Jackson Bates

Python Fu #7: More complicated plug-ins, recreating the Lomo effect

September 14, 2015November 30, 2015 | jacksonbates | gimp, gimp scripting, python, python-fu, tutorial
See the other tutorials in this series. (https://jacksonbates.wordpress.com/python-fu-gimp-scripting-tutorial-pages/)

For our 7th Python-Fu tutorial we are going to make a more complicated plug-on. I made a tutorial for the Lomo effect 4 years ago, and we’re going to automate that. It’s a good idea for you to watch that tutorial first, and maybe even attempt to manually recreate the effect, before we script it. This will give you a good sense of how we will create our program. The code for this tutorial can be found here: lomo.py (https://gist.github.com/JacksonBates/05af1904e5a7bebcd95e).

And here is how we automate that:

The first thing I will do is figure out the steps I need to recreate and then write each of those as comments in the main function.

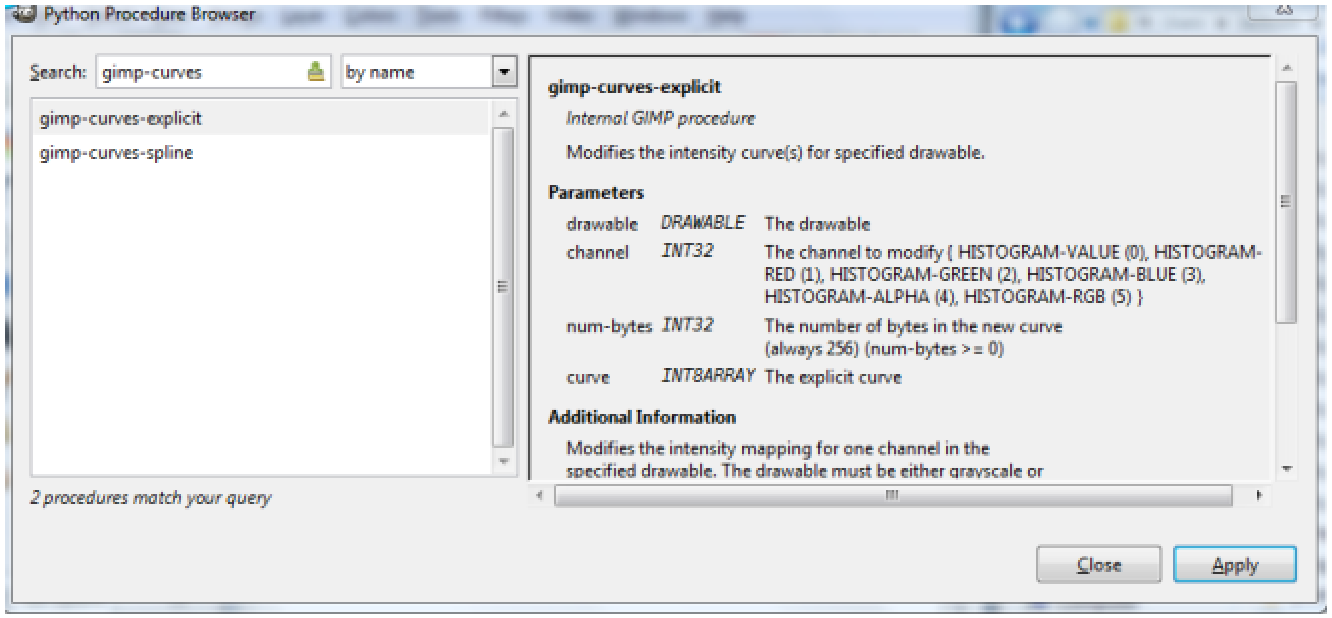
```
    # open curves, select red channel and create an 's' curve
    # 's' curve green channel
    # 'inverted 's' curve blue channel
    # add new layer
    # set layer to 'overlay'
    # blend tool, using linear gradient
    # merge layers
```

And I also know I will want to start and end an undo group for the image, so I’ll put that in comments too, and then code it straight away.

```
# start undo group
pdb.gimp_image_undo_group_start(image)
# end undo group
pdb.gimp_image_undo_group_start(image)
```

Now, when I first started making this I didn’t know how to do any of this, so I had to investigate the PDB and test some ideas. It took a little while, so don’t be put off if you find what you though would work doesn’t the first time. Just go back and read the documentation, google some key phrases, and eventually you’ll figure it out.

