Image Compositing in Jitter:

There are many ways to combine jitter matrices to produce composite images. One of the most basic is the wipe, which progressively replaces one image with another. The best tool for this is jit.qt.effects, and is explained in the effect spotter tutorial. Here are some details on various objects dedicated to compositing.

Jit.op

Simple math using jit.op produces some sort of overlay. The images blend or interact. In the case where most of each image is black, addition is all you need. (See the op spotting guide for a complete rundown of these effects)

Figure 1.

Many of the jit.operators will have no effect because they don't apply (or apply strangely) to type char. The fact that each layer is processed independently creates odd colorations. The effect is avoided by using jit.alphablend, demonstrated below. Note that this patch contains sends for the two movies and the metro tick to simplify the patches that follow. The dishes are on the left.

Crossfades

Figure 2.
Crossfading (as the jitter tutorial explains) is a simple multiply of each input and addition of the results. The xfade object does a fine job, and the qt.effects package has a somewhat more complex version. The control ranges from 0.0 (all left) to 1.0 (all right.). There are many more applications for crossfading than mixing the final image. Figure 3 shows how crossfade can give composite control images for jit.repos:

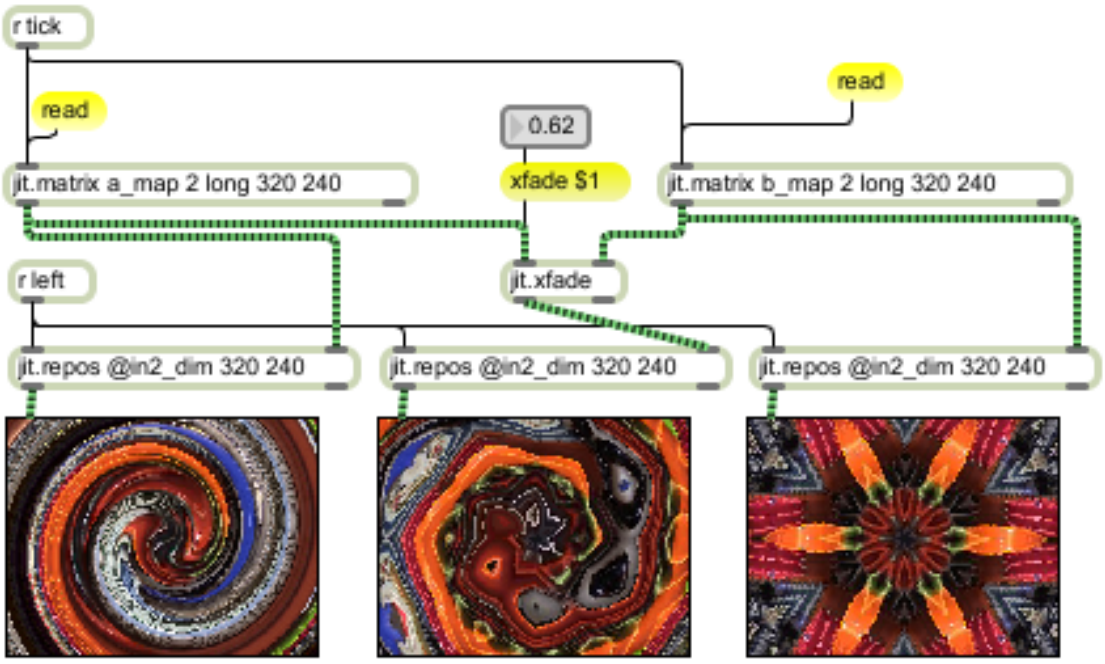

Figure 3. repos with spiral crossfaded with kaleidoscope
(To find out what's happening here, see the repos tutorial.)

