Full Circle

THE INDEPENDENT MAGAZINE FOR THE UBUNTU LINUX COMMUNITY

INKSCAPE SERIES SPECIAL EDITION Vol 7

INKSCAPE SERIES SPECIAL EDITION

INKSCAPE

Volume Seven Parts 43 - 49

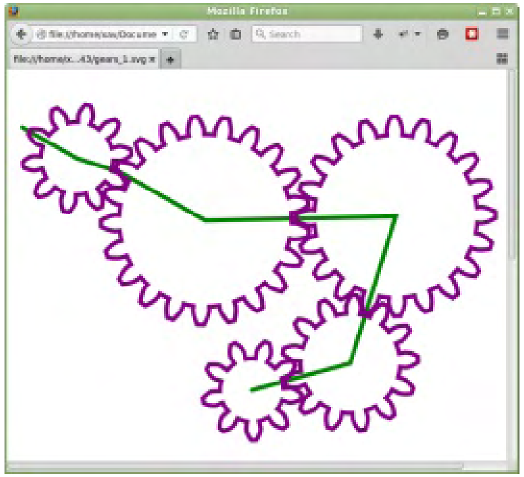
Full Circle Magazine is neither affiliated, with nor endorsed by, Canonical Ltd.

HOW-TO

Written by Mark Crutch

Inkscape - Part 43

Before we delve too much further into Live Path Effects, there are some implementation details that are worth pointing out. The first is that LPEs don't exist in the SVG specification. They're an Inkscape-specific thing, and no browser or other SVG editor knows how to render them. Go on, give it a try. Create a nice chain of gears, or a Spiro path, then save your SVG file. Open it in a modern web browser and see what you get. Here's my file, opened in Firefox.



Well, it certainly looks like my original Inkscape file, but how can that be if the browser doesn't know anything about LPEs? The answer can be found by looking at the XML code for the file, either via Inkscape's XML editor (Edit > XML Editor, or CTRL-SHIFT-X), by viewing the page source in your browser (CTRL-U in Firefox), or simply by opening your SVG file in a text editor. You'll see that the main body of the image is made up of an SVG <path> element. The “d” attribute contains a series of letters and coordinates that tells an SVG-aware application how to draw the final path, after any visible LPEs have been applied. It's like a snapshot of the result, in a format that your browser understands.

Notice that there are some other attributes, in the “inkscape” namespace. In particular you'll find “inkscape:original-d”, which holds the path definition of the original, skeleton path. There's also an “inkscape:path-effect” attribute, which holds a semicolon-separated list of XML IDs. These refer to <inkscape:path-effect> elements up in the <defs> section of the XML, which is where all the parameters for your effects are stored.

So, in summary, Inkscape uses the “original-d” attribute and <path-effect> elements to hold all the information it needs to draw the LPE. Other applications use the “d” attribute to render a snapshot of the final path, with the LPE applied. When you modify an LPE within Inkscape, it automatically updates the “d” attribute to match the rendered output, so other applications should always be able to display your drawing as intended, even though they don't know anything about LPEs.

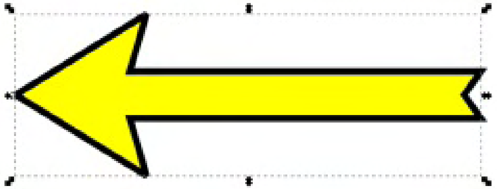
Inkscape doesn't always do a great job of clearing out unused elements in the <defs> section of a file, and path effect definitions are no exception. If you add and remove a number of LPEs whilst experimenting with your drawing, old definitions tend to build up there. They don't do any harm, but do slightly increase the size of the file. You can clear them out, together with other unused definitions, by using the File > Clean Up Document menu entry (File > Vacuum Defs on 0.48).

Because Inkscape calculates the final path from the original path and LPE parameters, using live path effects places more of a burden on the processor, resulting in slower rendering speeds. Usually this isn't an issue, but when zooming into a very complex drawing it can become noticeable. If you're happy with the LPE output, and don't need to change it any further, you can “fix” the path so that it looks the same, but is no longer based on path effects. Essentially this process just removes the Inkscape-namespaced attributes from the path element, leaving it with just the same “d” attribute that any other application uses. To do this, simply use the Path > Object to Path menu entry (CTRL-SHIFT-C). It may seem odd to use Object to Path on something that's already a path, but think of it as converting an LPE path to a plain SVG path, and it makes more sense. Like any other Object to Path conversion this is a strictly one-way affair, so make sure you keep a backup of the file

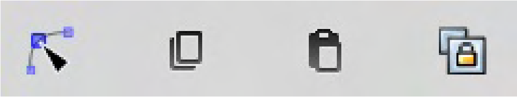
contents ^

from just before the change, in case you subsequently find you need to modify your LPE parameters after all.

That's enough behind-the-scenes detail for now, let's press on with another path effect! As usual we'll need a path to work on, so let's start by drawing a simple arrow shape.



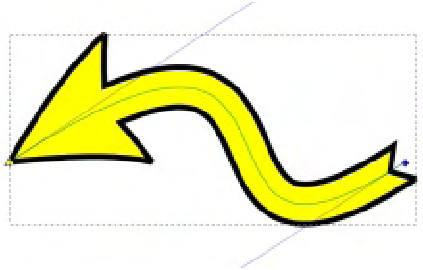
The path effect we'll look at this time is “Bend”, so add that to your path following the instructions from the previous instalment. As before, there's no immediate change to your image, but the LPE dialog has gained a few controls at the bottom. Of particular note is this quartet of buttons:



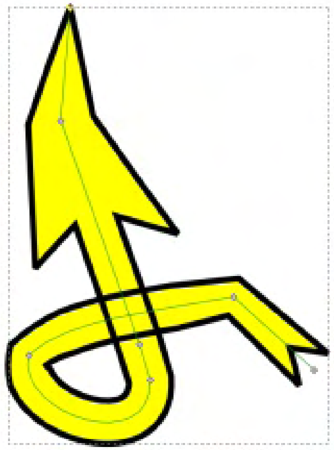
This arrangement of buttons appears frequently in LPEs, whenever an extra path is required as part of the input parameters. In the case of the Bend effect, two paths are required: the original skeleton path (the arrow shape, in this case), and a bend path whose shape dictates how the skeleton path should be distorted. These buttons are for managing the bend path, as follows:

- The first button allows you to edit the bend path directly on the canvas. This is the most commonly used of the four.
- The second button lets you copy the bend path to the clipboard. From there you can paste it into another LPE, or even paste it directly into the canvas as a new path in its own right. These copies maintain no connection to the original bend path.
- The third button is for pasting a path to use as the bend path. This could be one that you've copied from another LPE using button two, or it could be a path you've constructed elsewhere in your canvas. Again, there's no connection maintained to the original path.
- The final button lets you link to an existing path, rather than create a new bend path. In this case there is a live connection to the original path, so any changes you make to that are immediately reflected in the LPE. I'll discuss this button in more detail a little later.

If you press the first button you should find that a straight green path appears on the canvas, directly over your skeleton path. This is the bend path, and you can manipulate it in the same way as any other. Try dragging the path itself, or use the node handles, to distort its shape, noticing how the skeleton path is morphed in real-time to match your changes. You can also move the nodes themselves, in order to stretch, compress or rotate the skeleton path. If the bend path disappears – usually due to a mis-click causing the skeleton path to become selected – just click on the first button of the quartet in the LPE dialog to make it reappear. With barely any effort the Bend path effect can turn your straight arrow into a curved or sinuous shape that would take a lot more time and work to produce using normal path editing techniques:



But there's more! Your bend path doesn't have to be limited to a pair of end nodes connected by a curve. You can add extra nodes, turn them into corners, mix straight and curved sections, have the path double-back on itself, or even split it into several sub-paths. Admittedly, getting too complex with your bend path can lead to a degree of contortion that's hard to control, but the options are there for you to explore.

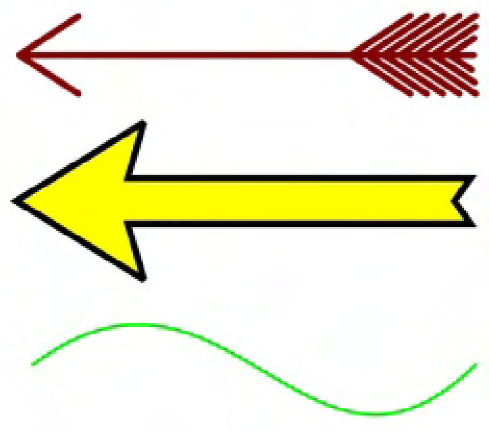


Using the second and third buttons you can copy and paste the bend path from one LPE to another, which can be handy if you want several skeleton paths all distorted in the same way. Each bend path will be an independent copy, though, so changes to one won't affect the others.

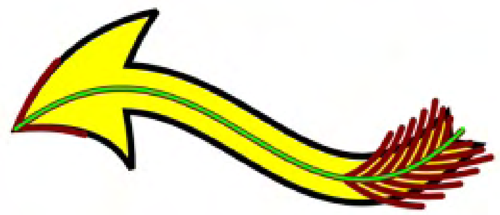
contents ^

Sometimes it's useful to have multiple bend paths all linked to a single “master” path, such that changes to the shape of the master are immediately reflected in each individual LPE. The fourth button allows you to achieve that effect, but it's not without its difficulties.

For this example I'm going to use two different kinds of arrow, and I want to apply the Bend LPE to both of them such that they follow the shape of the green path at the bottom of the image.



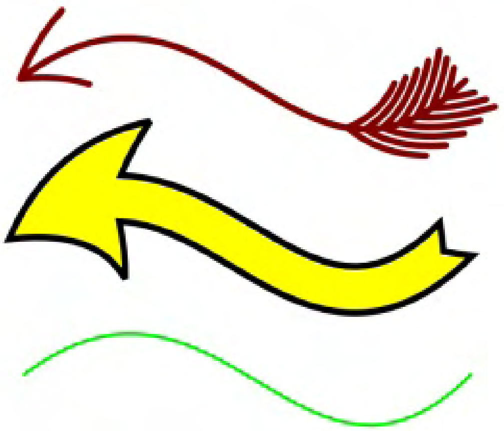
The first step is to select the green path and copy it to the clipboard. As well as copying the path data itself, Inkscape also stores a reference to the original object. Next I need to select one of the arrows, add the Bend LPE, and click on the fourth button to use the stored reference to define the bend path. Clicking this button has two immediate effects: the arrow is distorted to match the bend path, as expected, and the arrow is moved to the same location as the bend path – which is not what I wanted! If I add a Bend LPE to the second arrow and link that to the bend path, that also gets moved. I've got all the right shapes, but not necessarily in the right locations.



At first this might seem like a fairly trivial problem. Just drag the arrows back to where you want them, right? Unfortunately that doesn't work – drag them away and they'll spring right back to the location of the bend path. Drag the bend path away, and they both follow along after it. Being able to link to a common path seems a lot less useful if it means that your linked shapes all have to sit on top of each other.

Fortunately there are a couple of ways around this problem. Inkscape has a setting hidden away in Edit > Preferences > Behaviour > Transforms labelled as “Store Transformation”, with options of Optimised or Preserved (it's in File > Inkscape Preferences > Transforms on 0.48). Use Optimised and you'll see the behaviour I've described above – LPE paths strongly bound to their linked bend path. Set it to Preserved, however, and you can move them around with impunity. Of course there's a trade-off: Optimised results in slightly smaller, more efficient files, whereas Preserved potentially stores additional data for any object that's been transformed, not just the ones that are causing us problems.

If you want to leave the setting as Optimised, there is a second alternative which allows you to add extra data to just the problem paths. It's a little counter-intuitive, but it does the job perfectly: just select your path and add a second Bend effect to it. You don't even have to modify the bend path – just adding the effect is enough to let you drag your path around independently of the linked bend path once more.



Whichever approach you take, you should now have two separate, independently positioned arrows, both of which are linked to the shape of the master bend path. Modify that path and you'll see the arrows shape change accordingly. If you don't want to see the bend path in your final design, simply hide it behind another object, set its opacity to 0 (use View > Display Mode > Outline to find it again) or just move it onto a hidden layer.

