does 103 % 50.

min

Minimum lets the lesser values through. Note you get the black holes from the bars. If you crank up the contrast, you can matt against the light areas of an image.

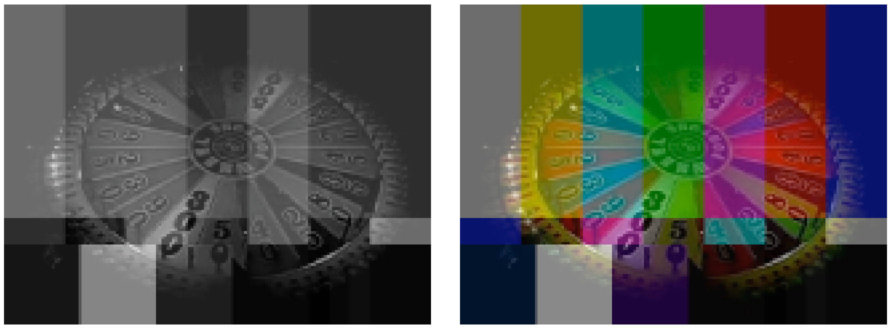


max

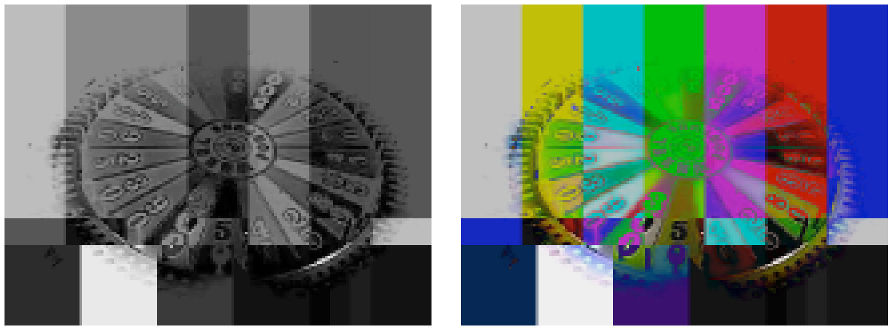
Maximum goes the other way. Note that in the black areas of the bars, the wheel is unaffected.

abs Since the char data type is unsigned absolute value does nothing.

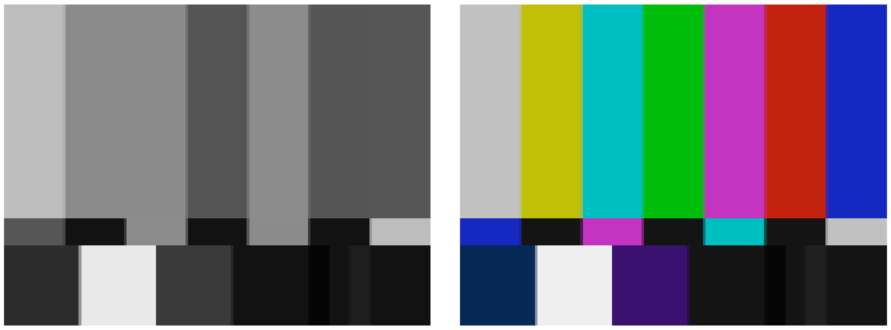
Op Spotting



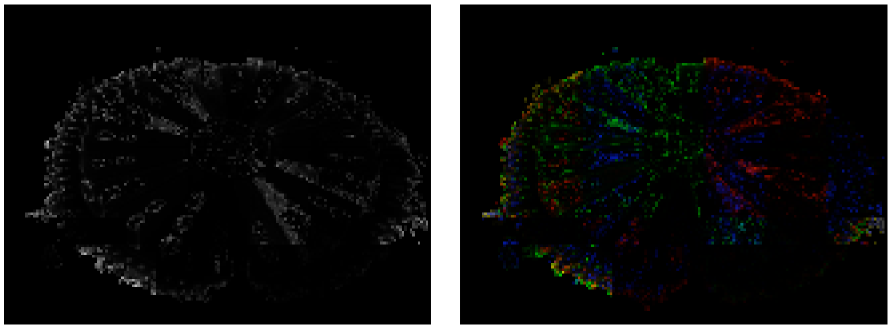
The average of pixel values is a washed out mix.



