Spinner- an exercise in UI development.

I was asked to make an on-screen version of a rotating disk for scratching effects. Here's what I came up with, with some explanation of the process I went through in designing it.

Spin a record

I started with fairly standard patch that displays an image with an arbitrary rotation. Jit.rota is the key object here. The theta message sets a rotation angle, expressed in radians. 3.141592654 gives a half turn counter clockwise, and –π will turn it a half turn the other way.

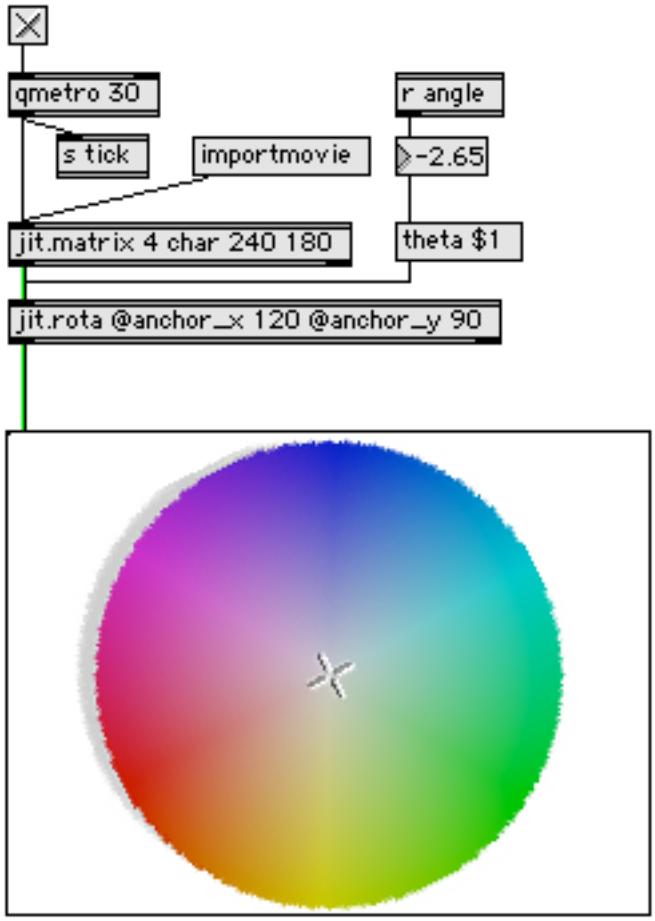

Figure 1.

Send and receive objects are here to connect to other portions of the patch.

Clicking

The next problem is to connect the image rotation to mouse action. This is discussed in Jitter Tutorial 15, and I'm taking a similar approach. A click in the pwindow produces a message such as "mouse 217 171 1 1 0 0 0 0". The first value is the horizontal location in

the pwindow, the second the vertical location (counting from the top) and the third is a 1 if the mouse button is down and a 0 if the mouse button is up. If you set Idle Mouse Reporting in the pwindow inspector, you will get messages whenever the mouse is over the window. In the default case messages are sent as long as the button is down, with one extra after the button is released. This extra message has the same coordinates as the final button down message. There are 5 more values in the mouse message, indicating the state of the modifier keys.

Figure 2 shows a way to convert this information to an angle. I have encapsulated this to keep the main window uncluttered. This subpatch attaches to the right outlet of the pwindow.

