Projet industriel promotion 2009

PROJET INDUSTRIEL 24

De-Cooman Aurélie
Tebby Hugh
Falzon Noé
Giannini Steren
Bouclet Bastien
Navez Victor

Commanditaire — Inkscape development team
Tuteur industriel — Johan Engelen
Tuteur ECL — René Chalon
Date du rapport — May 19, 2008

Final Report

Development of Live Path Effects for Inkscape, a vector graphics editor.

Industrial Project
Inkscape

FINAL REPORT

Contents

Introduction **4**

1 The project **5**
1.1 Context . . . 5
1.2 Existing solutions . . . 6
1.2.1 A few basic concepts . . . 6
1.2.2 Live Path Effects . . . 7
1.3 Our goals . . . 7

2 Approaches and results **8**
2.1 Live Path Effects for groups . . . 8
2.1.1 New system . . . 8
2.1.2 The Group Bounding Box . . . 12
2.1.3 Tests . . . 13
2.2 Live Effects stacking . . . 15
2.2.1 UI . . . 15
2.2.2 New system . . . 16
2.2.3 Tests . . . 17
2.3 The envelope deformation effect . . . 18
2.3.1 A mock-up . . . 18
2.3.2 Tests . . . 23

Conclusion **28**

List of figures **29**

Appendix **30**
0.1 Internal organisation . . . 30
0.1.1 Separate tasks . . . 30
0.1.2 Planning . . . 31
0.1.3 Sharing source code . . . 32
0.2 Working on an open source project . . . 32
0.2.1 External help . . . 32
0.2.2 Criticism and benefits . . . 32
0.3 Technical appendix . . . 33
0.3.1 The Bend Path Maths . . . 33
0.3.2 GTK+ / gtkmm . . . 33
0.3.3 How to create and display a list? . . . 34
0.4 Personal comments . . . 36

Introduction

As second year students at the École Centrale de Lyon, we had to work on "Industrial Projects", the subject of which were to be either selected in a list, or proposed to the school. Being a regular user of the vector graphics software Inkscape himself, our team leader Steren proposed to work on an improvement of Inkscape: the envelope deformation effect, a feature long requested by users. Since Inkscape is a free and open source software, everyone can freely study its source code and participate in its development. It seemed the perfect occasion to work on a big and very popular piece of software, even if not for an actual company.

Our supervisor in the Inkscape development team was Mr. Johan Engelen, who wrote the initial piece of software necessary to make effects such as the envelope deformation. With his help, we refined our goals and objectives for this project, which finally went a lot further than simply develop a new effect.

In the school, our coordinator was Mr. René Chalon, teacher and researcher in the Computer Engineering and Mathematics department.

This report describes the three objectives of the project, and the work done to fulfill each of them. The appendix gathers complementary information about the project: its organization, technical details that could be used by others, and our general feelings about the project.

Chapter 1

The project

1.1 Context

Inkscape is a vector graphics editor application. Its stated goal is to become a powerful graphic tool while being fully compliant with the XML, SVG and CSS standards. Released under the terms of the GNU General Public License, Inkscape is a free open source software which means that its source code is open to everyone and can be shared freely.

Inkscape deals with vector graphics. Contrary to traditional images (or raster graphics), a vector graphic is not defined by a matrix of pixels. All the shapes are defined by points and curves or any other mathematical primitives. This means for instance that vector graphics usually make smaller files and when you zoom in, the shapes will always remain crisp with no aliasing.

The difference can easily be understood with the figure 1.1.

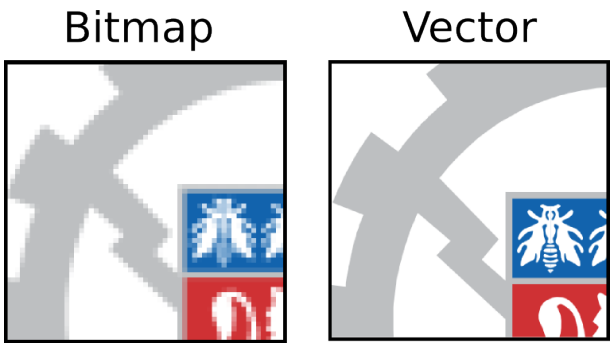


Figure 1.1: The difference between a traditional image and a vector graphic

Inkscape being open source means that anyone can obtain the sources, modify them, and then have these modifications applied to the main branch of the software. There is a list of many new features and enhancements that is available, and our project is to complete some of those tasks.

Johan Engelen is a Dutch student and is responsible for creating the very recent Live Path Effects in Inkscape. The system is fully fonctional, but some enhancements could be made in order to make it even more powerful. We contacted him and he accepted to help us developping these enhacements.

1.2 Existing solutions

1.2.1 A few basic concepts

A path is a set of curves. Each curve is defined by at least two control points. Inkscape uses Nonuniform rational B-splines to define curves but most of the Inkscape curves are included in the subset of cubic Bezier paths. A closed curve defines an area.

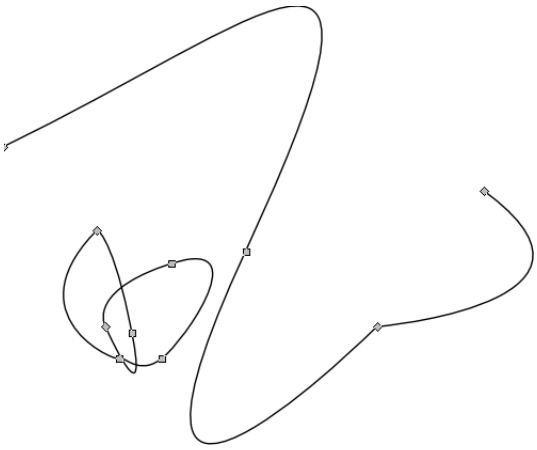


Figure 1.2: A path made of two cubic bezier curves.

A group is a set of objects. Those objects can have different natures (paths, shapes...) and keep their nature and all their properties when grouped with others. Transformations can be applied to the whole group but each object remains editable separately. Grouping objects is different from combining them. The combination converts a set of objects into a single path. It's a destructive transformation while grouping is not.

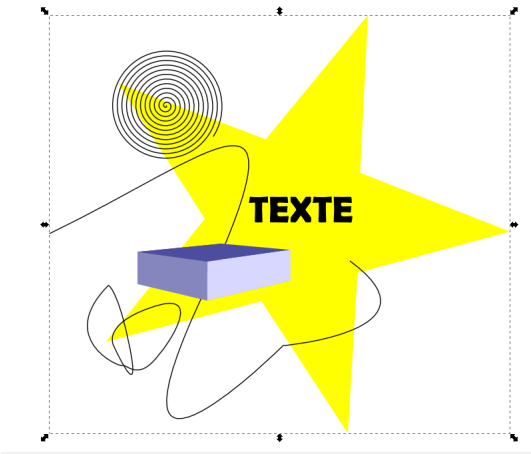


Figure 1.3: A group made of several object

The Bounding Box of an item is a rectangle that surrounds it. It is defined by the two horizontal lines at the minimum and maximum abscissa the item reaches

and two vertical lines at the minimum and maximum ordinate.

1.2.2 Live Path Effects

Traditional effects transform an original object into a new object, but the original object is not memorized so you cannot modify it after the effect has been applied.

On the contrary, Live Path Effects are entirely non-destructive. They can be assigned to a path and the result of the effect is a new path, but you can still edit the original path even after the effect is applied and the new path is immediately updated in real time. Many different effects can be thought of, and adding them is made as easy as possible.

It is worth noticing the existing Bend Path Effect for instance. This Live Path Effect can easily deform a shape along a defined path as you can see on figure 1.4.

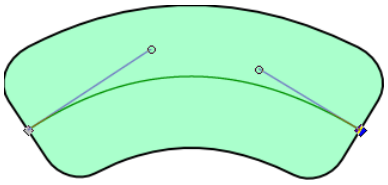


Figure 1.4: The Bend Path Live Path Effect applied to a single shape

Stacking LPE (i.e. applying several effects to a single path) can only be performed by applying definitly the LPE and then assigning a new one. You then lose the information on the original path.

1.3 Our goals

Our project focuses on the enhancement of Live Path Effects. Our goals are to :

- Enable Live Path Effets assignment to groups.
- Enable Live Path Effets stacking and create the UI (user interface).
- Create a new effect : Envelope deformation.

Of course the file generated when saving must keep his total compatibility with the SVG standard.

Chapter 2

Approaches and results

2.1 Live Path Effects for groups

A drawing very often consists of multiple shapes of various colors. It seems natural to be able to apply an effect to this drawing, as you would to a raster graphics picture. It should thus be the case for Live Path Effects.

The aim is to allow the user to assign a LPE to a group of items that can receive LPEs. The effect is supposed to be applied to each item of the group, may it be a path, a sub-group or anything else, and it should behave as one would expect intuitively.

2.1.1 New system

Existing architecture

Our first task was figuring out how the path effects were applied to objects. It involved understanding the global architecture of Inkscape. For each kind of SVG elements, Inkscape implements a class. For each object in an SVG document, Inkscape instantiates the class of the appropriate type and stores it in memory. The class hierarchy follows the SVG specifications (see figure 2.9), but not exclusively. There are some classes, used to implement common features, which don't correspond to any actual type of SVG elements and thus aren't instantiated.