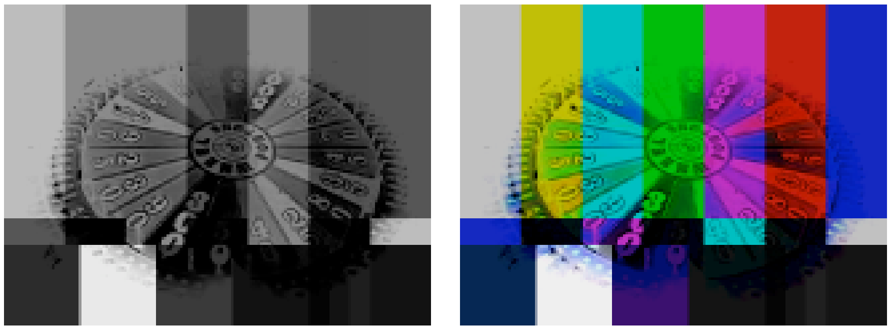
The absolute difference will give either image where the other is black, inversions where one is white, and black where they are the same.



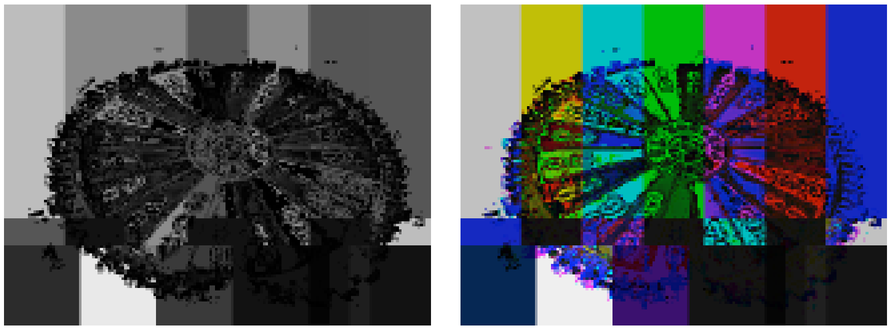
I'm tempted to call this don't pass, because ! means not in mathematics, but in Max ! means reverse inlets. So this passes the right input.



Reversed Division.

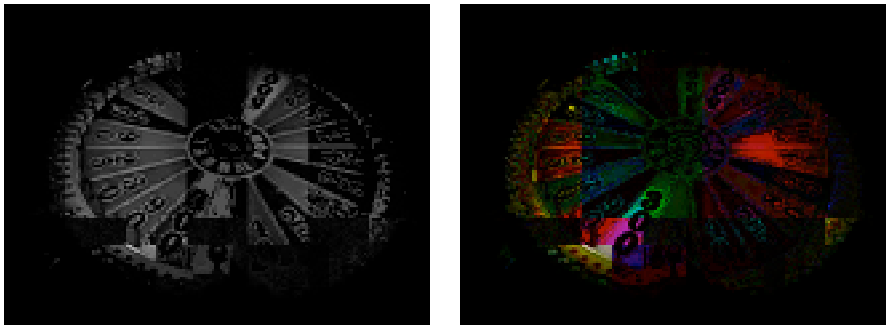


Reversed Subtraction.



Reversed Remainder.

Op Spotting



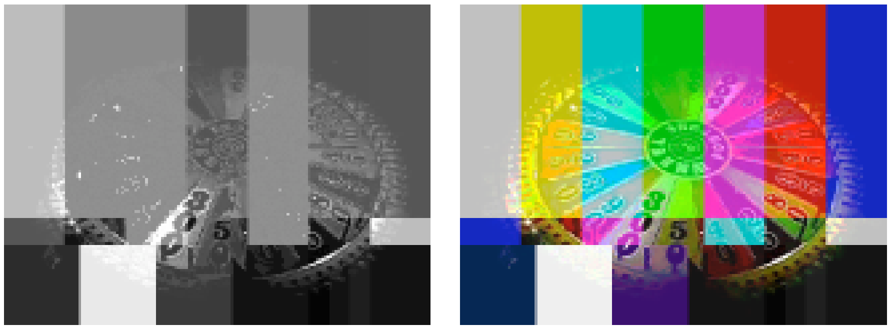
&

This is bitwise AND, where 10101001 & 11010111 = 10000001.
Slight changes in values give big jump in results, so images are broken up.