Alpha blend

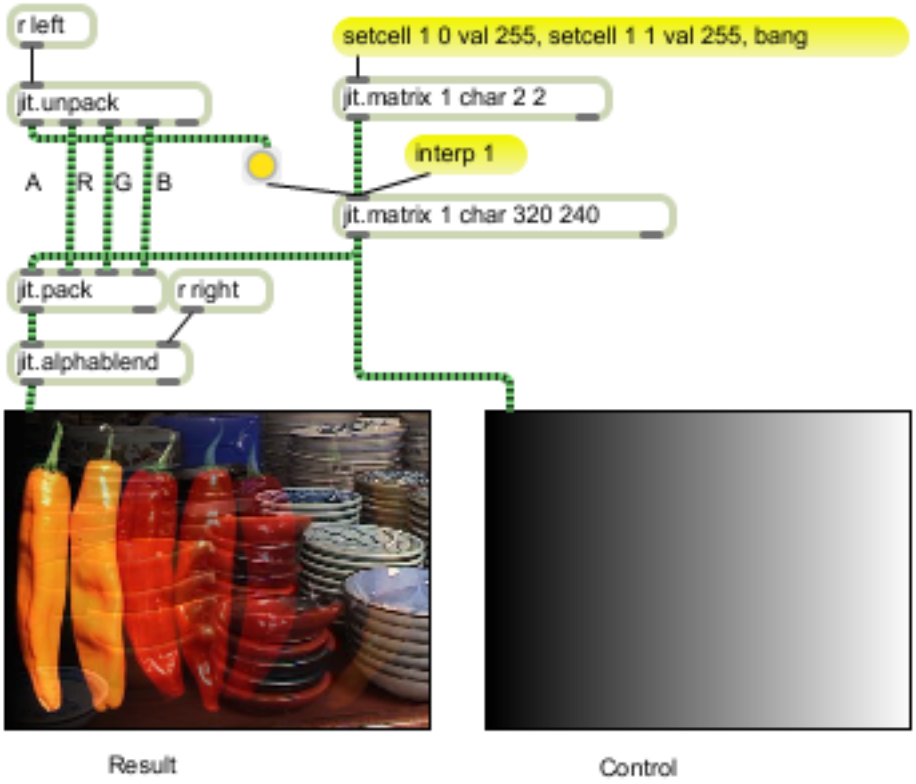

Figure 4.
The alpha channel can be a bit of a nuisance, because most of the operations process it but it is never used. (In fact, if you have a lot of jit.ops, you can gain speed by specifying pass as the first operator in a set of four.) Here is an object that does use the alpha channel. Most sources don't have any alpha information, so we construct some, or at least replace what's there with something of our choice. The gradient (constructed by using the jit.matrix interpolation feature) controls the mix of the two images on a pixel by pixel basis.

Chroma Key

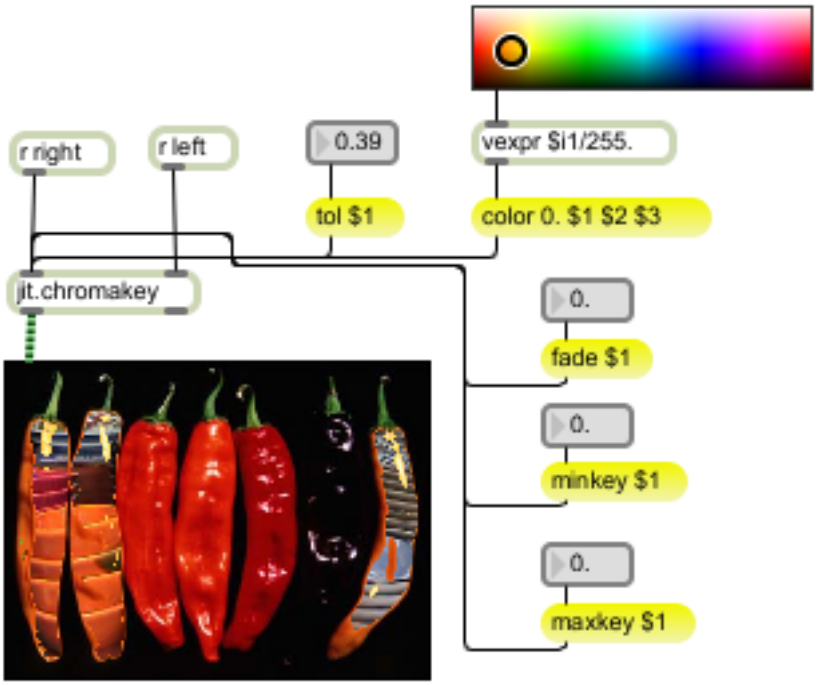

Figure 5.
Key operations differ from blends in that you get pixels from one image or the other, with little or no transition. In chroma keying, the choice is dictated by the color of the primary image. Any part of the primary image that is the key color will be replaced by the secondary image. Precise match of color to the key is unlikely, so there is a tolerance factor. In figure 5, the key color is a sort of orange, so the orange chilies are replaced.

