A plane transformation from 16 control points

The definition of the transformation

Assume we are given four cubic Bézier curves, B_0, B_1, B_2, B_3. Each is defined by four control points, so we have 16 control points $p_{ij} \in \mathbb{R}^2$, and we write

$$B_r(t) = \sum_{s=0}^{3} b_s^3(t) p_{rs} \qquad (r = 0, 1, 2, 3) \tag{1}$$

where the b_s^3 are the Bernstein polynomials. By grouping the 16 control points differently we have another set of four Bézier curves:

$$C_s(t) = \sum_{r=0}^{3} b_r^3(t) p_{rs} \qquad (s = 0, 1, 2, 3). \tag{2}$$

From the 16 *control points we construct a transformation* $F_0(u, v) : \mathbb{R}^2 \to \mathbb{R}^2$ as follows:

$$F_0(u, v) = \sum_{r=0}^{3} \sum_{s=0}^{3} b_s^3(u) b_r^3(v) p_{rs} \qquad ((u, v) \in \mathbb{R}^2). \tag{3}$$

This can be written in terms of the B's

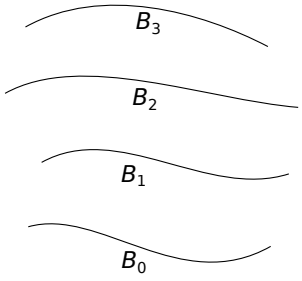
$$F_0(u, v) = \sum_{r=0}^{3} \left(b_r^3(v) \sum_{s=0}^{3} b_s^3(u) p_{rs} \right) = \sum_{r=0}^{3} b_r^3(v) B_r(u), \tag{4}$$

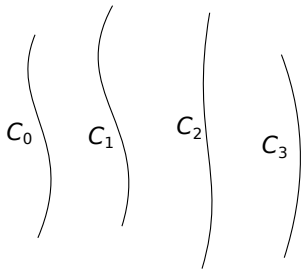
or in terms of the C's

$$F_0(u, v) = \sum_{s=0}^{3} \left(b_s^3(u) \sum_{r=0}^{3} b_r^3(v) p_{rs} \right) = \sum_{s=0}^{3} b_s^3(u) C_s(v). \tag{5}$$

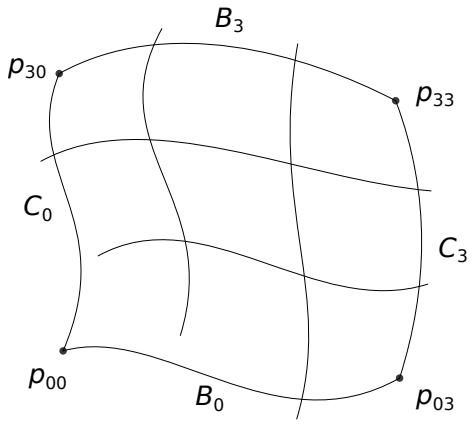
An effort to visualize

The parameter t in (1) and (2) runs the whole $\mathbb{R}$. Restricting it to the interval $[0, 1]$ we get what we call here *Bézier arcs*. We try now to visualize the arcs B_i and C_j and the effect of F_0. To this end, we assume that the arcs B_i run as in the picture on the left, each horizontally, one above the other. It follows that the arcs C_j each run vertically as in the picture on the right.





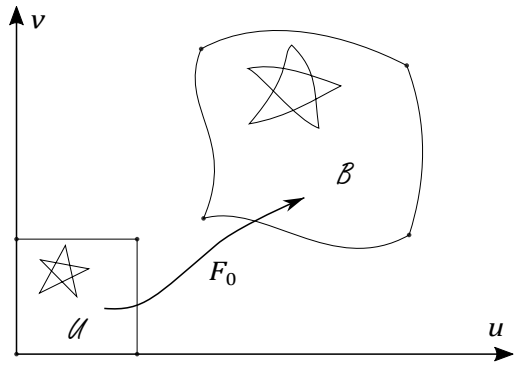
The two sets of arcs are not independent: they share the control points p_{rs}, and they cross each other like in the picture below. In particular, the outer arcs B_0, C_3, B_3, C_0 have common end points so that they in fact form a closed curvy quadrilateral:



Let us call the curvy quadrilateral $\mathcal{B}$. We shall need also the unit square with vertices $(0,0)$, $(1,0)$, $(1,1)$, $(0,1)$; we call it $\mathcal{U}$. Easily from (4) and (5), for all u and v,

$$F_0(u,0) = B_0(u), \quad F_0(1,v) = C_3(v), \quad F_0(u,1) = B_3(u), \quad F_0(0,v) = C_0(v). \tag{6}$$

This means that F_0 sends the sides of the unit square $\mathcal{U}$ onto the sides of the curvy quadrilateral $\mathcal{B}$. The following picture shows this very schematically.



Along with the unit square, everything appearing on the plane is mapped by F_0 to some other place and distorted. As an example I drew here a pentagram and what might be its image in F_0: a curved pentagram.

Note that F_0 is *not* conformal. It is not bijective either.

A simplified version

If a plugin is implemented directly from the above formulas, the user will have to feed to the plugin all 16 control points p_{rs}, and they should be somehow compatible to obtain anything useful. This is too awkward. A better idea is to offer the user some restricted way to feed the inputs, and the missing inputs are then internally computed from those using some reasonable default formulas.

I propose the following scenario. A plugin is implemented to expect as input the outer Bézier arcs B_0, C_3, B_3, C_0 as one path; this is the curvy quadrilateral $\mathcal{B}$. That gives 12 of the 16 control points. To get the missing four, the inner control points $p_{11}, p_{12}, p_{22}, p_{21}$, we require that the four high terms u^3v^3, u^3v^2, u^2v^3, u^2v^2 in $F_0(u, v)$ vanish, the idea being to avoid too strong fluctuations in F_0. It turns out that these four vanishing conditions indeed enable us to solve from (3) the control points $p_{11}, p_{12}, p_{22}, p_{21}$ in terms of the 12 input control points. Calculations (rather long) lead to the equations

$$\begin{cases} p_{11} = \dfrac{1}{9}\Big[-4p_{00} - 2(p_{03} + p_{30}) - p_{33} + 6(p_{10} + p_{01}) + 3(p_{13} + p_{31})\Big], \\ p_{12} = \dfrac{1}{9}\Big[-4p_{03} - 2(p_{00} + p_{33}) - p_{30} + 6(p_{13} + p_{02}) + 3(p_{10} + p_{32})\Big], \\ p_{21} = \dfrac{1}{9}\Big[-4p_{30} - 2(p_{00} + p_{33}) - p_{03} + 6(p_{20} + p_{31}) + 3(p_{23} + p_{01})\Big], \\ p_{22} = \dfrac{1}{9}\Big[-4p_{33} - 2(p_{03} + p_{30}) - p_{00} + 6(p_{23} + p_{32}) + 3(p_{20} + p_{02})\Big]. \end{cases} \tag{7}$$

These work well in the cases I have tried.

The plugin

I implemented formula (3) with the simplifying idea and wrote a Gimp plugin to transform paths. (It includes an approximation algorithm since the resulting paths must be drawn as Bezier curves. I say nothing of that problem here. It would be a story in its own right.)

Basically the plugin takes as input

- Path, the path to be transformed;
- Base, a path with two anchors A and B;
- Target, a path with two anchors P and Q;
- Shaper, a closed path with four anchors; this is the curvy quadrilateral $\mathcal{B}$ with sides B_0, C_3, B_3, C_0;

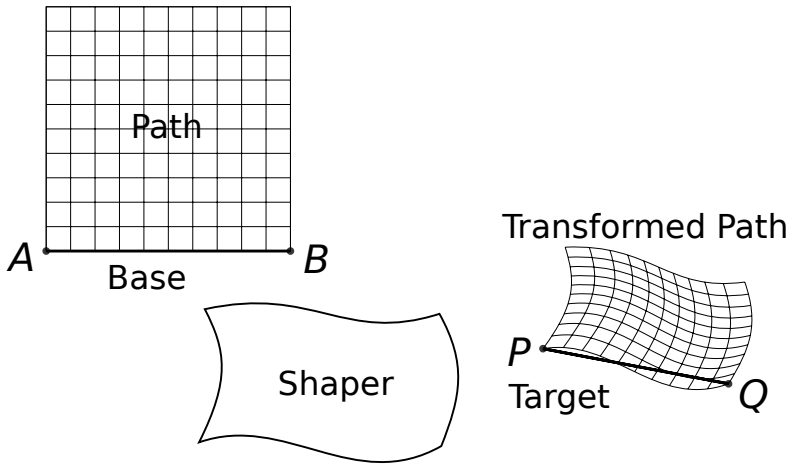
and some others. The idea is to embellish the transformation F_0 with suitable wrapper functions which cause the anchors A and B of the Base to be sent onto