The remaining controls for the Bend LPE are fairly simple. The Width spinbox lets you control the scaling of the skeleton path, perpendicular to the bend path. Play with it to see the effect. The “Width in units of length” checkbox has a slightly misleading title: “keep width proportional to

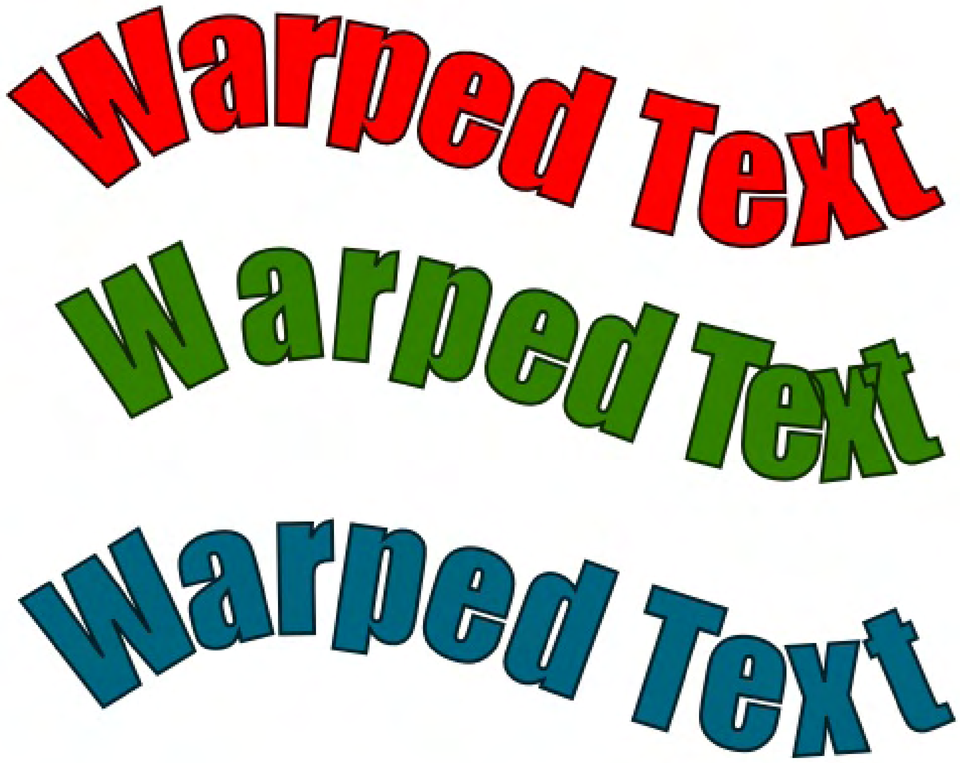
contents ^

length" would be a better name. Check this, and the width of the path is scaled as the length of the bend path changes; leave it unchecked to keep the width unchanged regardless of the shape of the bend path or the position of the end nodes. The final checkbox is quite self-explanatory: if you wish to bend a path that's more vertical than horizontal (e.g. an upwards facing arrow), then check this box, otherwise you'll be distorting along the width of the shape, rather than its length.

The Bend LPE is one that's well suited for use with text, to produce the sort of "Word Art" effects so beloved of parish newsletters in the 1990s. Because LPEs won't work directly on a text object, you first have to perform the one-way conversion of your text into a complex path. Using Path > Object to Path will result in a group of individual paths, one for each letter. We really want a single path encompassing the whole text, so it's easier to use Path > Combine, which will convert your text into paths, and combine them into a single complex shape, all as one operation. The final result will be a group of one object, so you'll probably want to ungroup as well.

From there you're free to add a Bend effect and distort your text as you would with any other path.

Before you race off to convert your text into a path, however, it's worth considering the downside: the shape is no longer a text object, so you can't subsequently edit the content if you find a mistake. Often a similar result can be obtained by drawing a separate bend path, then selecting both your text and path before using Text > Put on Path. You may need to manually kern some of the characters to get the right result (see part 11), but it has the distinct advantage of keeping your text editable. In this image the red text was converted to a path and bent, the green is the same text put onto a copy of the bend path, and the blue is the same as the green, but with some manual kerning applied.



One noticeable difference between the approaches is that the LPE distorts the shape of the letters, whereas text-on-a-path maintains their original shapes. Sometimes the distortion effect is desirable, in which case I can only recommend that you save a copy of the file just prior to converting to a path, in case you do need to edit it later.

Next time we'll move beyond simple path bending and into the kind of full-on distortions that can turn some simple text into a 1970s album title, as we continue to look at Inkscape's live path effects.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

HOW-TO

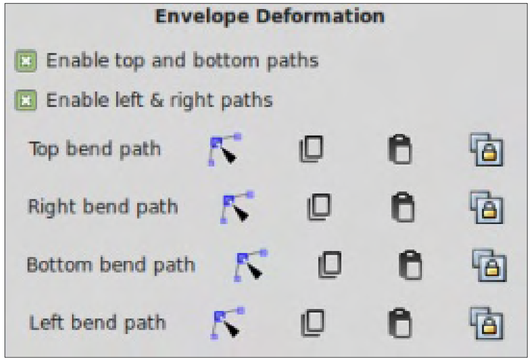
Written by Mark Crutch

Inkscape - Part 44

When I'm not drawing things in Inkscape, I like to spend my spare time playing lead guitar for a 1970s style prog rock band called “Envelope Deformation”. So when we decided to record our first album, it naturally fell on me to come up with a suitable logo for the band. My starting point, of course, was the name of the band in a bold font (Impact), with a golden gradient applied (when we hit the big time, no doubt our record company will pay for proper gold embossing on all our merchandising, so we may as well start as we mean to go on).

Envelope
Deformation

That's a good start, but it's a bit plain. I could use the Bend LPE from last time to add a bit of a curve to the whole logo, but 70’s style prog-rock really calls for something more indulgent. A quick scan through the list of LPEs reveals the suspiciously coincidentally named “Envelope Deformation” path effect, so that seems like an obvious one to try. After converting the text to a path (CTRL-K, then ungroup), then applying the LPE, I found myself faced with this user interface in the Path Effect Editor dialog.



Envelope Deformation

☒ Enable top and bottom paths

☒ Enable left & right paths

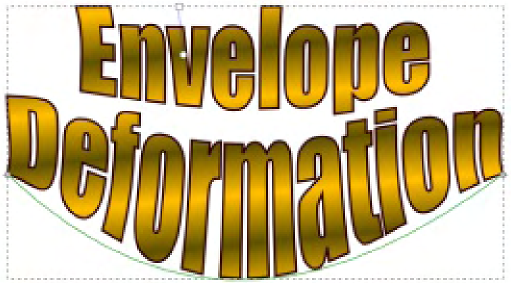
Top bend path

Right bend path

Bottom bend path

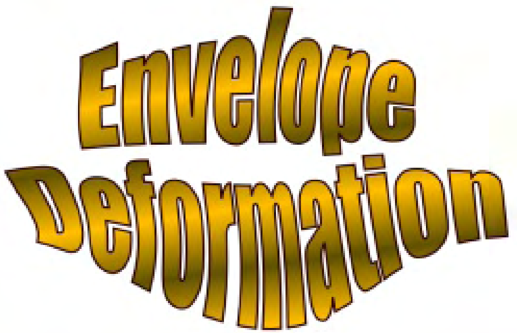
Left bend path

Skipping the two checkboxes for now, what will be immediately apparent is that this looks like four sets of the UI from the Bend LPE. The four lines correspond to the four sides of the path's bounding box: by default they are straight horizontal and vertical paths, but by using the buttons in the same way as we did for the Bend LPE, you can deform each side along a bend path. The result is as though your skeleton path is printed on a rubber sheet whose sides are stretched, distorting the shape. For example, clicking the “Edit on-canvas” button for the Bottom bend path allows me to quickly change the logo to something more suitable for an album cover.



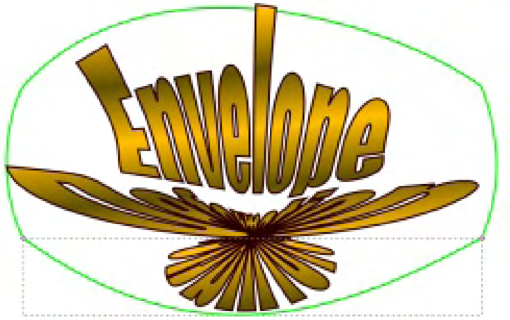
As you can see, the path currently being edited is displayed in green. Notice also that the deformation stretches across the whole height of the skeleton path, even though we only modified the bottom path. For more fine-grained control – such as keeping the top of the text horizontal – you have little choice but to engage in some manual node editing

With just the Envelope Deformation LPE and a bit of tweaking to each of the four sides, we get closer to a classic prog-rock logo.



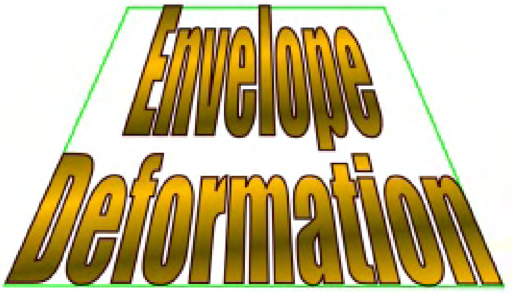
One thing I find frustrating is that three of the four bend paths are not displayed when you use on-canvas editing. For tweaking the shape of the path, that's not too much of a problem. But, if you decide to move the end nodes, it can be tricky to keep things in sync so that you don't lose the sharpness of the corners. Like the Bend LPE, however, our four lines in the UI also offer the ability to link to an existing path. It can be a bit fiddly to draw four lines that match the bounding box (Object > Objects to Guides can help), and then there's a lot of clicking to link them all up, but it does at least mean that you can keep an eye on all four paths at once – and even select nodes from more than one path at a time in order to move them in unison. Once your editing is done, simply set the opacity of the paths to 0 to make them disappear from sight.

One thing to watch out for with this LPE is the direction of the paths. If your paths don't match the directions that the effect expects, you can easily end up with a result like this:



If that happens to you, just use Path > Reverse to change the order of your path's nodes without affecting their positions.

As to those checkboxes… as their labels imply, they are used to enable or disable the top/bottom or left/right paths in the effect. Be aware that disabling a pair of paths is not the same as setting them to a straight line, which can lead to some confusing results. Where I find these options most useful is for creating trapezoid shapes. Here's the logo with left and right paths disabled, and the top path edited to be smaller than the bottom one:



The alternative, with left and right paths enabled, led to distortion around the bottom of the shape.

Of course you can use the Envelope Deformation LPE with any path, not just one created from text. You can also stack it up with other LPEs, in case you want to stretch some Spiro Splines or add some fake perspective to a set of gears – although you sometimes get better results if you "fix" the earlier LPEs using Path > Object to Path, at the expense of the live editing capability.

Like all good 70's bands, however, halfway through writing this tutorial we had "artistic differences" and split up. The keyboard player and bassist got custody of the name; I got the singer and drummer. So we needed a new name for the band. The singer suggested "Live Path Effects".

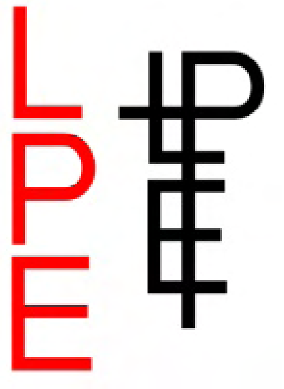
"We would abbreviate it to LPE," he said, *"with the logo being something more geometric made up of the three letters crossing over and under each other."*

"So something like the Emerson, Lake and Palmer logo?" I asked.

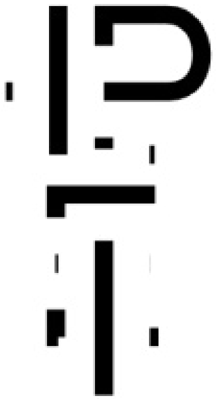
"Erm… no, not at all like that. Their's is ELP, whereas ours is LPE. See the difference?"

"I do, but I'm not sure their lawyers will."

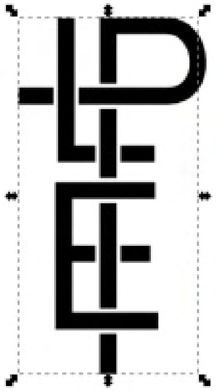
Despite my misgivings, I began work on the logo. Starting with letters made from simple paths (red), I extended and arranged them to create something more logo-like (black).

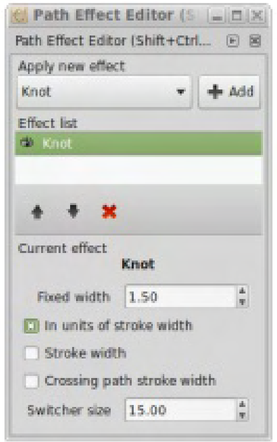


For the "crossing over and under" requirement, it's another trip to the Path Effects dialog. First, as usual, I turned my separate paths into a single composite path using Path > Combine (CTRL-K). Then I added the Knot LPE, and watched in horror as much of my path seemed to vanish completely!



Unchecking a couple of the checkboxes got things back on track a little. Before explaining what each of them does, it will probably be more helpful to see the final result, with the parameters that produced it:





Apply new effect: Knot

Current effect: Knot

Fixed width: 1.50

☒ In units of stroke width

□ Stroke width

□ Crossing path stroke width

Switcher size: 15.00

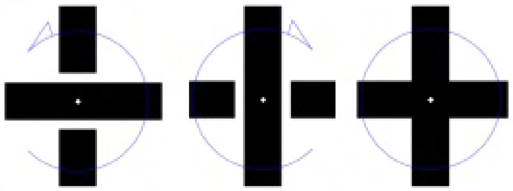
As you can see, this LPE automatically introduces gaps into a path where it crosses itself or any other sub-path. Most of the controls are used to simply alter the width of the gaps. With all the checkboxes cleared, the Fixed Width spinbox allows you to set a fixed size, in pixels, for the gaps. Check the “In units of stroke width” box, and it instead becomes a multiplier of the stroke width. The value of 1.50 that I've chosen just means that the gap will be 50% larger than the stroke, giving a nice 25% gap on either side.