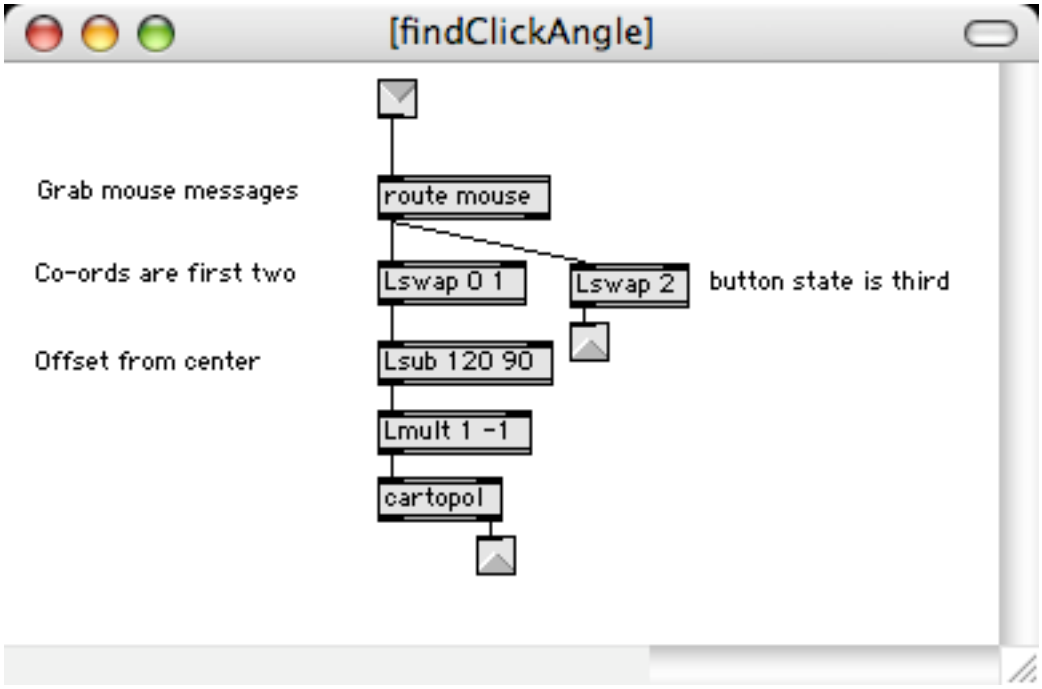

Figure 2. Mouse click to angle.

The route object filters out any other messages the pwindow may provide. The Lswaps break out the data I am interested in. The button state is just sent to an outlet. The X and Y coordinates go through some transformation. First the coordinates of the center are subtracted to make them relative to the center, then the Y value is negated make upward motion positive. Finally the handy cartopol object does the trigonometry and provides the angle. Cartopol considers the 3:00 position to be 0 radians. Clockwise from there is negative down to $-\pi$. There is a jump from $-\pi$ to π at 9:00. We will have to deal with that at some point.

This angle can be sent to the [r angle] object at the top of the patch and the rotation will nicely follow the mouse.

Auto rotation

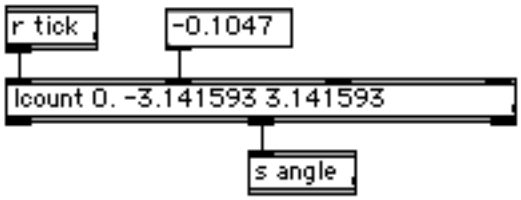

Figure 3.

Figure 3 shows a simple mechanism to spin the picture. Lcount in float mode will add the increment (first argument) to the current angle with each bang. With an increment of 0, there is no motion. An increment of – 0.1047 will turn the image clockwise at 33.33 RPM. Here's a bit of math:

1 revolution takes 60000 ms /33.33333 = 1800 ms

In 1 ms a record turns 6.283185307/1800 = 0.003490658 radians

In 30 ms the image turns 0.104719755 radians.

The tick arrives each 30 ms and Lcount provides the angle to rotate the image. Note in figure 1 the tick is sent before the image is passed through jit.rota. This mechanism gives a smooth rotation, and is appropriate anywhere radians are used.

Calculating increments

The next step is to connect the mouse to the lcount somehow. Figure 4 shows the portion of the patch that does that. The click angle we just found is stored in a float object. The mouse messages are sent at irregular intervals ranging from 5 to 60 ms, and we need to synchronize this with the drawing clock. The [r tick] object will bang the most recent angle when it's time to draw a new frame. This bang is switched by the mouse button, because we don't want that last coordinate of the mouse up. This coordinate is sent twice, and would (as you will see in a moment) cause the disk to stop.

