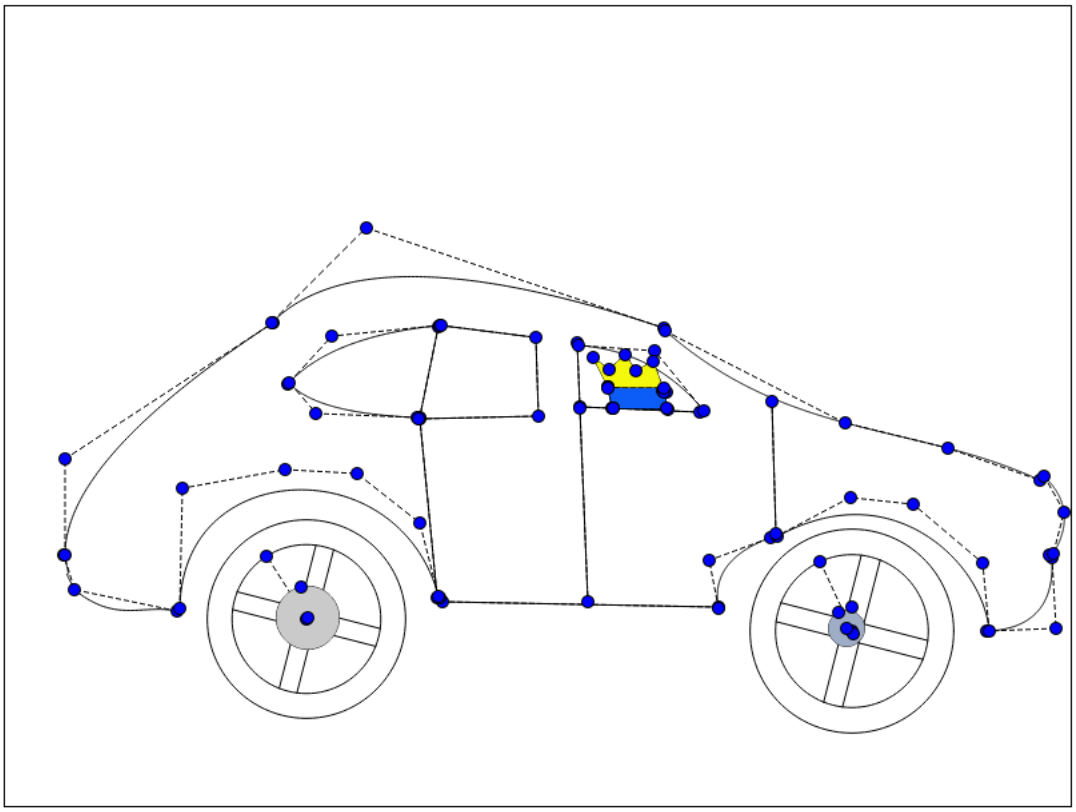
Geometric Modeling

Manuel Camargo
Mohamed Traoré
Nicolas Gindrier



Organization

We used Github. Manuel worked on parameterizations of 2D curves (Aiken Centripetal/Chordal), 1D curves and delCurve. Mohamed made 1D curves (sinus) and 2D curves (Lagrange/smooth) and Nicolas did essentially 2D curves (Hermite1-closed, Bezier and Gear, Wheel, even if there are not curves).

Curves

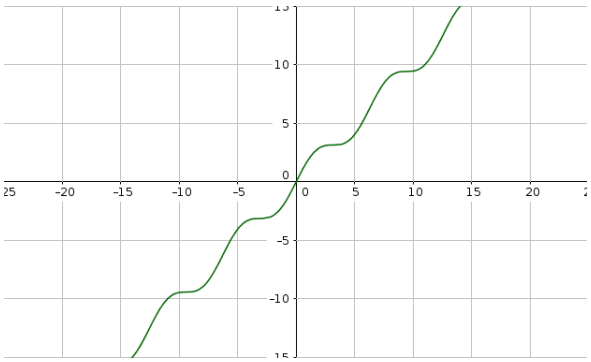
Curves 1-D

Step curve

This is a very basic curve to familiarize ourselves with the concept of 1-D curves: the Path function, draws in the animation Widget the way our points should move during an animation, while the EvalAt function returns the value of the points a different moments in time following the Path.

Sinusoidal

The idea comes from the fact that the function $t \longmapsto t + \sin t$ has an interessing form as shown on the following figure. It will help us to do some kinds of movement.