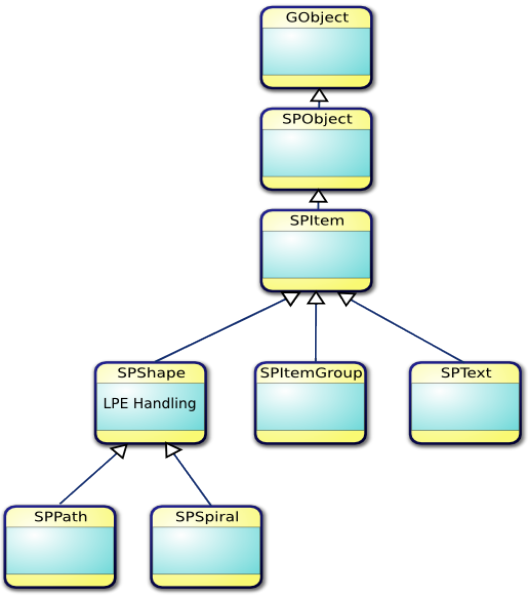


Figure 2.1: Partial class diagram of the existing architecture

Currently, Inkscape doesn't use plain C++ classes. It uses a library named GObject that allows to do object oriented programming in C. This is why all the objects subclass the GObject class. The SPObject class implements common features such as retrieving the XML node of the instance, and handling the updates made to the object. The SPItem class represents an object that can be drawn on the canvas. SPItemGroup allows handling several SPItem as if there were only one. It is Inkscape's implementation of the SVG "g" element. SPShape is the parent class of all the items that are drawn using a single path. This is where the live path effect code used to be, since LPEs could only be applied to shapes.

Updated architecture

Basically, we had to copy all the LPE related code from SPShape into SPItemGroup and adapt it so that it is possible to apply effects to groups of shapes. This is what we first did as a quick test. The issue was that it introduced a lot of code duplication, and we had to separate the case of shapes and groups everywhere.

To solve these issues, the code used by the LPEs had to be the same for both shapes and groups. To achieve a full unification of LPE handling, the LPE related code had to be located in a common parent of SPShape and SPItemGroup. We first thought that SPItem was the ideal candidate, but we soon realized that the LPE handling code couldn't be moved in that class because the SPText class subclasses it. Live path effects cannot be applied to text items because those are not stored as paths in the SVG document.

The only remaining solution was to create a new class that would allow an object to have a path effect. This solution solves all the problems stated above. We named the new class SPLPEItem and integrated it in the existing architecture as seen on figure 2.2.

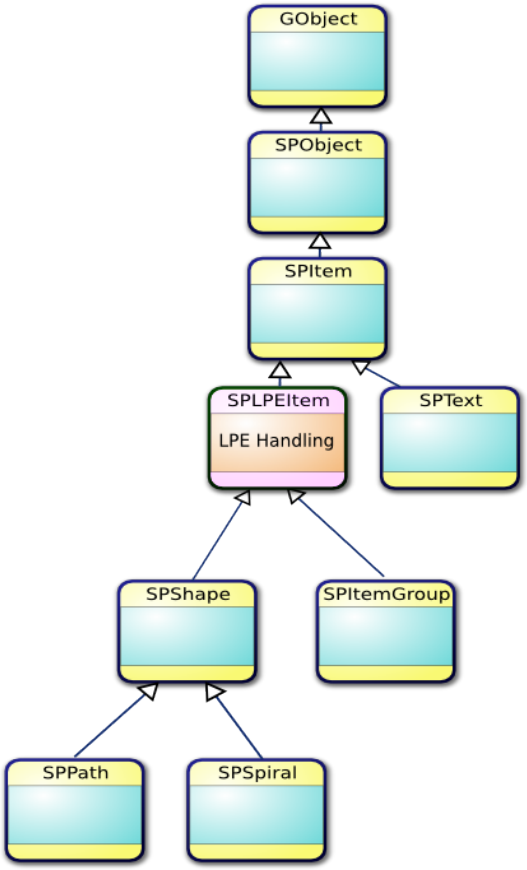


Figure 2.2: Partial class diagram of the new architecture

First approach

Live Path Effects transform an original path into a transformed path. Before we started our work, Inkscape could only apply one LPE for each path. In order to add the ability to apply a path effect to a group, we need to be able to apply several path effects to paths. That means that groups have to store a path effect reference, like paths did. This can easily be done in an unified way with paths thanks to the new class SPLPEItem.

Our first attempt to implement path effects for groups is described in figure 2.3. When a path is modified, it first applies its path effect on its own path, and then asks its parent to apply its path effect on the same path. The result of this recursive algorithm was the original path, with the LPEs of all of its parent groups applied.

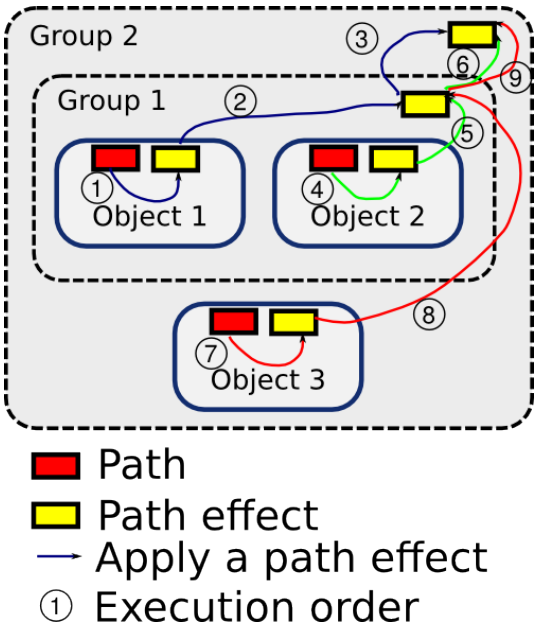


Figure 2.3: Initial LPE for groups algorithm

This algorithm had the advantage of being extremely simple. At first, it seemed to work fine, but as we tried more complex examples we realized it was wrong. Some LPEs need to know the bounding box of the item it is applied on to compute the transformation they make to paths. However, in the case of a group with several items that already have a LPE, it is impossible to compute the right bounding box for the group path effect using this algorithm. Indeed the paths used to compute the bounding box are the original versions of the subitems' paths, except for the subitem that caused the path effect update. We had to find an other algorithm to solve this issue.

Final algorithm

To fix the problem described above, all of the subitems of a group need to have their path effect applied when a group's LPE is computed. This way, the right paths are used when calculating the bounding box of the group, and the path effect applies as expected.

This algorithm can be implemented by considering the item hierarchy as a tree, and then using tree traversal techniques to apply the path effects. We chose to use a top to bottom approach instead of the previous bottom to top strategy. This way we know which LPEs have to be applied starting from the beginning of the process. The steps taken by the algorithm we designed when a path is modified are (see figure 2.4):

1. Find the topmost item in the hierarchy that has a path effect.
2. For each child re-apply this point until the bottom of the tree is reached, and then, if the child is a simple path, apply its path effect. Else, if it is a group, apply the path effect to the group (see below).

This post-order tree traversal ensures that the group path effects are applied only when all of the subitems have their LPEs applied.

To apply a path effect to a group: For each child of the group, if the child is a group apply the path effect to the group. If it is a path apply the path effect of the group the algorithm was called from to the path.

This pre-order tree traversal applies the path effect of a group to all of its sub-paths (even if they are inside another intermediate group).

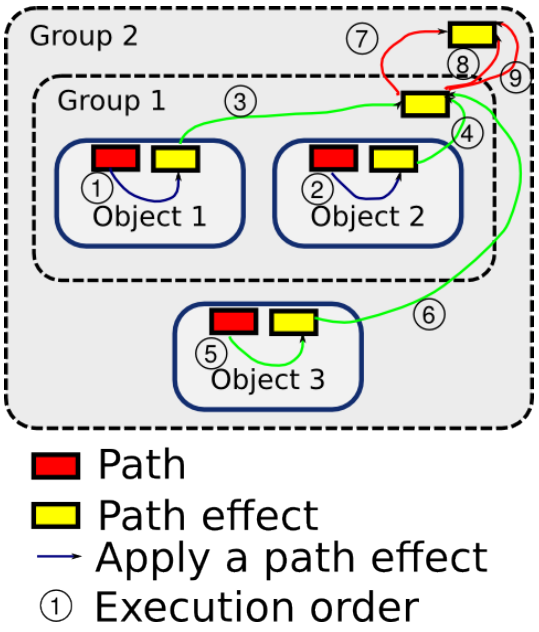


Figure 2.4: Final LPE for groups algorithm

The global algorithm is of course more complex than the previous one, since it is a double tree based recursive algorithm as opposed to a single linear recursive algorithm. However it fixes all of the issues we noticed. The current implementation could be optimized as it triggers unneeded redraws for LPEs that don't use bounding boxes. Bounding boxes could also be cached to speed up the process.

2.1.2 The Group Bounding Box

For many LPEs assigned to a group of paths, the effect is done recursively without any special behaviour. When we consider Bend Path (see picture 1.4 page 7), it seems obvious that the whole group must be deformed by one single path. Thus, the behavior of this effect depends on the size of the group, and not only the size of each sub-shape.

Effects using the Group Bounding box

Many effects will need the group bounding box in the future. That's why we settled a new system to allow people to easily create new effects implementing this. In the end, an effect that needs it only has to inherit from the new `GroupBBoxEffect` class. The function `original_bbox()` can be called. It stores the bounding box dimensions in `Geom::Interval boundingbox_X` and `Geom::Interval boundingbox_Y`.