The new angle is packed into a list with the previous angle by the llast object. Applying this list to the !- object gives the difference between the new angle and the last, which is exactly the increment needed by lcount. Note that llast is cleared when the mouse button is released. This prevents a big jump that would happen if the next click is not in the same region of the picture.

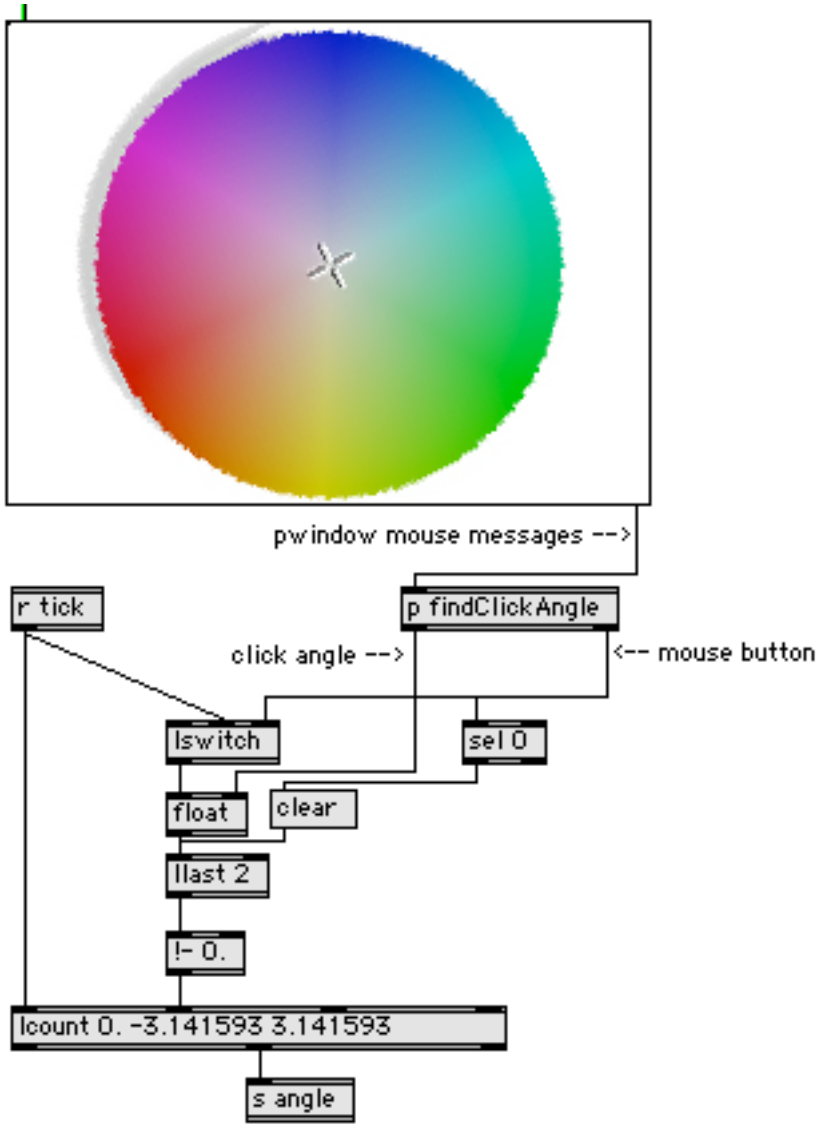

Figure 4.

The behavior of this is kind of entertaining. The disk rotation will follow a mouse drag, and if you stop the mouse before releasing the button, the disk will stop. However, if the button comes up as the mouse is moving, the disk will coast with the direction and velocity of the final motion.

Some refinements are desirable. One issue is that when the mouse is in the left side of the image, there will be that occasional discontinuity when the angle wraps from $-\pi$ to π. This wouldn't necessarily be visible, but it would be audible when we finally get around to controlling sound. To fix it I replaced the !- object with the subpatch shown in figure 5.