Now, the goal is to find a function like the previous sinusoidal function which connects two given points $A(t_A; x_A)$ and $B(t_B; x_B)$. And after looking for it, we find the function defined by :

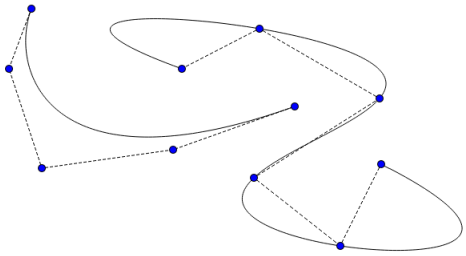
$$t \longmapsto x_A + (x_B - x_A)\frac{t - t_A}{t_B - t_A} + \Delta \sin(\Omega(t - t_A)) \sin(\Omega(t_B - t))$$

Where Δ is the amplitude and Ω is the pulsation in terms of signal.
This function is a sinusoidal function and we can also remark that $f(t_A) = x_A$ and $f(t_B) = x_B$.

Curves 2-D

Aitken

A recursive function on the control points is called. We store points in a static tab and we linked these points after. It is also used for *Hermite1* and *Hermiteclosed*. For *Aitken* we use a uniform parameterization.



$$x(t) = \sum_{i=0}^{n} \mathbf{b}_i B_i^n(t)$$

where the $\mathbf{b}_i$ are the control points and the B_i^n are the Bernstein polynoms defined as follows:

$$\forall t \in [0\,,1] \quad B_i^n(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

A change of variable $t = \frac{u-a}{b-a}$ might be needed if our parameters are in $[a\,,b]$ instead of $[0\,,1]$.
For a given t, we can compute the value of the curve using the De Casteljau algorithm based on the following theorem:

$$Let \quad \begin{cases} \mathrm{b}_i^0 = \mathbf{b}_i \quad \forall i \in [\![0;n]\!] \\ \mathrm{b}_i^k = (1-t)b_i^{k-1} + t\, b_{i+1}^{k-1} \quad \forall k \in [\![1,n]\!], \quad \forall i \in [\![0,n-k]\!] \end{cases} \qquad Then \quad x(t) = b_0^n$$

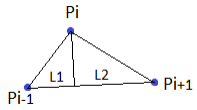
As said previously, the algorithm was implemented recursively.

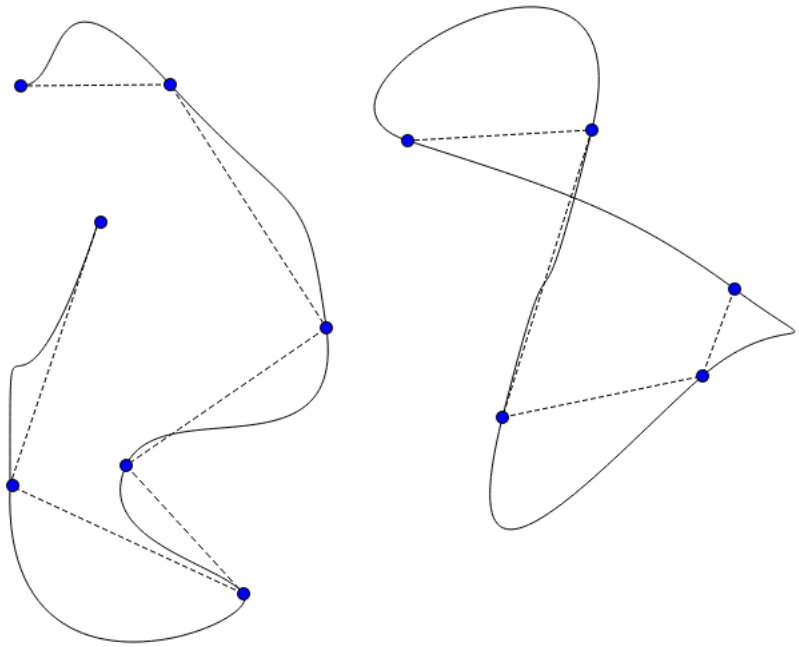
Hermite1-Hermite closed

These curves are made with some Hermite cubic splines, based on P1, P2 and their tangent. That allows to have C^1 closed curves. The difficulty is to impose good direction, weight and sense for tangents.
The general formula is :

$$\begin{cases} D_{xi} = lL_1 + mL_2 \\ D_{yi} = lL_1 \dfrac{Y_{i+1} - Y_i}{X_{i+1} - X_i} + mL_2 \dfrac{Y_i - Y_{i-1}}{X_i - X_{i-1}} \end{cases} \tag{1}$$

We adapt it for the first and the last point. Moreover $\dfrac{Y_{i+1} - Y_i}{X_{i+1} - X_i}$ is bounded.
l and m are equal to 1 or -1 according to tests of the form $X_{i+1} - X_i$, that changes the sense of the curve. L_1 and L_2 are the weight of derivatives, they are a projection. Actually this is a weighted average.

