

Max and Graphics

Translating the LCD help file

The LCD (and jit.lcd) is the main drawing environment for max. It is based on the underlying Macintosh graphic routines called QuickDraw.

The drawing space is made up of numbered pixels. Each pixel has two numbers, the first is horizontal location, the second is vertical. The upper left corner is 0 0, and the pixels across the top are 0 0, 1 0, 2 0, 3 0, 4 0, etc. The right edge is 0 0, 0 1, 0 2, and so forth. This is similar to the familiar Cartesian coordinates except that y increases going down the screen. Instead of X and Y, I'm going to call the points H V.

Technically speaking the point H V refers to the upper left corner of the pixel. Thus, when you draw a line from 0 5 to 10 5, and another from 10 5 to 20 5, they will butt against each other instead of overlapping. Some commands draw inside of lines, so you occasionally find a 1 pixel difference on the right and bottom edges.

All commands are in lowercase letters. Some commands are switches that are turned on with something 1 and off with something 0.

Commands that affect the LCD itself

The size H V command will change the size of the LCD object, a bit disconcerting, but makes it easy to figure out where the edges are.

Getsize will cause the message size H V to come out the right outlet.

Clear removes all drawing and restore the background color.

Reset removes all drawings and restores all defaults for pensize, colors, etc.

Onscreen 1 starts drawing directly to the screen instead of to a hidden buffer. Drawing to a buffer gives a slicker effect because all of the pixels appear at once rather than as the computer calculates them. Also, if a buffer is used, the computer can automatically restore the contents of the LCD after it has been hidden by another window. The only advantage of onscreen mode is it saves memory, which may be important if you use a lot of LCDs. When you are in onscreen mode, the right outlet will send the message "update" when the display has been messed up. You can use that to trigger your own redrawing.

Colors

Some of the commands can take colors as arguments. There are two ways to specify color: indexed and rgb.

An rgb color is specified by three numbers, which set the intensity of the red, blue or green component. Possible values are for each color range from 0 to 255. 0 0 0 is black, 255 255 255 is white. 255 0 0 is a very intense red, and so on. Yellow is 255 255 0, magenta is 255 0 255, and Cyan is 0 255 255.

Indexed colors use a single number that refers to a set of predefined colors. The set defined is rather strange, being permutations of the values 255, 204, 153, 102, 51, and 0. That gets us up to 214. 215 to 224 are darkening shades of red, 225 to 234 are shades of green, and 235 to 244 are blues. 245 to 255 are darkening shades of gray.

There is a color designated as the foreground color. This is used by any drawing command that does not specify a color. It defaults to black, and will be changed by any color drawing. Or, it can be explicitly set by the command `frgb r g b`. There is a color designated as the background color with the command `brgb r g b`. You won't see it until you issue a clear command. The default is white.

`Getpixel H V` will tell the color of the pixel at `H V`. The response out the right outlet is "pixel r g b h v".

The pen

Drawing is done at the location of an invisible pen. To put the pen somewhere, use the command `moveto H V`.

Click in the LCD also moves the pen.

The similar command `move H V` is relative to the last location. Thus if the pen is at 15 44, `move 5 5` will place it at 20 49.

`Getpenloc` will tell the position of the pen.

The pen starts out one pixel across. The command `pensize H V` changes it to a rectangle. This affects line drawing and the various frame commands. If the shape is not square, vertical lines will be a different width from horizontal ones.

`Penmode` sets the behavior of the pen in relation to what is already drawn. It determines whether you get the foreground or background color, or something else. Some of these functions use a third color called `opcolor`, which is set by the command `oprgb`.

The modes are called by number, but have names as shown in the help file. The following table describes what happens with simple shapes, with effect on colored pict's in parens.

