Manipulating curves by innovative plastic multitouch interactions

Yoann Bourse (ENS Paris)
Mentor : Laurent Grisoni (INRIA MINT Lille)

Summer 2010

Abstract

This papers focuses on the design of a multitouch software enabling tactile drawing and manipulation of one-dimensional functions. It is destined to a wide public, that is to say to be usable without any mathematical knowledge. Therefore, we established an user-intent driven pattern based on a sketch paradigm. To that aim, we tried to maintain as an invariant the global shape of the drawn curve during the various manipulations, which resulted in technical challenges.
We also wanted to get as far as possible from the conventional mathematical tools, and use the possibilities brought by multitouch technologies to develop innovative fluid interactions to manipulate curves in a plastic way. This resulted in very intuitive tools leading towards a reconsideration of the way we view curves.

Contents

This project was motivated by the need of various people without mathematical background to deal with curves representing the variation of a given parameter in respect to another one (being often the time). They were forced to rely on other specialized people whom they could have difficulties communicating with, given their different formations. They often had to draw little sketches to make their intent understood.
The gap between their background and the very technical existing tools lead us towards a reconsideration of curves. Thanks to multitouch interactions, we tried to get closer to human's intuition and to provide new tools to manipulate curves that require no mathematical knowledge. Since we wanted our software to answer the need for intuitive manipulation tools, we decided right away to focus on user's intents. Therefore, the semantic of the curve in the situations we considered lead us to focus on the case of one-dimensional functions (at a given time, the represented system is in a single state). Moreover, we wanted our manipulations to be smooth and plastic, and thereby distort the curve without losing its general shape.
All those constraints resulted in the development of a very user-friendly software enabling anyone to draw and manipulate curves by plastic operations with multitouch interactions.

1 Related work

1.1 Curve modelling

For various reasons discussed later *(see 2.1)*, we were brought to use a cardinal spline interpolation paradigm. We transformed the user input (a list of points) to a piecewise polynomial function using cubic Hermite spline polynomials, inspired by the success of this model in modelling and visualization.

1.2 Intent-driven paradigm

During all the development, we focused on the user's intents. We wanted a very simple and user-friendly interface, so that it would not take any expertise at all to master. As a result, we ended up with almost no buttons, and nothing else than the fingers to interact with the curve. Therefore we needed to focus on the context to retrieve informations about user's intents. The mechanisms involved are discussed later *(see 3.3)*. We were inspired by several successful examples such as Google Sketch Up [2].

1.3 Shape manipulations

In order to stick to the intuition, we analyzed the psychological aspects of curve manipulations. We were inspired by Michael Leyton's Shape Form Deformation theory [3], which states the hypothesis that a shape is mentally defined as a succession of deformations applied to a basic shape. We therefore provided the user with a corresponding experience : they will first draw a simple shape and then modify more precisely differents aspects of it in order to obtain the final required result.

1.4 Multitouch technologies

Our manipulation are designed to fit the movements of a human hand, in order to create a direct contact between the user and the manipulated curve. Multitouch technologies play therefore a major role in enabling the intuitive manipulation of curves. But there are currently many providers of such technologies, and almost no standards, making the interoperability very hard. Although Apple platforms are widely spread, they are very restrictive as regards programming. Therefore we chose to focus on Microsoft Windows Seven.

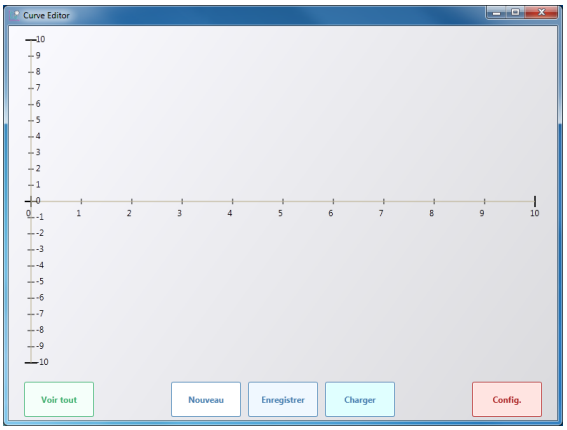
2 Global orientation

We will present and discuss the reasons of the guiding lines we adopted in this work in order to enhance our user-friendly plastic interaction experience.