So, lets look at curves first. I check the PDB and see that there are two entries for curves, and they both look pretty similar.



gimp-curves-explicit
gimp-curves-spline

2 procedures match your query

gimp-curves-explicit

Internal GIMP procedure

Modifies the intensity curve(s) for specified drawable.

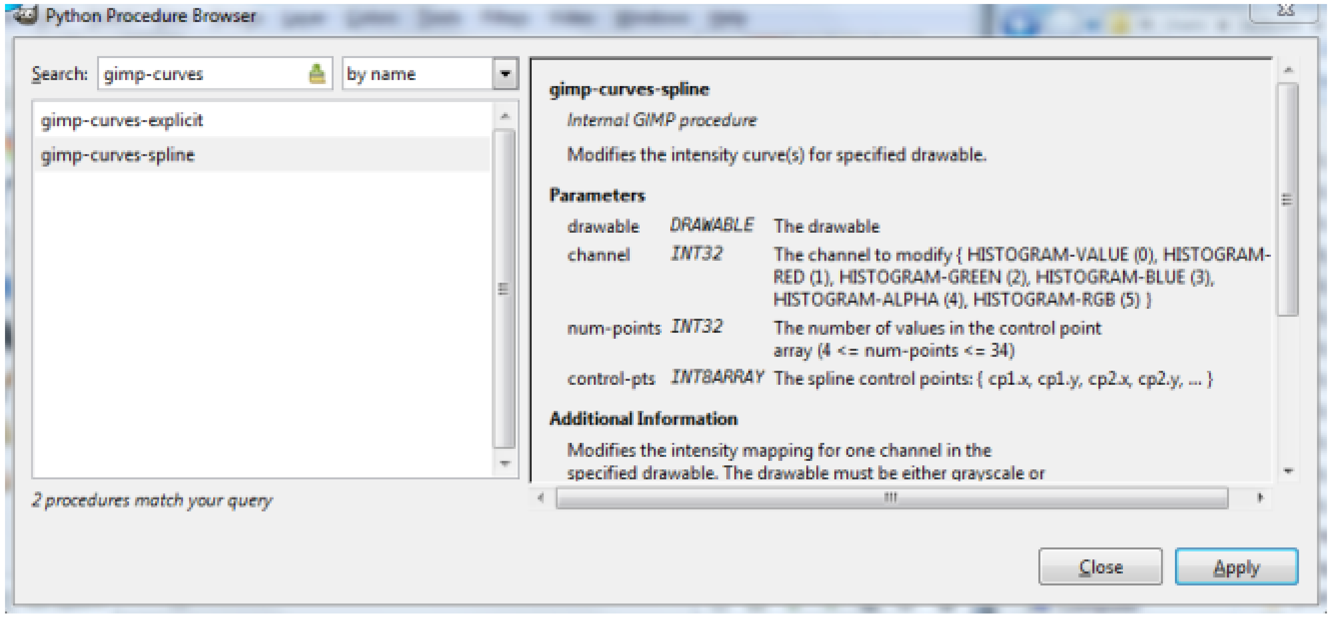
Parameters

drawable	*DRAWABLE*	The drawable
channel	*INT32*	The channel to modify { HISTOGRAM-VALUE (0), HISTOGRAM-RED (1), HISTOGRAM-GREEN (2), HISTOGRAM-BLUE (3), HISTOGRAM-ALPHA (4), HISTOGRAM-RGB (5) }
num-bytes	*INT32*	The number of bytes in the new curve (always 256) (num-bytes >= 0)
curve	*INT8ARRAY*	The explicit curve

Additional Information

Modifies the intensity mapping for one channel in the specified drawable. The drawable must be either grayscale or

(https://jacksonbates.files.wordpress.com/2015/09/e-curves.png)



gimp-curves-spline

Internal GIMP procedure

Modifies the intensity curve(s) for specified drawable.

Parameters

drawable	*DRAWABLE*	The drawable
channel	*INT32*	The channel to modify { HISTOGRAM-VALUE (0), HISTOGRAM-RED (1), HISTOGRAM-GREEN (2), HISTOGRAM-BLUE (3), HISTOGRAM-ALPHA (4), HISTOGRAM-RGB (5) }
num-points	*INT32*	The number of values in the control point array (4 <= num-points <= 34)
control-pts	*INT8ARRAY*	The spline control points: { cp1.x, cp1.y, cp2.x, cp2.y, ... }