It is possible to apply this LPE to a group of paths, rather than a single composite path. In that case you could be dealing with different stroke widths as a thick line crosses over, or under, a thin one. The last two checkboxes let you add the width of the “under” stroke (the one that gets the gap inserted) and the “crossing path” stroke, respectively. This can be useful to automatically compensate for line differences in complex arrangements or to have gaps that automatically adjust if you change the stroke width, whether explicitly or just by scaling your design.

Which brings us to the last control: Switcher size. In order to understand what this does, I first need to introduce you to the switcher. With your path selected, highlight the Knot LPE in the Path Effects dialog, then switch to Node Edit mode (F2, double-click on the path, or choose the second icon in the toolbox). You should now see the nodes of your path, as normal, but with one small addition. At one of the path crossings there will be a small, white, diamond-shaped handle. That's the switcher.

It can be really hard to spot in Inkscape 0.48 and, despite the name of the control, changing the Switcher Size parameter will have no effect whatsoever. On 0.91, however, the switcher is surrounded by a blue arc or circle. Modifying the parameter will affect the size of the circle, making it easier to spot the switcher on a busy path with lots of intersections.

But what does the switcher actually do? Clicking on it cycles the crossing between three states: the first two determine which path has the break (and therefore, which path appears to go over the other), whilst the third state removes the break entirely. Version 0.91 indicates these three states using a blue arc with an arrowhead pointing clockwise or anti-clockwise for the first two states, and a circle with no arrowhead to indicate the third state (0.48 offers no such indication). Unfortunately there's no fourth state to break both paths, leaving a large void. If you want that effect you'll have to manually break the paths yourself.



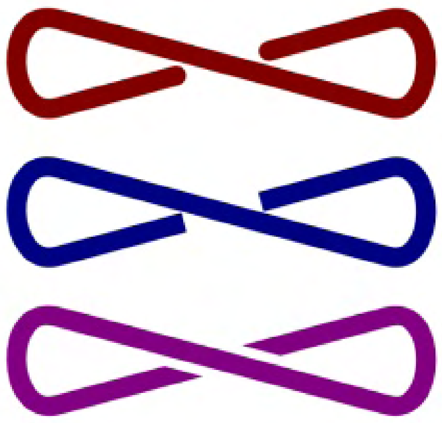
Whichever state you choose, however, only affects that one crossing point. You can drag the switcher to another crossing point in your design then change the state of that point by clicking. Being able to change only one point at a time like this, with a dragging process in-between, can quickly become tiresome on a complex design, but unfortunately there's no way to select or change multiple crossings at once.

With the basics of the design complete, I added a couple of finishing touches to turn it into a proper logo. First I copied the original path and removed the LPE before using Path > Stroke to Path. This resulted in an outline version of the logo that I could then apply an extra stroke to in order to thicken it. I copied this version again, leaving me with three paths, one of which has the Knot LPE applied. By setting the fill and stroke to white on one of the copies, and setting a thicker black stroke on the third, I was then able to stack the paths on top of each other to produce the final effect.

There are a few things worth mentioning about the Knot LPE. Trying to edit the skeleton path while the LPE is visible can lead to Inkscape crashes, especially if you're still using version 0.48. Make sure you save regularly, and know where any auto-save files are stored. Simply turning off the visibility of the LPE in the Path Effects dialog is enough to mitigate this problem and you can then make it visible again afterwards.

Depending on how your skeleton path was produced, you can also end up with unexpected breaks in it when using the Knot LPE. If this occurs, check for nodes that are doubled up on top of each other, perhaps as the result of a boolean operation. These can be fixed using the Node tool by rubber-band selecting the two nodes in question, then using the "Join Selected Nodes" button on the tool control bar to combine them into one. Where misplaced breaks are not due to doubled up nodes, your only recourse is to reshape your path a little. Try adding a node at a nearby intersection, then removing the one at the break, or adding another node close to the breaking one.

It's also worth reiterating that a Live Path Effect takes a path as its input, and produces a path as its output. Therefore your knotted path is still just a path, so is limited by the choice of end-caps that are available in SVG. If your paths cross at 90°, as in the logo example, butt or square caps will usually produce a good result. For anything else, however, you might find that rounded caps are better. This restriction does limit the artistic effects you can get from this LPE when lines have to cross at shallow angles. In the following example, the red and blue lines are broken using the Knot LPE with round and square caps. To get the effect of the purple line, however, it was necessary to convert the stroke to a path, then manually cut out the gaps.



All the practice of designing our new logo looked like it would be particularly useful when the lead singer declared that our album would be called "Celtic Knot". I quickly designed a potential album cover.

Thanks to the Knot LPE it didn't take too long to produce that design – which is a good thing, as the following day the singer decided that he was leaving the band to go on a spiritual retreat to India. Perhaps I should form an 80's style synth pop group instead. "Spiro Spline" sounds like a great band name to me…

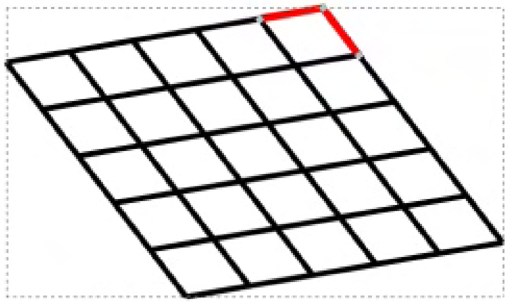
Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Written by Mark Crutch

Inkscape - Part 45

If you've read the previous few articles, you should now have an idea of how to use Live Path Effects, and just how capable they can be. Rather than go through every remaining effect in detail, I'm going to spend the next couple of articles presenting a whistle-stop tour of those that are present in version 0.48. These are all also in 0.91, and their respective interfaces are the same between the two versions, so these examples should apply to either version. In each example, I'll present the original skeleton path in red with the results of the applied LPE in black.

Construct Grid



A simple LPE to start with, the Construct Grid effect does exactly as its name suggests – it constructs grids. It uses the first three nodes of the skeleton path to define two sides of a parallelogram, extending the shape to form a grid of cells based on the Size X and Size Y values in the UI.

The "live-ness" of this effect can make it useful if you want to drag the nodes around to produce the correct perspective by eye, rather than by creating a grid with numeric angles.

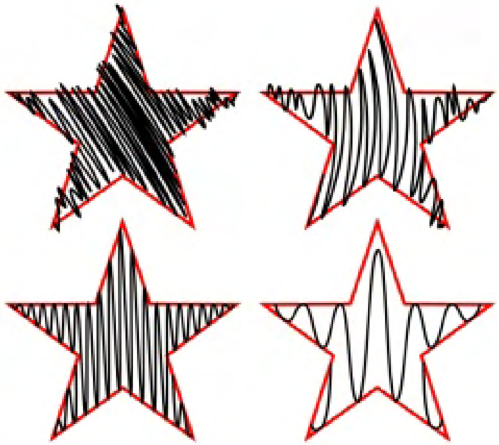
Hatches (rough)

This is a peculiar effect. Its main use is to simulate hand-drawn scribbles as a fill inside your (usually closed) path, but given the rough nature of the results – it even says "rough" in the LPE name – I don't think it really needs the huge number of fine-grained controls it presents. For most people, the key to using this LPE is to just use the on-canvas controls and a few of the main UI elements, without getting too bogged down in the many other options.

When you apply this effect to a shape, Inkscape draws one or more sine waves that try to fill the available area. The waves can be modulated in both frequency and amplitude by the parameters you set in the UI, and their angle, base frequency, and the amount of bend applied to them, are set by on-canvas handles.

To get a feel for the effect, draw a closed path, then add the LPE. You'll see your path replaced by a squiggly line that approximates the original shape. Now switch to the Node tool (F2), and towards the middle of your shape you should see four handles – two circular and two diamond-shaped. If you see fewer than four, then it's simply because some are positioned on top of others. Drag them around until all four are visible.

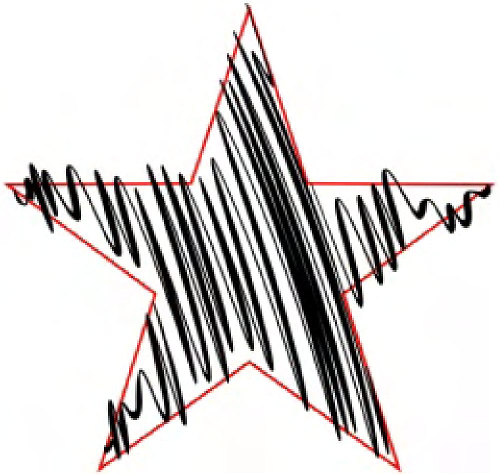
The four handles represent the end nodes of a pair of vectors (which, confusingly, aren't actually drawn as lines), and are used to set the main parameters for the effect. In each case the circular node is the reference point – drag that one, and the corresponding diamond will move in sync. This can be used to move the nodes to a clearer part of the canvas, or to some specific reference point in your drawing. Moving one of the diamonds adjusts both the angle and frequency of the sine waves used to fill your shape. The other diamond sets the amount of bend that is applied. It has an effect only if the Bend Hatches checkbox is ticked in the LPE dialog, so, if you don't want the additional curvature applied to your sine waves, simply uncheck that control.



Of the remaining controls in the dialog, it's probably the top two that have most effect: Frequency Randomness is used to adjust the amount of variation that is applied to the base frequency, whilst Growth causes the frequency to increase from left to right. Set both values to zero if you want to use just the base frequency that you've set with the on-canvas handles.

With these basic controls, it's possible to produce a variety of effects, running from the appearance of a hand-drawn scribble, to a simple shaped sine wave:

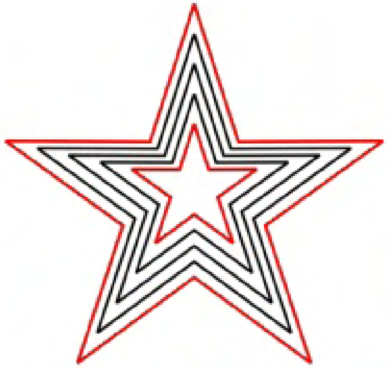
Many of the settings in the dialog have pictures of dice next to them. Despite their appearance, they don't actually set the fields to random values. Rather, these are buttons which change the seed value in the random number generator that's used to produce the corresponding value in the hatching algorithm. Their only real use is to ensure that one copy of a shape using this LPE has a different hatching pattern to another copy – if you need to produce many similar shapes then clicking a few of the dice will ensure that they all look slightly different from one another.



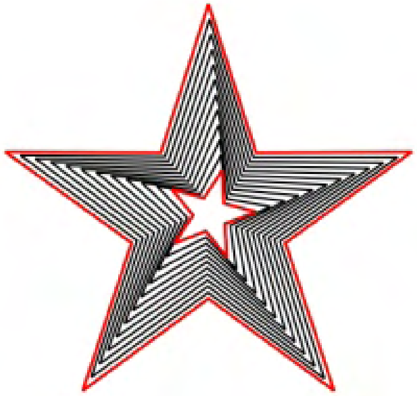
The final checkbox, “Generate thick/thin path”, is worthy of a mention too. With this enabled, two sets of paths are created that move in and out of sync with each other on each half-cycle of the underlying sine wave. The specifics of the synchronisation between them are set by the last few fields in the UI. These two sets of paths are actually joined at each end, forming a single path that can be filled to give a calligraphic effect to the hatching:

Interpolate Sub-Paths

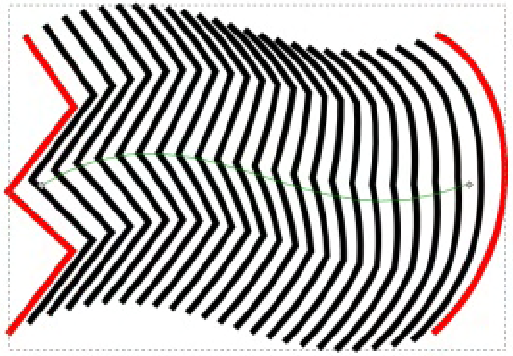
This effect requires that your skeleton path is made up of two sub-paths (if it has more than two, only the first and last ones are used by the LPE). Typically, sub-paths are created by combining multiple paths – through Boolean operations such as removing one path from another object that completely encloses it, or by breaking a single path into smaller sections by hand using the Node tool's Delete Segment or Break Path buttons. Consider this simple example of one star inside another, drawn separately, then combined using Path > Combine (CTRL-K). When the LPE is applied, a number of additional sub-paths are created, interpolating between the two sub-paths of the skeleton:



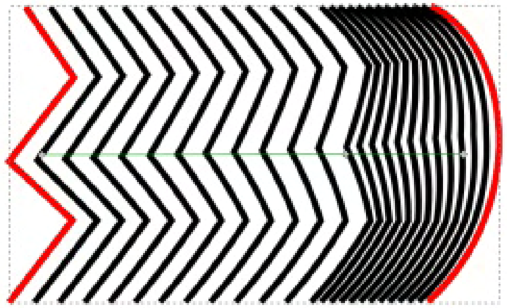
The total number of sub-paths in the final result is set using the Steps parameter. Increasing this, and turning the inner sub-path a little, demonstrates the sort of effects that you can easily create with this LPE:



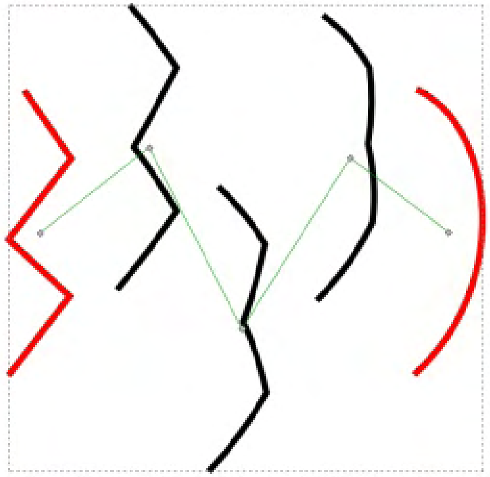
The Trajectory control in the LPE dialog shows the familiar group of four controls for setting a path. These allow you to specify a path along which the rendered sub-paths will be spaced, allowing for more than simple linear projections.



