

## Filter Factory Tutorial

by Neal Tucker

8/30/94

The "Lite Pegs" filter takes an image and reduces it to a grid of evenly spaced, uniform size circles on a black background. The color of each circle is taken from the pixel at the circle's center on the original image. This tutorial takes you step-by-step through the thought process of making the "Lite Pegs" filter with the Filter Factory.

Start by setting the expressions for all of the channels to 0 and then modifying the expression for the red channel, leaving the others alone. This makes it easier for you to see the effects of the changes you make.

First, make a circle:

$$c2m(x,y) < 10 ? 255 : 0$$

This sets all pixels whose distance from the origin is 10 pixels or less to 255, and all the rest to 0. The result is a circle of radius 10 centered at the origin. To move the circle, we need to displace the x and y parameters:

$$c2m(x+15,y+30) < 10 ? 255 : 0$$

This creates a circle of radius 10 centered at (15,30). The resulting filter is somewhat boring, however, unless we give the user a bit of control over what is going on:

$$c2m(x+ctl(1),y+ctl(2)) < ctl(0) ? 255 : 0$$

This will create a circle whose radius is defined by control 0 and whose x and y coordinates are defined by controls 1 and 2, respectively.

Now we want to draw the circle we created above, centered at each multiple of 20 pixels. If you picture a series of vertical lines, spaced 20 pixels apart, you may notice that the expression  $x\%20$  (the remainder when x is divided by 20) represents the distance to the closest line on the left of a given point. Consequently, we can use  $x\%20-10$  as the distance from the current point to the nearest line (we will end up with values from -10 to 9, but the sign of the value is ignored). So if we use the expression:

$$abs(x\%20-10) <= 2 ? 255 : 0$$

we end up with all locations within 2 pixels of our imaginary lines lit up. The resulting lines are 5 pixels wide. When we drew the circle above, we were testing to see if each pixel was within 10 pixels of a given point. In order to draw the grid of circles, we need to test to see if each pixel is within 10 pixels of any point where  $x\%20-10 == 0$  and  $y\%20-10 == 0$ . This is easily accomplished by the following:

$$c2m( x\%20-10, y\%20-10 ) < ctl(0) ? 255 : 0$$

Now we are very close to having a functional "Lite Pegs" filter. The only thing missing is that instead of setting the circle's color to 255, we want the circle to take on the color of the pixel at its center. For any given pixel, the x coordinate of the center of the nearest circle is the x coordinate of the pixel, minus its distance from the center of the circle. We have already figured out an expression for the distance from the center of the circle, so the coordinates of the nearest circle are trivial to compute:

$$\text{center of nearest circle} = ( x-(x\%20-10), y-(y\%20-10) )$$

To obtain the pixel value of that point, we use the src() function:

$$\text{src}( x-(x\%20-10), y-(y\%20-10), z)$$

With all that rolled together, the expression looks like:

$$\begin{aligned} & c2m( x\%20-10, y\%20-10 ) < \text{ctl}(0) ? \\ & \quad \text{src}( x-(x\%20-10), y-(y\%20-10), z) : 0 \end{aligned}$$

Tip:

Note that we use z as the last parameter to src() in this case. We could have indicated the channel explicitly, which is 0 since we're editing the red channel's expression, but z is a variable that represents the index of the current channel, and using it means that we don't have to change the expression if we want to reuse it in other channels. For example:

R: src(x,y,0)

G: src(x,y,1)

B: src(x,y,2)

Each of these expressions could be replaced with src(x,y,z) and still have the same effect.

We can also allow the user to control the spacing between the dots by changing "x%20- 10" to x%ctl(1)-ctl(1)/2, and do the same thing with y. That's a fairly obvious feature, but it's also nice to make the range of allowable values a bit smarter, rather than simply 0 to 255. We can make the spacing depend on the size of the dots by scaling the value returned by slider 1:

$$x\%val(1,\text{ctl}(0)*2, \text{ctl}(0)*4)-val(1,\text{ctl}(0),\text{ctl}(0)*2)$$

This forces the minimum spacing to be twice the radius (so that no circles will overlap) and the maximum spacing to (arbitrarily) be four times the radius.