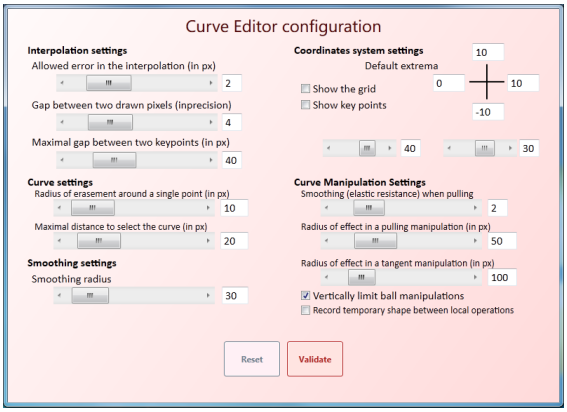
2.1 Curve

Given our specifications, we decided to work only with one-dimensional curves. Yet, the user has the possibility to click anywhere on the screen and thereby to draw loops or parametric shapes that do not fit the one-dimensional curve constraint. We had to maintain this invariant in an intuitive way. Our solution was to consider the user input like a sorted list of Y-coordinates, indexed by X-coordinates : therefore, to each X-coordinate corresponds only one Y-coordinate. Drawing a new point with the same X-coordinate overwrites the old one, which turns out to be very intuitive.
This, on the other hand, caused problems with the frequency of the sampling of the input : indeed, we do not receive a continuous signal but a sampled list of positions of a finger. Therefore a finger is rarely twice in the exact same spot, and registering every position poses the risk to end up with a lot of very thin spikes. Therefore, several points must be erased, and some data are bound to be lost *(see 3.1)*.
Moreover, the frequency of the sampling forces us to find a smooth way to join the points. Hence the use of an interpolation. It also eased the manipulations to come, and improved greatly the performances. The various shapes of the drawn curves lead us to use a cardinal spline interpolation, with Hermite polynoms *(see 3.4)*. Therefore, the curve is defined as a list of key points (two coordinates plus one tangent).

2.2 Interface



The interface of the software had to be the starting point of an immersive experience for any user, regardless of their background in mathematics or their familiarity with computers. Therefore, our goal was to keep it to the strict minimum, without any unnecessary buttons, all the more so that we wanted our application to be usable only with fingers. As a result, we provided large buttons and a configuration panel (to change the various parameters of the software) whose settings could be modified through scrollbars.



The main screen is the drawing board, where the user can draw directly with their fingers. He will then be able to complete what he has drawn by other dots or lines, or to manipulate and transform what is already on the screen thanks to the various tools at his disposal *(see 3.6)*.
One of our other guiding line was to provide a omnipresent visual feedback *(see 3.6.1)*. Whatever the user does, the program always provides a notification to inform the user that their action has been taken into account. Colors, sizes and positions are used to distinguish as much as possible the different states in order to provide a clear environment. The notifications however are as light as possible so as to keep a sober and purified interface.

2.3 Manipulations

To conceive the different manipulations, we tried to get as far away as possible from the mathematical perspective, and consider the curve as a plastic object. We wanted innovative operations, though their bases would be theoretically solid. We also aimed at providing different levels of expertise for those manipulations, to increase performances and precision when the user gains experience *(see 3.6.3)*.
Without buttons in the interface, the only tool at the disposal of the user was their fingers, thanks to the multitouch technology. Hence the crucial importance of the context in order to stick to the user's intents. The type of the manipulation is thus determined by various factors including the number of fingers down, their relative position, their movement direction... Yet, the decision between the numerous possibilities had to be very fast in order to maintain a real-time interaction between the user and the curve *(see 3.3)*.
One of the most basic operations we imple-

mented was the possibility to stretch the axis in order to zoom on parts of the curve. Started this moment, the screen could display only a part of the curve. Therefore, the precision of a manipulation could be directly given by the zoom : we operate on the pixels, and the manipulation is applied to the points they represent, according to the current zoom.

2.4 Implementation

We will present later *(see 4)* a diagram summarizing the consequences of those constraint and choices on the implementation. Several manipulations were programmed, including :

- Zoom and translations of the axes
- Translation of the curve
- Horizontal stretching of a part of the curve
- Vertical stretching of the whole curve
- Gaussian smoothing of a part of the curve (sanding)
- Rotation of a part of the curve (tangent modification)
- Various local applications of a translation vector

All of those moderated and modified in order to stick to our constraints *(see 3.6.4)*

3 Features

Let us now focus on different aspects of the program. We will describe the various features, their conceptual implications as well as the technical challenges we had to overcome. In this section, we distinguish the user-drawn curve designed by a direct finger input (initial drawing or new drawing overwriting previous parts of the curve), and the manipulations which are only transformation applied by the user to it later on thanks to the tool we developed *(see 3.6)*.