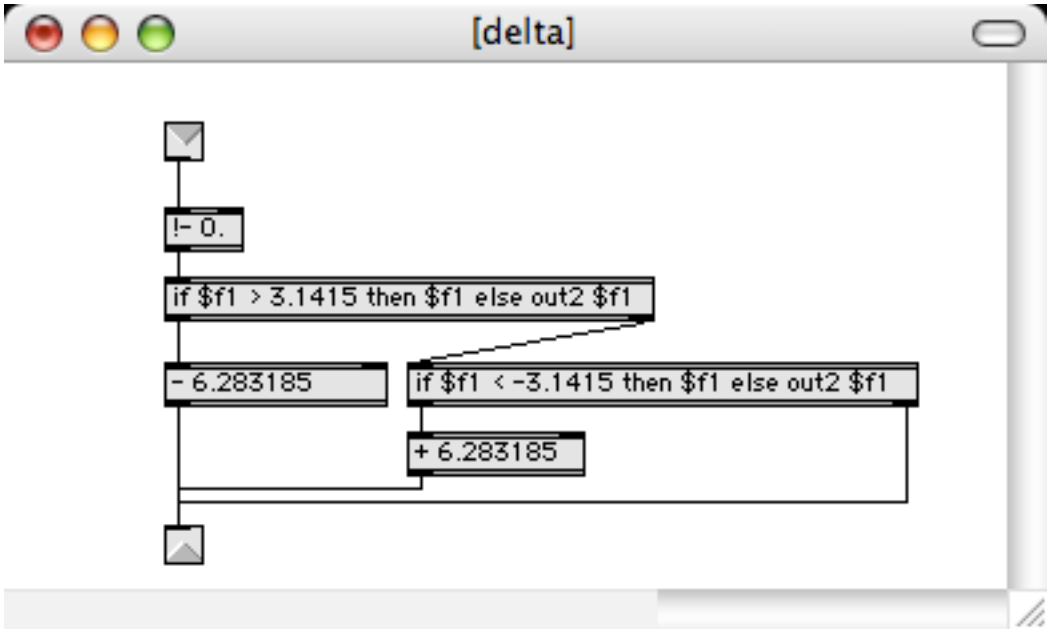

Figure 5. Fixing the angle wrap problem.

There's nothing tricky here. I just detect out of range values and fix them.

Another refinement is to add a feature so the disk rotates at normal speed when the button is released. This is done by attaching a default increment to the button up selector. These additions are in figure 6.

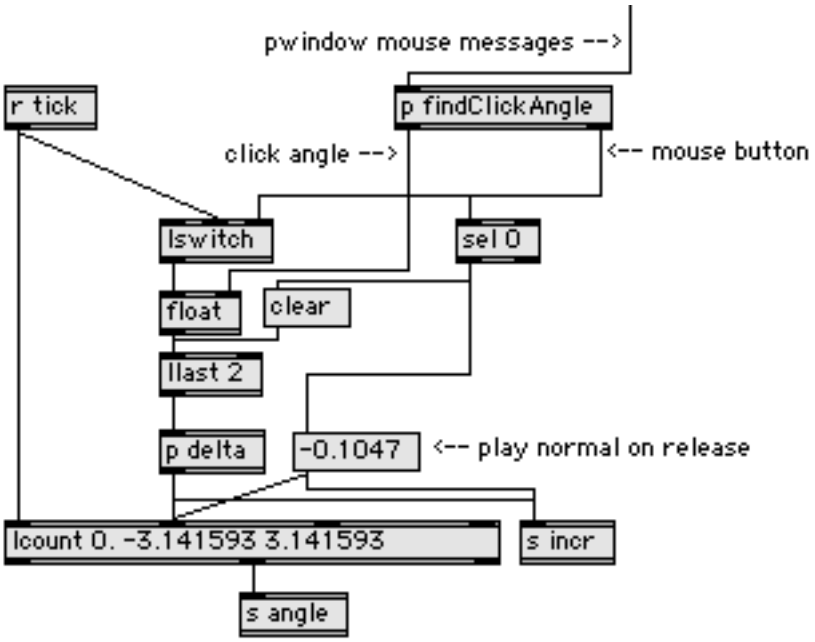

Figure 6.

Making it play

Now we need to connect this to audio. It's not too hard, as the increment we are using to turn the image is similar to the play rate required by groove~.