0 Copy	You get the foreground (and background of the pict)
1 Or	You get the foreground of the source and destination
2 Xor	Black if over white space, white if over black (in colors, you get compliments.)
4 Bic	Background color (erases things)
5 NotCopy	Background color (negative of colors of source)
6 NotOr	Doesn't draw at all (negative of pict, destination shows through)
7 NotXor	Doesn't draw (negative of pict, negative of destination)
8 NotBic	Doesn't draw (source & destination where they overlap)
32 Blend	Gives a translucent effect , based on <code>opcolor</code> .
33 AddPin	Adds the source and destination -generally gives stronger colors. Uses <code>OpColor</code> as maximum
34 AddOver	Adds, but with "wraparound" which can be really bizarre.

35 SubPin	Gives the difference, but opcolor is the minimum. Also strange
36 transparent	Will ignore pixels in the source that match the current background. Use this when drawing PICTs if you want the original to show through.
37 Addmax	Will pass the maximum of the source and destination color, which tends toward white.
38 SubOver	Uses the difference between the colors, but if negative wraps around.
39 AdMin	Uses the lesser of the colors.

The Not and OR functions would do something if the pen is in "Pattern mode", which means something other than a solid rectangle. Pattern modes aren't implemented in LCD yet. Since all this math occurs on the individual color components, it's really hard to predict what will happen with any two colors.

Text

Text is drawn starting at the pen location, and moves to the right with each letter. If you want text to restart at the left and next line down when the right is reached, you have to do that yourself.

Font `n s` changes the font to number `n` and size `s`. The number for specific fonts varies from machine to machine so this will need some experimenting.. Most applications list them in alphabetical order in the font menu, which isn't the number order. The first 24 are system fonts and will be predictable. (There are third party externals to find font numbers, I'll try to chase some down)

The command `[write something \, something]` will write text in the LCD. Note that some punctuation, like comma, must be preceded by a backslash to be written properly.

The command `ascii n n n n` will translate the numbers to ascii equivalent letters and write those. There's a chart of ascii in the Max and text tutorial.

Lines

The command `lineto H V` draws a line from the pen location to `H V`.

The command `line H V` draws to a point `H` pixels right and `V` pixels down. `H` and `V` can be negative numbers to go the other way.

`Linesegment h1 v1 h2 v2 c` draws a line from `h1 v1` to `h2 v2` with indexed color `c`. `Linesegment` is really a `moveto` and a `lineto`.

Shapes

Various shapes can be drawn with a single command. The shapes are fitted into a rectangle you define as `H V` upper left and `H V` lower right. The shapes can be painted or framed. `Paint` gives a solid shape, `frame` gives the outline.

`Paintrect`

`Framerect`

`Paintoval`

`Frameoval`

`Paintroundrect`

`Frameroundrect`

(remember, all commands are lowercase. Bill gates won't let me start a sentence with a lowercase.)

Framearc needs two more numbers, which are starting angle and ending angle. These are given in degrees, where 0 is straight up.

The paint commands paint the area inside the bounding lines, and the frame commands draw the bounding lines, so if you paint on top of a frame with the same numbers, the right and bottom of the frame will still show. Paint commands do not move the drawing pen.

Poly

Polygons are painted with the paintpoly and framepoly commands. The arguments to the poly are a list of points (H,V) the poly will be drawn to. It's analogous to a moveto followed by series of linetos. If the lines cross, the paintpoly command will only fill areas on one side of any of its lines. It looks for the best enclosure. Note that the poly will only be closed if the last pair of numbers is the same as the first pair.

Regions

You can group a more complicated set of commands than the lines of poly. This is done by the region procedure. To do this, you issue these commands:

- Recordregion
- Any number of drawing operations
- Closeregion somename
- Paintregion somename H V

You won't see any drawing until the paintregion command. The H and V of the paint region will be added to whatever Hs and Vs were in the recorded drawing commands. You may be surprised at what you see when you include linetos in the region recording.

You can have as many named regions as you want to, but when you are done with one, you should call deleteregion to free up the memory. Clearregions deletes all regions, as does reset.