3.1 Input frequency/precision

We can distinguish two drawing operations : pointing to draw a single point, or moving one finger in a continuous move to draw a line. Both raised challenges as regards the frequency of the sampling of the input, considering the fact that we view the one-dimensional curve as a list of Y-coordinates indexed by X-coordinates, without doubles, thanks to an overwriting paradigm.

Pointing operation

Pointing in a single location is interpreted by the will to add a single point to the curve. However, there are chances that a point of the curve already exists only a few pixels away from the location desired. This will result in the creation of a thin spike. Even though it enables precision drawing, this conflicts with our overwriting paradigm : a point is almost impossible to erase willingly. All the more so than the main input tool is supposed to be the user's finger, which is not at all precise. Therefore, adding a point erases neighbouring points in a given area of effect (default is 10 pixels). This loss of data seems necessary to deal with the bad precision of a tactile input.

Moving operation

Depending on the speed of the finger, the gap between two following signals in a moving operation might be significant, and therefore those two signals may surround existing points. However, those two signals are bound by the semantic connotation of the user's moving operation : they correspond to the same finger which was not lifted of the screen, and therefore are expected to be directly linked. Hence the need to keep the previous sample registered, however close it is, but delete any point between the previous and the current sample. Moreover, we have to be careful not to delete the previous sample even if it is in the radius of erasing mentioned in the pointing operation (which is most of the time the case).

3.2 Coordinate system

As seen in *2.4*, the coordinate system acts as the control of the buffering between the whole function stored in memory with actual X/Y coordinate, and the graphical part of the curve displayed on the screen with pixels coordinates. The ability to zoom in and out of the curve is the crux to the setting of the precision of any manipulation, as they operate on pixels *(see 2.3)*.

Graduations

The displayed coordinate system is able to adapts itself automatically to the scale, seeking the optimal scale for a better rendering. The step between two axes graduations is chosen between powers of ten, according to the maximal number of graduations specified, thanks to the following law :

$$step = 10^{\lfloor log(\frac{Max-Min}{MaxNbGraduations}) \rfloor}$$

Various trials with other automatic step system proved inefficient : the possibles step have to be multiples of each other so that no graduation appears out of nowhere and thereby disturb the user.

Stretching

Any axis manipulation is started by putting a finger down on one axis. The translation of the viewed zone is trivial (the coordinate system follow the finger). But changing the scale of the coordinate system was a little more complex. This was a perfect opportunity to study stretching manipulations in order to make our zoom as intuitive as possible.
We distinguish two different types of stretching : either the user puts one finger down and then uses another one to stretch the axes in respect of the first one *(figure 1)*, or they put down two finger in the same and move them around *(figure 2)*. In the first case, the user expects the first location they pointed to pinpoint the curve. In the second one, the pinpoint would be the middle of the two initial position of the fingers.

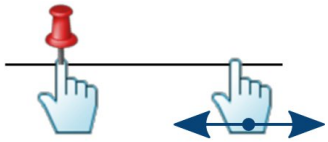


Figure 1: Pinpointing and moving finger

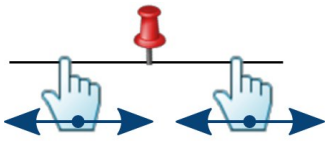


Figure 2: Two moving fingers

In order to determine which stretching is intended, we decided not to rely on a temporal difference which could be misleading, but rather on the movement of the first finger down : if its speed raises above a given threshold, we enter the second kind of stretching.
We wanted to enable the manipulation of either one axis at a time, or both in the same time. This was another user-intent determination challenge. It is settled by the consideration of the respective position of the two fingers : if the vertical component of the vector they are forming is low enough, the stretching operates only on the horizontal axis, and vice versa. If both component are significant, we stretch the horizontal and vertical axis proportional to their corresponding component, so that the user has a control over the direction of the stretching by the direction of the vector joining their two fingers.

3.3 User-intent determination

Our goal was to develop a user-intent driven software without any other tool for the user than their fingers. Therefore, context-based determination of the ongoing process is omnipresent. We will present here an overview of the general mechanisms.
We distinguished three zones on the drawing board : the curve, the axes and everywhere else (free space). When the user puts one finger down, the program will answer in consequence. The lines constituting the curve and the axes being very thin, there is a settable tolerance threshold to select it. Each zone will allow a different set of actions. The number of fingers put down following this first one is then monitored and will be useful to determine the exact function to provide. Movement and position of all the fingers are of course considered. The time however is not, due to the fact that users can have very different manipulating speed depending on their experience with the program. We will present briefly the detection mechanisms involved, without describing the various manipulation which will be dealt with later *(see 3.6)*.