Logical truth tables:

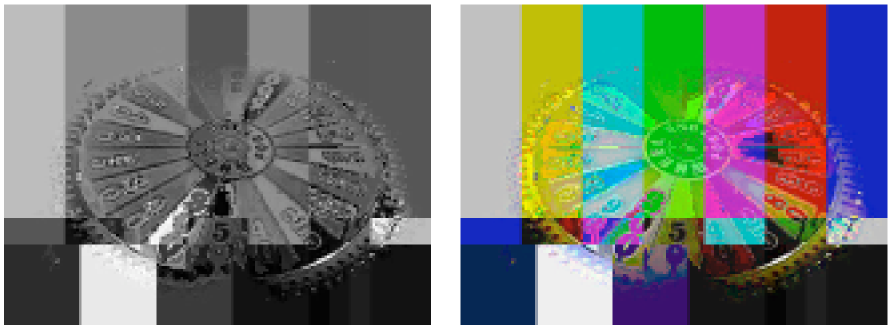
A	B	A AND B	A OR B	A XOR B
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

And of course NOT 1 is 0 and NOT 0 is 1. Bitwise NOT is called compliment.



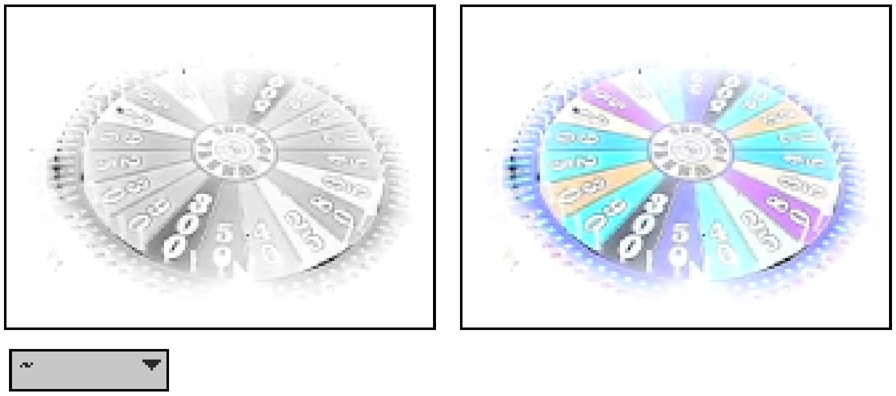
|

OR'ed images tend to be light, as all 1s survive.