With the Equidistant Spacing checkbox ticked, the sub-paths will be placed evenly along the trajectory path. Un-check that, however, and their spacing will be determined by any additional nodes in the path. The nodes split the path into segments, then the total number of sub-paths is distributed between the segments. For example, a trajectory with three nodes will result in two segments, each holding half the rendered sub-paths. Moving the middle node, therefore, results in the spacing of the paths being adjusted – one half bunched together and the other half spread out.



By creating a trajectory with the same number of nodes as the Steps value for the LPE, each sub-path is tied to a single node, letting you accurately position them simply by moving the nodes around. This example uses Steps=5 together with a trajectory path that has five nodes, to demonstrate this possibility:



Next time we'll look at the remaining effects that are available in 0.48: Pattern along path, Ruler, Stitch sub-paths, and VonKoch.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

HOW-TO

Written by Mark Crutch

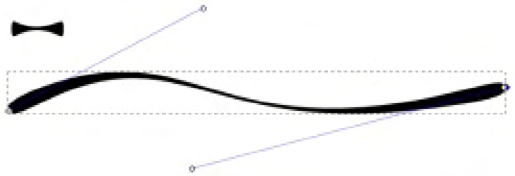
Inkscape - Part 46

This month we'll conclude our tour of the Live Path Effects that are available in both versions 0.48 and 0.91 of Inkscape, starting with perhaps one of the most useful.

Pattern Along Path

This effect is often referred to as “PAP” in forum posts and bug reports so, for brevity, I'll do the same here. Like the Spiro Spline effect, PAP can be applied automatically as part of the normal drawing process. To do so, you simply have to draw a shape for use as your “pattern”, copy it to the clipboard, and select the “Shape: From Clipboard” option when using the Pencil or Bézier tools. See part 17 of this series for more details on the technique, but suffice to say that the result is that your path will have the PAP effect applied to it – albeit with some default options selected. Whether you initially apply the LPE like this, or by explicitly adding it via the dialog, you'll find more controls available to you within the dialog's UI, in order to tweak the effect.

“Pattern along path” is something of a misnomer; it should more correctly be called “path along path” as the effect is one of taking a source path (the “pattern”) and stretching or repeating it along the skeleton path. In part 17, for example, I drew a rounded bow-tie shape, copied it to the clipboard, then used it to provide the shape for the Bézier tool, resulting in strokes which appear thinner in the middle.

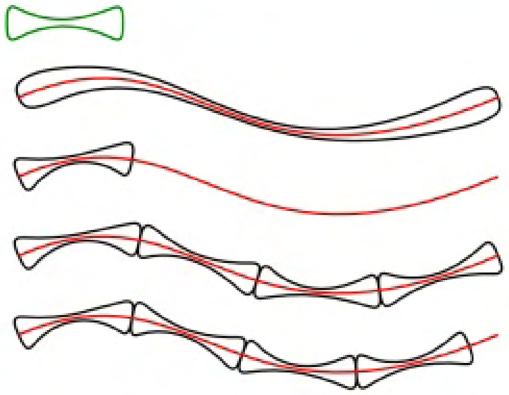


It's a useful technique, but the same visual result could also be achieved by using the Bend LPE. There's a philosophical difference between the two approaches, though: the Bend LPE uses your “pattern” as the skeleton path, then lets you distort it with an on-canvas path; the PAP approach, however, lets you copy the “pattern” to the clipboard, then stretches it to match the shape of the skeleton path. On the surface, therefore, it all boils down to whether you want the skeleton path to be your pattern or your target, but there's more to the PAP effect than that. The difference really becomes apparent only when you start to change the settings.

Perhaps the most important setting is the “Pattern Copies” pop-up. This defaults to “Single, stretched”, but there are three other options available, giving the following possibilities:

- Single, stretched – puts a single copy of your pattern onto the skeleton path, stretching it to the length of the latter (or shrinking it, if the skeleton path is shorter than the length of your pattern).
- Single – puts a single copy of the pattern onto the skeleton path. It's distorted to match the shape of the skeleton, but isn't stretched or compressed in length.
- Repeated, stretched – puts multiple copies of the pattern onto the skeleton path, stretching each of them in order to fill the length of the latter.
- Repeated – puts multiple copies of the pattern onto the skeleton path, but does not stretch them, usually resulting in a path that falls short of the skeleton's length.

You can see these four possibilities shown in order in this example – the green bow-tie at the top is the pattern that's being used, whilst the red line is the skeleton path.



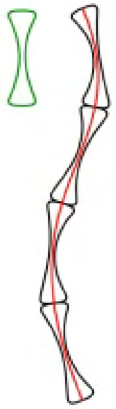
The Pattern Source buttons are the usual quartet used to define the path that's used as the pattern – by editing an on-canvas path, using one from the clipboard, or linking to an existing path. In this case, it's usually easiest to draw

contents ^

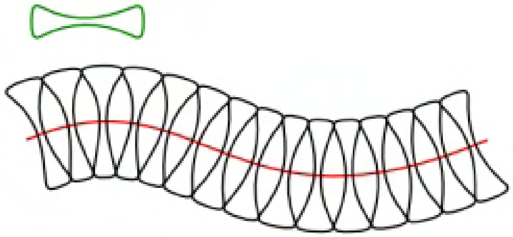
your pattern separately, then copy it to the clipboard and use the third button to apply it to your skeleton path. The first button can then be used to display a copy of the pattern on the canvas, for fine-tuning the shape.

The “thickness” of your pattern, perpendicular to the skeleton path, can be set with the Width parameter, with a checkbox to determine whether to use a fixed pixel width, or a multiple of the pattern's length. When using either of the “repeated” options, the gaps between shapes can be set using the Spacing field. Negative values are allowed, but only up to -90% of the pattern width. The Normal Offset can be used to push the pattern to one side or the other of your skeleton path, whilst Tangential Offset pushes it along the length of the path. The latter can be used to adjust the space at the end of the non-stretched options, but also works on the stretched modes to add some space at the start of the path, before the pattern begins. Once again, there's a checkbox to determine whether Spacing or Offsets are in fixed pixel values or proportional to the pattern length.

The penultimate control indicates to Inkscape that the pattern has a vertical orientation rather than a horizontal one. This is particularly useful to apply a vertical pattern to a vertical skeleton path:

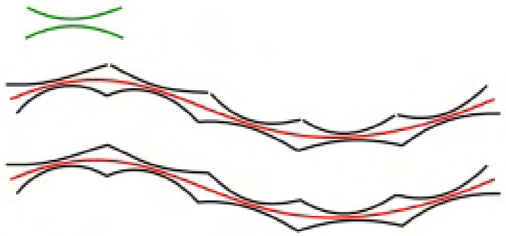


Because it effectively rotates the pattern through 90° before applying it to the skeleton, it can also be used to produce a different effect, when used with a horizontal pattern and path.



Of course, the same effect could be produced by simply rotating the pattern before it's used in the LPE.

The final control is used when your pattern is not a closed shape, but has unconnected ends. By setting this to a positive number, any line ends that are separated by less than the specified amount will be fused together to produce a continuous line. In this example, I've lopped off the ends of my bow-tie and slightly shortened the top section to exaggerate the effect. The two PAP examples show the effect of using this pattern with Fuse Nearby Ends set to 0, then set to a suitable positive value.

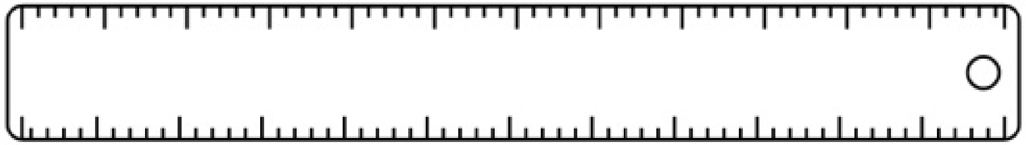


Ruler

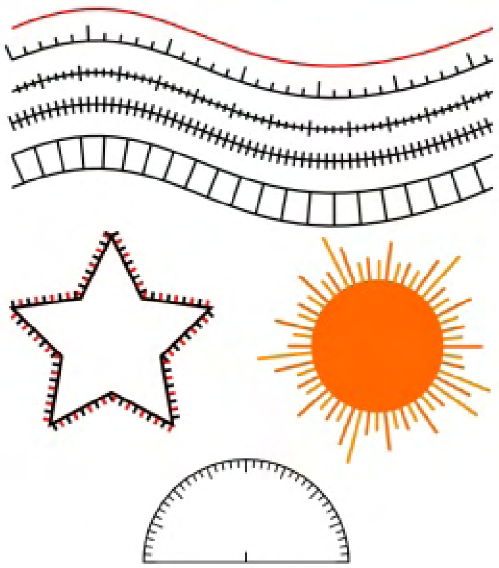
This LPE doesn't really require much explanation – it simply draws tick marks perpendicular to your skeleton path to give the appearance of the graduations on a ruler. You can define the distance between tick marks, the frequency of major marks, and the length of both the major and minor ticks. You can also determine which side of the skeleton the ticks will be drawn on, or have them centered to appear evenly on both sides.

One use for this effect is, as the name suggests, to create a ruler. For this example I've used a combined pair of parallel lines for the skeleton path, and overlaid them onto a rounded rectangle and circle.

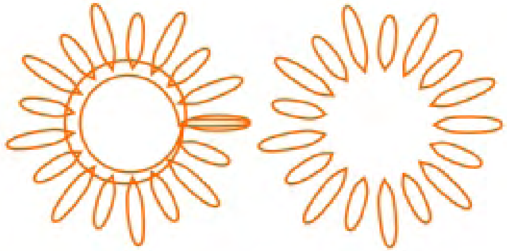
It's also possible to use this LPE for more artistic effects, though. When combined with other shapes it's easy for your ruler to become a zipper, a simple pathway, or even a protractor. By copying the object and applying different settings and colors to the LPE, you can easily



get the effect of coloured rays emanating from your shape.



One frustrating omission from this LPE is the ability to suppress the original skeleton path in the output. This makes it more difficult to chain this effect with others. A workaround is to use Path > Object to Path to "fix" the LPE before manually removing the skeleton path, then applying other effects – but you do then lose the ability to do live edits to the parameters. For example, chaining the Ruler, then PAP effects on a circle, gives a result like the one on the left here, whereas the example on the right shows the "fixed" version with the skeleton removed before the PAP effect is added.

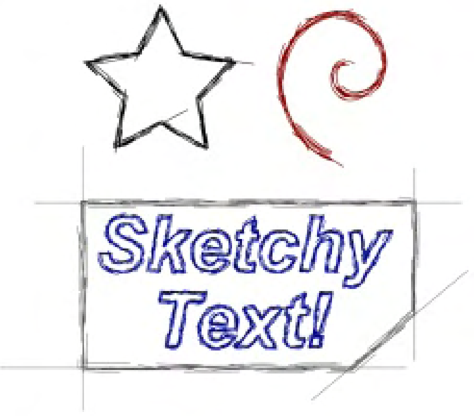


Sketch

This LPE does one simple thing, but, like the Hatches effect, the number of controls greatly exceeds any practical requirements for such a deliberately un-tamed result. In short, this effect simply replaces each part of your path with a number of smaller paths, overlapping and with their ends offset from the skeleton by a semi-random amount. It gives the effect of having sketched your path with repeated strokes of a pencil – and can even include construction lines for extra effect.

At the top of the UI are the controls for setting the number of paths that will be used for approximating each section of the original, and for determining how long each can be and how much they can overlap. Usually it's sufficient to adjust only the top couple of controls to set the "density" of the sketch strokes – fewer, longer strokes for a light sketch effect, more short strokes for the appearance of a more heavily scribbled line.

The Average Offset and Max Tremble controls are useful for determining the "thickness" of the sketched result. There's also a control for the number of construction lines – set it to zero if you don't want any. In this same area of the UI, the Max Length parameter is useful to sufficiently extend your construction lines from the original shape. As with the Hatches LPE, the dice buttons can be used to set a new random seed used for some of the parameters, which is only really of use for making otherwise identical copies look dissimilar.