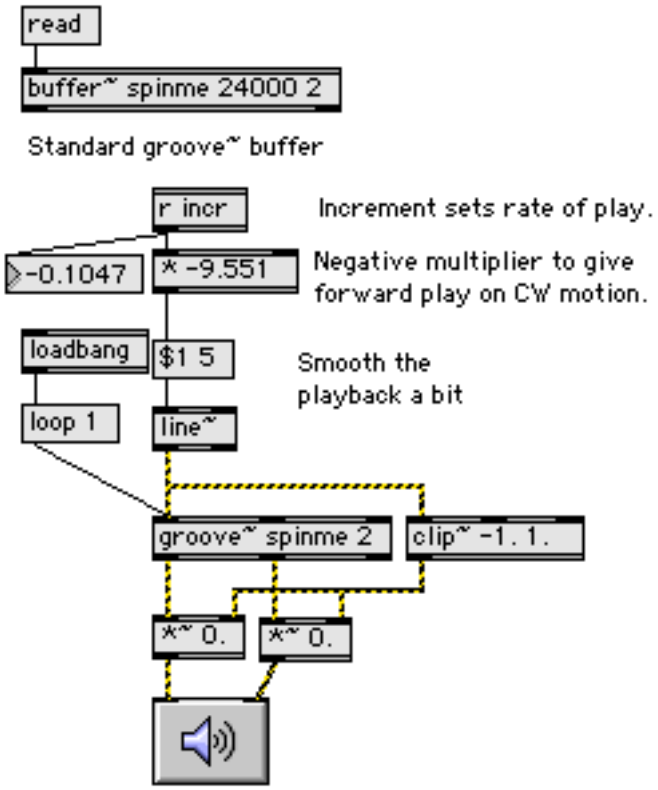

Figure 7. Playback

In figure 7, you see a fairly standard buffer~ with groove~ set to play it back. Groove~ is loadbanged into loop mode. The value from [r incr] merely has to multiplied by a constant to convert 33.33 RPM clockwise (-0.1047 you will remember) into a signal with value of 1.0. This is massaged slightly to prevent pops as the image is scrubbed. The line~ provides a short envelope to smooth rate transitions. This is also applied to the amplitude to fade when playback is stopped and started. This makes the volume sensitive to speed of playback. Since real records behave the same way[1], this is not objectionable. The clip~ object prevents the signal from distorting when playback is faster than normal.

The complete patcher (with one or two other options) is shown in figure 8.
One invisible option is I added an output to the findClickAngle subpatch to give a 1 when a modifier is down. This is used to gate the normal speed setting on mouse up. I like the coasting effect.

[1] One thing that is missing is the bass boost you get when playing records slowly. This is because vinyl is recorded with a reduced bass (de-emphasis) to keep the grooves from getting too wide. On playback, the bass is boosted (RIAA curve), so if the speed is not 33.33, the wrong frequency band is boosted. You could sort of simulate this by adding a shelving filter below 500 Hz and manipulating the gain.