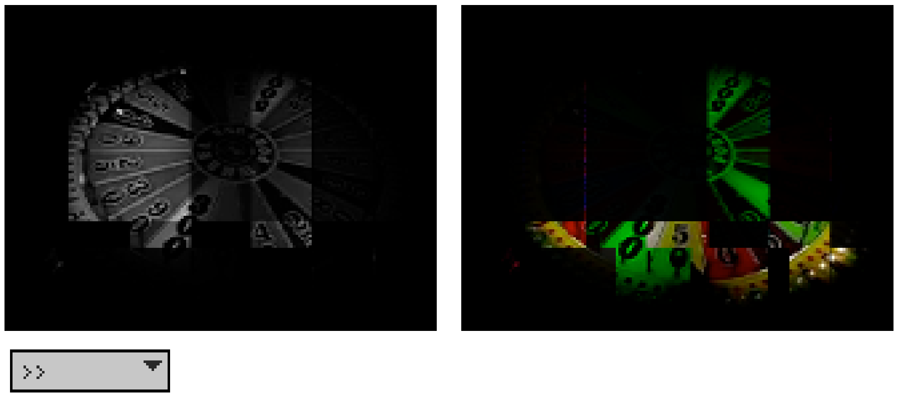


^

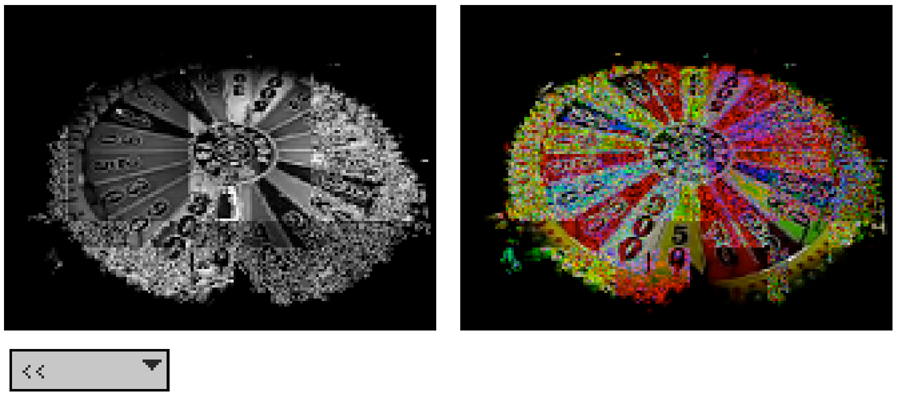
Exclusive OR (XOR). Order of images does not matter in bitwise operations.



Compliment of left image.

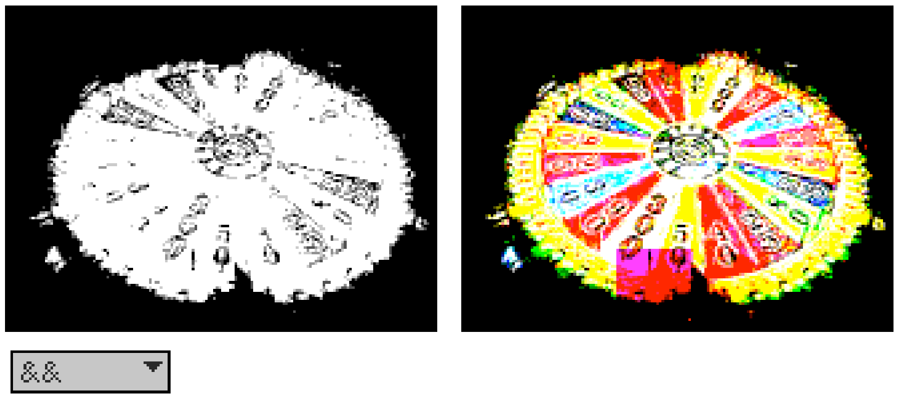


Right shift is effectively divide by a power of two. The right input determines how far to shift, but when the shift exceeds 8 (which will always yield 0) the right input is wrapped modulus 8. Right images with gradual shadings give interesting striated effects.

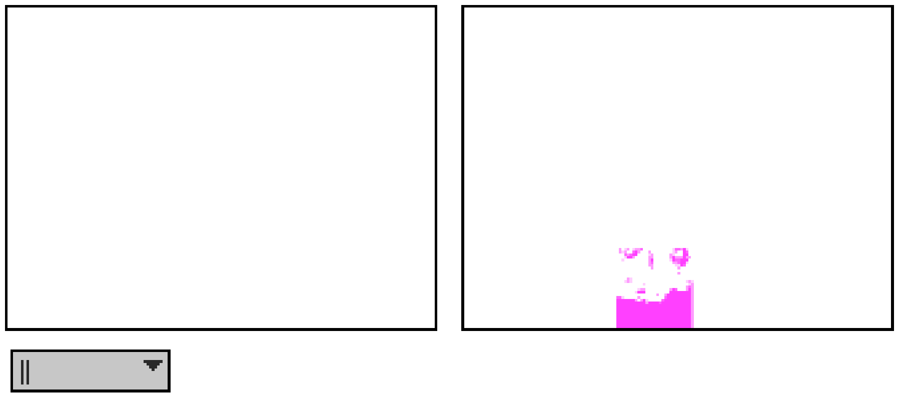


Left shift is multiply by a power of two. Note that black shifted any distance remains black.