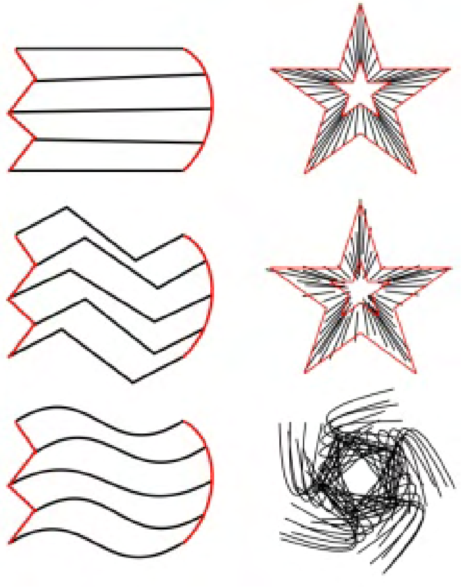
Note that this effect can easily produce lots of new nodes, so be careful when using it as anything other than the last LPE in a chain. Here's a small showcase of the kind of results it can produce.

Stitch Sub-Paths

This effect can be thought of as a perpendicular version of the Interpolate Sub-paths LPE that I described last month. Whereas that creates a connection between two sub-paths by introducing interstitial versions that gradually distort from the shape of one path to the other, the Stitch effect joins the two sub-paths directly with a series of new paths that link evenly spaced points on one path to evenly spaced points on the other. In other words, it draws some lines from one sub-path to another.

Once again there are too many controls to be useful. You really only need the first parameter – for setting the number of new paths to draw – and the quartet of buttons for manipulating the "Stitch path". Most of the other controls are there to let you add some randomness to your stitches, should you wish.

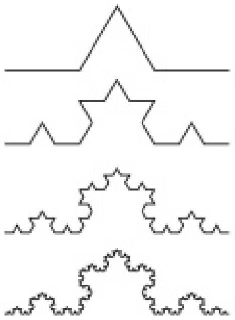
In its simplest form, this effect just draws straight lines from one sub-path to another. It can work on shapes with more than two sub-paths, but, for anything other than simple shapes, it can be rather unpredictable as to what the result will be. By using the buttons to paste, link to, or modify a stitch path, you can replace the straight-line stitches with something more complex. It can be good for adding a little curvature to the lines, but once again it becomes difficult to control the result as things get more complex.



In the example above, you can see, in the left hand column, some simple stitching between two basic shapes. The middle and bottom examples show the effect of altering the shape of the stitch path. The stars show a simple stitching, the application of some randomness via the UI, and the result of bending the stitch path (skeleton path omitted for clarity).

VonKoch

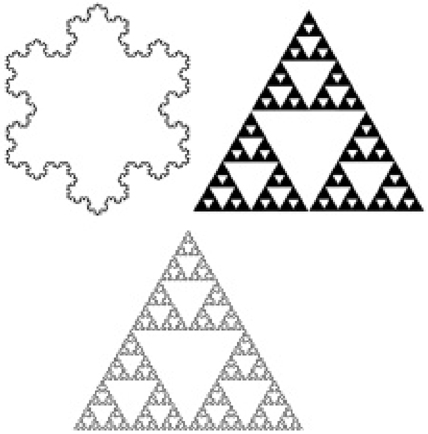
Finally we have the VonKoch LPE. If the name sounds familiar, then perhaps you've come across the Koch Snowflake – a fractal shape created by recursively replacing the middle of each side of an equilateral triangle with a smaller equilateral triangle. It was derived from a paper by Swedish mathematician Helge von Koch, who described the process for one side of the snowflake shape, creating a “Koch curve”. This image shows the first four iterations of the curve:



When you apply this LPE, you'll find that a pair of new copies of your path appear on the canvas. Within the UI you'll also find path buttons for a “Reference Segment” and a “Generating Path”. The two new paths you can see correspond to two sub-paths within the Generating Path. If you use the third button to paste a different path in, you'll see the number of copies change to reflect the number of sub-paths in the new Generating Path. Somewhat ironically, you have to paste in a new path to create a Koch curve, as this requires four copies, not two – so immediately the LPE makes it difficult for a layman to create its eponymous fractal! The Reference Path is used to position the copies on their Generating Path segments – essentially the skeleton is scaled and positioned such that the Reference Path lies on top of each Generating Path segment.

If that makes it all sound rather confusing, that's because it is! To adequately explain the operation of this LPE would require an article of its own. If you do wish to explore this one further, I recommend reading Tavmjong Bah's description in the official Inkscape manual, which also includes step-by-step instructions for creating a Koch curve: http://tavmjong.free.fr/INKSCAPE/MANUAL/html/Paths-LivePathEffects-VonKoch.html

If you do persevere with this LPE, it can produce some impressive results – as well as a lot of frustration. I did manage to create a Koch Snoflake, the Sierpinski triangle (another stalwart of fractal geometry), and a Sierpinski arrowhead curve, but it is a far from intuitive process.



Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

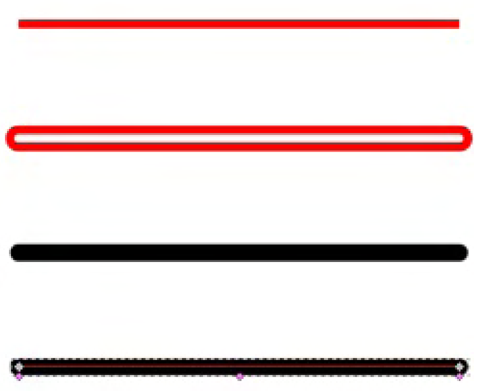
Written by Mark Crutch

Inkscape - Part 47

This month, we'll conclude our tour of LPEs by looking at the two new additions to Inkscape 0.91: Power Stroke and Clone Original Path. One limitation artists often find with Inkscape is its inability to produce variable thickness strokes. There are ways to fake it, which pretty much all rely on the "stroke" actually being a filled path in its own right. That inevitably leads to the follow-up problem of how to fill a shape drawn using such fake strokes. These two LPEs are Inkscape's answer to those problems.

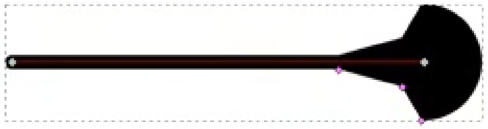
Power Stroke

This effect lets you vary the width of your path by adjusting "stroke knots" along its length. For cases where you might otherwise use PAP to provide some variability to the stroke width, Power Stroke will often achieve a similar effect but with more flexibility. Let's start with the simple example of applying this LPE to a straight line:

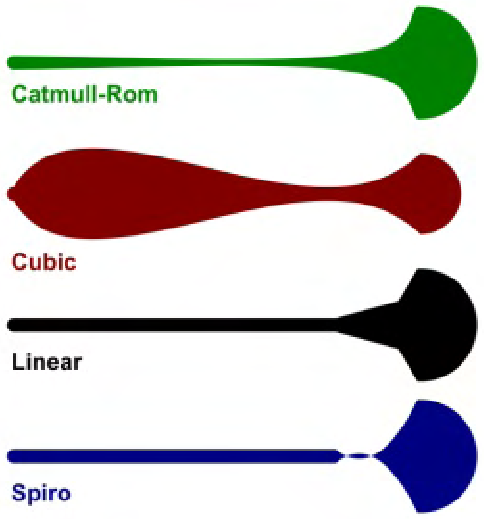


The top line, in red, represents the original skeleton path. On applying the LPE the first result you'll see is that the skeleton is replaced by a closed path that encloses the original shape. The closed path takes on the attributes of the original skeleton path, so, in this case, it has a red stroke and a transparent fill, resulting in the second shape in the image. Usually you'll want your Power Stroke to be filled with no outline, so in the third image I've changed the style to a black fill with no stroke. You would be forgiven for thinking that we've just gone through a convoluted way to produce a slightly thicker black line with rounded end caps, but the fourth image shows the real secret to Power Stroke: this is the same as the third object, but with the Node tool selected (F2), revealing not only the normal start and end nodes, but three additional pink or purple colored handles (the "stroke knots") sitting on the periphery of the shape.

The effect automatically adds these nodes at the start and end of the path, and somewhere towards the middle. Using the Node tool they can be dragged perpendicularly to the skeleton path to set the stroke width at that point, but can also be dragged along the path to change the location at which the thickness changes. Taking the previous example; dragging the handles around a little lets us easily produce this result:



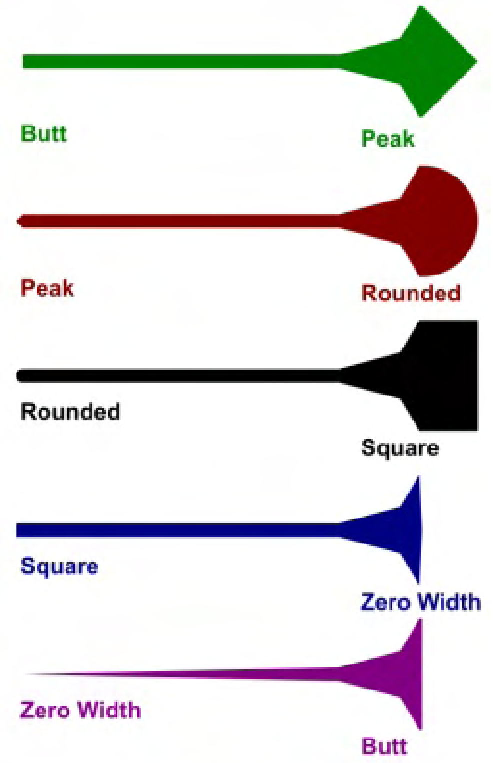
As you can see, the thickness of the line is set by each of the nodes, with sections in-between ramping linearly from one node to the next. Looking at the LPE's UI you'll notice a pop-up menu for the Interpolator Type. This is what is producing the linear change; pick another value to alter the way in which the width of the path is modified from one node to the next.



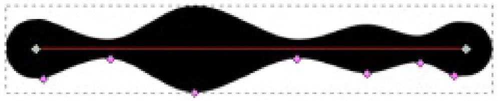
You can't fail to have noticed the rounded ends to the line. They weren't present on the original skeleton path, so where did they come from? A quick scan of the effect UI will show that several of the controls from the Stroke Style tab of the Fill and Stroke dialog are replicated as part of the LPE. From there, you can set start and end

contents ^

caps (the source of the rounded ends in this case), as well as the join type and miter limit for paths with angles in them. These all operate in a similar way to the equivalent controls in the Style and Stroke dialog, except that the LPE offers more options.

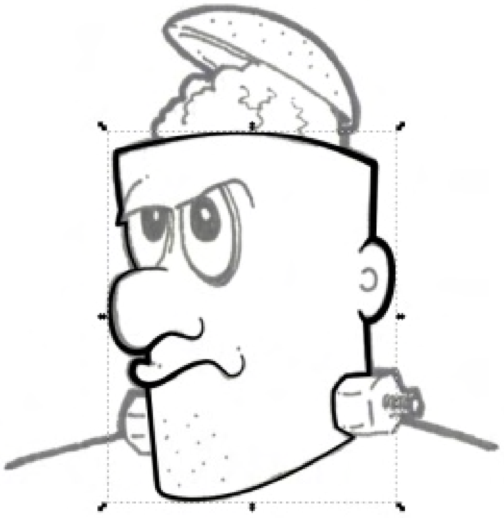


With three handles for manipulating the Power Stroke, you can already produce some useful results, but the real power comes when you add even more handles. Unfortunately, this is done via a rather clunky approach that can sometimes be a bit unstable, particularly on more complex paths. The first step is to select your path and switch to the Node tool, so that the handles are visible. Next you have to hold the Control-key whilst clicking on a handle. This will create a second handle, directly on top of the first one, which you can then drag to its new location. If you do experience problems, such as the handle becoming detached from the path and having no effect, undo your changes, and then try again, duplicating a different handle instead. Our simple line with three handles quickly turns into something more bumpy once a few more are added:



To delete a handle you have to click on it whilst holding the Control and Alt keys. There's no way to select multiple handles in order to move or delete several at a time. You've probably noticed that you can drag handles past one another with ease – that's thanks to the Sort Points checkbox in the effect's UI. Uncheck that for a different behaviour in which the shape is drawn from handle to handle based on their original ordering rather than their final position along the line. It's useful for some effects, but generally it's better to leave the box checked.

With all that background out of the way, let's take a look at this LPE when used on a more complex path. It's time to return to my efforts to manually trace "Frankie" (see parts 16-21 for my previous attempts at this):



The parts of the path that stretch inside the outline – the mouth, nose and ears – were particularly fiddly to get right. In practice it's probably faster to just use a simple path for the outline, then draw those parts separately. But you can clearly see how the LPE allows the path to thicken and thin to give more of a dynamic feel to the character than a simple fixed width line could achieve.

Clone Original Path

Our final LPE follows on directly from those efforts to manually trace Frankie. One problem with the Power Stroke or PAP effects is that they draw strokes as filled paths, so setting a fill color on them actually changes the color of the "stroke", not the area inside it. To clarify, if I were to select the Power Stroke path in the Frankie example and set its fill color to red, the result would just be a red Power Stroke, not a black Power Stroke with a red fill inside the face area.

Previously, I've shown you how to work around this limitation using the Bucket Fill tool or by manually drawing a second path that you can fill and then send below the outline. Neither of these solutions is ideal, and both can take a lot of manual tweaking to get right. Wouldn't it be easier if you could just fill the original skeleton path with a different color, without affecting the fill that's used for the Power Stroke? Effectively that's what the Clone Original Path effect lets you do.

