Symmetrical Drawing with jit.lcd

A lot of art pieces are symmetrical in some way. If full symmetry seems too rigid, a composition may still be made of symmetrical elements. Jit.lcd is a good way of creating symmetrical drawings.

The most flexible way to draw is to build images up pixel by pixel.

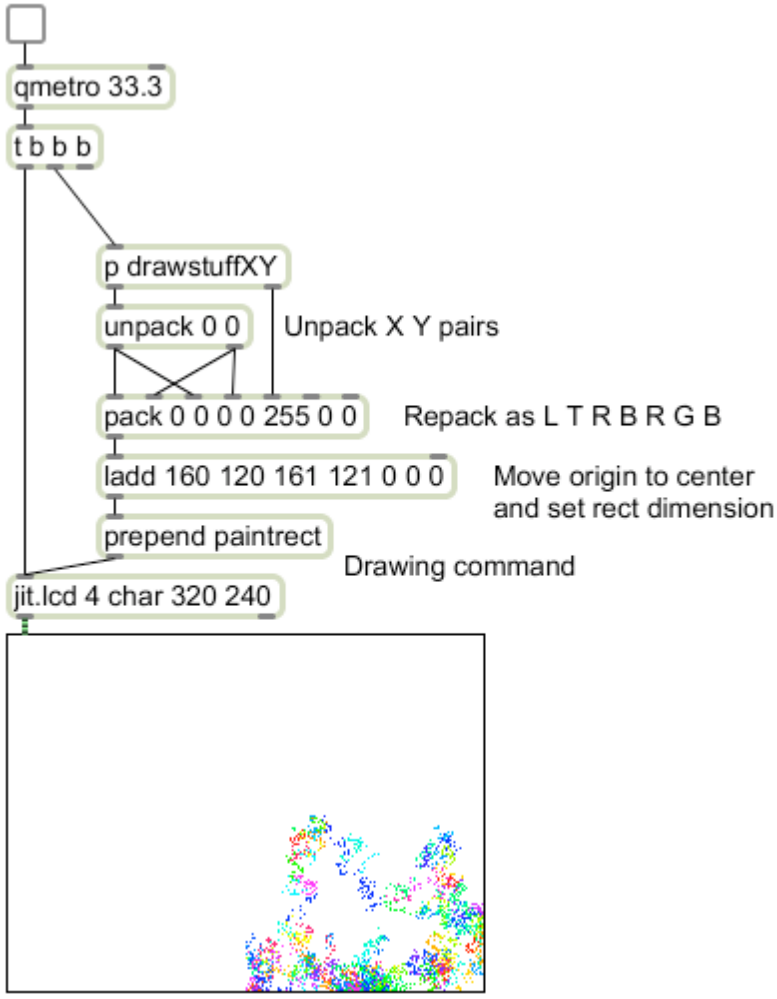

Figure 1.

Figure 1 shows a basic patch for pixel drawing in jit.lcd. Everything is driven by the qmetro, which bangs once per frame. The trigger provides three bangs in controlled order (the third will be used later.) The subpatcher drawstuff contains the calculations that create the image. It has two outlets, one for lists of X,Y pairs, the other for lists RGB color values. Anything at all can be in there, as long as the outputs are formatted right. This patch assumes that anything generated will have an origin in the center of the image, with negative values for X and Y when necessary.

The XY pairs are unpacked and then repacked to provide the numbers for left, top, right, and bottom required by the paintrect command. The RGB list can be used as is if it is attached to the last but third outlet. The Ladd object serves two functions. It moves the drawing to the center of the window and adds 1 to the right and bottom values to give the rectangles about to be painted a size of 1 by 1. I split these functions out from the drawing subpatcher in order to make the image mobile-- by changing the list in Ladd, the image can be moved around the screen and the size of the pixels can be changed.

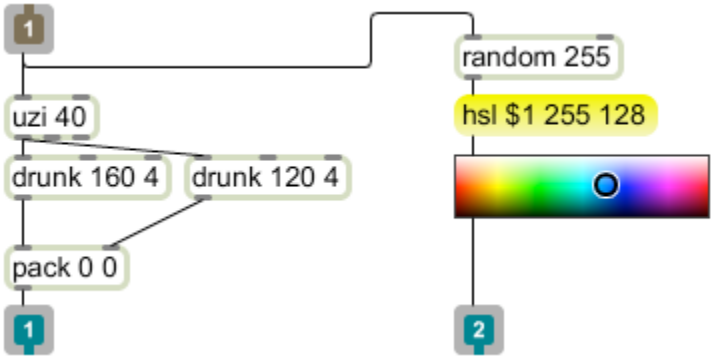

Figure 2.