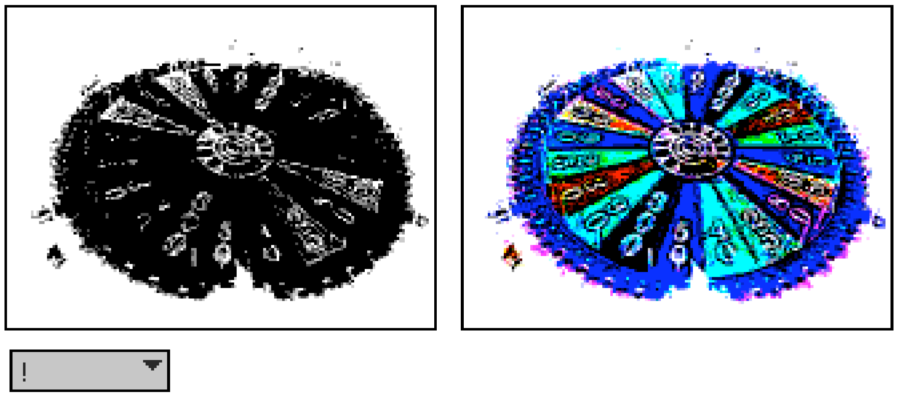
Op Spotting



This is logical AND, where any number that is not 0 is considered 1, and the output is 0 or 255. This is a great way to make masks for use with the min or max ops.

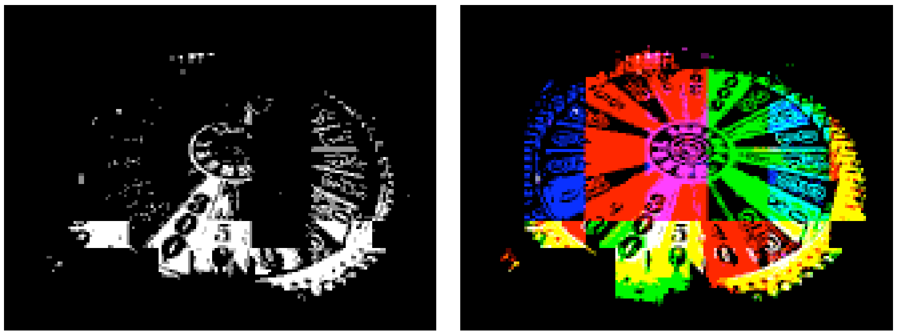


Logical OR gives mostly white.

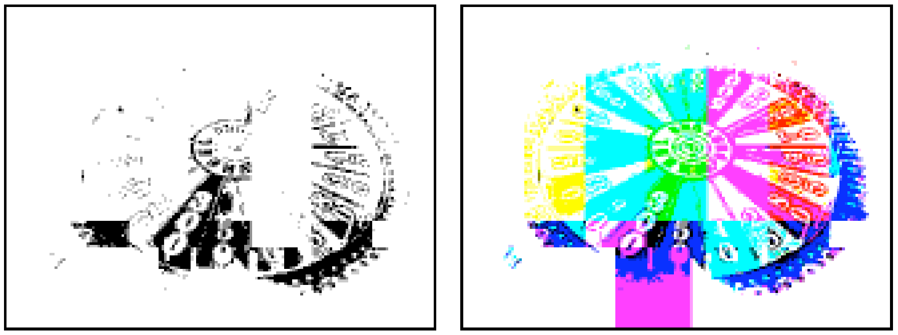


This really is logical NOT. The right input is not involved.

