Vectors and Motion.

We have learned to represent an object's position with and X and Y (and maybe Z) values. These are distance from an origin point of 0,0 found somewhere on the screen.

We can easily move the object by adding a pair of values to X and Y. We call this kind of value pair a velocity vector, because the values determine a direction and magnitude of motion. We can visualize this as a line from 0,0 to the point X,Y. Adding vectors is done by adding the Xs and Ys (and Zs) individually. To move the object again, we simply add the velocity vector to the new position. If we keep doing this, drawing the new location each time, we will get smooth motion in the direction the vector points.

When the position of the object moves off the screen, we need to do something to keep it visible. One option is to simply move the object to the other edge with code like:

```
if(X>width) X = 0;
if(X< 0) X = width;
if(Y>height) Y = 0;
if(Y< 0) Y = height;
```

This will make the motion "wrap". A more interesting motion is to change the sign of the X or Y part of the velocity vector. This will make the object "bounce" off the edge. Here's some processing code:

```
// vector motion
float X, Y;
float cY, cX;
float vX,vY;

void setup(){
 size(400, 300);
 cX = width/2;
 cY = height/2;
 X = cX;
 Y = cY;
 vX = 4;
 vY = 4;
 stroke(255);
}

void draw(){
 background(0);
 X += vX;
 Y += vY;
 ellipse(X,Y, 10,10);
 if(X > width) vX = abs(vX) * -1;
 if(X < 0) vX = abs(vX);
 if(Y > height) vY = abs(vY)*-1;
 if(Y < 0) vY = abs(vY);
}
```

Note that I am using the absolute value of vX * -1 to change the sign, this approach guarantees that the object will return to the screen eventually. It should never be able to get more than a vector's distance off the screen, but as we add code that possibility may arise.

To change the speed of the motion, we simply modify the motion vector. This code will do it:

```
void mouseDragged(){
 vX = (mouseX - cX)/10.;
 vY = (mouseY - cY)/10.;
}
```

You can add this to the draw routine to show the mouse action:

```
if(mousePressed)
   line(cX,cY,mouseX,mouseY);
```

With vectors, it is easy to include complex motion effects. For instance, gravity can easily be added to the draw routine:

```
void draw(){
 background(0);
 if(mousePressed)
   line(cX,cY,mouseX,mouseY);
 ellipse(cX,cY, 2,2);
 X += vX;
 Y += vY;
 ellipse(X,Y, 10,10);
 if(X > (width-5)) vX = abs(vX) * -1.;
 if(X < 10) vX = abs(vX);
 if(Y < 10) vY = abs(vY);
 if(Y > (height-5)) vY = abs(vY)*-1.;
 if(Y < (height-5))
   vY += g;
}
```

The variable g is set as a global to be 0.5. Note that it is added to the Y vector on each cycle unless the ball is resting on the bottom. (Gravity does not change the velocity of anything in contact with the floor.) When this is included and vX and vY are set to 0, the ball appears in the center and drops. It will bounce forever, because there is nothing slow it down. We can easily add that as friction at the end of the draw routine.

```
  vX *= (1.- f);
  vY *= (1.- f);
```

The friction variable should be pretty small, 0.01 or less. Now all motion will eventually come to a stop.