Free space

Putting one finger on the free space will enable the drawing mode, that is to say direct user drawing input (line or dot). However, putting a second finger will allow the user to stretch a part of the curve. Therefore, when only one finger is down, we are in an undetermined state. We need to buffer the input point and wait for either a lift or a move from the finger (and confirm the will to draw) or a second finger down (and confirm the stretching).

Axes

A finger down translates the function until another is down, starting the stretching mode. The stretching mode then will not exit if one finger is up, enabling to stretch several times in a row.

Curve

A finger on the curve enables the various manipulations, that is to say transformations of the curve with the conservation of its global shape. All those operations have a default area of effect, but it can also be specified by pinpointing two points of the curve which will stay fixed and become the boundaries of the manipulation's influence. Hence the apparition of another undetermined state.
The number of fingers down is not sufficient to settle the conflict. Indeed, the rotation manipulation uses two fingers, and is triggered by putting down the second finger on the tangent of

the curve, next to the first one. Therefore, the position of the second finger down is crucial : if it is anywhere else than the tangent, it will set the area of effect, and those two first fingers will become virtually inexistant : it is exactly like if they did not exist, but we cannot remove them per say from the actual list of fingers detected because it will start a new interaction. Hence the need to operate on a virtual list of "free" fingers, the fingers that are not pinpoints, that is to say all the fingers down but the first one and the second if it was anywhere else than the tangent. The position and movement of the fingers in this list will then trigger manipulations, locally limited or not.

3.4 Interpolation

As we use a cardinal spline interpolation to connect the points of the curve, we can optimize the process by getting rid of all the unnecessary points. Therefore, any drawing input is finalized by the update of the interpolation which preserve only what is necessary. This operation is done by a little algorithm described below. An option in the configuration panel enables the update of the interpolation after every manipulation, in order to register more precisely the result of it. In order to stick to a sketching paradigm and to focus on the user-drawn curve, it is by default disabled.
Although the drawing board is not especially oriented, we can notice that this algorithm is. However, it does not seem to cause any relevant consequence. This algorithm enables us to work only with a minimal description of the user's input, so as to optimize performance and to give us more flexibility in the various manipulations of this input that will be possible later on. As a result, the curve is stored and manipulated as a list of key points (coordinates plus tangents). Hence the obligation to set a maximal distance between the kept points, in order to maintain the global shape of the user-drawn curve during all the manipulations to come, even if it may result in the keeping of unnecessary points.

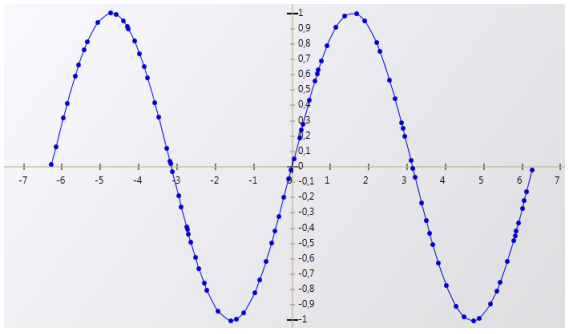
Let us note $H(begin, end)$ the Hermite polynomial obtained by an interpolation on the segment $[begin, end]$ using the coordinates of the points *begin* and *end*, and their neighbouring points to compute the appearing tangent in those points.

```
KeyPoints = [];
OriginPoly = DrawnPoints.First();
PreviousPoint = DrawnPoints.First();

foreach (CurrentPoint in DrawnPoints)
{
    Check that all the points in
    [OriginPoly;CurrentPoint]
    are close enough from their value
    in H(OriginPoly, CurrentPoint)
    else {
        Add PreviousPoint to KeyPoints;
        OriginPoly = PreviousPoint;
    }

    PreviousPoint = CurrentPoint;
}

Add DrawnPoints.Last() to KeyPoints;
```



3.5 Maintaining the shape of the curve

Attenuating the manipulations

Maintaining the shape of the curve was one of the hardest challenge we had to face, and one of the reasons for our use of key points. They were easier to manipulate and enabled more complex operations, often constructed around geometric bases, which enhanced the plastic view of the curve we wanted for our program. Yet, those geometric operation, as intuitive as they might be, raised the major problem to enter in conflict with the invariant we wanted to keep : the curve had to stay the representation of an unidimensional function. Therefore, the problem was how to handle manipulations which would change the order of key points (that is to say if a key point that was right to another one ends up left to it after the manipulation).