Clipping Regions

A clipping region is a area that limits drawing, sort of a cookie cutter effect. The following define clipping regions.

Cliprect
Clipoval
Cliproundrect
Clippoly

Cliparc is apparently planned for a later version.

Only one clipping region is in effect at a time. You can have a complex clipping region by making a region, and using the command:
Cliprgrn somename H V

If a clipping region is in effect, the clear command only clears the clipping region. Clipping regions are not removed by the reset command, but I suspect this is a bug. The command noclip gets rid of the clipping region.

Scrollrect is similar to a region command. It copies what ever is in the specified rectangle, erases the rectangle, then draws the copy into a rectangle that is moved by the specified distance. That's all done by
Scrollrect H V H1 V1 mH mV

It's most useful for moving the entire LCD contents.

Sprites

So far all drawing has been permanent. If you draw something, change the H and V values and draw it again, you get two copies. To get rid of something, you have to draw it with the background color, and then redraw anything you wanted to keep. Sprites allow you to define drawings that will move around the screen.

To use sprites, you need to do enablesprites 1. This creates more off screen space for drawing. Then:

- recordsprite
- (a bunch of drawing commands)
- closesprite aname

will create the sprite. Drawsprite H V will draw it. Drawsprite H1 V1 will move the sprite to a different position without affecting any background. To make a sprite go away temporarily, use hidesprite. Free up its memory with deletesprite when you are done with it.

Clear and reset do not affect sprites.

If you use counters to calculate H and V, sprites will glide nicely around on the screen. (Jit.lcd does not have sprites. You can composite images in other ways.)

picts

Pict files are a Macintosh graphics files format. Most graphics programs such as photoshop can create them. You can use Picts in an LCD. The command

Readpict aname filename

will load the pict into the system. (It can apparently also read jpeg files.) The name you give it is just an identifier for LCD, it has nothing to do with what's on the drive.

Then drawpict will put it in the LCD. The arguments to drawpict can make some interesting changes:

Drawpict aname H V	will place the upper left corner of the pict at H and V. Many picts include white space around the edges, so you may have to allow for that. (Try transparent mode)
Drawpict aname H V H1 V1	will squeeze (or expand) the picture into the rectangle defined.
Drawpict aname 0 0 0 0 sH sV sH1 sV1	will draw only part of the picture. sH etc determine which part of the picture.
Drawpict aname H V H1 V1 sH sV sH1 sV1	will draw part of the picture in the rectangle with appropriate scaling.
Deletepict aname	removes the pict file from memory but doesn't erase it from the LCD.
Clearpicts	removes all picts from memory.

You can put a pict into a sprite. You can also tile picts with the command:

tilepict H V H1 V1 sH sV sH1 sV1

Tiling repeats the image as many times as necessary to fill the specified rectangle or the LCD. (You can shape this with a clipregion.) The sH etc. arguments determine the size of the tiles and where they come from in the image.

Once you have built up an interesting image in your LCD, you can immortalize it with the writepict command.