We also created a function that takes as a parameter the item on which the effect is applied. This function is automaticaly called before the calculation of the effect. We named it `doBeforeEffect (SPLPEItem *lpeitem)`. Therefore, it is interesting to call the function `original_bbox()` in `doBeforeEffect()`.

Modifications of the Bend Path effect

The existing algorithm of Bend Path was rather simple to understand due to a reliable documentation. See appendix 0.3.1 page 33 for detailled maths. We only had

to do the modifications described earlier. Then we replaced the built-in calculation of the shape bounding box by `boundingbox_X` and `boundingbox_Y`.

2.1.3 Tests

We test an effect that doesn't need the group bounding box calculation. We create a group of shapes on which we apply a *Sketch* effect. We can see on figure 2.5 that the LPE applies to each sub-shape. Each original shape can still be modified inside the group.

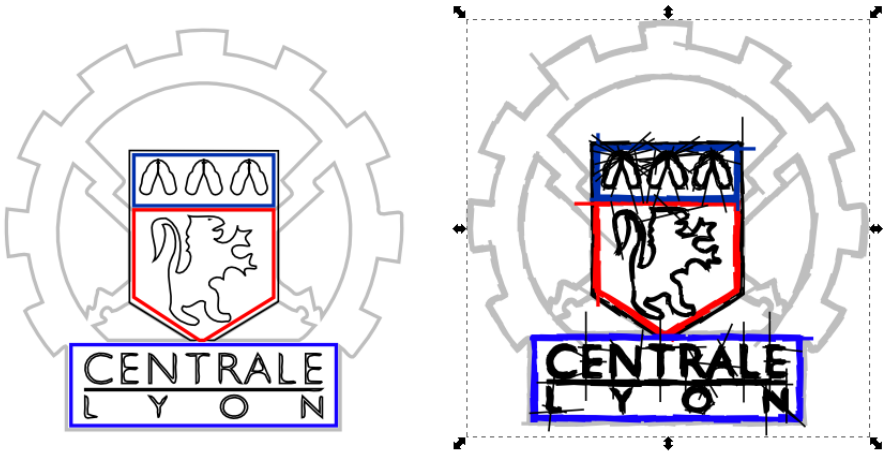


Figure 2.5: Effects on Group : Test for Sketch

The next test is done with an effect that needs the group bounding box dimensions. We apply a *Bend Path* on a group of shapes. As we would expect (see figure 2.6), a single bend path is created automatically over the whole group. When we modify it, the whole group is deformed.

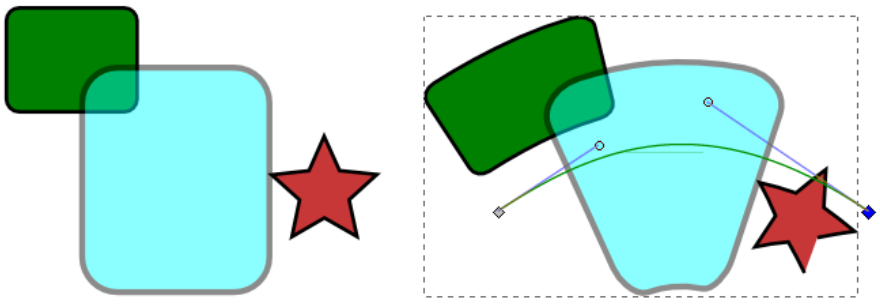


Figure 2.6: Effects on Group : Test for Bend Path

As a conclusion, we can say that LPE on groups are very usefull because most of graphics are made of multiple sub-shapes. We can now play with the entire logo of the school, as you can see on figure 2.7.

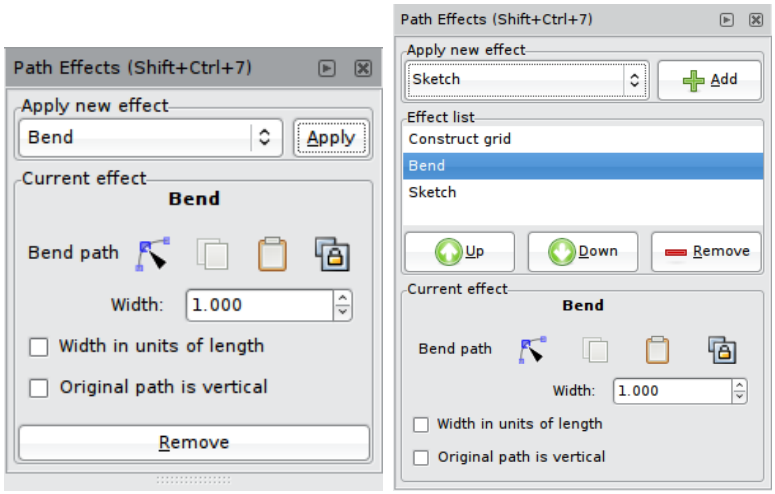
2.2 Live Effects stacking

In the original implementation of Live Path Effects, only one effect could be applied to a path. If the user needed to apply a second effect, the first one had to be permanently applied, therefore losing the benefits of the LPE system.

Our task was to make it possible to apply – or "stack" – several effects, and still have access to all the effect properties.

2.2.1 UI

To make LPE stacking user friendly and harmoniously integrated with Inkscape, the user interface (UI) was very important. On figure 2.8 you can see what the UI used to be, and what modifications we made to it.



(a) Original LPE UI (b) New LPE UI

Figure 2.8: LPE UI comparison

We separated the implementation of stacking into two different tasks. The first one was to imagine and create a user-friendly interface. The second one was to actually code the stacking functions and link them to the interface.

To create the interface, we based ourselves on applications which also used stacking, like the 3D graphics programs Blender and 3DS Max. We decided the effects should be added to a list, and when an effect would be selected, it's parameters would be accessible. We also used the Inkscape layer editor as a model.

Inkscape's UI is coded using the standard GTK+ libraries. Therefore we first had to learn how to code an UI using GTK+. The C++ interface for GTK+ is called gtkmm. But older parts of the Inkscape sources, coded in C, don't use gtkmm. The basic functions are identical – they are simply interfaces for GTK+ – but the syntax is different. Our first idea for the interface was copying the layer editor code and removing unnecessary elements. We found out that the layer editor UI was very complex, and didn't use gtkmm. We then decided to code the stacking UI from scratch.

The original UI was separated into two "frames", one for the effect selection and application, the other for the effect parameters. We added a third one named "Effect List". In the "Effect List" frame, we list the applied effects. The list is empty to start off. Elements are added each time an effect is applied. By default, the frame

size allows the viewing of 4 effects, but it is possible to add more: a scroll bar then appears. When an effect is selected, its properties are displayed in the "Current Effect" frame. There are also 3 buttons. "Up" and "Down" to move up or down the selected effect in the list and a button "Remove" to remove it.

We were easily able to add frames, buttons, and also link the buttons to the right functions. All of this was written in a single file : livepatheffect-editor.cpp/h.

The main difficulty was to code the actual display of the list. The implementation of lists and trees is very powerful in GTK+, but we only needed something very simple, at least for the time being. We tried to understand how it was done in the layer editor, without much success. We then looked for examples in the official documentation and in various tutorials found on the Internet. Finally we found a good and simple example we were able to modify to suit our needs.

You will find the detail of the implementation in the technical appendix.

There are still several enhancements we can think of for the UI. For instance it could be useful to enable or disable any effect in the stack without actually removing it and thus losing all the parameters. Being able to edit the name of the effect would also be useful when several effects of the same type are added. We may implement these features later on, but for the time being we will keep it as simple as possible and hopefully make it bug-free.

2.2.2 New system

Enabling the actual stacking of effects was in fact part of the refactoring we had to carry out in order to apply LPEs to groups, as explained previously.

In the SPLPEItem class we stored a reference to a LPE. But obviously it started to feel lonely and depressed, so we allowed it to have some new friends by replacing it by a list of references. These references are actually stored as URLs separated by semicolons in the SVG file. Here is an example:

```
<path
                //Defines the style of the path
      style="fill:none;stroke:#000000;stroke-width:1px;
         stroke-linecap:butt;stroke-linejoin:miter;stroke-
         opacity:1"
                //Defines the path that is displayed
      d="M 274.286,363.791 537.143,672.362"
      id="path2461"
                //List of LPEs
      inkscape:path-effect="#path-effect2463;#path-
         effect2465;#path-effect2467;"
                //Original path, before LPE
      inkscape:original-d="M 274.28571,363.79075
         537.14286,672.36218"
/>
```

A list of LPE references is created directly from the SVG file, and any modification to the LPE list, such as changing the order, must be directly written to the SVG file. The list of references must then be updated.

The functions assigned to the “Remove”, “Up” and “Down” buttons work that way, writting directly to the SVG file and then updating.

The actual stacking is performed by applying each path effect of the list on the resulting path of the previous path effect.

2.2.3 Tests

To check the good functioning of LPE stacking, we applied several effects to one object. We started with two lines which allowed us to build a grid. Then we made a Bend Path effect, a Knot effect and finally a Sketch effect (see the figure 2.9).