the anchors P and Q of the Target. So the Base and the Target are used to determine from where to where the mapping goes.

There is another way to use the plugin: the Base and the Target each may have four anchors rather than just two. This way will be explained only at the end of this text.

An example

Here is an example figure:

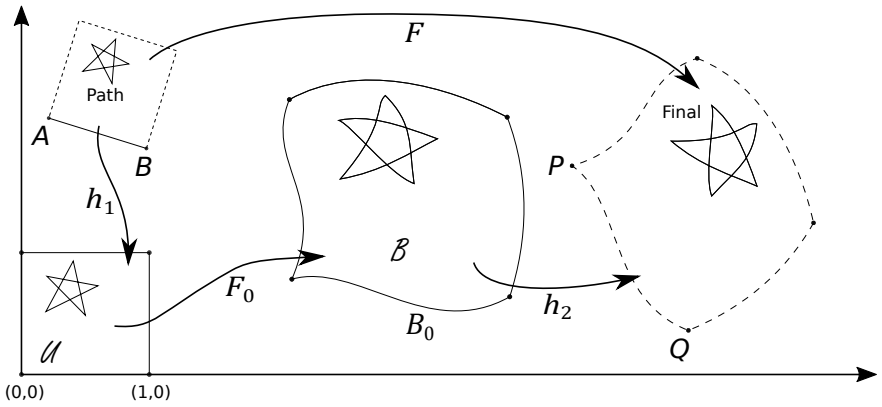


The Path to be transformed is the grid at top left. The Base (points A, B) is in this example set precisely at the bottom edge of the grid. The Target (points P, Q) is at bottom right. The Shaper is the curvy quadrilateral at middle bottom. The effect of the whole transformation is that the grid is deformed to the shape of the Shaper (this is the effect of F_0) and simultaneously the points A, B are sent to P, Q (thanks to the wrapper functions). For how this is accomplished, see the next section.

The reason why in this case the shape of the Shaper shows so clearly in the transformed Path, is mainly that the Base coincides with the bottom edge of the grid.

How the plugin works

For those interested, here are some words about the inner working of the plugin. See the following picture.



The implemented transformation is

$$F = h_2 \circ F_0 \circ h_1 \tag{8}$$

where F_0 is the core transformation (3) and h_1 and h_2 are the wrapper functions. (And the '$\circ$' is a mathematical symbol telling that the three functions are applied in succession: first h_1 is applied, then F_0, and finally h_2.) The maps h_1 and h_2 are direct similitudes, so they preserve all shapes. As the picture tries to show, h_1 sends the original figure (the Path and the Base A, B) to a *standard position*: A and B are sent to $(0, 0)$ and $(1, 0)$, respectively. Then F_0 is applied. The final map h_2 is chosen so that it sends $F_0(h_1(A)) = p_{00}$ and $F_0(h_1(B)) = p_{03}$ to P and Q, respectively.

This way of using h_1 and h_2 and going through the standard position ensures that:

- The absolute size and location of the Path do not matter, only its size and location relative to the Base. (Here "location" includes inclination.)
- The size and location of the Shaper will have no effect, only its shape. (But because of the way the plugin is written, it does matter which arc of the Shaper is the lowest on the screen; see below.)