Additional Information

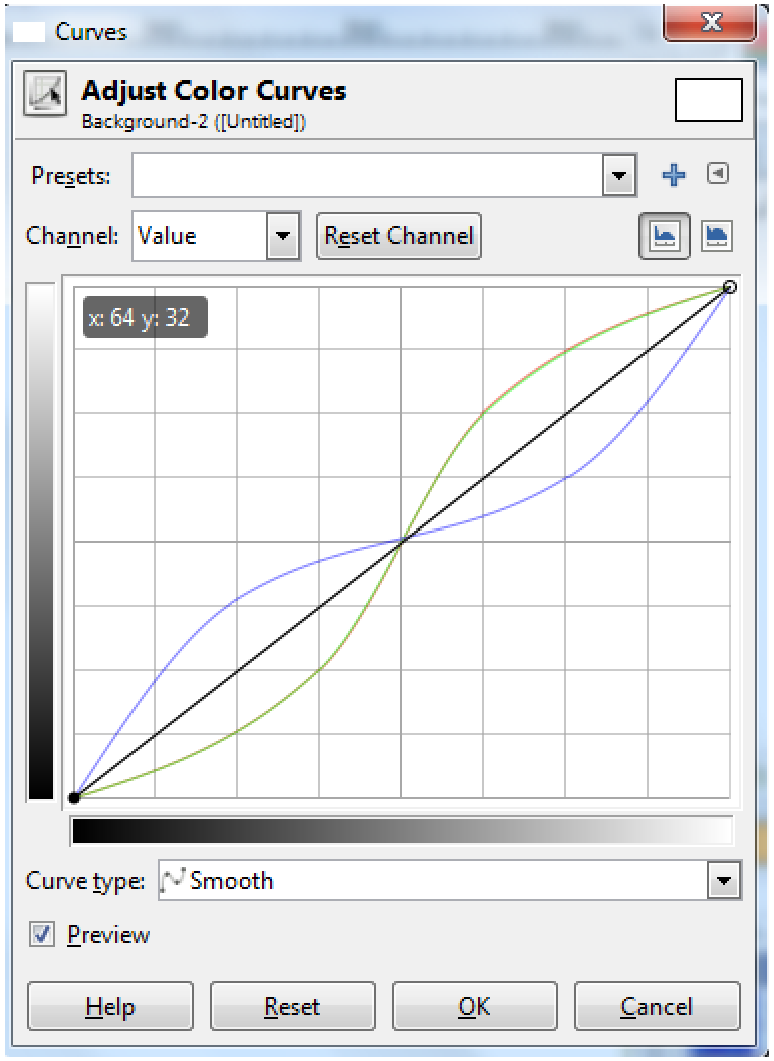
Modifies the intensity mapping for one channel in the specified drawable. The drawable must be either grayscale or

(https://jacksonbates.files.wordpress.com/2015/09/spline.png)

Both take a drawable layer and a channel, so that's good, since I know I need to be able to pick the channel. But the next two options for both are confusing. The explicit function asks for a number of bytes, whatever that means, tells me it's always 256, and has to be greater than or equal to 0. Then the curve is an ARRAY, which is another name for a list, and it just tells me it's the explicit curve. To be honest – I don't even know how to start understanding those options! So I take a look at spline, and even though the word spline is more off-putting, the arguments are more recognisable to me. The number of points is something between 4 and 34 – so worst case scenario, I can just try every number in between if I need to…(don't worry, I didn't actually need to do that – it took a little thinking and just 3 attempts…). Also, the control points also accepts an ARRAY, or a list, but the description is more informative. The list seems to take values labelled in numbers that increase and correspond to the x and y axes. So I still didn't really know what to do, but had mroe to go on – so I'll try spline.

I copied the procedure into my main function and then opened up curves again to see if I can get the coordinates.

```
    pdb.gimp_curves_spline(drawable, HISTOGRAM_RED, num_points, s_curve)
    pdb.gimp_curves_spline(drawable, HISTOGRAM_GREEN, num_points, s_curve)
```



Channel: Value

Curve type: Smooth

☑ Preview

(https://jacksonbates.files.wordpress.com/2015/09/curves.png)

Sure enough, the graph for the curve reports the coordinates when I move around. So lets write down the main ones and see if we can use them. My first point is roughly 95 – but I can be more precise if I think about it. The maximum coordinate values are 255, and the first coordinate on each axis is 0 – so there are really 256 points. There are also 8 grid boxes up and across. Each grid line represents an 8th of 256, or 32 points. Since I want to 'go in' 3 boxes the real value I want is 96, or 3 x 32. So I go pretty close by eye-balling it, ut knowing the principal means I can do the rest without even looking at the grid again! Work smart, not hard! So the coordinates I use in are 96, 64 for the first point and 160, 192 for the second.

```
pdb.gimp_curves_spline(drawable, HISTOGRAM_RED, num_points, (0, 0, 96, 64, 128, 128, 160, 192, 255, 255))
```

Now, to speed things up – here’s a spolier. If I use these 4 control points it does have an effect, but not the one I expected.