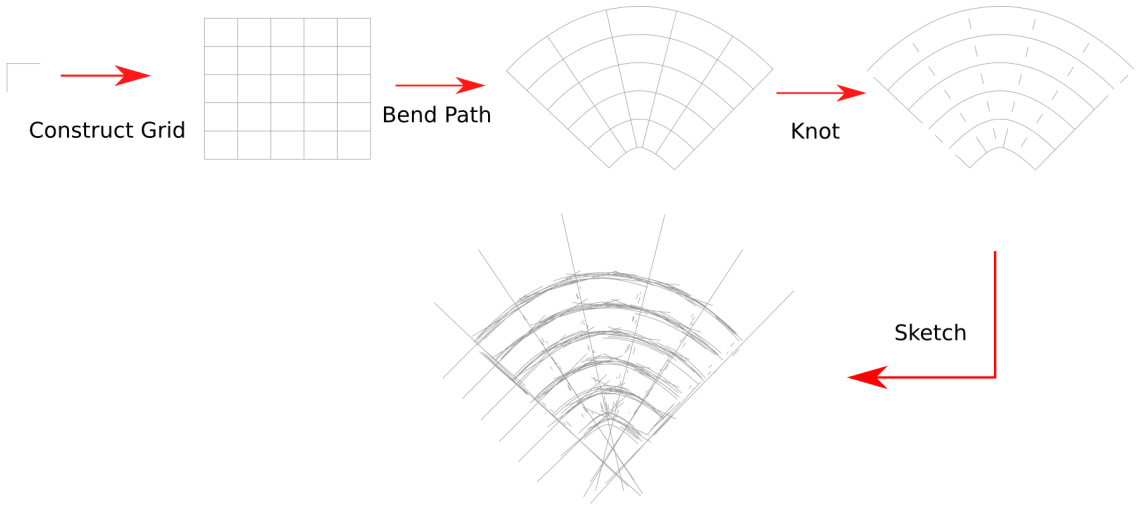


Figure 2.9: Example of stacking

Besides, we can see on figure 2.10 that the interface works too. The effects are added in the right order to the effect list. When we select an effect in the list, the attributes of the effect are displayed in the frame “Current Effect”.

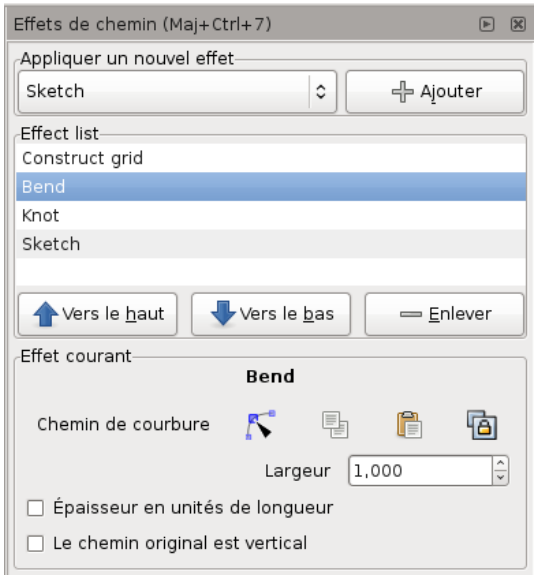


Figure 2.10: Interface for the stacking example

The buttons “Remove”, “Up” and “Down” also work exactly as they are supposed to.

2.3 The envelope deformation effect

Before our work, paths could only be deformed by simple transformations (translation, rotation or scale) or by the Bend Path LPE. Tools more powerful can be needed when manipulating paths, for instance modifying the global shape of the path, which is very often needed by designers. Drawing a flag is a good example of that need. With this effect, the flag is drawn as a basic rectangle, then the envelope deforms it.

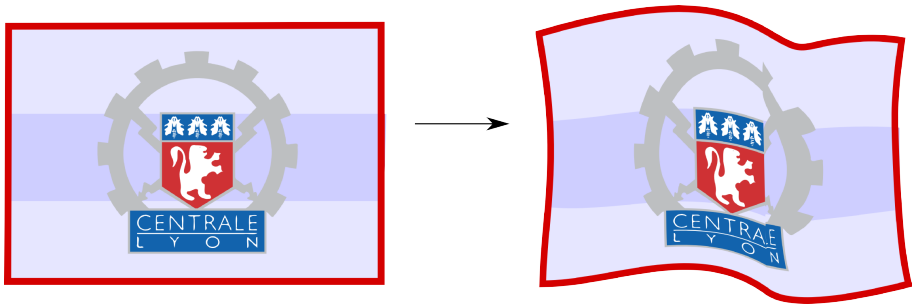


Figure 2.11: Envelope Deformation user case: Drawing a flag

2.3.1 A mock-up

Very early in the project we drew some examples of how an envelope effect would bend the space. The most important feature is that it must do what the user expects it to do. In the end we made the following mock-up in order to put forward some key examples:

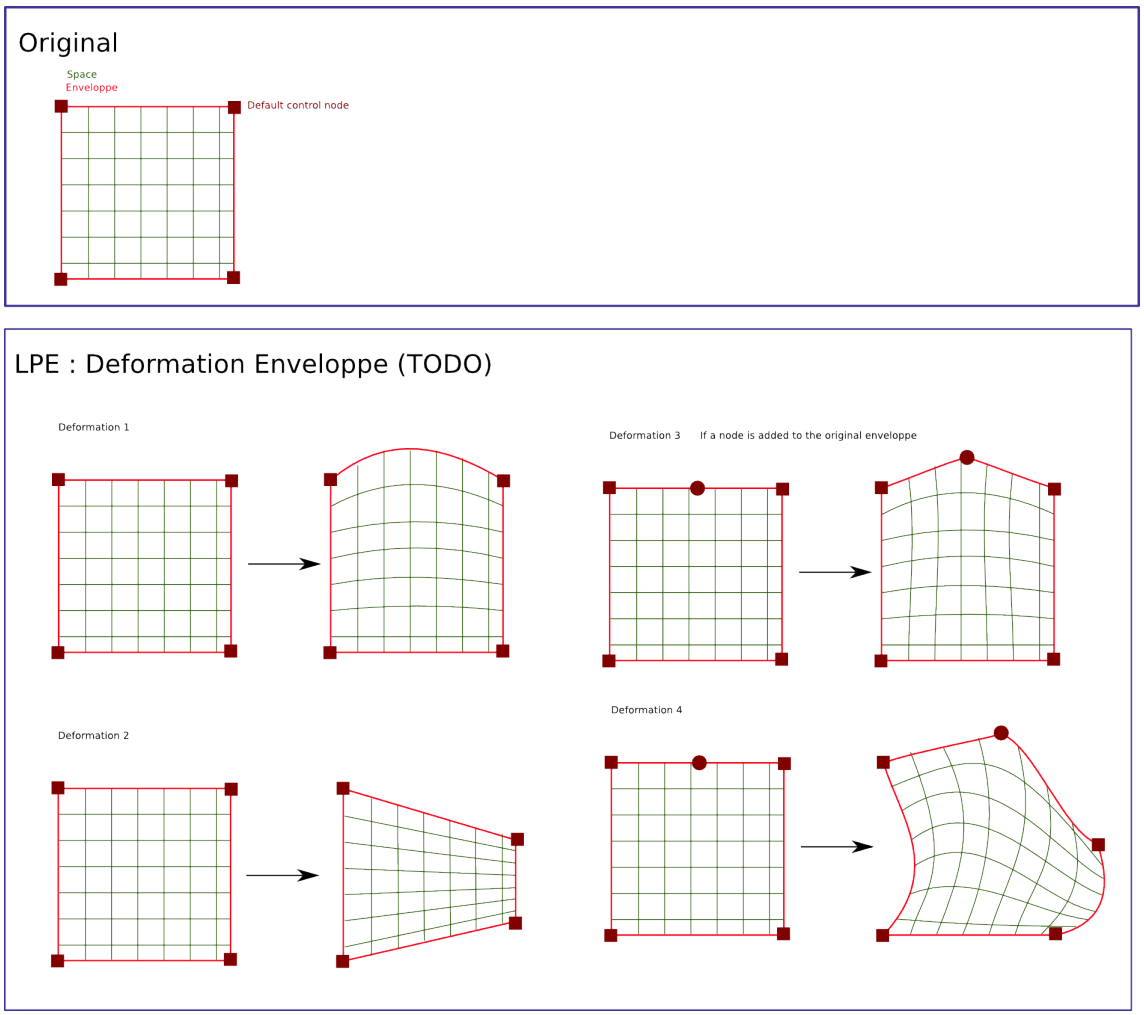


Figure 2.12: Mock-up of the Envelope Deformation effect

We also decided that when the LPE is assigned, the envelope should automatically take the size of the bounding box of the object. subsectionOur solutions

At first, we had had to think about the mathematics behind the effect itself. That is, define exactly what behavior we needed. It allowed us to discover some differences compared to a possible perspective effect.

Once the effect chosen and the mockup made, we were able to start searching for the actual way to program the transformation. Several ideas were proposed, including making use of the already existing "Bend path" effect: one bend path per side of the envelope, plus a few stitches to make things go well together. We thought of using a simple and well known Bézier Patch: the formula is easy to find, and the effect produced would have been very similar to what we were looking for.

Finally, we found an example code in the 2geom library which did partially what we had in mind: d2sb2d.

In the end we actually decided to create two separate effects, "Lattice Deformation", based on the 2geom library, and "Envelope Deformation" based on the Bend Path LPE.