The final expression with the spacing control added is much larger:

```
c2m( x%val(1,ctl(0)*2,ctl(0)*4)-val(1,ctl(0),ctl(0)*2),
      y%val(1,ctl(0)*2,ctl(0)*4)-val(1,ctl(0),ctl(0)*2) ) < ctl(0) ?
src( x-(x%val(1,ctl(0)*2,ctl(0)*4)-val(1,ctl(0),ctl(0)*2)),
      y-(y%val(1,ctl(0)*2,ctl(0)*4)-val(1,ctl(0),ctl(0)*2)),z): 0
```

Copy and paste this expression into all four channels, and you have a fully functional "Lite Pegs" filter. Note, however, that the first half of the expression is simply a test to see if the current pixel is inside a circle, and it doesn't need to be checked four times per pixel. We can use the put() and get() functions to cache the result of that calculation and speed the filter up considerably:

```
R: put( <test>, 0), get(0) ? <result> : 0
```

```
G: get(0) ? <result> : 0
```

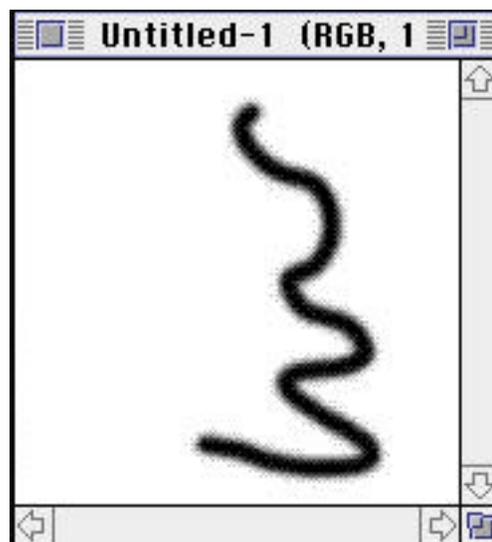
```
B: get(0) ? <result> : 0
```

```
A: get(0) ? <result> : 0
```

### Scaling the effects of your filter

When building filters using Filter Factory, you'll quickly find out that you need to account for the dimensions (number of pixels, not ruler units) of your image when making calculations, otherwise the effects of your filter will not be accurately displayed by the proxy in the dialog box. This can be illustrated with the following example:

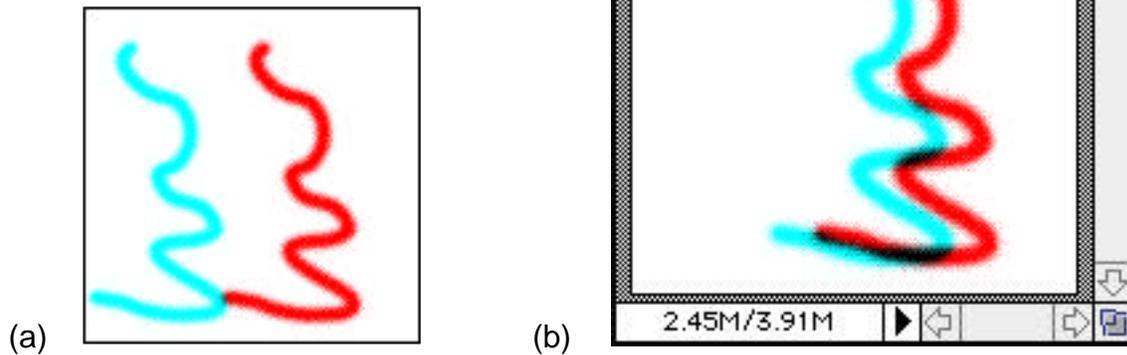
Create a new image that is 500 pixels by 500 pixels with a white background, and draw some black lines on it.



Open the filter factory, and enter the expression "src(x+50,y,z)"

Note the effect of the expression on the proxy (a).

Click OK and note the effect on the original image (b).



What happened here is that both images had their red channels shifted 50 pixels to the left, but the proxy is only 100 pixels wide, so 50 pixels is half of the image's width. The original image, however, is 1000 pixels wide, and a 50 pixel shift has a far lesser effect on it. The solution is to write your expression as follows:

```
src(x+50*X/100,y,z)
```

This scales the value so that the relative effect is the same for whatever size image you work with.

The sample filter factory expressions file accompanying this document contains a scaled version of "Lite Pegs", but beware – it isn't pretty.

Author's note: Any similarities between the look and feel of this filter and childhood toys that use translucent colored pegs poked through black construction paper with a light behind it are coincidental, and are certainly not worth suing over.