This case was of course to be forbidden. It was relatively easy to detect, but avoiding it was a lot more complex. We could have simply stopped the manipulation on detection of a conflict, but it was a dead-end. Indeed, if any conflict in the area of effect of a manipulation were to stop it, this manipulation would be drastically limited. Our trials resulted most of the times in rotations with no more than 40° of movement amplitude for instance. Moreover, such a solid block was totally against our will for a plastic view of the curve. Hence the need for another solution which would slow down the manipulation around any conflict zone without preventing it from continuing elsewhere (possibly slowed in order to notify the user of the conflict).

We therefore developed such a generic attenuation algorithm for all our manipulations. Instead of writing a complex unclear pseudo-code, we will explain its functioning. Its goal is to produce, for every point, a moderating factor in the range [0,1] which will be multiplied to the transformation vector the point is going to follow and thereby avoid any conflict. Let us call it maximal possible factor. The algorithm therefore needs all the transformation vectors for the studied manipulation, and it will maintain a maximal possible factor for every point, which can only decrease. It operates around a center (often the finger) and within the boundaries of the area of effect, in a centrifugal way. It will indeed consider every point from the center to the boundaries, and try to solve any possible conflict.

As the algorithm is symmetrical, let us consider for instance a point T right to the center. Let F be the next key point in the direction of the center. We compute nT and nF, the position of T and F after the modification, given the current maximal possible factors (being 1 if not initialized). If nT is right to nF, there are no problems. Else, we need to consider the direction of T.

- If it is leftwards, we need to moderate its movement so that T stays right to nF.
- If it is rightwards, we need to moderate the movement of F so that it stays left to T, and to set the maximal possible factor of T to 0 so that it does not move. But the moderation of the movement of F may have caused a conflict : therefore we need to recursively consider the point F and handle the conflict.

In the implementation, using detection threshold and margins in order to maintain some dis-

tance between the key points is recommended. This algorithm is quadratic, but the low number of key points makes it fluid to use. Moreover, to improve performances, there are some cases of manipulation where this algorithm can be only locally applied (only to the boundaries for instance).

Manipulating the tangents

The geometric manipulation of key points also causes the problem of the manipulation of their tangents, which can rarely be determined analytically. Therefore, we focused on the difference between the slope of the tangent on a key point and the slope of the line joining the previous and next key point. Saving it and restoring it after the manipulation by adding it to the slope of the line joining the new position of the surrounding key points brought good results, but sometimes lead to very sharp tangents, especially around the center of the manipulation.
Therefore, we developed a mean to attenuate it. We compute a weighted average between this retrieved tangent and a fictive tangent, which is the average of the slopes of the lines joining the current point to its neighbouring points, weighted inversely proportionally by their distance on the X axis. Thereby, this fictive tangent is more influenced by sharp rises between key points, which improves the behaviour around the center of the modification. The weight in the average between the two tangents is optimally a function of the distance between the considered key point and the center of the modification : the further you are from the center, the more important the virtual tangent is.

3.6 Curve manipulations

The different manipulations possible on the curve are really the heart of the program, given that the drawing mode is very casual. With the invariants of the programs maintained, we can now focus on enhancing the user immersive experience. Note that in the picture, the huge arrow are added and don't appear in the program.

3.6.1 Visual feedback

Putting a finger down on the curve will begin a user-friendly notification of most of the possible manipulation, in a non-invasive way : the tangent and the normal in the selected point are displayed. The user has then the possibility :

1. to pull the curve around, the normal becoming a sort of stick for them to use.
2. to follow the tangent and rub the curve in order to enter a sanding manipulation (smoothing).
3. to put a finger down on the tangent and rotate it around to start a rotation.

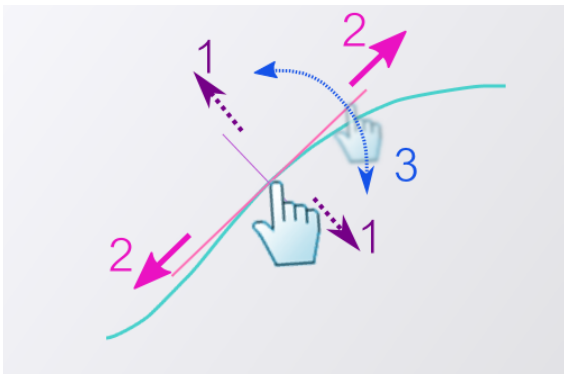


Figure 3: Possible actions are notified

They may also point another finger anywhere on the screen to determine an area of effect or beginning a translation of the whole curve.

3.6.2 Area of effect

All the manipulations have a default area of effect, which is settable in the configuration panel. It is centred around the finger. But putting down a second finger after having selected the curve will define a specific area of effect for the following manipulations. This is easily done by simply ignoring fingers outside the interval. It is advised to set the interval with one hand and manipulate in between with the other one.