The Lattice Deformation

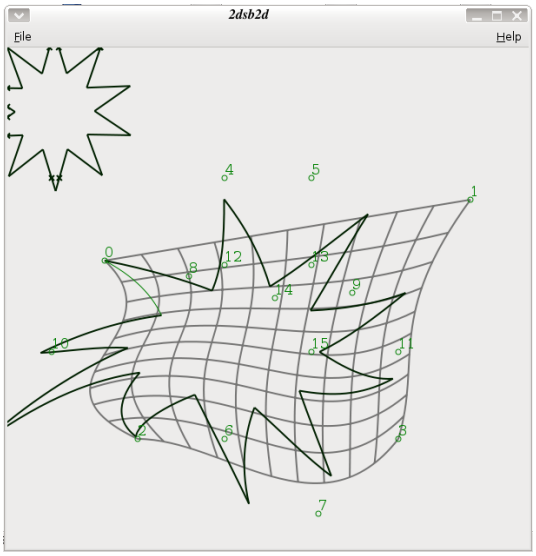


Figure 2.13: The d2sb2d toy as an example of the possibilities of the 2geom library.

All we had to do was to transpose the code from the toy framework (a test framework given with 2geom) to the LPE framework. Unfortunately, several difficulties occurred.

The main difficulty was the *absolute* lack of documentation about the 2geom library. Reading its source code didn't help much, since no comments are to be found (or really cryptic ones), and the data structures are specific enough to be impossible to understand without prior knowledge. The help we got from the creators of the library was very kind but unfortunately pretty much useless, since it required mathematic knowledge we do not have. Therefore, we had to translate the code without really understanding it.

This implies a lot of trials and errors: trying to figure out what the hard coded numbers meant, what they controlled in the toy, changing their values and testing, etc. The same pattern was true about the algorithm itself: since we didn't know what it did, nor how, we tried to establish links between what we saw in the code, and what happened in the toy. We finally figured out enough to be able to write the effect as a LPE for Inkscape.

Internals of the d2sb2d object The effect is based on a mathematical object, called "d2sb2d", in which we store the values necessary for the deformation, and the 2geom function "compose", which computes the mathematical compostion of functions. In this case, we compose the original path with the 2dsb2d (which holds the wanted deformation). Based on what we understood from our testing, here is how the 2dsb2d is used.

The effect needs 16 control points we called "handles" – they are meant to be positioned by hand – placed as a rectangular grid over the shape. The four corners are those of the bounding box of the path, and the others divide the rectangle evenly to form their original position. It the difference between the actual position of the handles (chosen by the user) and their original position that is stored in the d2sb2d structure. This is done with a simple formula, which depends on the numbering of the handles, which is itself quite odd:

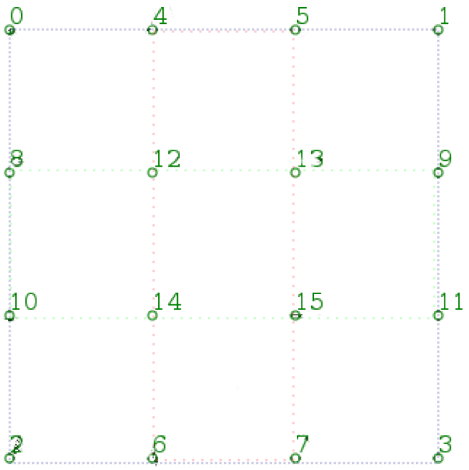


Figure 2.14: The d2sb2d uses a peculiar numbering system for its handles.

Inside the algorithm, two variables, `corner` and `i` are used to select the handle. `i` is the big rectangle ([0 1 2 3], [4 5 6 7], [8 9 10 11], or [12 13 14 15]), and `corner` is its corner, so that any individual handle is computed `corner + 4*i`. But it is even more complicated since the algorithm loops on four other variables, ui, vi, iu, iv, each taking either 0 or 1 as a value, in such a way that `corner = iu + 2*iv` and `i = ui + 2*vi`. Why so many complications ? Because the formula used to store values in the d2sb2d seems to need, for each handle, the associated value for `ui+vi...`

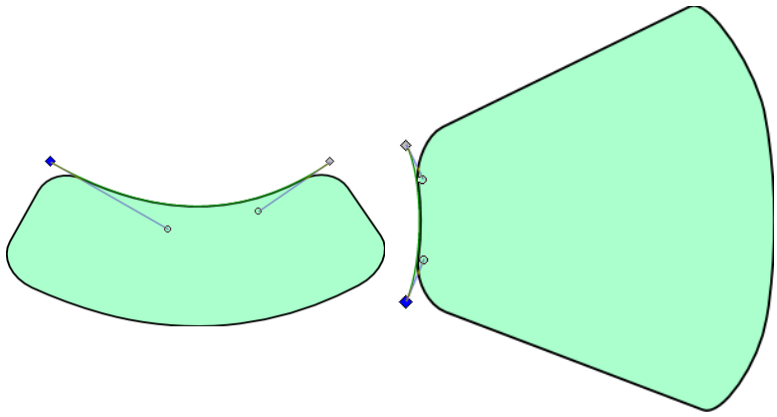
And it is not over yet! Apparently, for the effect to work properly, the path has to be translated to (0,0), and resized to be exactly 1 unit wide. Then, it should be translated back to its original position. The difficulty was to deduce this from the d2sb2d toy, which used a lot of "magic numbers" (hard coded unexplainable values) to perform the effect in a very specific context (and not the general one we needed for Inkscape).

The final difficulty was to deal with 2geom's data type for the paths ("Piecewise D2 sbasis"), again not documented, because the compose function works only with "D2 sbasis" objects. We had to find a way to cut the piecewise version, apply the effect to each piece, and combine them again later.

The Envelope Deformation

The code of Envelope Deformation uses the Bend Path code. Read the technical appendix (section 0.3.1 page 33) to understand the maths behind Bend Path.

We first define four path parameters around the item bounding box: Top and Bottom, Left and Right. Each path represents a side of the deformation envelope and acts as a Bend Path effect. For instance, the result of the deformation of the "Top Path" and the "Left Path" can be seen on figure 2.15



(a) Top bend path (b) Left bend path

Figure 2.15: A rectangle deformed by some particular bend paths

We will follow this example along this section. We consider that the "Bottom" and "Right" paths are not moved.

Then the idea is to blend those four deformed paths is order to generate the final deformed path. To do this, we will use weighting: The closer a point is to a Bend Path, the more it will be affected by this Bend Path. So we had to find simple weight coefficients to use.

The first step is to blend the Top and the Bottom deformation. To do this, the rule is simple: the Top deformed path must be dominant for small y and the Bottom deformed path must be dominant for small $(Bbox_y - y)$. We generate the blended path (named *output-y*) with the following formula:

$$(Bbox_y - y) * \text{Deformation-Up} + y * \text{Deformation-Bottom}$$

The figure 2.16 illustrates the process. It is important to notice that $y+(Bbox_y-y) =$ constant. Indeed, we only have then to divide the result by $Bbox_y$ to scale it to the original size.

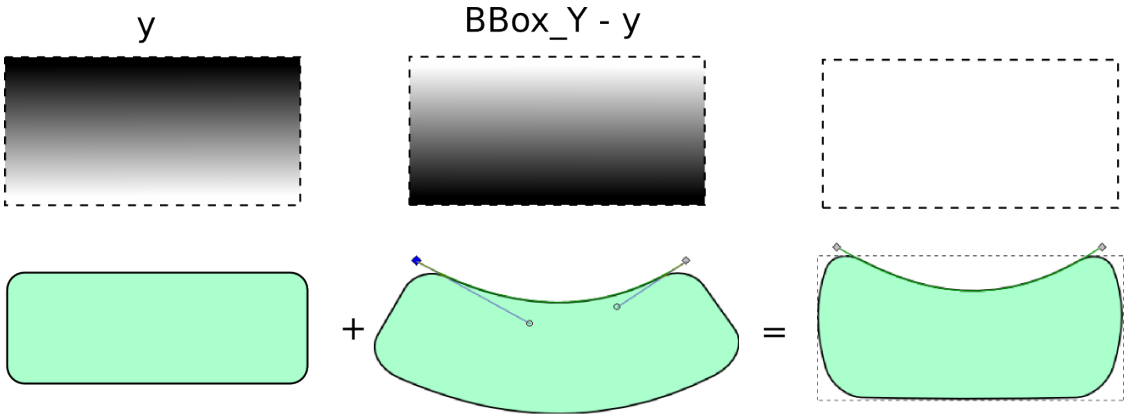


Figure 2.16: Blending the Top and Bottom Deformation

Using the same model, the second step is to blend the Left and Right Deformations (we name it *output-x*):

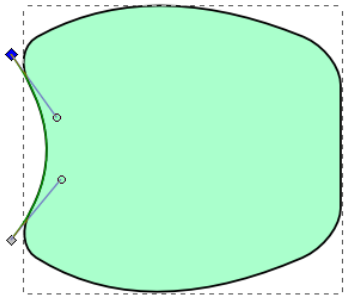


Figure 2.17: The envelope deformation considering Left and Right bend paths

The final step uses the same method but is not as accurate as the previous operation. Indeed, in the end we will average two different results.

Our first result is the calculation of *output-1*: The more a point is close to Top and Down, the less it will be affected by *output-x*. The figure 2.18 illustrates the process.