Figure 2 shows a typical drawstuff subpatch. This just puts some random blobs in the window. The use of drunk will keep the blob pretty much contained, whereas random would scatter dots all over. Note that this only generates positive numbers. Figure 1 shows how this is confined to one quarter of the screen. (It doesn't have to be.) Figure 3 shows the addition of one object:

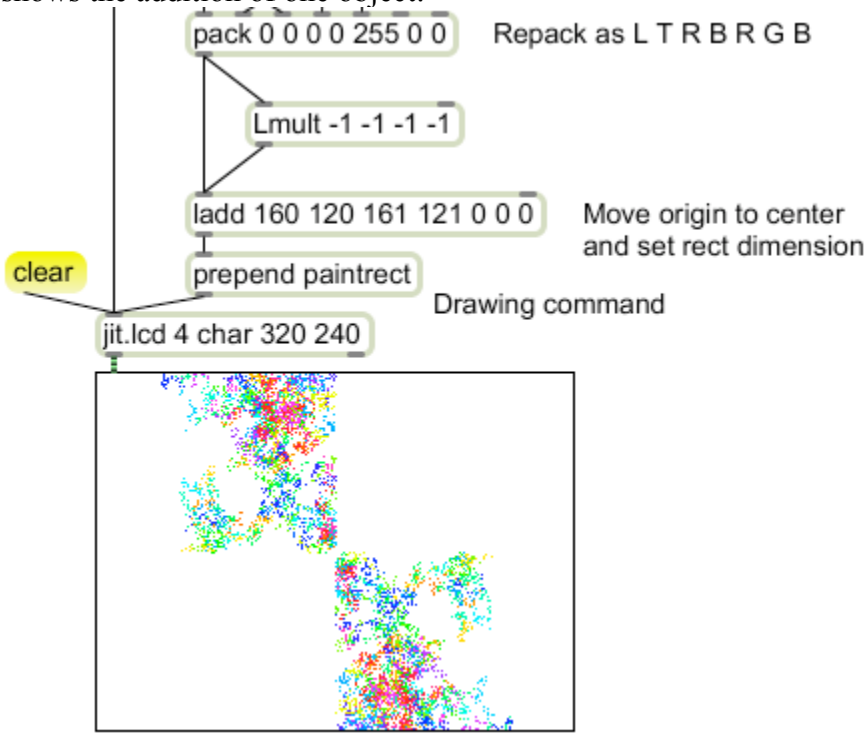

Figure 3.

The Lmult object added just below the pack object duplicates every drawing command with the signs reversed. Points with negative X and Y coordinates will be drawn above and to the left of the origin. Adding two more Lmult objects will fill the window:

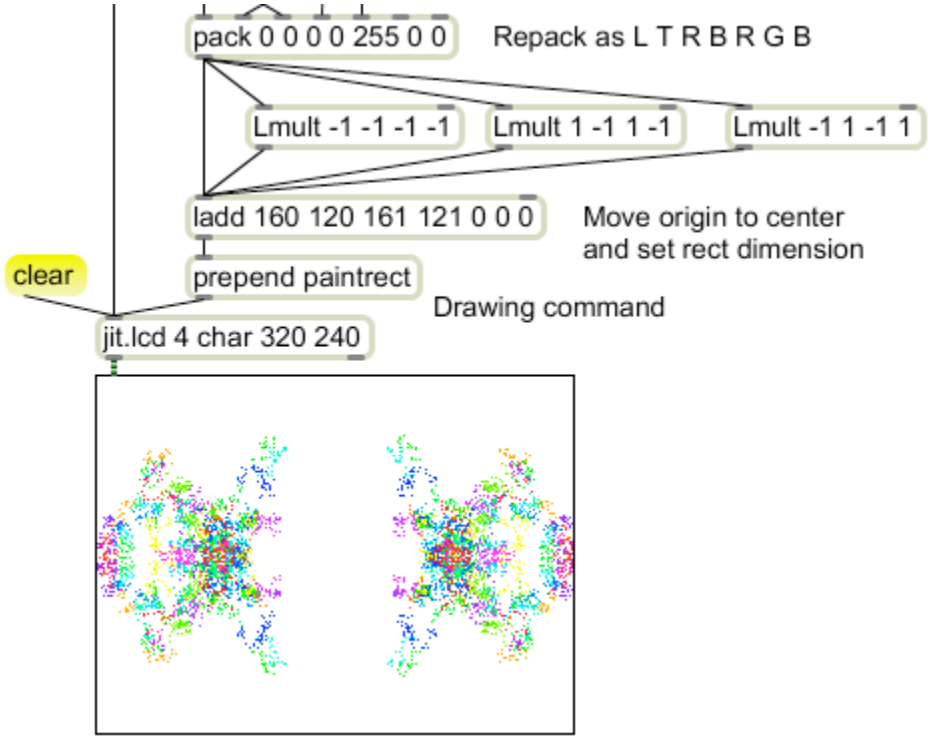

Figure 4. Quadrilateral symmetry

Figure 4 shows quadrilateral symmetry. Each point is now drawn 4 times. Can you see how bilateral symmetry can be achieved with fewer objects?