Max Version

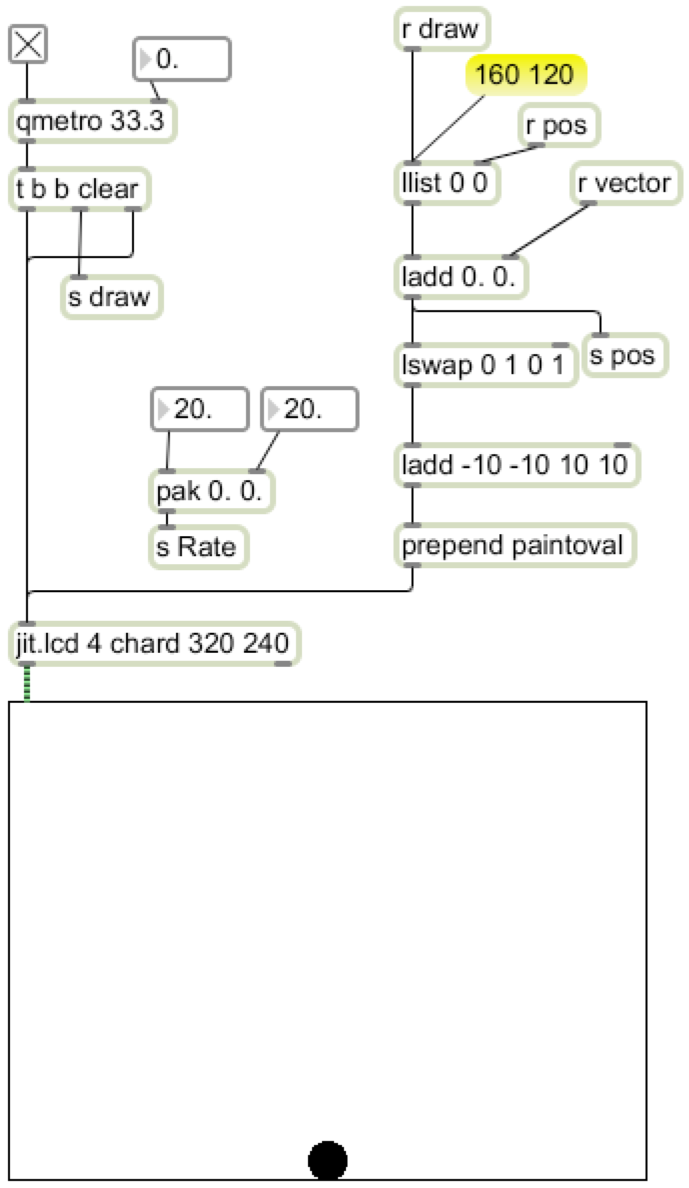


Figure 1

To do this in Max, I use a two part patch. Figure 1 shows the drawing apparatus. The qmetro clears the jit.lcd then sends a bang to receivers named draw. There's one on the right, which will cause a circle to appear somewhere in the window, We aren't sure where, since the location just comes in as pos. This is held in the llist[1] object until the bang at draw. This is next added to the vector then sent back to pos. You can see this makes a little loop that is iterated by the draw bang. If vector has any value, the location will change on each cycle. This is passed on to the lswap[2], which converts [X,Y] to [X,Y,X,Y] This is added to [-10 -10 10 10] to set the location of the following paintoval command.

This mechanism can be used with any patch that modifies vectors. For bounce with gravity and friction, it looks like this:

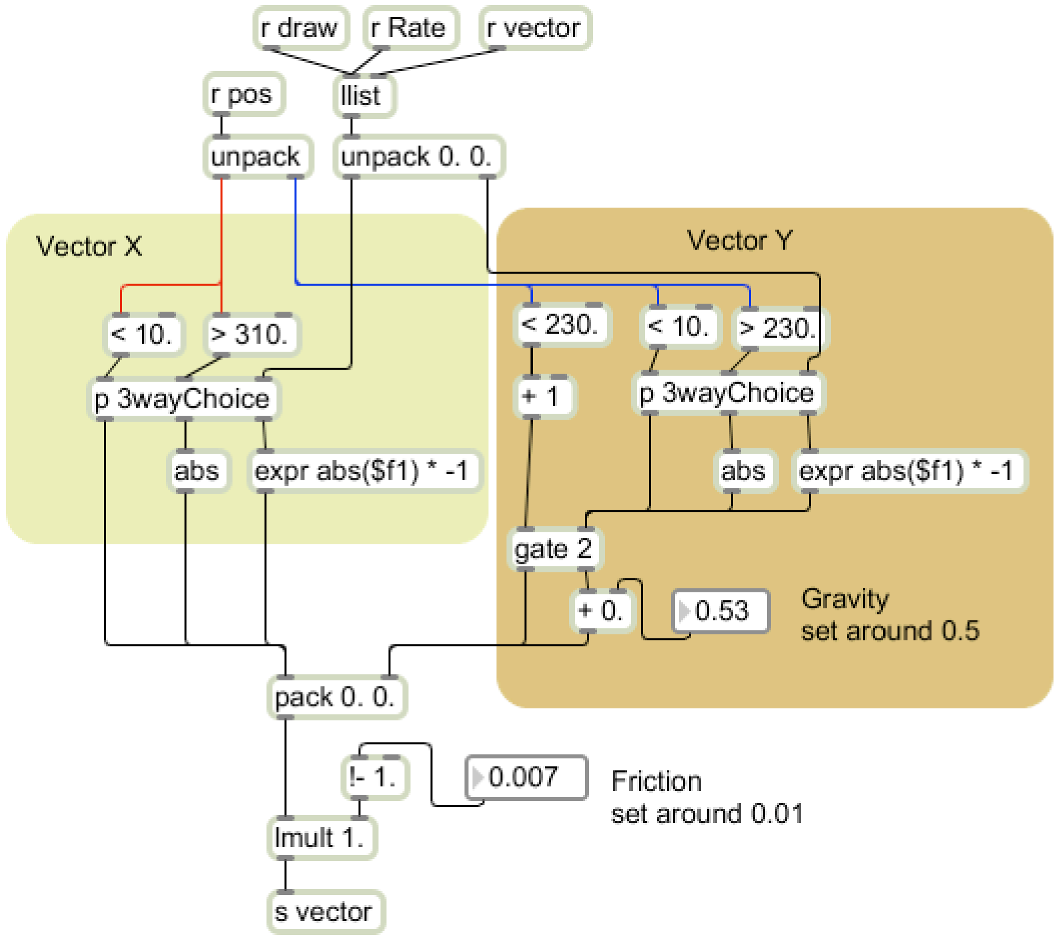
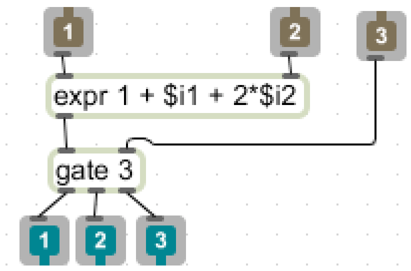


Figure 2.

---

[1] Llist stores lists the same way float and int store numbers. If applied to the right inlet, the list will be held until the left inlet is banged.
[2] The arguments to lswap are indices in the incoming list. 0 is first, 1 the second and so on. The argument 0 1 0 1 will repeat a two element list.

This is also an iterated system, with vector as the input and output. Pos is also needed to control the modifications of the vector. The vY side is slightly more complicated, so I'll talk through it. The vY value is tested to see if it is less than 10, grater than 230, or somewhere between. This test occurs in the 3 way choice sub patch:



The data at input 3 is switched by both inlet 1 and 2.

- if they are both 0, the data goes out 1.
- if inlet 1 is 1 and inlet 2 is 0, the data goes out 2,
- if inlet 1 is 0 and inlet 2 is 1, the data goes out 3.

Inlets 1 and 2 cannot be 1 at the same time. This switching performs the following operations.
Out 3 `expr abs($f1) * -1` forces the value to be negative.
Out 2 `abs` forces the value to be positive.
The third option is no change.

This mechanism handles the bounce at the top and bottom. vX gets similar treatment, but vY has one more possibility. If the value of vY is less than 320 (above the bottom) the gravity value is added in, This will make the ball drop faster with each frame[3].

Friction is applied after the velocity vector is reassembled from vX and vY. Friction is a coefficient which is subtracted from 1 and multiplied by both elements of the vector. Friction eventually slows everything down to 0.

[3] Gravity is like that- it's an acceleration, meaning it constantly increases the velocity in the down direction.