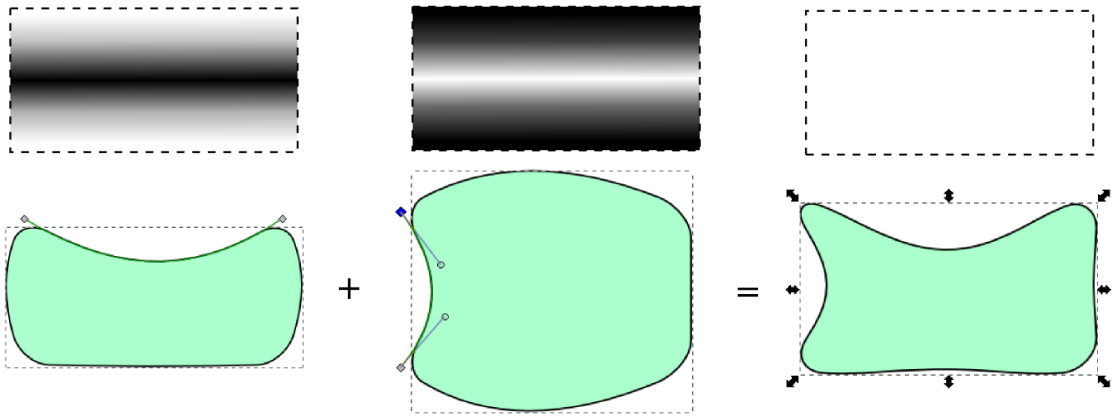


Figure 2.18: The envelope deformation considering all bend paths.

The weight coefficients used were $y(Bbox_y - y)$ and $\frac{Bbox_y}{4} - y(Bbox_y - y)$. We notice again that their sum is constant.

We then calculate *output-2*: The more a point is close to Left and Right, the less it will be affected by *output-y*

In the end, we do the mean between *output-1* and *output-2*. Of course, from a mathematical point of view, the result is not perfect, but on a graphical one, we can consider that this is sufficient.

2.3.2 Tests

Two tools

It is important to underline that the Envelope and Lattice tools were not built for the same use. Indeed, while the lattice offers an accurate deformation, this deformation is only determined by control points. It also allows the user to move inner control points, which can't be done with the Envelope deformation. The power

of the Envelope deformation is the absolute control of the shape of the envelope, but as we will see, the result may show some artefacts.

Lattice deformation

The Lattice effect we made accomplishes its primary goal. It is quite intuitive to use and the visual effect is what we wanted. We renamed it to Lattice deformation because it provides 4 inner handles in addition to the envelope. Here you can see a screenshot of the effect on a grid with the original path in red and the new path in grey.

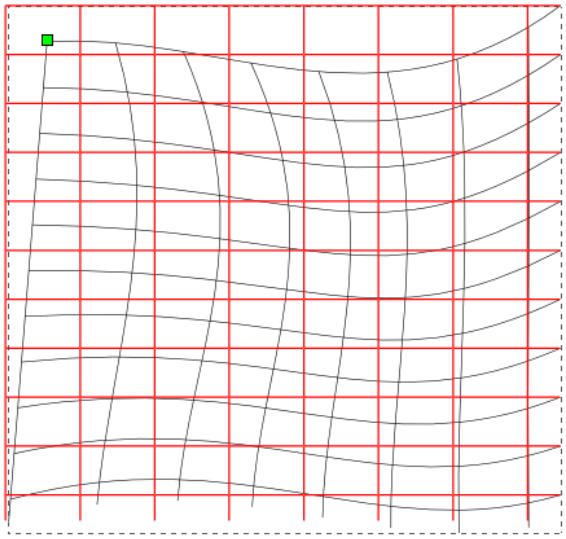


Figure 2.19: A screenshot of the Lattice effect

Known Bugs Some of them come from the PointParam class we used which is not completely finished:

- The PointParams are written in the svg only when moved by hand whereas they should also be written when created.
- Exceptions occur when they are loaded.
- They cannot be all shown on screen at the same time and that's not very ergonomic.

An other problem is that when you modify the original path and if this modification changes the bounding box the distortion of the final image is modified (see figure 2.20). This issue comes from the fact that the handles' coordinates are absolute – therefore they don't move when the original path is modified – and their original position is calculated from the item bounding box which varies when the path is modified. The deformation being based on the distance between the handles' positions and their original position, it thus changes with the bounding box.

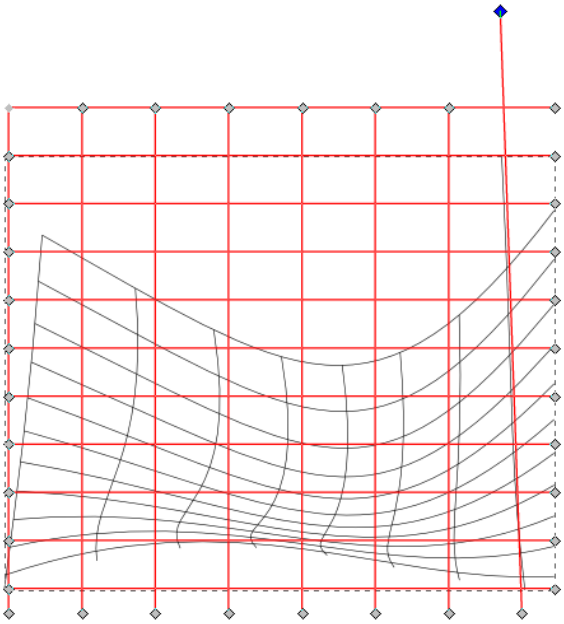


Figure 2.20: The Lattice effect: known bug

The main question is about those original handles positions (called reference points).

With absolute parameters, the reference points are made when the effect is created and are not changed after. The handles can only be handly moved. This solution fits with intuition when the original path is modified but not when it is moved or resized.

With absolute parameters, the reference points are create when the effect is and are not changed afterwards. The handles can only be moved by hand. This solution corresponds to what one would expect when the original path is modified but not when it is moved or resized.

On the other hand with relative parameters the reference points are recalculated each time the original path changes. The handles follow the reference points. This solution corresponds to what one would expect when the original path is moved but not when it is modified.

On the following pictures the intuitive position of reference points and handles are in green, the absolute solution in yellow and the relative one in blue. The original path is in red, the deformed path is not displayed.

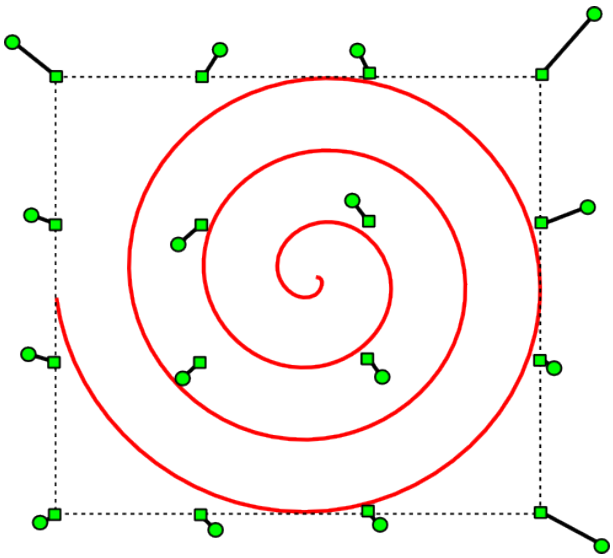


Figure 2.21: Lattice Effect : the effect is applied to a spiral, handles are moved

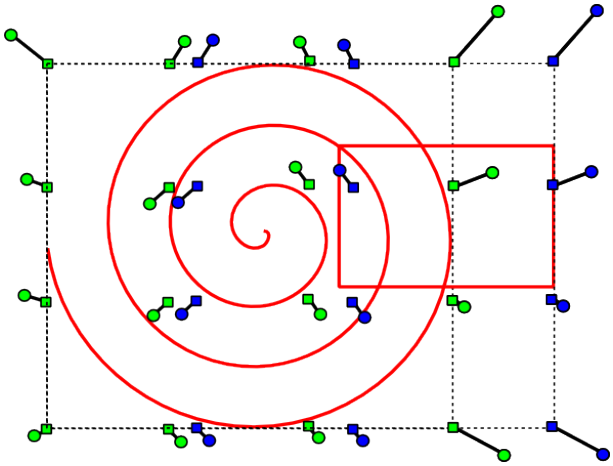
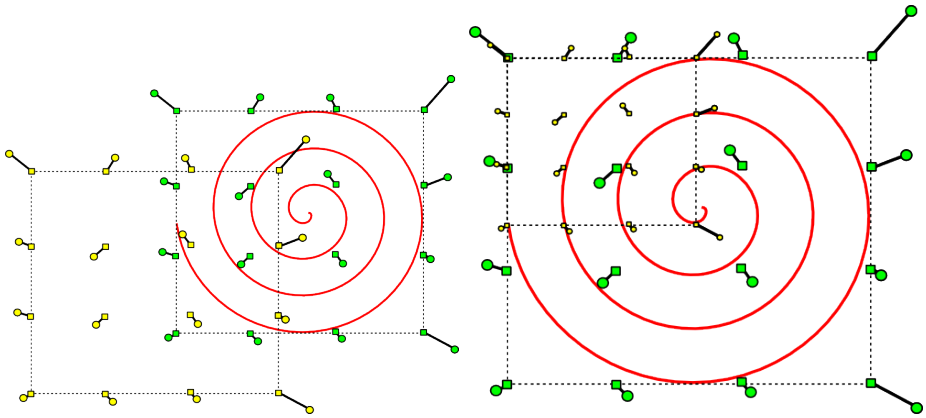


Figure 2.22: Lattice Effect : the original path is modified and so is its bounding box

When the bounding box of the original path is changed, the absolute parameters correspond to what one would expect but not the relative ones.