Mouse Actions

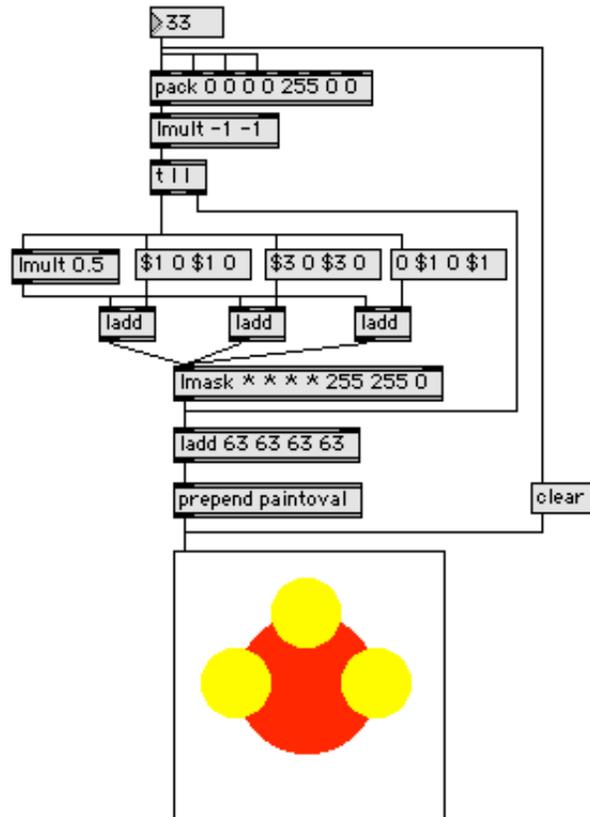
When the patcher is locked, you can draw directly into the LCD with the mouse. The mouse location is sent out the left outlet while drawing is going on. If the mouse button is clicked, a 1 for mouse down or a 0 for mouse up are sent out the third outlet. The command local 0 will turn mouse drawing off, but position and button messages are still sent.

If the command idle 1 has been received, the location of the mouse will be sent out the second outlet when the mouse is over the LCD but not drawing.

With these mouse features, the LCD makes a nice performance interface. You can load a pict in, and then respond when the user clicks over an interesting part of the pict.

A Drawing Example

Drawing on the screen is a matter of figuring out where to put the available shapes. Some lovely patterns can be created by placing dots at mathematically determined locations. Here's an example.



The number box at the top sets the radius of a circle. The paintoval command requires a box to inscribe the circle in, which is provided by the pack. This is modified by the lmult object, which with two arguments only affects first two list members. The list at this point is [-34 -34 34 34 255 0 0]. This corresponds to left edge, top, right edge, bottom, with three numbers to provide the color in RGB format.

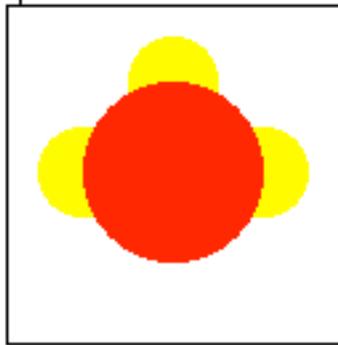
This list is fed to the Ladd at the bottom which centers everything in the LCD. I find this approach is usually much simpler than trying to do all the math in the positive quadrant, and changing the size of the LCD requires only one change in the patch.

I also want some circles half this size, which are created by lmult 0.5. Lmult with one argument will process everything in the list. (This will affect the color, but it's about to be changed anyway.) The Ladds make three of these and move them to different places. Messages box tokens are a good way to copy specified

members of a list. In this case they are chosen to move the center of the small circles to the edges of the large one.

Lmask sets the color of the smaller circles.

Note that the two sets of circles are separated by a trig (t) object. This kind of drawing is very sensitive to the order of operations. Here's what I get if the two trig outlets are switched:

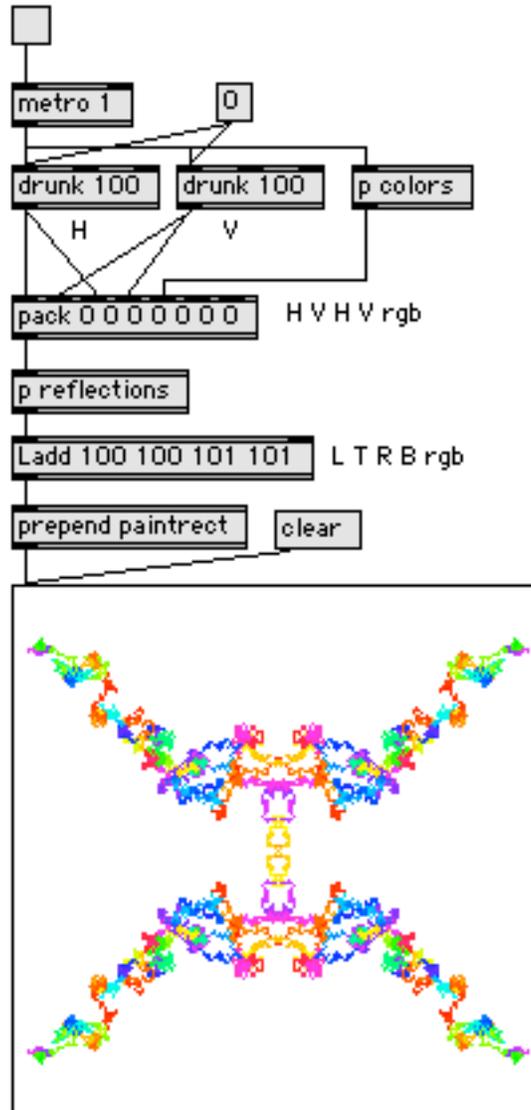


I use Lobjects extensively for sizing and positioning graphics. Individual objects could do the job, but I find it easier to keep track of what is going on when the complete description of a rectangle is in a single list. When this format is used, the following operations are easy to apply:

Ladd 5 0 5 0 0	Move object right 5 pixels
Ladd 0 5 0 5 0	Move object down 5 pixels
Ladd 0 0 5 0 0	Widen object by 5 pixels
Ladd 0 0 0 5 0	Increase height by 5 pixels
Lmult 1 1 5 5 1	Make object 5 times size (only works if left top is 0 0, so this will usually be done first. You can also do this if you are using negative coordinates for left and top.)
Lmask * * * * 5	forces index color to be 5. (See the Lmask help window to see how to change the mask.)

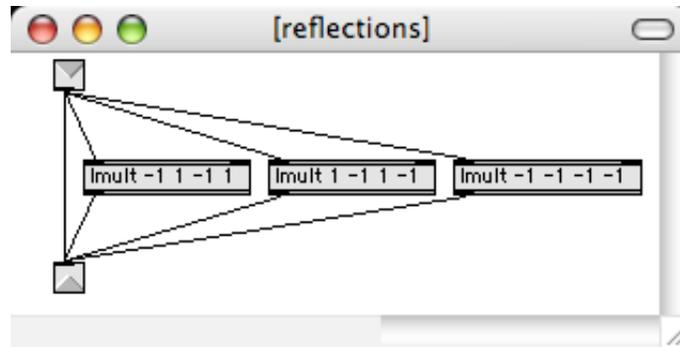
Rorschach

Here's a more elaborate drawing.



This patcher uses drunk objects to generate a random scribble made of tiny rectangles. A new horizontal, vertical and color value are created on each metro tick. Note that the rectangle to be drawn has no dimensions at the start. This is added later. The list that comes out of the pack object is H V H V color.

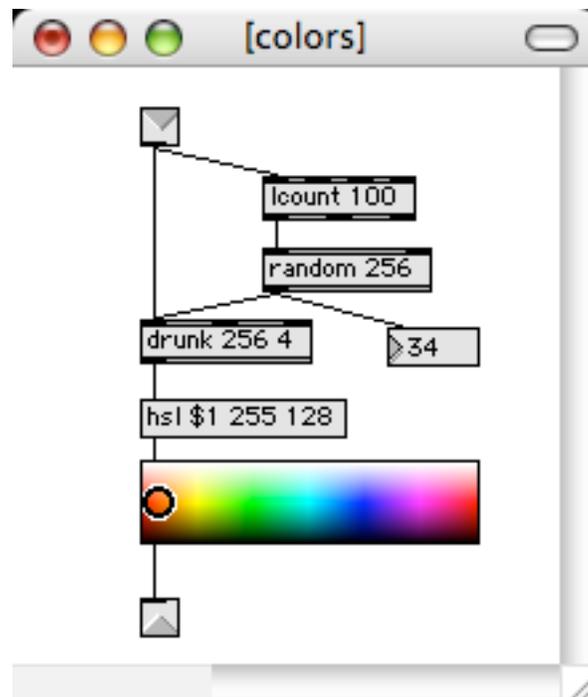
The reflections subpatcher gives the characteristic symmetry of inkblots that are made by folding paper. This is what's inside:



This creates three new lists and thus four rectangles will be drawn. By just changing the sign of the H, V or both values, the shapes will be reflected into four quadrants.