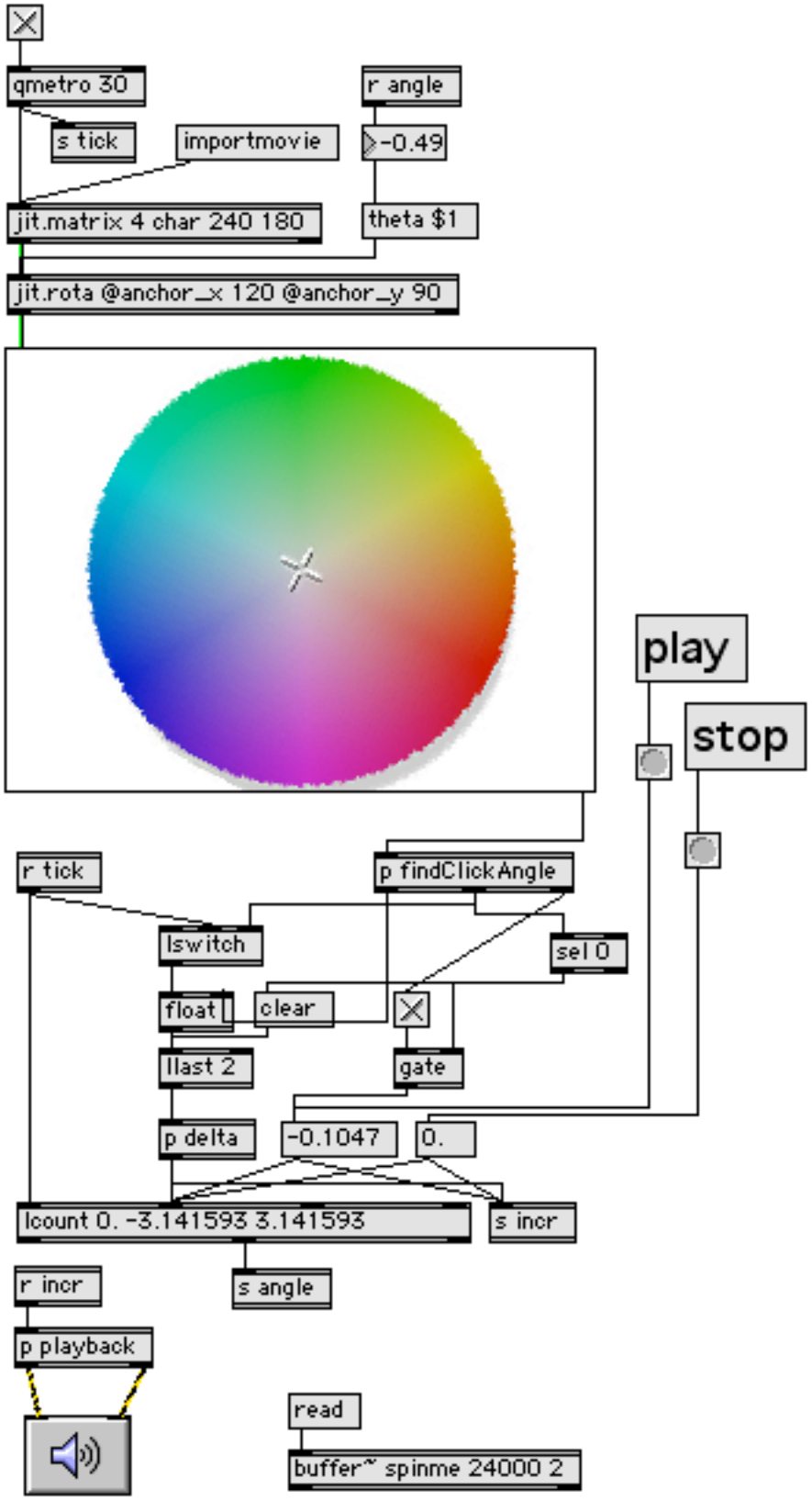

Figure 8. The complete Spinner Patch

File Playing Version

Groove~ is limited to fairly short files, of course. However, with recent improvements to sfplay~ the same principles can play a file directly from disc. All that is necessary is a modification to the playback part of the patch:

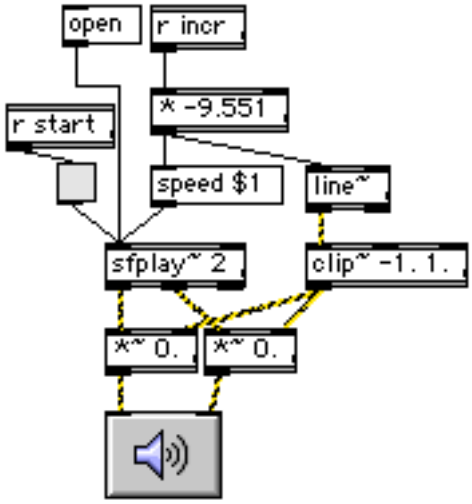

Figure 9. Spinning sound files.
The receive start object gets a 1 or 0 from the play and stop buttons in the main patch.