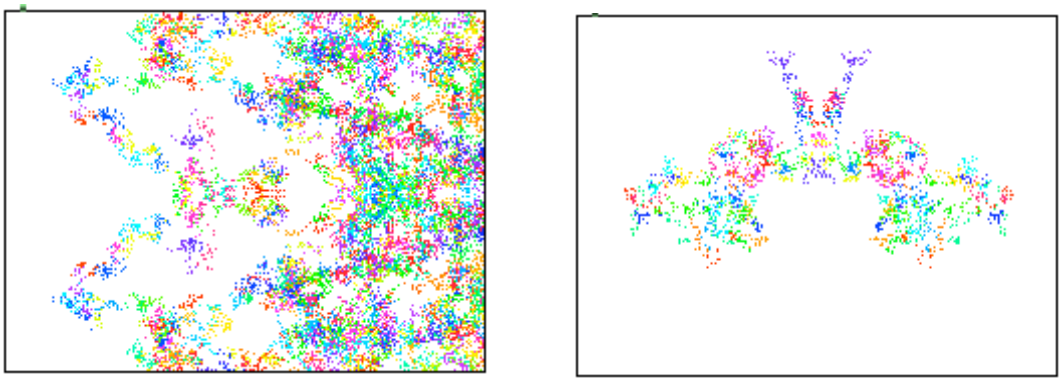

Figure 5. Bilateral symmetry

Radial Symmetry

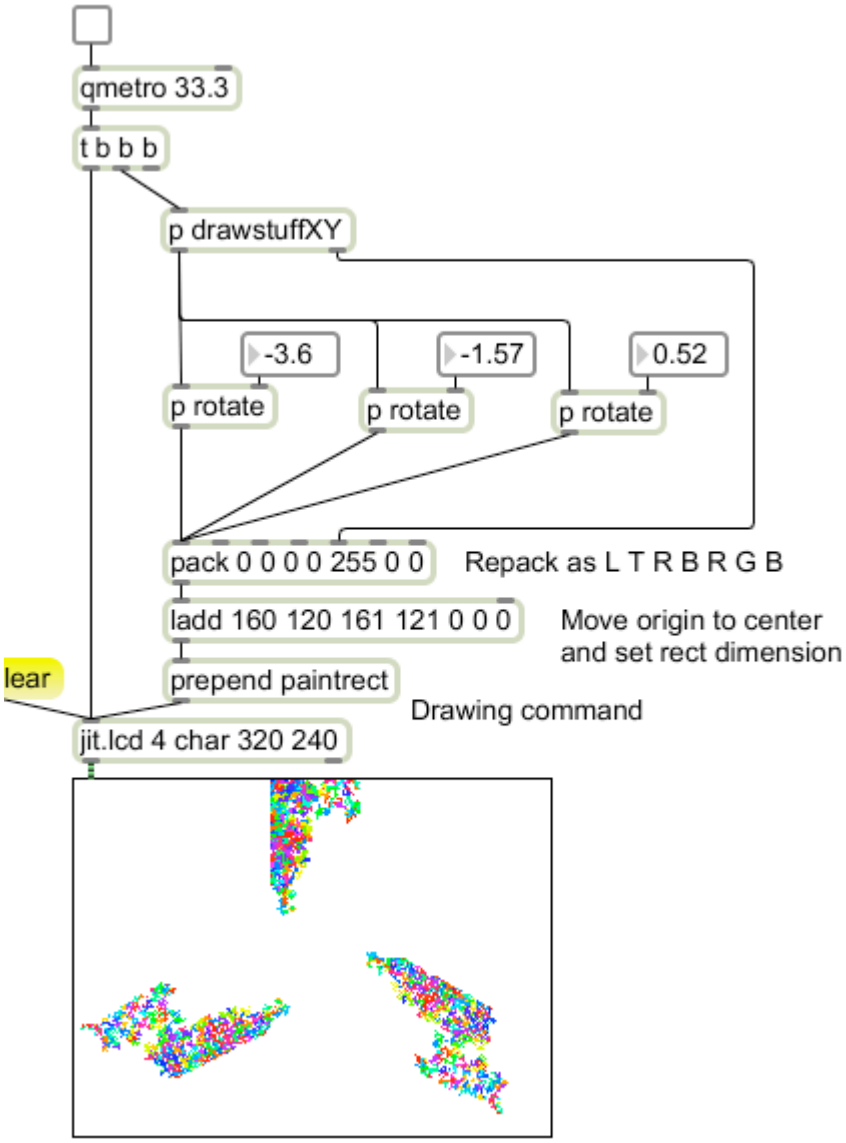

Figure 6.