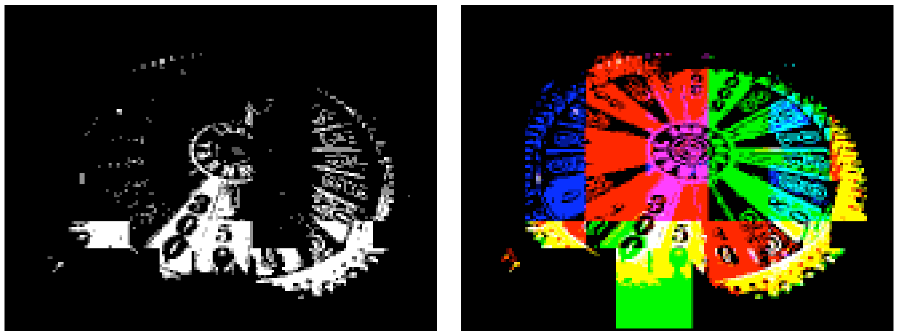
Op Spotting



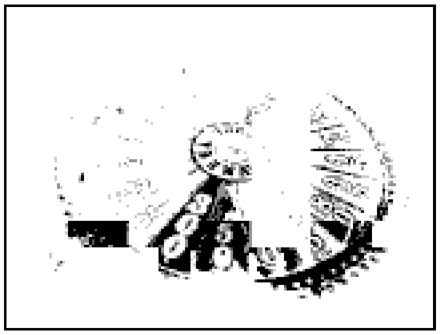
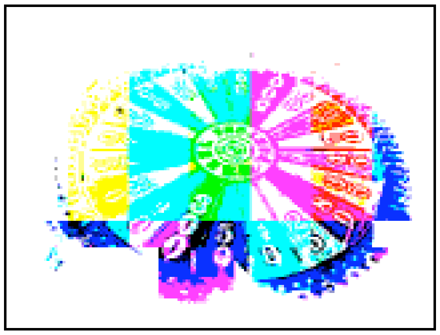
If left input is greater than right, you get 255.



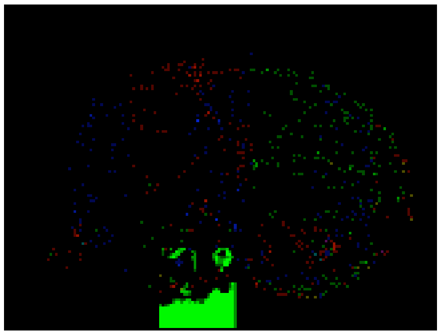
If left is less than right, you get 255.



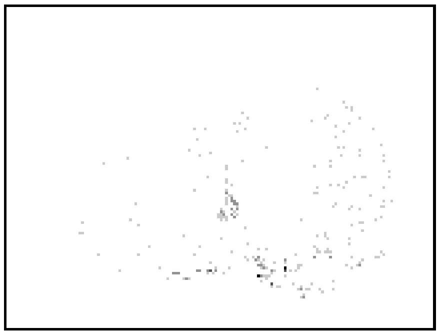
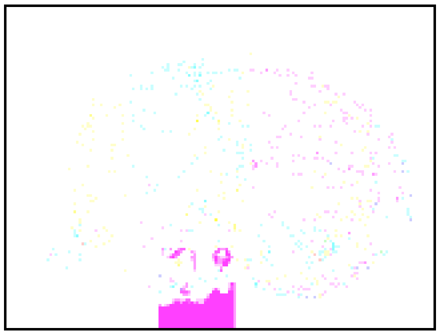
If left input is greater than or equal to right, you get 255. Can you spot the difference between this and <?

Less than or equal to gives pretty much the same results except where black meets black.



The equal test seldom comes true with images.

Not equal gives mostly white, unless you produce the right from the left.

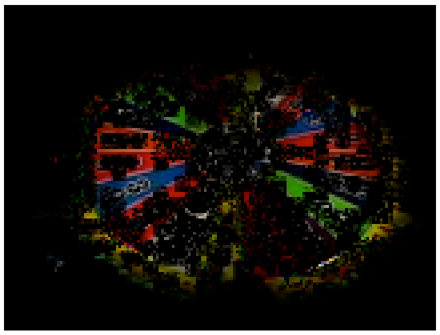
Op Spotting

If left is greater than right, it is passed through. Otherwise, black. That makes it different from max.

<p

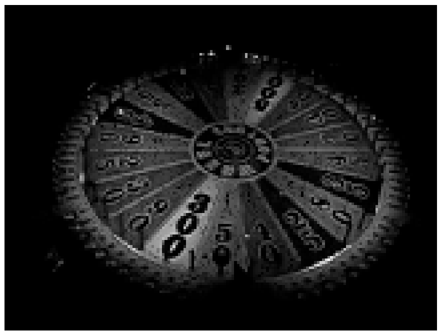
If a left input pixel is less than the corresponding right pixel, it is passed.

>=p   <=p   These two are only subtly different from the above.



==p

Pass when equal, will give you a black screen most of the time, unless the images are very similar. These are both the wheel, offset by one frame.