So I thought about it and realised I could give more points – I knew both the start point of the line, 0, 0, and the end point 255, 255 and I can also see that it goes throught the centre-point 128, 128.

So I put those values in, too, and that didn’t work initially, with the number of points set to 4, but when I set it to 10, it matched my expectation perfectly. How did I get the magic number 10? Another hunch – the list for the curve was 10 values long. I don;t know if that is the real reason, but it worked!

```
pdb.gimp_curves_spline(drawable, HISTOGRAM_RED, 10, (0, 0, 96, 64, 128, 128, 160, 192, 255, 255))
```

Now, you’ll notice that both the red and green channels use the same curve. A good principal in programming is ‘Don’t Repeat Yourself’. The way I avoid this is by creating a tuple called s_curve and typing the coordinates into that. I can then just put the name of the tuple in instead of those numbers. Another good rule is to avoid what we call magic numbers in your code. Numbers could mean anything, and replacing them with meaningful variable names is cheap. Using variable names may seem redundant, but it makes your code more readable and it will be easier for you to read it days or months later and still know what is happening. So I will create a variable called num_points and set that to 10. I can set the channel either using the numbers in the PDB, or the text label. Obviously to make it as readable as possible, I’ll use the text. The only other thing is that the blue channel has an inverted S curve, so I’ll write the tuple for that and then put it in the code…

```
    s_curve = (0, 0, 96, 64, 128, 128, 160, 192, 255, 255)
    inverted_s_curve = (0, 0, 64, 96, 128, 128, 192, 160, 255, 255)
    num_points = 10
    pdb.gimp_curves_spline(drawable, HISTOGRAM_RED, num_points, s_curve)
    pdb.gimp_curves_spline(drawable, HISTOGRAM_GREEN, num_points, s_curve)
    pdb.gimp_curves_spline(drawable, HISTOGRAM_BLUE, num_points,
                           inverted_s_curve)
```

So, I'm not showing you that I'm testing this every time I add a line, but I did this painstakingly when actually developing this script. It's slow, as I said before, but it makes it easier to debug.

The curves are taken care of, so now we add a layer. The PDB says we can create a variable called layer, with all the appropriate settings, but that doesn't actually put the layer in the image – it just holds it in memory until we do something with it.

The layer arguments are: image, which is the one we're working on, width and height can be accessed using the image.width and image.height variables we discussed in tutorial four, type of layer (as in RGB or whatever), name, which is just a label, opacity, and mode. This last one is very encouraging because setting the mode was a separate step in our comment version of the code, so we've probably just saved a line of code. To avoid magic numbers in my code I will create a variable for the opacity. This is all straightforward enough, so I put in the values I know I need:

```
  #add new layer & Set to 'overlay'
  opacity_100 = 100
  layer = pdb.gimp_layer_new(image, image.width, image.height, RGB_IMAGE,
                             "Overlay", opacity_100, OVERLAY_MODE)
```

Actually, putting the layer in the image took some trouble, because some of the differences between Scheme and Python aren't that well documented. A little digging in the PDB shows that to put the layer in the image we use **gimp.image_insert_layer** and there are some complicated instructions in the description. It talks about 'parent layers' and layers being in groups. It also says 'if the parent is 0…' and this stumped me. I figured I didn't have a parent group, since I hadn't made one but the value 0 didn't work. After half an hour of Googling and reading some forums I found that someone else figure out that Python wants that value to say **None** *instead* of 0 – as soon as I put that it, it worked. Again – this will happen to you occassionally when you start trying to make your own plug-ins. Just keep at it and you'll find the solution. The position is straightforward – I want my new layer to be the first in the list, so the position will be 0 – a variable will take care of the magic number.

```
layer_position = 0
pdb.gimp_image_insert_layer(image, layer, None, layer_position)
```

Finally, I need to use the blend tool. The PDB tells us we can do this and gives us a huge range of arguments! To be honest – I guessed at these and got it right first time, which I was glad for since the new layer took ages.