There are a few ways to use this effect, but we'll start with the long-winded method, to give you a better understanding of what's happening. First you'll need a sacrificial skeleton. Don't worry, we're not heading into Voodoo territory, but rather you'll need a skeleton path that will completely disappear once you use the LPE. Its only purpose is to serve as an object to apply the effect to, so a simple straight line will suffice. Select the path and add the Clone Original Path LPE to it, then gasp in amazement as… nothing happens. There are a few more steps to go through before the effect has any visible impact.

Having applied the effect, you next need to select your Power Stroked path, then copy it to the clipboard. Re-select the sacrificial skeleton and use the first button in the effect's UI to paste the path from the clipboard. Your skeleton will disappear, and it will seem that the Power Stroked path has been selected instead. Appearances can be deceptive, however – use the arrow keys to move the selected item and you'll realise that you've actually got a clone of the skeleton path used in the Power Stroke. Fill it with a color and send it back in the stack and you've achieved in seconds what would have taken several minutes to do manually.

You can fill your clone with a gradient or pattern, if you prefer, or change the stroke style. In principle you can also apply other LPEs – although they don't always chain as well as you might hope. And because it's a clone, you can change the shape of the original skeleton path (the one used for the Power Stroke), and your filled version will automatically update to match it – comic creators rejoice! No more does every tweak to an outline have to entail a corresponding manual update to the fill shape!

With the cloned path selected, the second button in the UI will select the original – though the classic Shift-D shortcut or Edit > Clone > Select Original menu entry both also work. Cloning a path in this way isn't restricted to the Power Stroke LPE, so if you need a copy of the skeleton that you used with any other effect, just follow the same steps.

You can streamline the creation process a little by copying the Power Stroke path to the clipboard before you create your sacrificial skeleton rather than afterwards. But the Inkscape developers are nicer than that, and have added an option that will automatically create an infinitesimally small sacrificial path that has just a single node, add the Clone Original Path effect to it, and connect it to your original path, all from a single menu entry. Just select the Power Stroked path, then use Edit > Clone > Clone Original Path (LPE), and then set the fill and stroke you want for your clone. The only way it could be any faster is if there was a keyboard shortcut for the menu option.

But wait! Inkscape 0.91 does include a keyboard shortcut editor. Click on the Edit > Preferences menu item, then in the dialog, drill down to Interface > Keyboard Shortcuts. Expand the Edit section of the pane on the right and you should find Clone Original Path (LPE) in the list. Click in the Shortcut column for that entry, then press the new keyboard shortcut you wish to use (I went for CTRL-ALT-SHIFT-D to keep it in line with the other cloning shortcuts).

There's one final trick up the sleeve of this LPE. Back in part 30, I introduced the use of “unset” fills and strokes to allow different clones to have different styles and colors. This effect offers another way to achieve a similar result. You can either select an original object that you wish to clone, and then use the Edit > Clone > Clone Original Path (LPE) menu option, or you can select a clone that you've already created and just click the “+” button in the Live Path Effects dialog. Now you can change the color and style of your clone with impunity, safe in the knowledge that changes to the original shape will still be

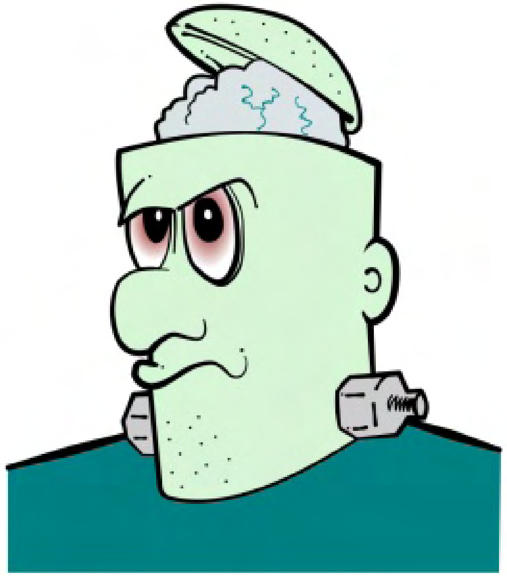
contents ^

reflected. For obvious reasons this works only where the parent object is a path or can be trivially converted into one – so it does work with stars, spirals and even text objects, but doesn't work with groups or bitmap images.

It's worth noting that this approach does result in a second copy of the path data being stored in the clone's XML (see part 43 for details), unlike a normal clone which just hold a reference to the original. This means that not only is the file size a little larger, but any edits made to the original outside of Inkscape won't be reflected in the LPE clone, whereas they would with a “real” SVG clone. One advantage of this approach, however, is that you don't have to unset the fill and stroke on the original, so you won't be left with a black shape that you might have to hide under another object or by placing it off the side of the page.

Conclusion

The Power Stroke and Clone Original Path LPEs are worthy additions to Inkscape 0.91. For a comic artist they could be reason enough to upgrade from an older version. Using little more than these two effects produced the best manual trace of Frankie so far:



We've now reached the end of our tour of LPEs. There are more being added to the development builds all the time, so do check out the dialog with each future version of Inkscape. They represent perhaps the most common way in which the developers have broken through the limitations of the SVG format in order to add functionality that far exceeds what any normal SVG editor could offer. It's true that the UI for some is confusing, they're sometimes a little unstable, and that they often don't chain as well as they should, but it's well worth spending some time to play around with them as they can open a path to drawing possibilities that would be impossible or, at least, impractical to produce any other way.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

Written by Mark Crutch

Inkscape - Part 48

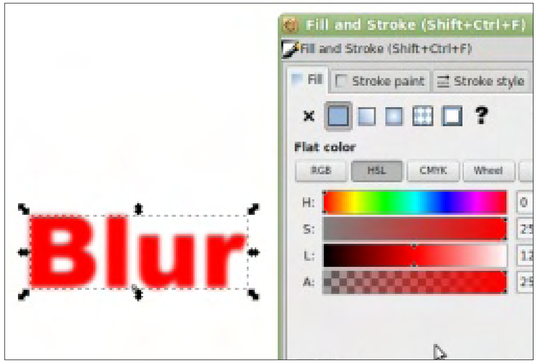
If there's one word that can stir up controversy amongst experienced Inkscape users, it's “filters”. Attitudes range from “they're bitmaps, and therefore bad”, through “it's easier to add filters to your image in GIMP”, right up to “filters are great”. I'll go on record as putting myself in the last group, but I would definitely extend the statement to add “(if a little slow and clunky)” to the end of it. So what are filters, in Inkscape terms? And why are they so divisive?

In short, filters are part of the core SVG specification that offer a way to perform bitmap operations on your vector objects. Filters consist of a number of “filter primitives” that can be linked together to create a “filter chain” that produces the desired effect. They apply at the point that your elements are being rendered to a bitmap for display, export or printing, and operate at the same resolution as the output device. So, although they are pixel-based, they can be just as crisp and scalable as the unfiltered vectors in your drawing. It does mean, however, that they are purely a display feature and have no effect on the underlying geometry of your image – so they're of no use to anyone trying to produce drawings for use on a vector output device such as a vinyl cutter or laser engraver.

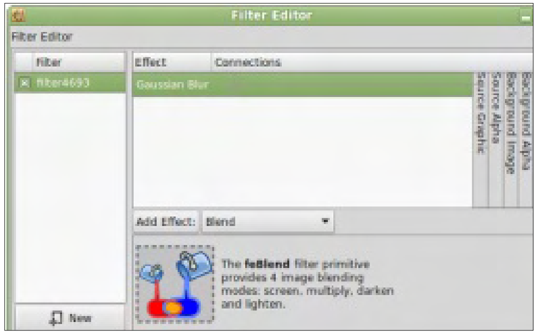
Filters are also “live”. You can tweak and modify the parameters used in your filter throughout the life cycle of your drawing, whereas filters in a bitmap editor like The GIMP are usually fixed and permanent once they've been applied. This is both a blessing and a curse: SVG filters are incredibly flexible because you can change them as you go along; conversely they eat up a lot of processing power as they need to be re-calculated not only when you change their parameters, but potentially each time you pan or zoom. The performance penalty can be severe, especially when zooming far into your drawing, leading some users to avoid filters entirely, although the worst problems can usually be avoided by using a few simple techniques that I’ll describe in a future article.

Finally, filters are rather unintuitive – and Inkscape's UI for editing them only makes this problem worse. The developers have included a great set of predefined filters, which was improved further in 0.91, but it's still useful to understand how to use the editor in order to tweak and extend them. So let's start by taking a look at the editor courtesy of the most common filter primitive: Gaussian Blur.

Gaussian Blur (or just “Blur” from now on) is the most commonly used primitive – principally because it’s exposed directly in the Fill and Stroke dialog. Many people happily use the Blur slider in that dialog without ever going anywhere near the full Filter Editor, but it also provides a convenient mechanism for creating a “stub” filter chain that you can develop further. Of course you’ll need an object (or group) to apply your filter to, so begin by creating a text object, give it a nice bright fill color, then add a little blur using the slider in the Fill and Stroke dialog.



Now open the Filter Editor using the Filters > Filter Editor menu entry. If it opens in a pane within the main Inkscape window, I suggest dragging it out as a separate floating dialog. This will let you resize it to give you more space to work with – filter chains can quickly become long and some of the primitives have a lot of parameters to tweak.



The **feBlend** filter primitive provides 4 image blending modes: screen, multiply, darken and lighten.

contents ^

The left of the editor is given over to a list of the filters in your document. Assuming you started with a blank drawing, you should see only a single entry here, given an automatically generated name along the lines of “filter1234”. That entry will have a mark in the checkbox, indicating that it’s the filter that’s in effect on the currently selected object. If you want to apply the same filter to another shape, just select that object in the canvas window, then check this box in the editor; you can use a single filter chain on multiple elements – which is useful when you want multiple text objects to share a single drop shadow, for example. Finally, in this section you can create a new filter from scratch using the “New” button at the bottom, or right-click on a filter entry to duplicate or remove it completely. You can also rename it from that context menu, but it’s usually easier to just double-click on the filter’s name and enter a new one. Giving slightly descriptive names to your filters makes it easier to keep track of the important ones as your drawing develops. For now, why not rename “filter1234” to “Blur”?

With your filter selected, you should see a single entry in the list on the right of the dialog: Gaussian Blur. This is your filter primitive, and it’s this list that’s used to stack and combine primitives into chains. For now, click on the Gaussian Blur entry to select it, and then look to the bottom of the dialog where you’ll find its parameters. Gaussian Blur takes two parameters, but by default Inkscape locks them to the same value via the “Link” button to the right. By toggling that to un-link the parameters, you can provide different values for horizontal and vertical blur, providing the opportunity for “motion blur” effects that make it look like your object has moved in one direction. Note that the scales are labelled in units of “Standard Deviation”, whereas the slider in the Fill and Stroke dialog shows a percentage value. The former is used in the SVG spec, whereas the latter is probably more understandable for users who just want to add a little blur without going near the filter editor. Suffice to say that the two fields are just different representations of the same underlying value, so the fact that they usually hold slightly different numbers isn’t really a problem.

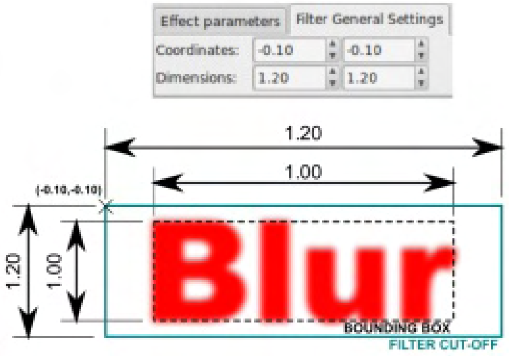
If you're still on 0.48, and drag the sliders to the right, you may notice that the blur on the text starts to get cut off at around the 20% mark, and by the time you reach 50% it’s become a ghostly rectangle. You’ve just encountered one of the most common problems reported by users who take their first steps into filters, but don’t worry, it’s an easy one to fix. In principle, some filters – including Gaussian Blur – could continue off into infinity: mathematically speaking, a blur represents an infinite series of calculations, although the results quickly drop off to the point that the calculated values have no visible effect on the drawing. Obviously performing an infinite series of calculations isn’t possible for even a high-end machine, so the SVG spec allows for a window or cutoff to be defined, beyond which the rendering engine shouldn’t bother performing any more calculations. By default this cutoff is set to allow a 10% margin all around your filtered object, which is fine for a small blur, but clearly not enough as the blur value increases.

The cutoff is adjusted via the “Filter General Settings” tab, and, as the name suggests, it affects the whole filter, not just the currently selected primitive. Within this tab you’ll find two sets of parameters, labelled “Coordinates” and “Dimensions”. The former sets the position of the top left of the filter window, relative to the width of the object. The default values of -0.10 mean that the cut-off rectangle starts 10% up and to the left of the object’s bounding box. The latter pair of values sets the width and height of the filter window, so the default value of 1.20 results in a cut-off that’s 20% larger than the bounding box. Because the Coordinates fields have offset the window by 10% to the top left, the result is a cut-off that symmetrically surrounds the original object with a 10% margin. To use a large Gaussian Blur value, you might want to increase this window to give you a 50% margin all round: that would entail setting the Coordinates to -0.50 and the Dimensions to 2.0. Most of the time you don’t need to adjust these values, but when you start to see your filtered objects being unexpectedly cut off at the edges, the Filter General Settings are almost always the cause.