The abrupt switching often gives a rather harsh effect, with orphan pixels scattered around the screen. To soften the edges, there is a parameter called fade. This number is added to the tolerance. Areas within fade + tolerance but not within tolerance of the key color get a transition between the two images.

Minkey and Max key have the effect of crossfading between primary and secondary images. Minkey applies to the areas not switched, and defaults to 0.0, leaving the primary image. Increasing it to 1.0 will fade in the secondary image. Maxkey applies to the regions switched to secondary. Reducing it from 1.0 will fade back to the primary.

There is also a mode function. Mode 0 is the usual operation. In mode 1, the control matrix is output. This is the difference between the key color and the primary image, with the tolerance and fades applied. This is useful for operations that require a mask. Mode 2 sends the primary image with the control data in the alpha channel.

Luma Key

Figure 6.
Luminance keying is based on the brightness of the primary image. Luma key fills the dark areas first, so I switched the chilis and dishes. With a lum value of 0, only the black parts of the chili image are replaced by crockery. Look what happens if lum is increased:

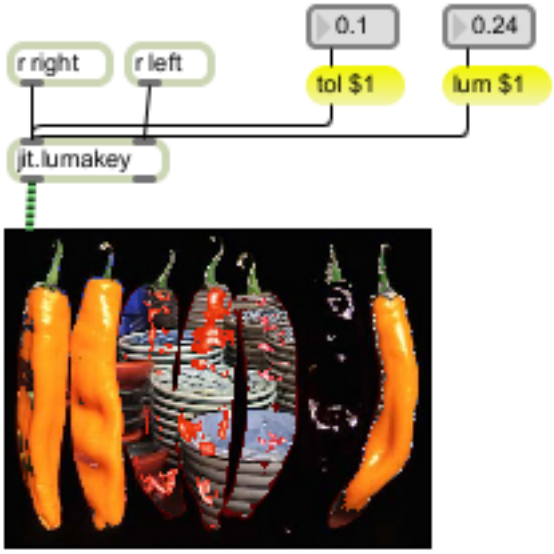

Figure 7.
tol and fade work pretty much like their counterparts in chroma key. You also have arguments to adjust the relative contributions of R G and B to the luminance. It's not equal, because the eye responds differently to the colors. The defaults, with bscale at 0.11 and gscale of 0.59 demonstrate the standard difference.

KeyScreen

Figure 8.
Jit.keyscreen allows even more control, but can be tricky to use. The fundamental operation is that one input is used to create a key shape (based on color in the input image), that is used to shape one image and overlay it on another.
The three images are called key, target and mask. These default to left, middle and right inputs, but can be reassigned. The key shape is based on color matching as in chromakey. Where the key image matches the key color, you get the target. Where it does not, you get the mask. In other words, the key color defines a hole in the mask image for the target to show through.
(Unfortunately, the description in the help file and HTML documentation is a bit confusing on this point.)

In the default case, keying will happen on black as shown. The lcd drawing is in the key inlet, the dishes are in the target inlet, and the chilis in the mask inlet.

I get best results with unambiguous colors as shown in the example. There are individual tolerances for each plane, and it's a good idea to set alphatol to 1.0. Keying is a simple switch of cells, but there is a mode to make it more interesting.