About designing user inputs

It follows also that arc B_0 has a special meaning. Namely, the Base is sent to the line segment $\big[(0, 0), (1, 0)\big]$ which is sent to the bottom edge B_0 of the Shaper and finally that is sent to the Target:

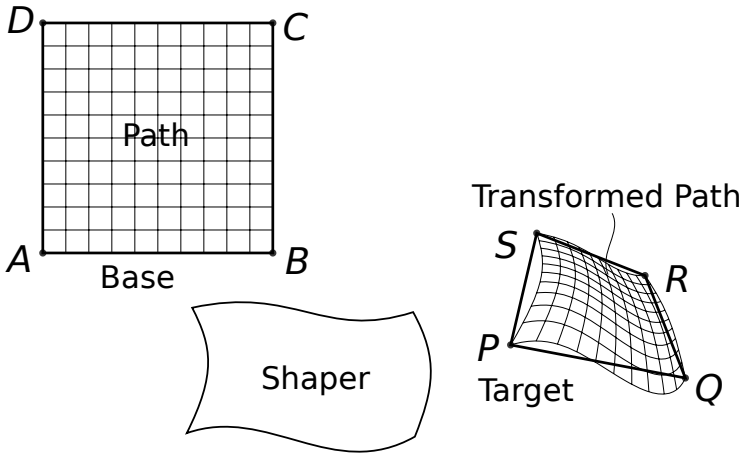
$$\begin{aligned} \text{Base points } A, B &\xrightarrow{h_1} (0,0), (1,0) \\ &\xrightarrow{F_0} \text{ end points } p_{00}, p_{03} \text{ of } B_0 \\ &\xrightarrow{h_2} \text{ Target points } P, Q. \end{aligned} \tag{9}$$

Therefore, it is crucial in the plugin to know, which of the four edges of $\mathcal{B}$ is B_0 and in which direction it runs, and it is clearly also crucial that the directions of the Base and the Target are chosen correctly. This problem must be addressed to in the design of the GUI, since in Gimp the user generally does not know in which direction a path runs, or in the case of a closed path, which is the starting anchor.

The plugin is deliberately written with the situation as in the above example in mind: The Shaper is quadrilateral with its sides horizontal, vertical, horizontal, vertical, running counter-clockwise, and B_0 is the bottom edge; the Base and the Target are line segments with direction from the left to the right. If the input paths do not obey such rules (for example, if the user inputs a Base which runs from the right to the left), the plugin first internally arranges things to what it thinks is good: It takes as B_0 that edge of the input $\mathcal{B}$ which is located lowest on the screen, and sets the direction of that edge to be from the left to the right; the Base and the Target are set to run from the left to the right on the screen. This way of doing things presumably best fulfills user's intention. But in the GUI of the plugin the user can override those decisions.

Base and Target which have four anchors

But the plugin allows also the Base and the Target to have four anchors. Then instead of direct similitudes, the plugin uses projective transformations as the mappings h_1 and h_2. This introduces further distortion to the resulting figure. The idea is that, for example, one can choose four points from the original figure, say A, B, C, D, and have them mapped onto another pre-chosen set of four points P, Q, R, S. This is done by feeding to the plugin Base and Target formed from these points. An example:



Here the Base (points A, B, C, D) is set to the outer boundary of the grid. The

Target (points P, Q, R, S) is the skew quadrilateral. The transformed Path is further distorted (as compared with the previous example) so that A, B, C, D are mapped onto P, Q, R, S.

Admittedly, this effect could be achieved by applying first the plugin with the Base and the Target consisting of two anchors, and after that using Gimp's Perspective transform tool on the result. But this option seems good to be available. There is a snag, however: h_1 and h_2 are projective transformations, hence the whole process may fail (with a message "Hit an infinity") since infinities occur quite naturally for such mappings. Also, collinear anchors in the Base or in the Target, or even in the Shaper, are not allowed. Namely, instead of (9), we have

$$\begin{aligned} \text{Base points } A, B, C, D &\xrightarrow{h_1} (0,0), (1,0), (1,1), (0,1) \\ &\xrightarrow{F_0} \text{corner points } p_{00}, p_{03}, p_{33}, p_{30} \text{ of } \mathcal{B} \\ &\xrightarrow{h_2} \text{Target points } P, Q, R, S, \end{aligned} \tag{10}$$

and to define the projective transformations h_1 and h_2 no collinearities among the points are allowed. If a failure occurs, the remedy is to make changes in the input paths and try again.