(a) Moved (b) Resized

Figure 2.23: Lattice Effect : the original path is moved or resized

When the original path is moved or resized, it is the opposite.

We chose to implement the relative solution, because images are often moved and resized. The only remaining problem is when the original path is manually modified and consequently changes the original bounding box.

Envelope deformation

On the image below, we tested a complex deformation of a group of paths. The result is visually what we expected. The deforming paths are shown in green.

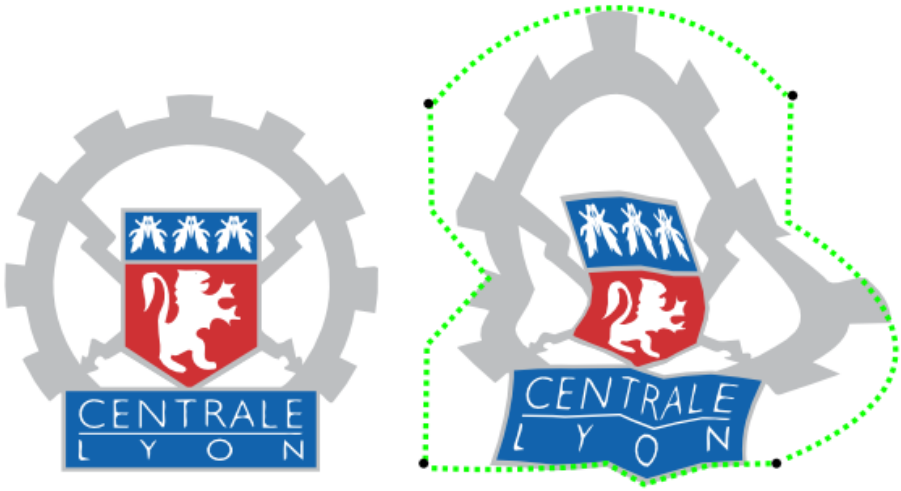


Figure 2.24: A screenshot of the Envelope effect

Known Bugs An unwanted effect may occur when working with lines. Indeed, the algorithm used to compute the final result is not mathematically accurate. As seen on figure 2.25, the edge of the deformed rectangle are quite "wavy", whilst we expect them to stick to the Left and Right bend path. This comes from the weight coefficients we chose.

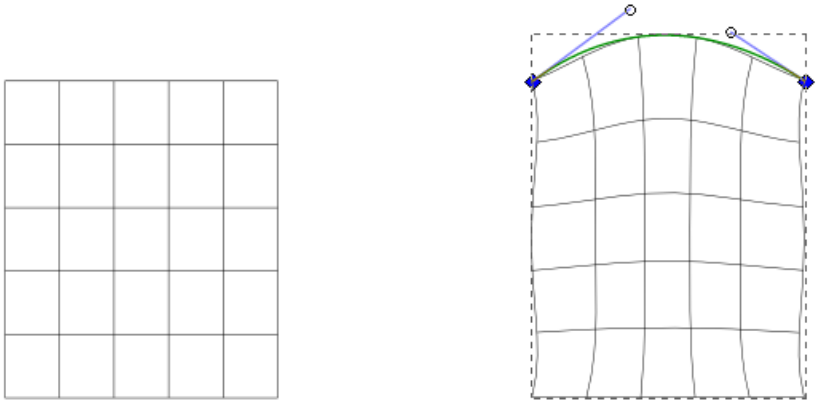


Figure 2.25: The "Wavy-edge" known bug (an artefact)

We must also underline that all the deforming paths can't be edited at the same time. The user has to switch between the four bend paths for the moment, which is not very ergonomic.

Conclusion

Our goals were:

- Enable Live Path Effets assignment to groups.
- Enable Live Path Effets stacking and create the corresponding user interface.
- Create a new effect: Envelope deformation.

Today, we can say they are reached. We are giving to the Inkscape community a work of quality that souldn't require much revision. Of course, we must recognize there still might be some unrevealed bugs even if we already have corrected many of them.

The next step will be to submit our work to the Inkscape team. We will also fill in the Inkscape wiki, that acts as a documentation reference for users and developers. Finally, we will write release notes and give some examples to the community.

We hope our work will be in the next Inkscape version and that it will be usefull for many users. That is what mainly motivated us when we were working hard on the project. At last, we are really proud of having contributed to the Inkscape adventure.

List of Figures

1.1 The difference between a traditional image and a vector graphic . . . 5
1.2 A path made of two cubic bezier curves. 6
1.3 A group made of several object 6
1.4 The Bend Path Live Path Effect applied to a single shape 7

2.1 Partial class diagram of the existing architecture 9
2.2 Partial class diagram of the new architecture 10
2.3 Initial LPE for groups algorithm 11
2.4 Final LPE for groups algorithm 12
2.5 Effects on Group : Test for Sketch 13
2.6 Effects on Group : Test for Bend Path 13
2.7 Group effect: test on a logo 14
2.8 LPE UI comparison 15
2.9 Example of stacking 17
2.10 Interface for the stacking example 17
2.11 Envelope Deformation user case: Drawing a flag 18
2.12 Mock-up of the Envelope Deformation effect 19
2.13 The d2sb2d toy as an example of the possibilities of the 2geom library. 20
2.14 The d2sb2d uses a peculiar numbering system for its handles. 21
2.15 A rectangle deformed by some particular bend paths 22
2.16 Blending the Top and Bottom Deformation 22
2.17 The envelope deformation considering Left and Right bend paths . . 23
2.18 The envelope deformation considering all bend paths. 23
2.19 A screenshot of the Lattice effect 24
2.20 The Lattice effect: known bug 25
2.21 Lattice Effect : the effect is applied to a spiral, handles are moved . . 26
2.22 Lattice Effect : the original path is modified and so is its bounding box 26
2.23 Lattice Effect : the original path is moved or resized 26
2.24 A screenshot of the Envelope effect 27
2.25 The “Wavy-edge” known bug (an artefact) 27
26 The Gantt diagram of the project 31

Appendix

0.1 Internal organisation

0.1.1 Separate tasks

Our project was seperated into three parts:
1 - Live Path Effects for groups
2 - Live Effects stacking
3 - The envelope deformation effect

In order to be more efficient, we decided to divide the team into three groups. In the beginning, each group worked on its own and informed the others of progress status on a weekly basis. After a while, we noticed that some of the parts required the knowledge of the same files. So we decided to mix the groups sometimes to make those tasks get along faster.

The first part to be almost finish was the 1. So Bastien and Steren who worked on it started to help the groups 2 and 3.

0.1.3 Sharing source code

Even if it has been splitted in small groups, the team had to work on the same source code. We early decided to create a Subversion repository. Subversion (SVN) is a version control system, it allows everyone to stay synchronized with our latest internal version of Inkscape. Each time a local modification is done and works well, its creator decide to commit it on the repository. The *revision number* is incremented and a new version of the source code is created. Then, each member of the team check out this new version and download it on his machine. It allows us to always work on our latest version of the code.

Moreover, Subversion is smart, even if you work localy on a file which has been modified in the new version of the repository, Subversion merges easily the new version to your local files when possible. Sometime the new version is in direct conflict with your local modifications (the same lines were modified). In that case subversion displays an alert and you have to solve it.

Subversion was also helpful to stay synchronized with the official Inkscape repository.

0.2 Working on an open source project

0.2.1 External help

Steren first contacted Johan Engelen, student at Twente University, Netherland, who introduced us to the Inkscape development team. Johan is the creator of most of the currently existing LPE implementation in Inkscape and took the responsibility to control the progress of our project and evaluate the final work. He also helped us to define the goals of the project.

If the envelope deformation were done with a Bézier patch, we would have needed to use 16 PointParam objects for the control handles. But the PointParam Class was left half finished. Johann worked on this class to make it usable for our transformation.

To use Lib2Geom, we had to ask for some external help. Indeed there are very few comments within the code and no documentation on how it works. Johann introduced us to Jean-François Barraud, a maths teacher at Lille University. Mr Barraud is one of the co-authors of lib2geom and gave us some pieces of information about the 2bsd2d.cpp toy which we are using for the Lattice transformation effect.

0.2.2 Criticism and benefits

The main drawback we faced while working with free and open source software is the general lack of documentation. It is unfortunate, but quite usual, open source software is written by enthusiast programmers who focus more on writing code and creating features than making it understandable enough for subsequent use. For instance, consider the 2geom library. The idea of a non documented library seemed odd at first: how would people be able to even *use* it ? In fact, we figured out that it was mainly 2geom's creators who used it to make live effects... Even Inkscape's code is not that easy to understand.

The general benefits of free software were true in our case too. When we unveiled flaws in the parts of the software we used, we could warn the community, and they were corrected quickly. An example is the PointParam, used for control handles in our Lattice effect. It was largely unfinished, and the corrections were made as we needed them.

One of the main benefits for us in particular was the possibility to work on a large scale full grown software, without any restrictions or confidentiality contracts that we could have had in a commercial company. Furthermore – and this is more linked to the actual philosophy of free software – another benefit is simply the feeling of having contributed to a community driven project, and a software used by people all around the world, instead of having worked to help a company get richer.

0.3 Technical appendix

0.3.1 The Bend Path Maths

The following text was originally written by J.F. Barraud. It explains the mathematics behind the Bend Path effect.

Let B be the skeleton path, and P the pattern (the path to be deformed).