Figure 6 shows the patch of figure 1 modified for radial symmetry. The objects between the drawstuff subpatch and the big pack have been replaced by subpatchers that take the drawing coordinates and rotate them by a desired angle. Figure 7 reveals the contents:

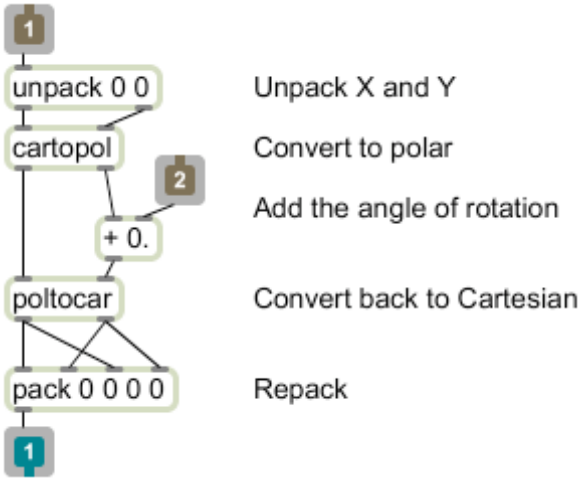

Figure 7. The contents of the rotate subpatch

The X and Y coordinates that the drawing patch throws out are converted to radius and angle of the polar coordinate system[1]. The angle is changed and the results converted back. Packing the numbers into an XYXY list just simplifies the main patch. The only difficult thing about this is figuring out the angles. Divide 2π by the number of arms you want, and add a common value to all angles to rotate the entire figure.

You will often combine lateral symmetry and radial symmetry in the same patch.

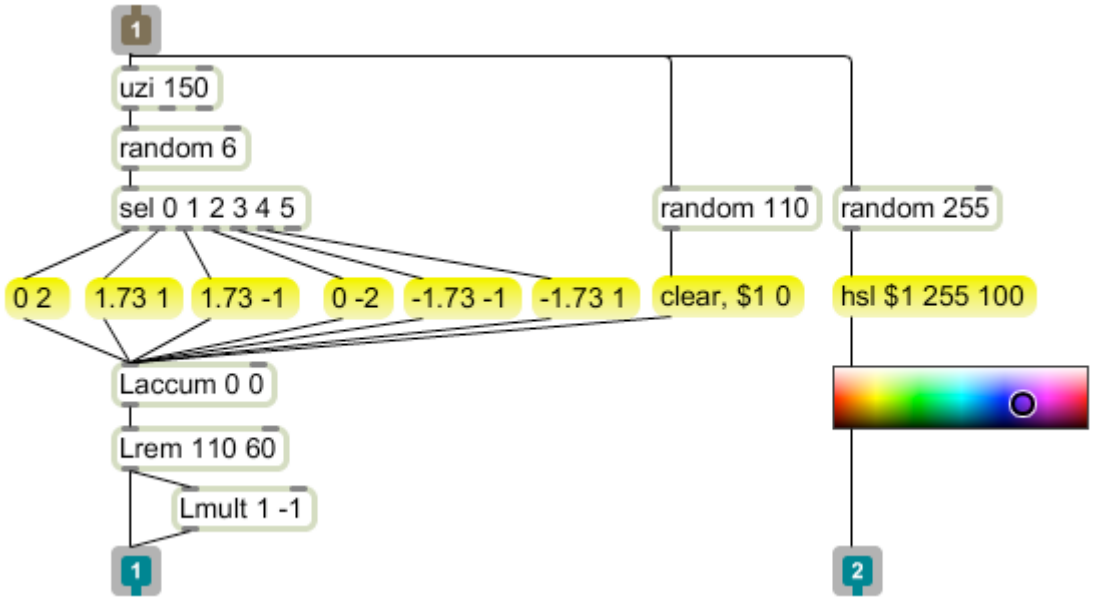

Figure 8.