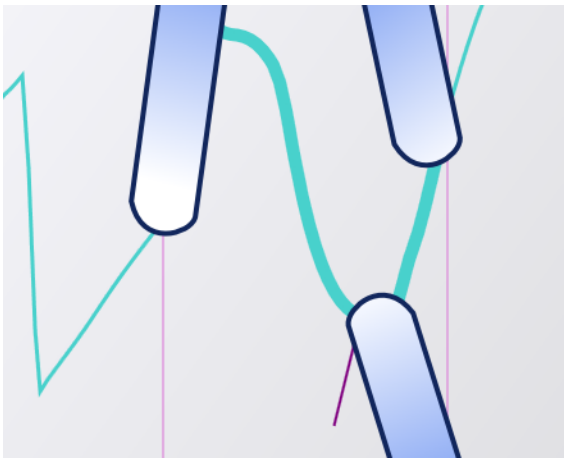


Figure 4: Setting the area of effect

For any geometrical manipulation, the transformation vectors are generally applied within the area of effect with an attenuation factor so that the manipulation is more effective in the center. A gaussian attenuation was often too rough, therefore we used most of the time a linear factor which turns out to be the smoothest and most fluid of everything we tried. Furthermore, the area of effect is always defined on the X-axis, and not geometrically centered on the finger.

3.6.3 Expertise and occlusion

Although we wanted our program to be very accessible, we also wanted to provide content for more expert users, especially in order to compensate the lack of precision caused by our sketching paradigm. Our way to do so is very simple. Once any operation has started, it is registered, and the user can slide their fingers to continue it anywhere else on the screen. They just have to do the same relative movement with their fingers as if they were actually on the manipulating point. This is a way to get more visibility about the ongoing operation by removing any occlusion, which is often annoying *(see figure 4)*, especially in the case of rotation, where several fingers are on the spot of the manipulation *(see figure 9)*. The expert can also adopt a more comfortable finger positions.

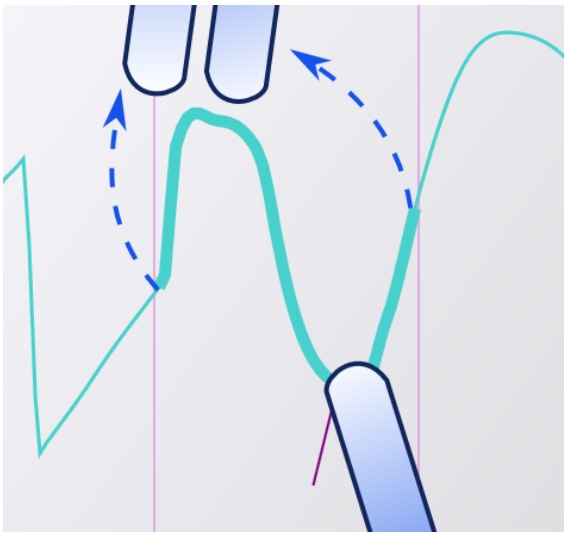


Figure 5: Without occlusion

3.6.4 Types of manipulation

Let us finally review the different manipulation featured and their theoretical meaning.

Moving

The user can move the curve around without moving the axes (translation) by putting down one or more additional fingers after having selected the curve.

Stretching

By stretching in the free space, the user can stretch a part of the curve without moving the axes. The rest of the curve is translated. The stretching can be either vertical or horizontal, and is limited by the two finger. Its direction is given by the major component of the vector formed by the two fingers, that is to say by the angle between the line joining the finger and the horizontal one.

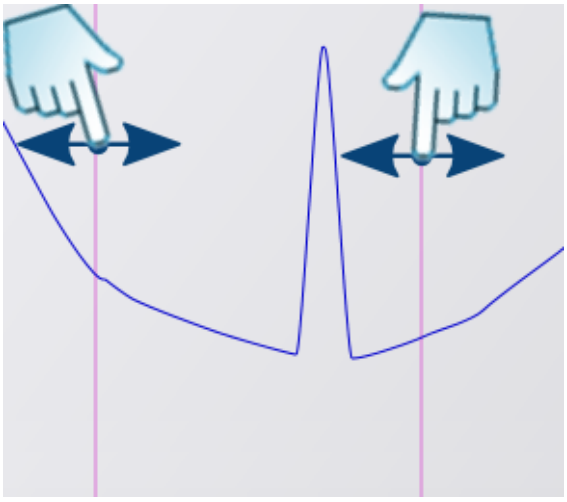


Figure 6: Stretching

Sanding

When the finger follows the tangent (that is to say the curve), it enters the sanding mode. By rubbing the curve, it will smooth it. This is simply achieved by applying a gaussian smoothing on the points hoovered by the finger, with $\sigma = \frac{Areaofeffect}{3}$, so that it covers all the zone. If the area of effect is limited and therefore not symmetrical, we use a piecewise gaussian function composed by two halves joined in the center.