In the copy of 0.91 on my machine, this problem seems to

contents ^

have been addressed by automatically modifying the settings to sufficiently encompass the blur. However I can find no mention of this change in the 0.91 release notes, so it's not clear if this only applies to blurs or simple filter chains, or if the algorithm being used is robust enough to handle complex chains as well. Therefore I recommend familiarising yourself with this tab, even on 0.91 – though with luck you'll never have to use it.


Effect parameters | Filter General Settings
Coordinates: -0.10 -0.10
Dimensions: 1.20 1.20

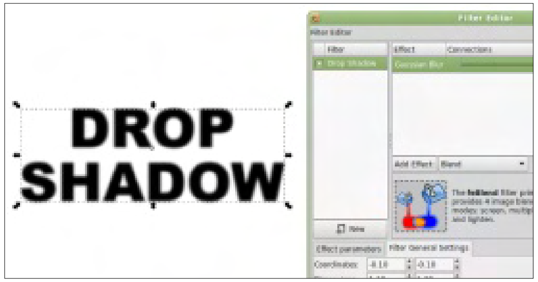
Before moving on, it's worth having a quick recap to make sure you're clear about the dialog so far. On the left is the list of filter chains, each with a checkbox to apply it to the currently selected object. From here you can create new chains (though just adding some blur to your object has a similar effect), and manage existing ones. On the right is the list of filter primitives that constitute your filter chain – though so far we've only dealt with a rather short chain consisting of a single primitive. At the bottom of the dialog is a tab for the currently selected primitive's parameters, and another for setting the filter cut-off window position and size.

Now, let's move back to the list of primitives that we so quickly glossed over previously. Looking more closely at the Gaussian Blur entry you'll notice that the "Connections" section contains a barely visible triangle, from which is emanating a line that connects to a column on the right with a "Source Graphic" label running vertically down it. The triangle represents an input into the filter primitive, and the column is one of several possible sources for that input. Unfortunately, of the six inputs shown in the UI, two of them require special treatment (and will be covered in a future article), and another two don't work at all! Of the two that do work, the "Source Graphic" column is exactly what it sounds like – it's used to 'inject' a bitmap representation of the selected object into the filter chain. The "Source Alpha" column is used to inject a bitmap representation of the object's alpha channel with solid black pixels representing opaque parts of the object, transparent black pixels for the transparent parts, and translucent black pixels for those parts that are somewhere in-between. In other words, it injects a blackened silhouette of the object.

As well as having the triangular input nodes, each primitive also has a single output. There's no obvious output node, instead it's the bottom edge of the primitive that acts as the output in the Inkscape UI. It's possible to connect the bottom of one primitive to the input triangles of other primitives, thus building a chain, but the output from the last primitive is always used as the output from the chain as a whole.

Let's build on our Gaussian Blur to create a simple drop shadow filter. During this process, you'll learn how to link primitives together to form a chain, and hopefully begin to understand a little more of the power of filters. Start by renaming your existing filter to "Drop Shadow", re-link the parameters if you need to, and set the blur to a fairly small number – enough that you can clearly see it applied to your object, but not so much that the it just turns into a fuzzy cloud. A value of 2-3 should do the trick.

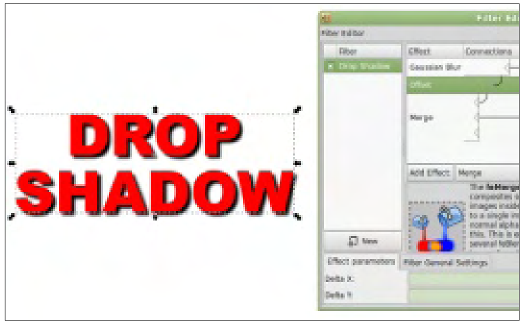
Our shadow is going to be dark, made of translucent black pixels, so the first thing to do is to generate a silhouetted version of our object to pass as an input to the Gaussian Blur primitive. But, of course, we already know of a source of silhouettes – the Source Alpha column. In a slightly back-to-front operation, we can link this source to the Blur's input by clicking and holding on the triangle, then dragging the mouse to the Source Alpha column before releasing (yes, you drag from input to source, rather than the other way around). If all went smoothly, the line that previously ran to the Source Graphic column has been replaced by one to the Source Alpha instead. Take a look at your text object and you should find it's turned into a blurry black version of the original.


Coordinates: -0.10 -0.10

If we were to display the original over the top of the blurred alpha version, you would just see a halo of darkness around your text. To make it work as a drop shadow we need to offset our blurred image from its original position using the imaginatively titled “Offset” primitive. Select it from the pop-up list just below the filter chain, then click the adjacent “Add Effect” button to add it to your chain. It should automatically be connected to the output of the Blur, as indicated by a small line running from the triangular input of the Offset to the bottom of the Gaussian Blur. Adjust the new primitive’s Delta X and Delta Y parameters to shift your shadow down and to the right – a value of 6.0 in each is a good start.

The final step in creating our filter is to add the original graphic back on top of the blur using the Merge primitive. Once again you should select it from the pop-up list and add it using the Add Effect button, but this time it won’t be automatically connected to the rest of the chain. The Merge filter combines multiple input images by stacking them on top of each other, honouring any transparency they may have in the process. The first input goes at the bottom of the stack, the last input at the top, so we need to add the offset blur first and the original source graphic second.

Begin by clicking in the Merge filter’s sole input triangle and then, holding the mouse button down, drag to the triangle in the row above (the Offset filter). Release the button and you should see a connection made, running from the base of the Offset to the input of the Merge. You’ll also notice that the Merge filter has gained a second input triangle. Click and drag from this second triangle to the Source Graphic column. Check the canvas, and you should now have a glorious drop shadow. See, filters aren’t so tricky… right?



Now tweak the Gaussian Blur and Offset parameters to change the softness of your shadow or its relative position. Then edit the text itself. Each change you make takes place live, and you can re-open the filter editor at any time to make further changes. Try creating a “hard” drop shadow by merging an offset Source Alpha with the Source Graphic, but without using the Gaussian Blur. Or try a bit of motion blur by un-linking the horizontal and vertical sliders; adding some horizontal blur and a horizontal offset; then merging with the Source Graphic again.

HARD
SHADOW
MOTION
BLUR>>>

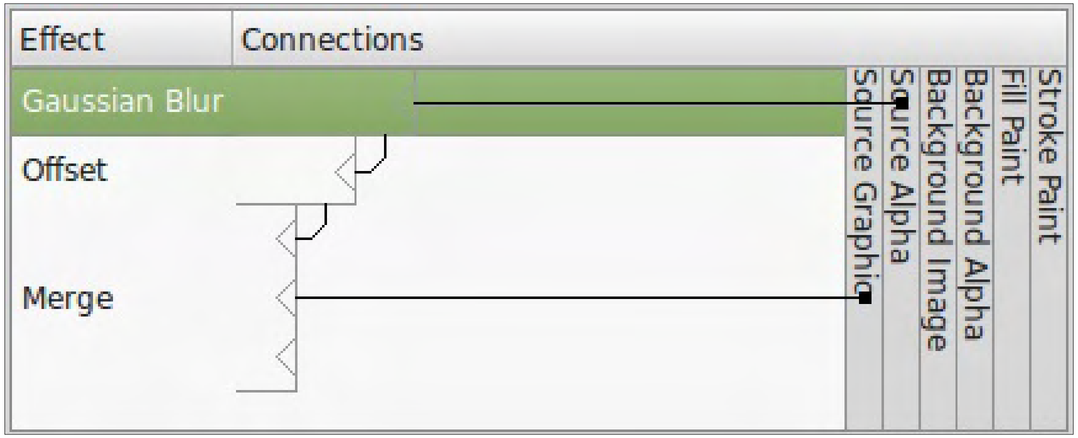
Make sure you understand what we’ve covered in this instalment, because, next time, we’ll build on this simple drop shadow to introduce some more filter primitives that will expand your repertoire further, giving you the capability to achieve effects that just aren’t possible without a little smattering of bitmap magic on your vector objects.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/

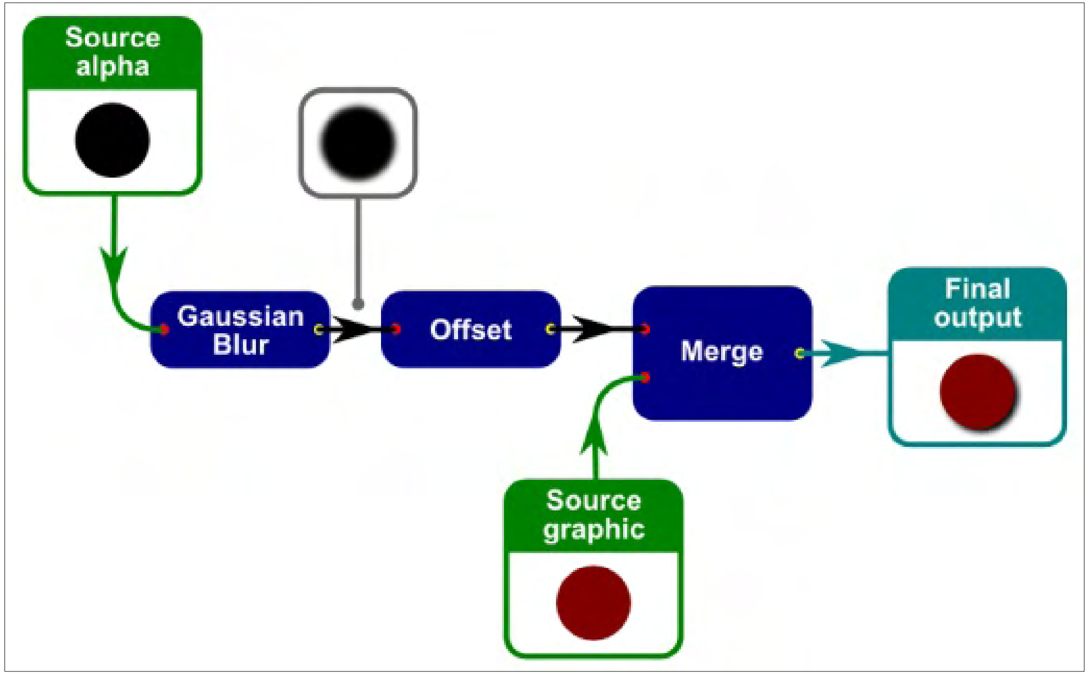
Inkscape - Part 49

Written by Mark Crutch

Last time, I introduced the Filter Editor dialog and demonstrated how to create a simple filter chain, resulting in a drop shadow effect. The chain consisted of three filter primitives: Gaussian Blur, Offset and Merge. I also used the Source Alpha and Source Graphic inputs. Recall that each primitive in the chain has one or more inputs, denoted by triangles, and a single output represented by the bottom of the primitive. The output from the chain as a whole is always the output from the last primitive. Therefore, in the Inkscape UI, our drop shadow chain looks like that shown below left



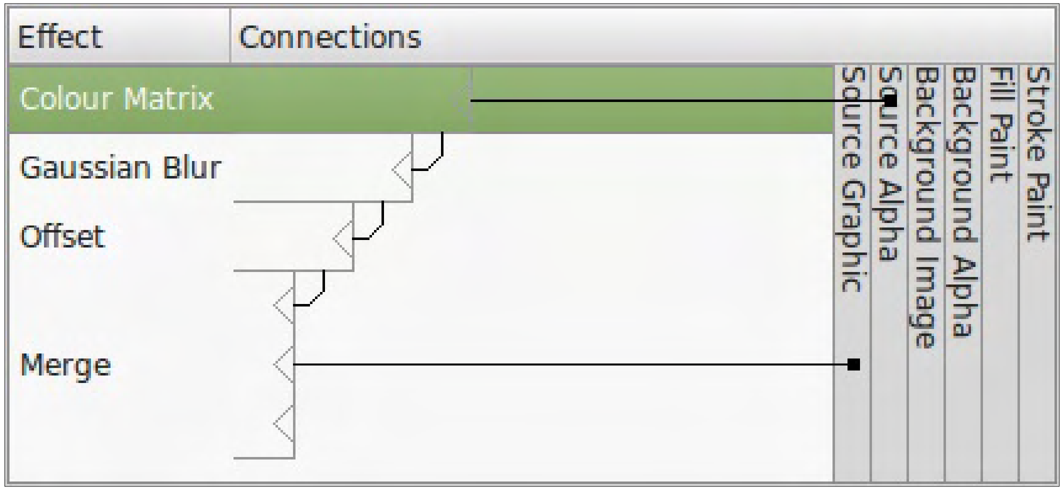
With a simple chain this is fairly understandable but, as the complexity of your filters grows, a simple one-dimensional list becomes an unwieldy tool for looking at the complex arrangement of primitives that evolves. Mathematically speaking, filters are a “directed graph”, consisting of a series of nodes (the primitives) and uni-directional lines connecting them. Such graphs are usually drawn in two dimensions, and you may find it easier to try to imagine your chains in that form. For example, our simple drop shadow could be represented like that shown below right.



Here I've used blue boxes for the primitives, green for the image sources, and teal for the final output. The gray box just shows the intermediate result that you would see if you could peek into the filter chain at that point. Hopefully you can see how this layout relates to the Inkscape UI, and I'll use this approach again to describe more complex filters as the series goes on.