I figured blend mode was FG_BG_RGB. Paint mode was normal. The gradient is definitely linear and opacity is definitely 100. Offset I guessed and stuck with 0, repeat I also guessed and stuck with 0, reverse I guessed was false, supersample I guessed was false, I guessed max depth was 1 and threshold was 0, and I saw on the blend tool options in the toolbox UI that dither was set to true. The coordinates were easy, but also presented a different challenge. In the original tutorial, I experiment with this until I like the way it looks. But, if I'm going to automate it I can only really make it do the same thing every time, or randomise it completely. There are actually other options, but those two are the easiest. In the next video, we'll give the user a little more choice, but for now we'll make it easy.

I will simply make the blend line go from the top right hand corner to the center of the image – that should be sufficient. To get the top right corner I know the x-axis for the first point is 544, and that the y-axis is 0 (because the coordinates for images are flipped on the y-axis). Well, what happens when I use a different sized image? that first value is wrong. So instead of using the value of my test image, I can use the generic **image.width** variable which will always be correct!

Similarly, to find the center point I just divide the height by 2 for the x axis, and divide the width by 2 for the y axis. image.height / 2 works perfectly for this.

So I put all of that in and test it.

```
# blend arguments and call to function
blend_mode = 0
paint_mode = 0
gradient_type = 0
offset = 0
repeat = 0
reverse = False
supersample = False
max_depth = 1
threshold = 0
dither = True
x1 = layer.width # top-right
y1 = 0
x2 = layer.width / 2 # centre
y2 = layer.height / 2
pdb.gimp_edit_blend(layer, blend_mode, paint_mode, gradient_type,
                    opacity_100, offset, repeat, reverse, supersample,
                    max_depth, threshold,  dither, x1, y1, x2, y2)
```

The final step is to merge the visible layers, which can be achieved with this PDB command:

```
layer = pdb.gimp_image_merge_visible_layers(image, 0)
```

It took a lot of trial and error, but I hope me explaining my thinking as I went helps you to investigate this kind of thing for yourself.

Related posts: Python Fu GIMP tutorials (https://jacksonbates.wordpress.com/python-fu-gimp-scripting-tutorial-pages/)

Here’s all that code:

```
#!/usr/bin/env python

# Tutorial available at: https://www.youtube.com/watch?v=X0_a6U6PkCA
# Feedback welcome: jacksonbates@hotmail.com

from gimpfu import *


def lomo(image, drawable):
    pdb.gimp_image_undo_group_start(image)
    s_curve = (0, 0, 96, 64, 128, 128, 160, 192, 255, 255)
    inverted_s_curve = (0, 0, 64, 96, 128, 128, 192, 160, 255, 255)
    num_points = 10
    pdb.gimp_curves_spline(drawable, HISTOGRAM_RED, num_points, s_curve)
    pdb.gimp_curves_spline(drawable, HISTOGRAM_GREEN, num_points, s_curve)
    pdb.gimp_curves_spline(drawable, HISTOGRAM_BLUE, num_points,
                           inverted_s_curve)
    #add new layer & Set to 'overlay'
    opacity_100 = 100
    layer = pdb.gimp_layer_new(image, image.width, image.height, RGB_IMAGE,
                               "Overlay", opacity_100, OVERLAY_MODE)
    layer_position = 0
    pdb.gimp_image_insert_layer(image, layer, None, layer_position)

    # blend arguments and call to function
    blend_mode = 0
    paint_mode = 0
    gradient_type = 0
    offset = 0
    repeat = 0
    reverse = False
    supersample = False
    max_depth = 1
    threshold = 0
    dither = True
    x1 = layer.width # top-right
    y1 = 0
    x2 = layer.width / 2 # centre
    y2 = layer.height / 2
    pdb.gimp_edit_blend(layer, blend_mode, paint_mode, gradient_type,
                        opacity_100, offset, repeat, reverse, supersample,
                        max_depth, threshold,  dither, x1, y1, x2, y2)
    #merge all layers
    layer = pdb.gimp_image_merge_visible_layers(image, 0)
    #pdb.gimp_displays_flush()
    pdb.gimp_image_undo_group_end(image)


register(
    "python-fu-lomo",
    "Lomo effect",
    "Creates a lomo effect on a given image",
    "Jackson Bates", "Jackson Bates", "2015",
    "Lomo",
    "RGB", # type of image it works on (*, RGB, RGB*, RGBA etc...)
    [
        (PF_IMAGE, "image", "takes current image", None),
        (PF_DRAWABLE, "drawable", "Input layer", None)
    ],
    [],
    lomo, menu="/Filters")  # second item is menu location

main()
```