Figure 8 shows an image generator that is vertically symmetrical . This creates a random walk that can step in one of six directions. Typical output is shown in figure 9. The

[1] This can be made more efficient by modifying the drawstuffXY to send polar coordinates.

stepping algorithm gives a lacy texture to the image. Figure 10 shows 6 stages of rotation applied to this type of image.

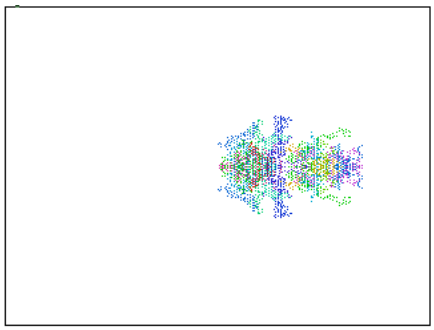

Figure 9.

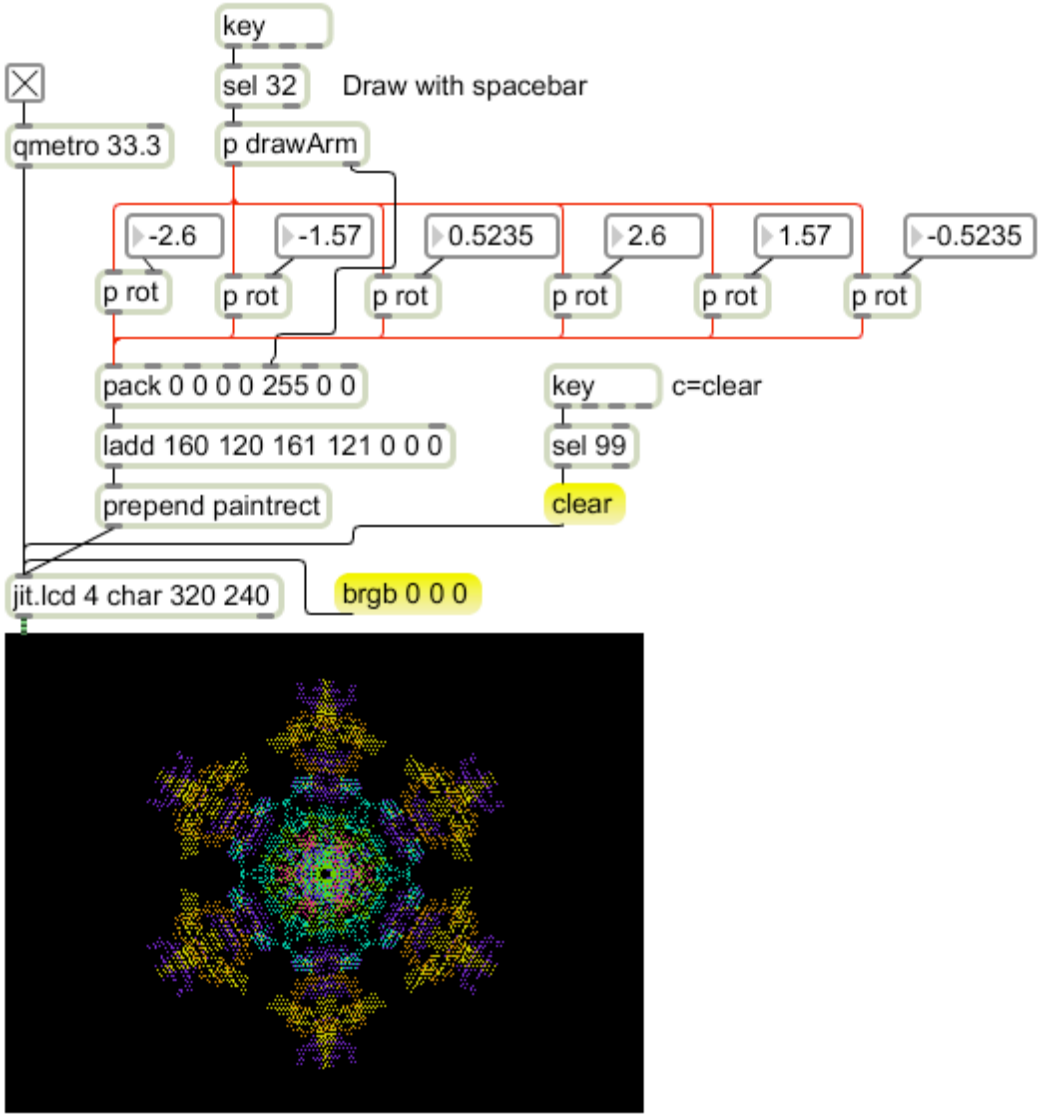

Figure 10.