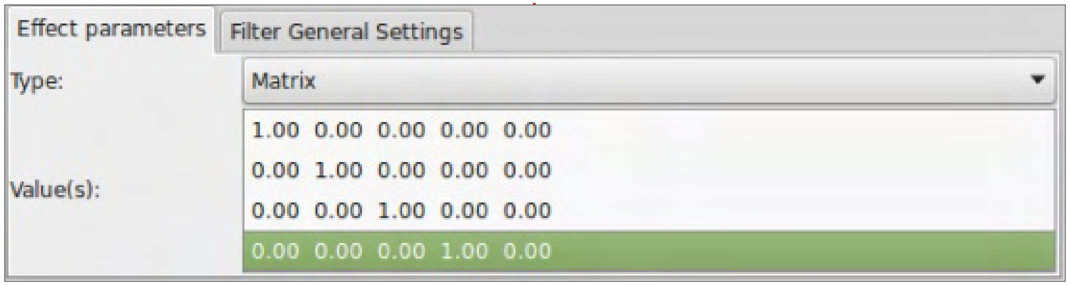
One problem with our drop shadow is that it's based on the Source Alpha of the original object, which is essentially a black silhouette. But what if you want your shadow to be more translucent – gray rather than black – or you want it to have a different color entirely? There is a filter primitive that lets us manipulate the color of the image in the chain, but unfortunately it's another case of a confusing UI that could have been made a lot more obvious.

Start by adding the Color Matrix primitive to your filter. It will appear at the bottom of the chain, but you can drag it to another



location. We'll start by changing the opacity of the shadow, so it either needs to go after the Gaussian Blur step (to change the opacity of the already blurred image), or right at the top of the chain (to change the opacity before the blur is applied). Either approach will give roughly the same result, so I've chosen to put it at the top of the list. You then need to modify the connections so that the Color Matrix gets its input from the Source Alpha column, and the Gaussian Blur gets its input from the Color Matrix primitive.

With the Color Matrix primitive selected, take a look at the parameters at the bottom of the dialog. First there is a Type pop-up which lets you select between four different varieties of color manipulation. Three of them have simple, easy-to-use interfaces… so of course we need the other one! Select the Matrix option (this is also the default when you first add the primitive), and you'll be presented with a grid of numbers with little extra explanation (there is a lengthy tooltip, but I'm not sure it helps very much).



Type: Matrix

Value(s):

```
1.00  0.00  0.00  0.00  0.00
0.00  1.00  0.00  0.00  0.00
0.00  0.00  1.00  0.00  0.00
0.00  0.00  0.00  1.00  0.00
```

Input				Fixed Offset		
R_{IN}	G_{IN}	B_{IN}	A_{IN}			
1.00	0.00	0.00	0.00	0.00	R_{OUT}	Output
0.00	1.00	0.00	0.00	0.00	G_{OUT}	
0.00	0.00	1.00	0.00	0.00	B_{OUT}	
0.00	0.00	0.00	1.00	0.00	A_{OUT}	

Above is that same matrix, presented with some headings to help clarify things:

Remember that filters are a way to manipulate the bitmap version of your vector image, just at the point it's converted to pixels. This matrix essentially holds some rules about how each individual pixel in your input image should be modified in order to produce the corresponding pixel in your output image.

Let's take the top row as an example. Suppose the first pixel in our image has an RGB value of (150, 128, 255) and it's completely opaque (an Alpha value of 255). To calculate the color of the output pixel we have to calculate its R, G, B and A values separately – the top row, therefore, is only concerned with the Red component of the pixel. The formula for calculating the Red output pixel value is:

ROUT = (RIN × **1.00**) + (GIN × **0.00**) + (BIN × **0.00**) + (AIN × **0.00**) + (255 × **0.00**)

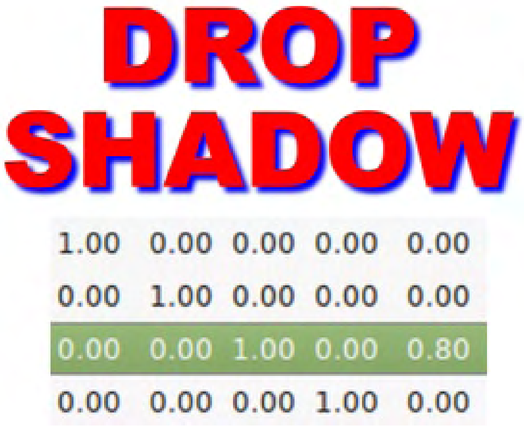
The bold numbers in the formula are taken from the first row of figures in the matrix. Clearly only the first value has an effect in this case, as all the others are zero, so ROUT is simply the same as RIN × 1.00. In other words, with these figures the red component is passed through untouched, with a value of 150. If you repeat the process for each of the remaining three lines, you'll see that the default color matrix simply passes the input color through to the output without modifying it. It's an "identity" matrix, in mathematical terms. Because the same matrix is used for every pixel in the input image, the result is that this filter primitive will just copy the input image directly to the output without changing it at all.

To make the drop-shadow more translucent, we need to modify the output Alpha value. On the bottom row of the matrix, click on the 1.00 field and change it to 0.50 then press the Return or Enter key. Immediately you'll see the drop shadow change. You can choose any value you wish (between 0.00 and 1.00) in order to create a lighter or darker shadow.

What about changing the color of the shadow? There are a few ways to go about this, but we'll start by using the fourth column in the Color Matrix – the one labelled as “Fixed Offset” in my diagram. Consider that the black pixels in the Source Alpha image have an RGBA value of (0, 0, 0, 1) – with all those zeros it's clear that no amount of multiplication will change the output of the red, green and blue components. But the fourth column lets us add (or subtract) a fixed value. If you change the fourth column on the third row to 0.80, the formula for the blue component of the output pixels becomes:

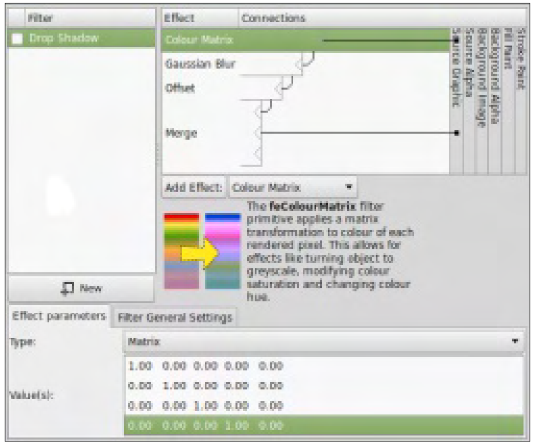
BOUT = (RIN × 0.00) + (GIN × 0.00) + (BIN × 1.00) + (AIN × 0.00) + (255 × 0.80)

The multiplied R, G and B values all come to 0, but then we add 255 × 0.8 onto the result, giving us a final blue component of 204. Our RGBA output value therefore becomes (0, 0, 204, 1), giving us a blue drop shadow. Try changing the values of the fixed offset for R and G as well and you'll quickly see that we can use this technique to produce any color of shadow we want, all from our black silhouette.



1.00	0.00	0.00	0.00	0.00
0.00	1.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.80
0.00	0.00	0.00	1.00	0.00

How about using the Color Matrix filter to go in the opposite direction – to generate a black shadow from a colored one? No problem, but first you'll need a colored image to work with. The easiest option is to move the input connection from Source Alpha to Source Graphic. You should also change your Color Matrix values back to the identity matrix to give you a known state to start from. With that done, your drop shadow should now be the same color as your original object (red, in my case).



The **feColourMatrix** filter primitive applies a matrix transformation to colour of each rendered pixel. This allows for effects like turning object to greyscale, modifying colour saturation and changing colour hue.

Type: Matrix

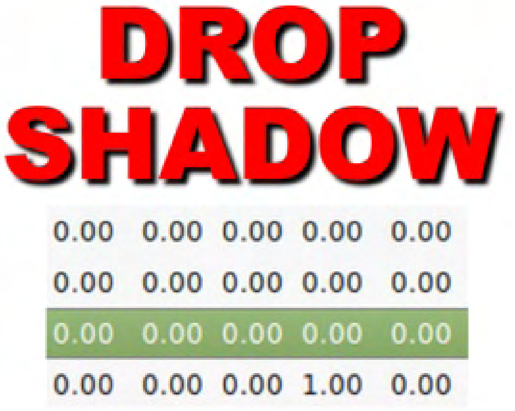
Value(s):

1.00	0.00	0.00	0.00	0.00
0.00	1.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.00
0.00	0.00	0.00	1.00	0.00

To convert our color to black, we have to set each component to zero. There are a couple of ways to do this:

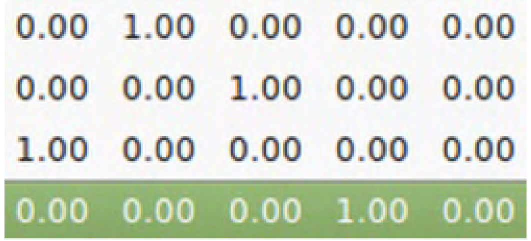
- Put -1.00 into the Fixed Offset field for the R, G and B output values. No matter what the input values are, this has the effect of subtracting 255 from the output. This has the effect of setting each output to zero, because it's not possible for a color component to go any lower than that.
- Change the 1.00 values in the first three rows to 0.00 instead. Regardless of the input value, multiplying it by zero will give a zero output.

I took the second approach, to give me a black drop shadow once more:



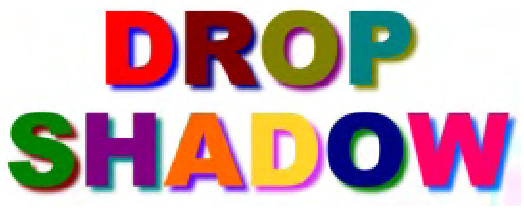
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00	0.00

Of course this is a terribly inefficient way to create a silhouette compared with just linking to the Background Alpha source, but it helps to demonstrate how output values are calculated from input values. So far, however, we've just looked at simple mappings, where red remains red and blue remains blue, but this filter primitive also allows you to map one input component to a completely different output. Consider a matrix like this:

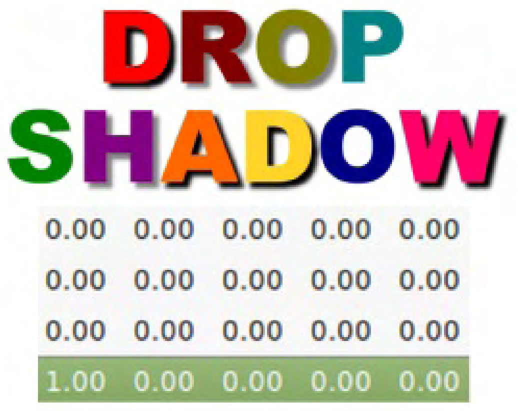


0.00	1.00	0.00	0.00	0.00
0.00	0.00	1.00	0.00	0.00
1.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	1.00	0.00

It's similar to the identity matrix, except that the R, G and B columns are shifted by one place. The result is that the red component of the output pixel is taken from the value of the green component of the input, whilst the green output comes from the blue input and the blue output is taken from the red input. Let's see the result on a multi-colored source image:



Notice that this moves away from having a simple fixed color for the shadow, and instead produces different colors based on the corresponding pixel in the input image. If you include the output Alpha channel, things can get even weirder. Here's our multi-colored text with the RGB values zeroed to give us a black drop shadow again, except that the fourth row has been changed so that the Alpha channel is taken from the Red input component. Notice that colors with a high red value have dense, solid shadows, whilst those with no red in them (such as the green S and blue O in “Shadow”) have no shadows at all.



0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00
1.00	0.00	0.00	0.00	0.00

Although I've shown only relatively simple examples here, it's possible to create complex mappings between color channels. If you really want your red output to consist of 90% of the red input, less 10% of the green, less 35% of the blue, plus a fixed offset of 64, you just have to put values of 0.9, -0.1, -0.35 and 0.25 into the top row. Of course, predicting the output from complex combinations like this becomes rather difficult, so for normal use I recommend sticking with simpler, easy to understand mappings.

This ability to flexibly map color components to each other, or to and from the Alpha value, can be useful on some complex and esoteric filters. Most of the time, however, you don't need that degree of flexibility, so the Type pop-up provides three other options to avoid you having to wrestle with the full matrix:

- **Saturate**: Provides a slider to let you change the saturation of your image. In other words, remove color from it, ultimately producing a grayscale result at the most extreme.
- **Hue Rotate**: Shift the color of your object by a fixed amount.
- **Luminance to Alpha**: Set the output alpha based on the RGB input values. In theory this makes dark areas more transparent and light areas more opaque, but RGB doesn't map neatly to the human perception of brightness, so this really works effectively only on grayscale input images. This can be used to punch holes in your filter output, based on the images produced in other parts of the chain.

Notable by its absence is a shorthand option for adjusting the opacity – where we came in at the start of this article. Unfortunately, if you want to make your drop shadow a little more transparent, you have no choice but to deal with the full matrix approach, even if you are only changing a single value in the bottom row.

Mark uses Inkscape to create three webcomics, 'The Greys', 'Monsters, Inked' and 'Elvie', which can all be found at http://www.peppertop.com/