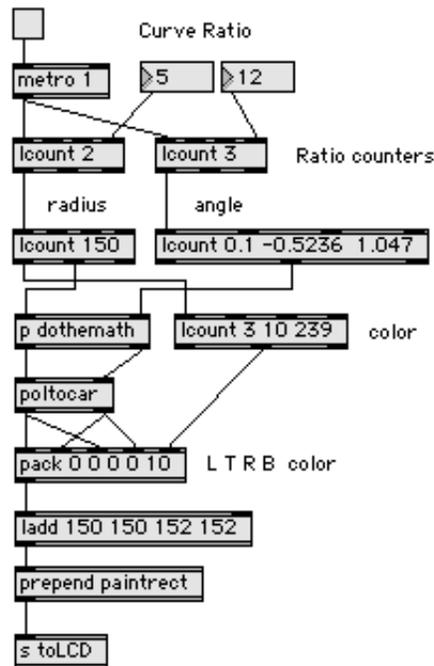
The final addition is performed to move the shape to the center, and at the same time give some dimension to the rectangle. Note that the rectangles are drawn to the right and below the actual coordinate.

A word about the colors subpatcher. This uses one of my favorite tricks to generate random RGB colors. The swatch object recognizes the hsl message, which specifies a color as hue, saturation and level. The arguments of the hsl message let me change colors randomly while guaranteeing they will be nice and bright.

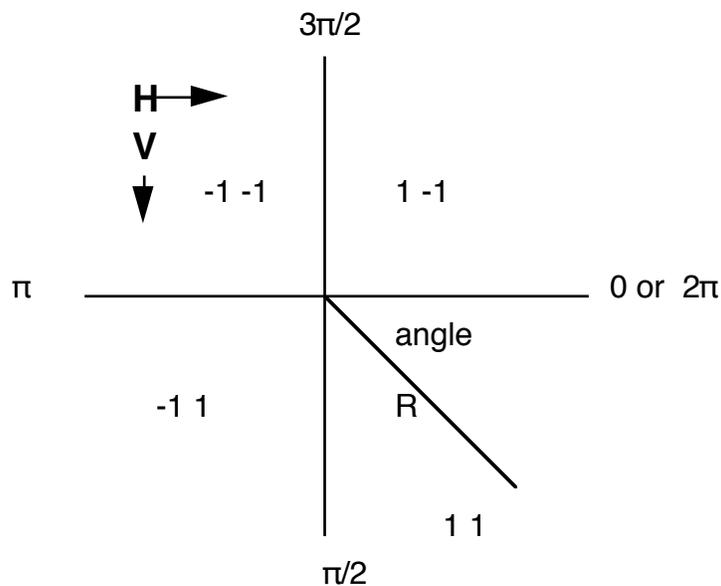


Polar Coordinates

Here's something even more complex:



This is going to draw some curves. It works by calculating an angle and radius for each point. This is how polar coordinates work in the Macintosh graphic world:

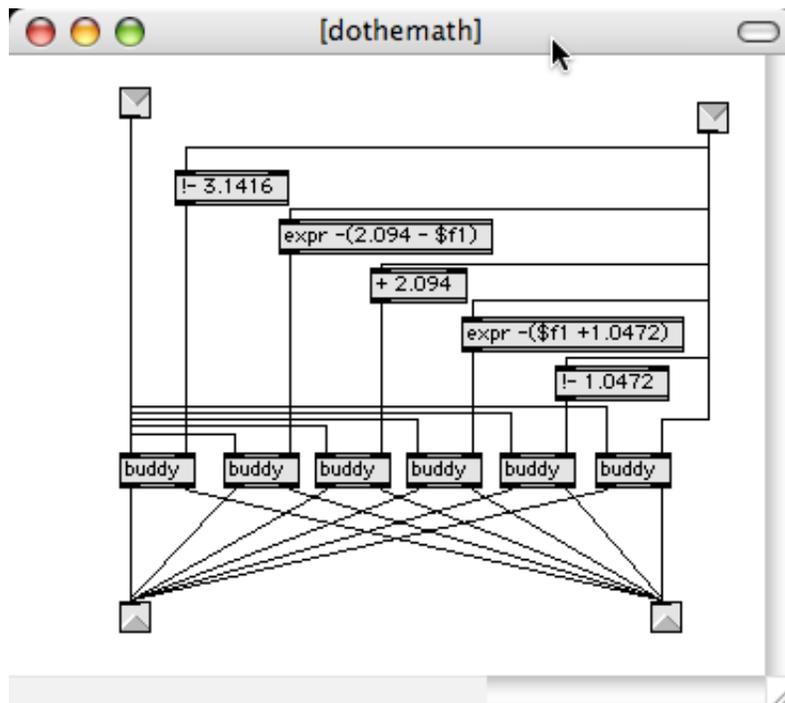


The point we are interested is found at the end of the line R. The radius R and the angle tell us where the point is. Of course, in order to draw the point, it has to be expressed in terms of H and V. H is equal to $R \cdot \cos(\text{angle})$ and V is $R \cdot \sin(\text{angle})$, but we can avoid the math with the poltocar object.

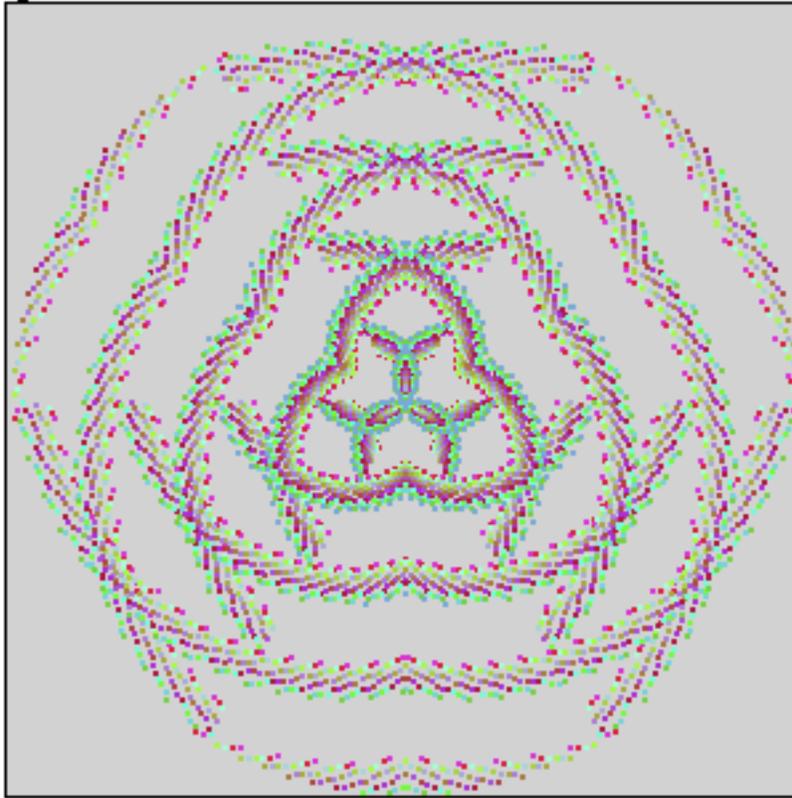
There are two things to be aware of when using polar coordinates. First off, the angle is expressed in radians, and a full circle is 6.283185307 radians. To get accurate placement out at the corners of large drawings, we should keep about 4 significant figures, so 6.283 works. We can generate angles by counting to 6283 and dividing by 100.0, or by using Lcount, which will do float counting.