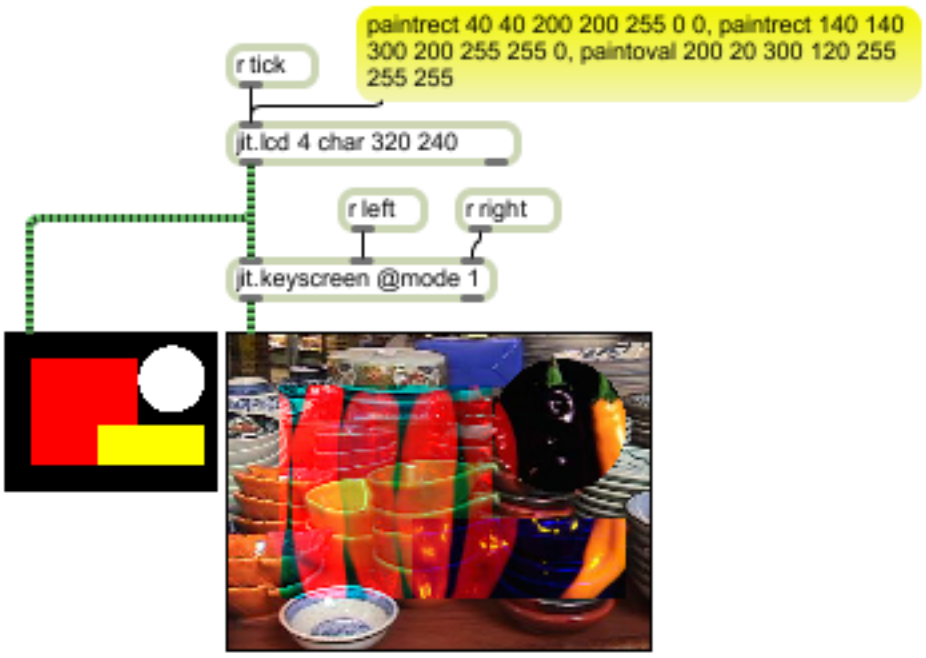

Figure 9.
With mode set to 1, keying is done plane by plane. Any non zero value passes the target value for that plane.

Picture in Picture

As a wrap-up, here's how to use a simple jit.matrix to do a rectangular insert.

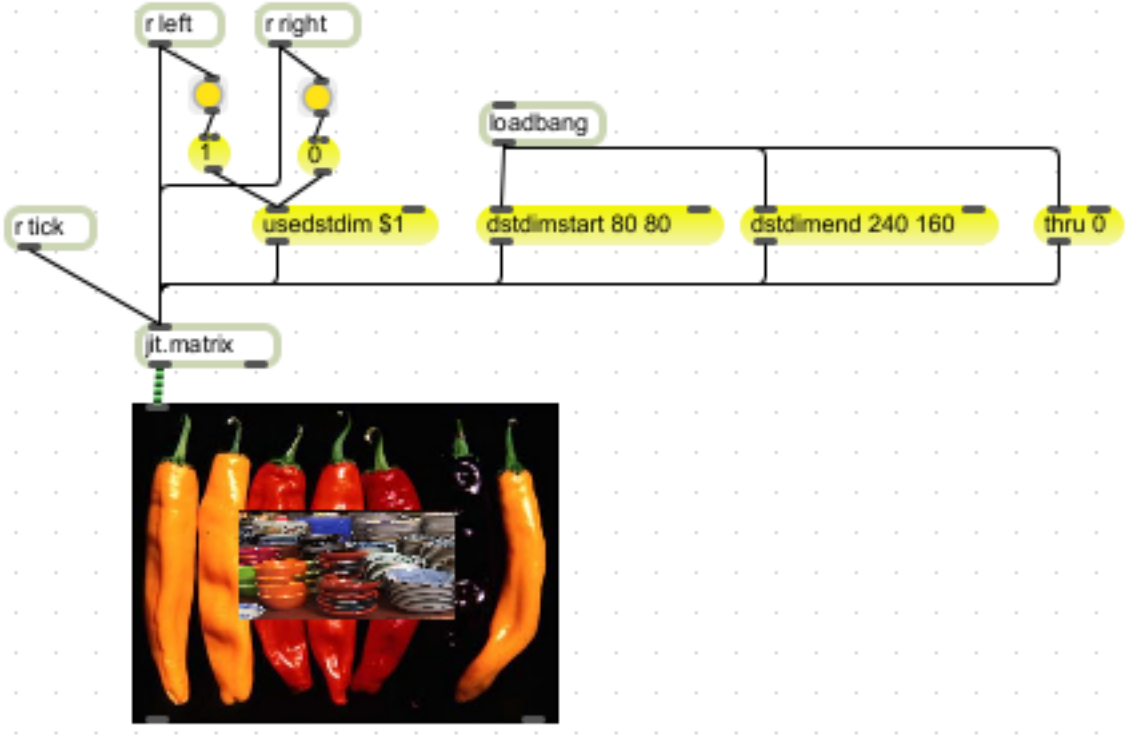

Figure 10.