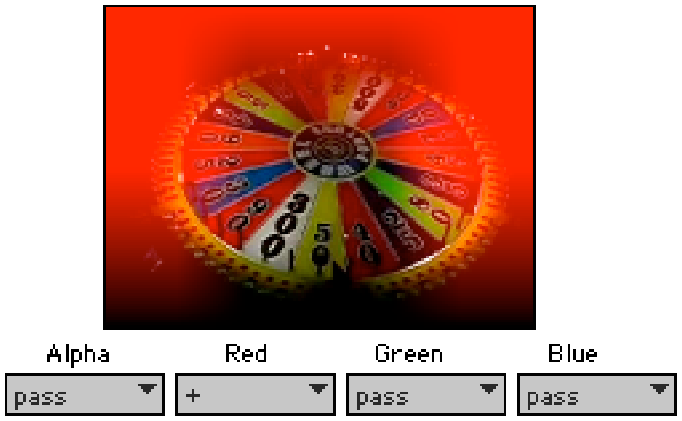


!=p

Even the same movie is not equal to itself most of the time.

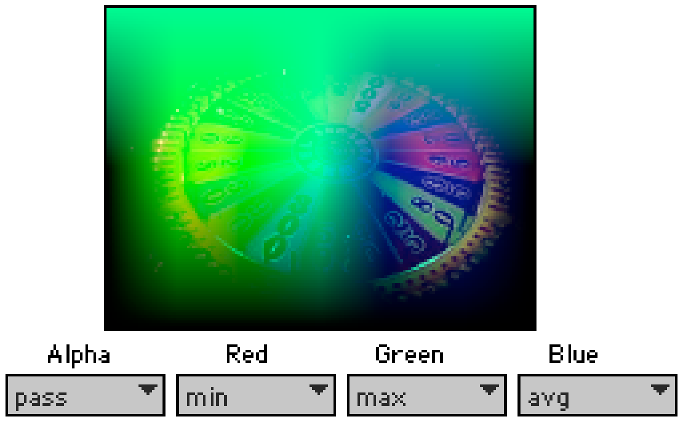
This is not the end of the jit.op ops, but the remaining functions don't do anything to images.

Plane by plane processing



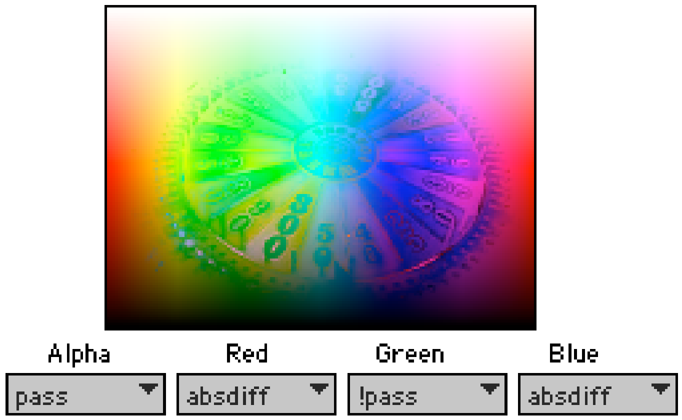
Alpha	Red	Green	Blue
pass	+	pass	pass

The help file shows how to send multiple ops to get different effects on each plane. Here I've replaced the color bar with the color swatch (for variety) and added on the red channel.



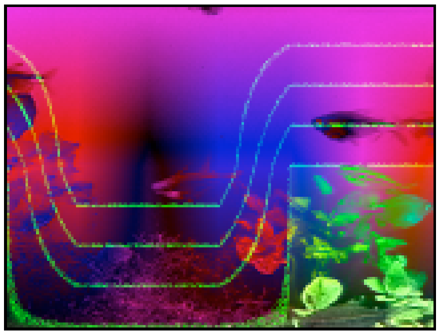
Alpha	Red	Green	Blue
pass	min	max	avg

Variations:



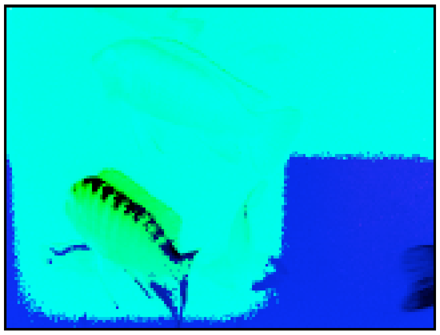
Alpha	Red	Green	Blue
pass	absdiff	!pass	absdiff