The second thing is that a full circle will include negative H and V values, so we will again use the offset origin trick.

The patcher includes all of those features but the angle generated is from $-\pi/6$ to $\pi/6$ (0.5236 radians, equivalent to 30° .) Hidden away in the subpatcher you will find these operations:



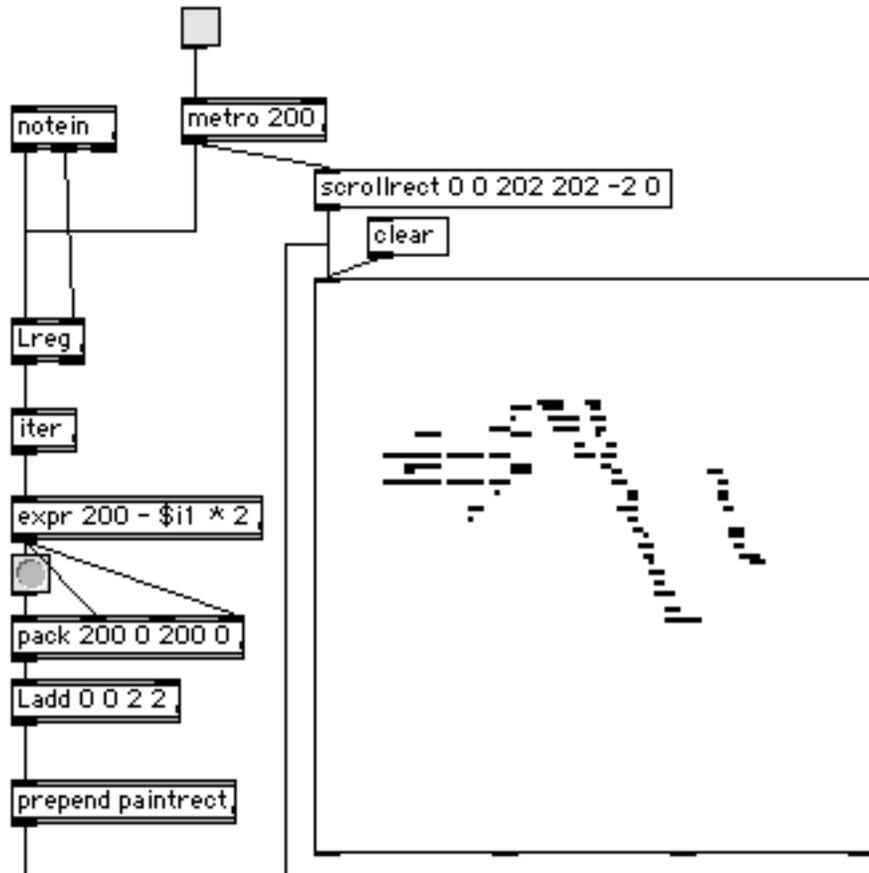
This forms 5 reflections for each point. The reflections are around axes set at 60° angles, which is the effect you get from most kaleidoscopes. The poltocar object converts these to the kind of list we are used to using, and rectangles one pixel wide are painted in the LCD. Here's the result:



The actual curves you get depends on the settings of the ratio counters. More complex patterns arise from changing these ratios at carefully chosen moments.

A real time display

The LCD object is quick enough that you can use it for metering and other real time tasks. Here's a simple piano roll patcher:



The key is the scrollrect command that moves everything to the left on each tick of the metro. Lreg maintains a list of active midi notes. (The same thing can be done with bag.) These are used to paint squares in the manner described before.