Figure 7: Sanding

Rotating

The operation corresponding to the change of a tangent is simply a geometric rotation of the key points around the pinpointed point. It will of course result in a change in the tangent, but it will be much smoother and more natural, due to its area of effect. The conservation of the key points enable the keeping of the global shape of the curve, but it will also result in the creation of

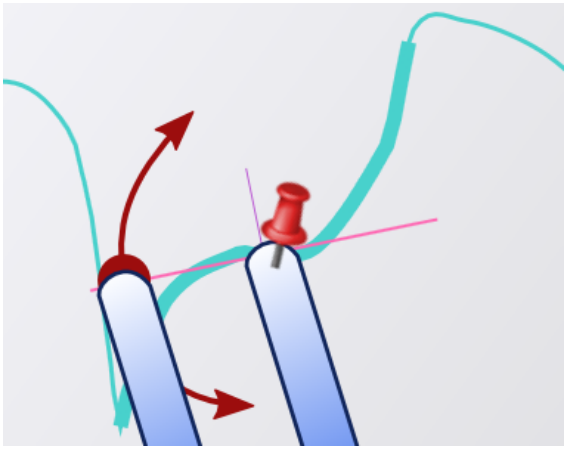


Figure 8: Rotating

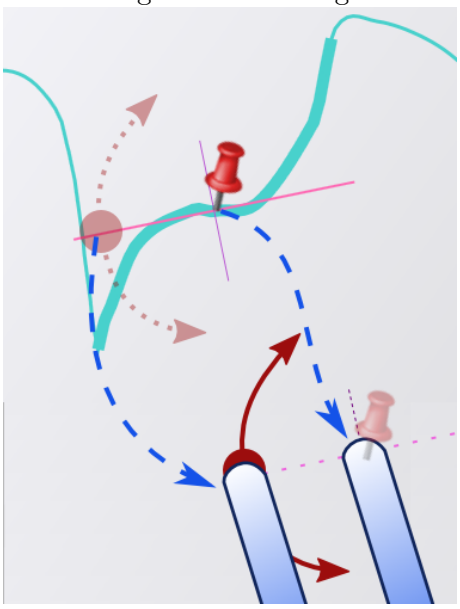


Figure 9: Rotating without occlusion

sharp slopes where key points are pushed against one another. The interpolation combined to the rotation provide here a whole new fluid interaction, which is able to go on even when there are conflicts.

Pulling

The user can pull and push the curve with the purple stick that once was the normal at the contact point. It is in fact the application of the translation vector that the finger follows, moderated by the same linear factor as the other manipulations, centred around the X-coordinate of the moving finger, wherever it is. The visual notification is a little more complex and links the fingers to the point whose X-coordinate is the middle of those of the application point (moving finger) and the extremum of the curve in the direction of the movement, in order to provide the impression that we grab the curve in its extremity to pull it. Like the sanding, this is a continuous operation.