Op Spotting

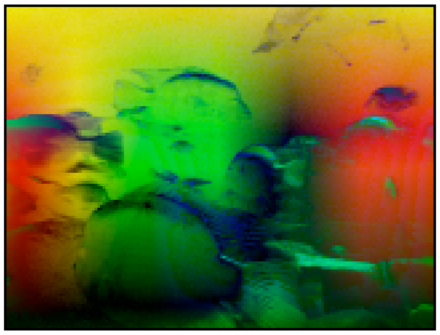


Alpha	Red	Green	Blue
pass	absdiff	<<	absdiff

Each image has its own possibilities.

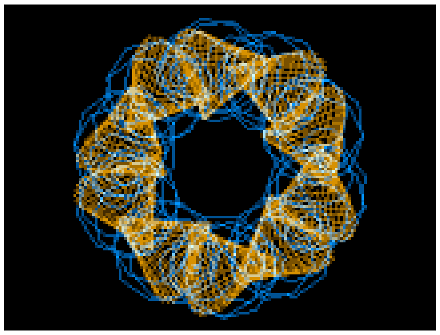


Alpha	Red	Green	Blue
pass	!/	<	~



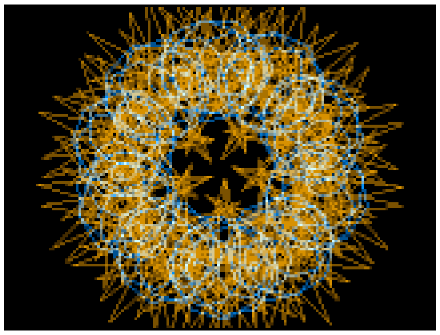
Alpha	Red	Green	Blue
pass	!-	absdiff	!%

Op Spotting

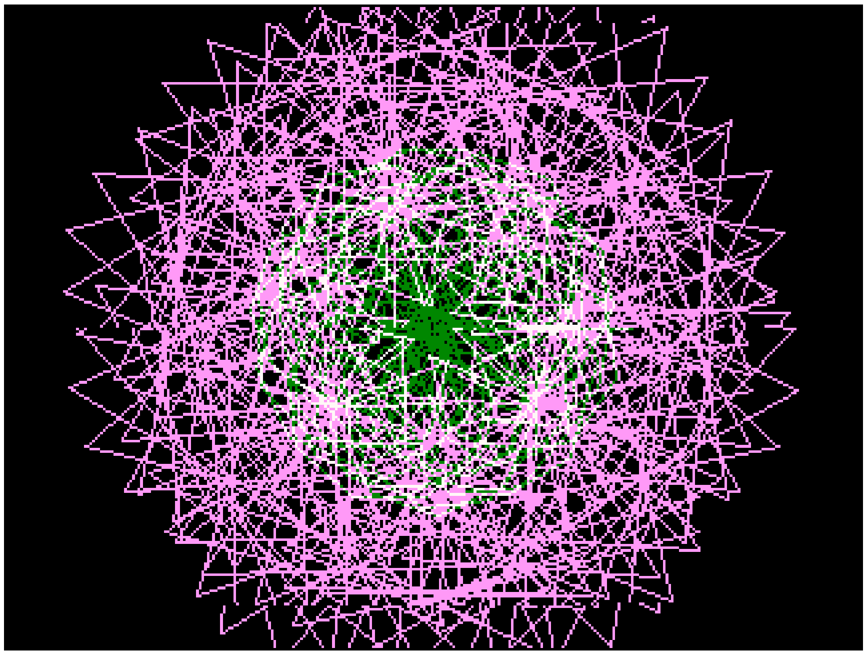


Alpha	Red	Green	Blue
pass	pass	avg	!pass

These are great for combining jit.lcd drawings (see drawing in jit.lcd)



Alpha	Red	Green	Blue
pass	pass	avg	!pass



Alpha	Red	Green	Blue
pass	||	avg	pass