P is a map $t \longrightarrow P(t) = (x(t), y(t))$ and B is a map $t \longrightarrow B(t) = (a(t), b(t))$

The first step is to re-parametrize B by its arc length: this is the parametrization in which a point p on B is located by its distance s from start. We obtain a new map $s \longrightarrow U(s) = (a'(s), b'(s))$, that still describes the same path B, but where the distance along B from start to $U(s)$ is s itself. We also need a unit normal to the path. This can be obtained by computing a unit tangent vector, and rotate it by 90^o. We call this normal vector $N(s)$.

The basic deformation associated to B is then given by:

$$(x, y) \longrightarrow U(x) + y * N(x)$$

(i.e. we go for distance x along the path, and then for distance y along the normal)

0.3.2 GTK+ / gtkmm

Inkscape's interface is entirely based on the GTK+ libraries, which is also used in many other open source programs, such as The Gimp or the GNOME desktop environnement. To use the GTK+ libraries with the C++ language, we had to use the gtkmm interface.

In order to create the GUI for the LPE stacking, we had to create a new frame, create boxes inside the the frame, then create buttons in those boxes and link them to the corresponding functions and finally display an editable list of effects. The only two files to modify were "livepatheffect-editor.h" and "livepatheffect-editor.cpp".

Basic tutorial

We will name the frame "test_frame". It shall contain a vertical box (VBox) named "test_vbox" which will contain a standard GTK+ stock button named "test_button" connected to the "on_test" function.

In the ".h" file :

```
Gtk::Frame test_frame;
Gtk::VBox test_vbox;
Gtk::Button test_button;
void on_test();
```

In the ".cpp" file

```
        // Defines the frame title
test_frame(_("Text written at the top of the frame"));
        // Creates a standard ''up'' button
button_up(Gtk::Stock::GO_UP);
        // Sets spacing between child widgets
test_vbox.set_spacing(s);
        // Inserts the button at the beginning of the VBox
test_vbox.pack_start(test_button, Gtk::PACK_SHRINK);
        // Adds the VBox to the frame
test_frame.add(effectlist_vbox);
        // Connects the button the the ''on_test''function
test_button.signal_clicked().connect(sigc::mem_fun(*this, &
   on_test));
```

Activating and deactivating the button In the function "LivePathEffectEditor::set_sensitize_all(bool sensitive)" add the following line

```
test_button.set_sensitive(sensitive);
```

0.3.3 How to create and display a list?

In GTK+, the storage and display of lists are seperated. All content is stored in a "Gtk::ListStore" variable, and a "Gtk::TreeView" widget is used to display it. Here is how we implemented the effect list in the UI.

In the ".h" file :

```
        // Defines the number of columns, their names and
           types. In our case we use two columns, but only
           one will be displayed, ''col_name''.
class ModelColumns : public Gtk::TreeModel::ColumnRecord
{
    public:
    ModelColumns()
    {
        add(col_name);
        add(lperef);
    }
```

```
11          Gtk::TreeModelColumn<Glib::ustring> col_name;
12          Gtk::TreeModelColumn<LivePathEffect::LPEObjectReference
                *> lperef;
13  };

15          // Creates the various variables
16  ModelColumns columns;
17  Gtk::TreeView effectlist_view;
18  Glib::RefPtr<Gtk::ListStore> effectlist_store;
19  Glib::RefPtr<Gtk::TreeSelection> effectlist_selection;
20          // We will display the list inside a scrollable
                window
21  Gtk::ScrolledWindow scrolled_window;
```

In the ".cpp" file :

```
1           //Adds the TreeView, inside a ScrolledWindow, with
                the button underneath:
2   scrolled_window.add(effectlist_view);
3           //Only shows the scrollbars when they are necessary:
4   scrolled_window.set_policy(Gtk::POLICY_AUTOMATIC, Gtk::
        POLICY_AUTOMATIC);

6           //Adds the columns to the ListStore and creates the
                Tree model
7   effectlist_store = Gtk::ListStore::create(columns);
8   effectlist_view.set_model(effectlist_store);

10          // Handles tree selections. We link the selection
                action to a function.
11  effectlist_selection = effectlist_view.get_selection();
12  effectlist_selection->signal_changed().connect( sigc::
        mem_fun(*this, &LivePathEffectEditor::
        on_effect_selection_changed) );

14          // Makes the column names invisible
15  effectlist_view.set_headers_visible(false);

17          //Adds the visible column to the TreeView
18  effectlist_view.append_column("Effect", columns.col_name);

20          //Adds elements to the list. In our case, this loop
                is embedded in the ''effect_list_update``
                function, which is called at each time the
                displayed list must be updated. The ''effectlist
                `` variable is a list containing the LPEs applied
                 to the selected object.
21  for( it = effectlist.begin() ; it!=effectlist.end(); it++ )
22  {
23      Gtk::TreeModel::Row row = *(effectlist_store->append());
24      row[columns.col_name] = (*it)->lpeobject->lpe->getName()
```

```
        ;
25      row[columns.lperef] = *it;
26  }
```

0.4 Personal comments

Tebby Hugh

Once I actually started to get familiar with the code and the locations of the various files, I found that it isn't actually that difficult to make modifications – hopefully enhancements. I hadn't had much experience of programming beforehand so it wasn't really easy getting started, but now I would feel confident enough to make some major changes – if needed, of course – to Inkscape, or any other open source program.

I mostly worked on the LPE stacking on the UI side, and I'm now quite familiar with GTK+ and gtkmm, which I had never used beforehand. I'll most likely continue working on the Inkscape code, for instance correcting the bugs the community will find or enhancing the interface for stacking. I might also have a look at other open source software, maybe less complex than Inkscape, to see where help is needed.

What we've done is not yet ready to be in the next official release of Inkscape, some aspects being quite rough in the corners, but it should soon be added to the official svn repository – as some bits already have been. I hope we get enough feedback from the community for bugs and various enhancements to allow us to polish our work. And I also hope our efforts will be appreciated. After all, what can we get out this job apart from glory?

De-Cooman Aurélie

I think that this project was quite interesting and difficult. My main difficulty was to get familiar with the Inkscape code (all the files without any comments and only long lines of code).

But it was very interesting to see all the background of open source software: the forums where you can go and ask questions, the strong mutual help between all the developers, the functioning of the svn repository. And it is really something to think that a lot of people will actually use what we've done for Inkscape.

Falzon Noé

This project was a great opportunity to work on a free and open source software like this one, for several reasons. First, being an everyday user of free software, it gave me a chance to contribute back to the community. Then, as an amateur programmer, I never worked on more than little personal projects. Adding my lines of source code to a software used by thousands, and praised by the critics, is an accomplishment per se. Finally, if we had to work without being paid, I do not think I would have appreciated to do it for the profit of a company.

The programming and designing itself was quite interesting, since it required a bit of mathematical and computer science background. It was a good way to practice our skills and knowledge.

Of course there were little disappointments, in particular about the lack of documentation that ruins – in a way – the quality of some of the libraries we used. The

debugging was generally very hard, since we – on the lattice effect – did not really understand how it worked. The bad times when nothing worked like we wanted were largely compensated by the pleasure of having something functional and real emerging from ideas and pure will.

I will probably follow for a while the evolution of "our" code in the official Inkscape SVN repository, and the comments made by users. Maybe I will even go on writing for Inkscape, or correcting yet unrevealed bugs in our effect.

Anyway, it reinforced my desire to participate in free and open source projects, or even maybe to start one myself someday.

Navez Victor

It was really great working on such a widely used tool. I mainly worked on the lattice effect with Noé which will, in my opinion, be quite useful for Inkscape users. We had much difficulties understanding the lib2geom code at the beginning, so we really enjoyed it when we finally had functional results.

It was also really interesting working on an open source project with a very active community since other people are making enhancement on other parts of the code while we were working on live path effects. Johann was working on PointParam – a class we are using for the lattice effect – and each time he committed on the the official Inkscape SVN repository we felt the benefits for our own work.

Bouclet Bastien

The greatest difficulty we encountered was getting into the code. We first spent a lot of time staring at the code and trying to decipher it. But once we understood it, our work soon started to show tangible results. Then, working on Inkscape became extremely rewarding and quite pleasant. However, I won't keep working on Inkscape after the end of the course, mostly because I don't use it on a daily basis, and I don't have enough free time, but I'll definitely keep an eye on it.

The strength of Inkscape's community impressed me. The developers are very friendly and open. Everybody can come and talk to them or ask for help. I think this is why Inkscape is doing so well.

Overall this project has been highly interesting. It allowed me to have a better understanding of the development of open source software.

Giannini Steren

Why Inkscape ? Mainly because I really wanted our *free* work used in a *free* project. I have been wanting for a long time to take part in a serious open-source software. This project was the ideal opportunity to do so in the scope of my studies.

As the leader of this project, I first had to gather a team. I would say this is a key point in the settlement of a project. Now I believe that the most important thing is to get interested people. As we saw during the project, members not really interested, be they technically skilled or not, won't do a lots of efforts. Nevertheless, I am fully satisfied with the work the team completed and it was a real pleasure to work in it. I think we together managed to achieve a great project and I am very proud of it.

This project was very interesting, I have learnt a lot concerning the structure of a real software. Of course, I must mention the general lack of documentation which, I think, slowed down the development.

In the end, it was for me a great pleasure to have contributed to one of my favorite software. I seriously intend to continue working on Inkscape, first by correcting and improving our actual work and then by adding many ideas I have in mind. I deeply hope people will like and use what we did.