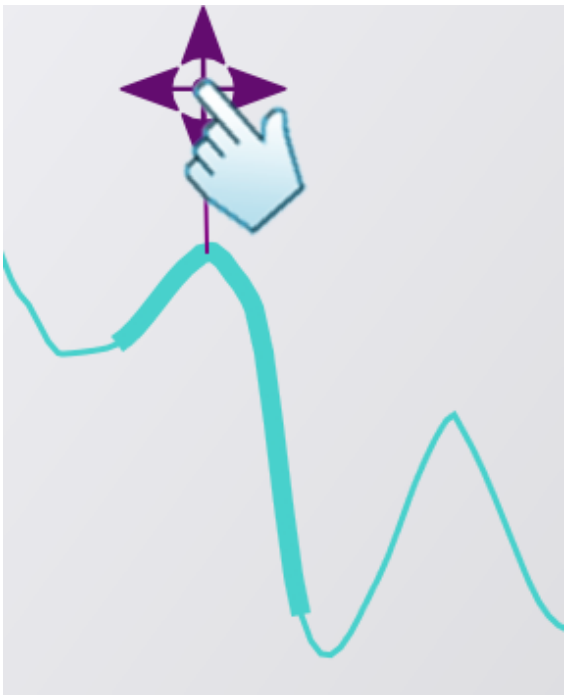


Figure 10: Pulling

Ball manipulation

The ball manipulation is a manipulation which is possible only in a limited area of effect. Indeed, when two fingers are already down, putting down another finger in the free space creates an "energy ball" which can either push or pull the curve from a distance. This very visual interaction is in fact the application of another translation vector, whose direction is given by the finger ("ball") movement, on the whole area. Contrary to the pulling operation, this one can be easily iterated several times in a row, which enables to stretch the curve by pushing it away.
The ball manipulation is by default vertically limited, that is to say vertical modifications happen only on half a plan (below or above the second finger down), and are proportional to the distance between this horizontal axis delimiting the two halves and the considered point. Therefore, the further away you are from this temporary Y-origin, the most important the effects of the manipulation are, with of course a limit factor in order to keep the behaviour reasonable.

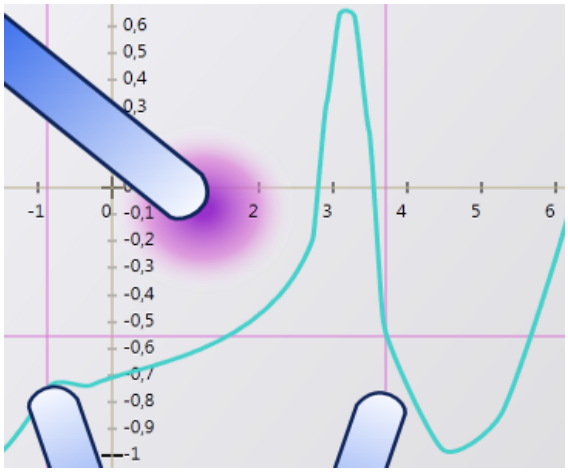
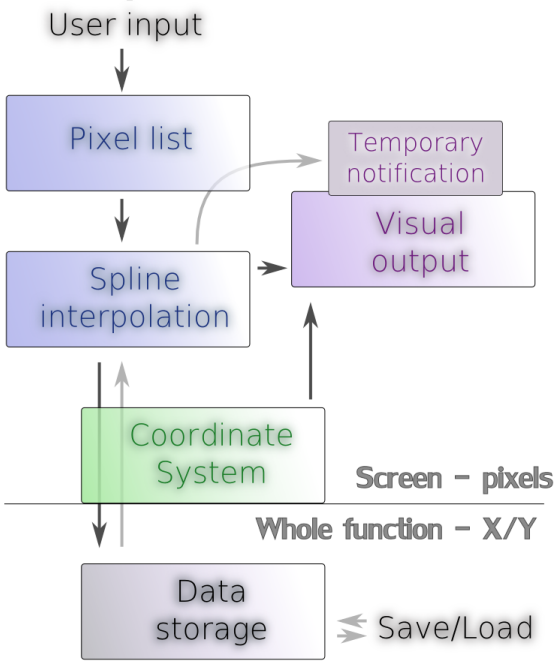


Figure 11: Ball manipulation

4 Implementation

We conducted the implementation of an embedded application using windows 7 multitouch API, and programmed in C# / WPF. The source code belongs to the INRIA laboratory LIFL, but a video displaying the various features of the program is available online [7].
Below stand a diagram underlining our architecture to answer to our constraints, especially the interpolation which acts as a buffer between the display and the storage.



5 Conclusion

This project was the occasion to develop innovative tools to manipulate curves in a new plastic way, regardless of the mathematics underneath, for anyone to use. The constraint we imposed on ourselves for our program to be user friendly and accessible brought us closer to the intuition. This trend was enhanced by our user-intent driven conception. Multitouch technology enabled a real-time immersive experience of manipulation, where the user can really connect with what they manipulate. A fixed finger pinpoints the curve, a moving finger drags it. Thereby we obtain an immediate link between the user and the computer, through the optimal use of the multitouch interface.

References

[1] Windows Touch Developer Ressources
http://code.msdn.microsoft.com/WindowsTouch

[2] Google sketch up : intuitive 3D modeling
http://sketchup.google.com/

[3] Leyton, M.
A Process Grammar for Shape, 1988

[4] Bartels, Richard H. and Beatty, John C.
A Technique for the Direct Manipulation of Spline Curves. Proceedings of Graphics Interface' 89, June 1989.

[5] Grisoni, L., Blanc, C., Schlick, C.
Hermitian b-splines. Computer Graphics Forum 18, 4 (Dec.), 237–248, 1999.

[6] Processing Approach to Fair Surface Design, G. Taubin, Siggraph'95

[7] My personal website, where a video of the resulting software can be found
http://www.YoannBourse.com

Special thanks to Damien Marchal whose help was invaluable.