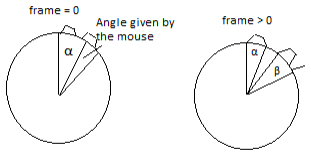




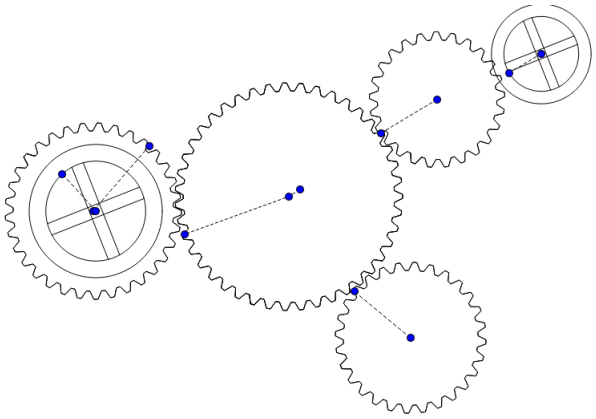
Wheel-Gear

It is not really curves. The goal here is to play with the variable *frame*. We are midway between cartoon and CAD.
In both cases *frame* is traduced as rotation speed.
For *Wheel* we draw two circles. We use the points of the little circle in scrolling the array according to *frame*. To draw a radius we link a point of the array at the size(array)/2 point. In playing with that and modulo we can draw a rolling wheel. For gear we implicitely use a rotation matrix. We have three

angles (see scheme) : α, to draw the gear, β, to make rolling the gear, depending on frame and an angle depending on the mouse position (it is for follow the mouse and engage another gear, the absolute value of this angle is 2 $arccos(d(\frac{mouse, center}{2R}))$These three angles are added. β is computed such that the angular speed of whatever gear be the same (β is inversely proportional to the radius).
) Actually there is a repetition of 4 points, at every rotation we choose to add or substract at the radius the half-size of a teeth. Besides, the radius is computed to have an whole number of teeth.



So the user can drow a gear in clicking one or two times to chose the sense of rotation (an odd number for clockwise, otherwise counter-clockwise) then right-click for the radius (in fact almost the radius as above explained).

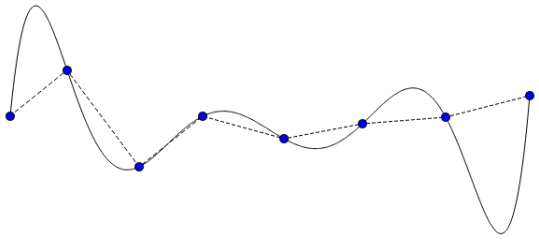


Lagrange interpolation

The aim of this part is to find a polynomial function with degree n that interpolates $(n+1)$ different given points. By using Lagrange method, the polynomial function that we are looking for is defined by :

$$L(x) = \sum_{i=0}^{n} y_i \left(\prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j} \right)$$

where $(x_i; y_i)$ are the coordinates of the i^{th} point.
We can see an exemple of Lagrange interpolation on the figure below.



Least square approximation

In this part we have to find a polynomial curve of degree p that approximates $(n+1)$ given points. We know that if $p = n$, we are in the case of parametrical interpolation which is done by *Aitken* method. So we will suppose that $p < n$.

Let $A(x) = \sum_{i=0}^{p} a_i x^i$ be the polynomial curve of degree p that we need.

$$\forall\ 0 \leq q \leq n, A(x_q) = y_q + \epsilon_q \iff \begin{cases} A(x_0) &= y_0 + \epsilon_0 \\ \vdots & \vdots \\ A(x_j) &= y_j + \epsilon_j \\ \vdots & \vdots \\ A(x_n) &= y_n + \epsilon_n \end{cases} \iff \begin{cases} \sum_{i=0}^{p} a_i x_0^i &= y_0 + \epsilon_0 \\ \vdots & \vdots \\ \sum_{i=0}^{p} a_i x_j^i &= y_j + \epsilon_j \\ \vdots & \vdots \\ \sum_{i=0}^{p} a_i x_n^i &= y_n + \epsilon_n \end{cases}$$

$$\iff \begin{pmatrix} 1 & \cdots & x_0^j & \cdots & x_0^p \\ \vdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \cdots & x_k^j & \cdots & x_k^p \\ \vdots & \cdots & \cdots & \cdots & \cdots \\ 1 & \cdots & x_n^j & \cdots & x_n^p \end{pmatrix} \begin{pmatrix} a_0 \\ \vdots \\ a_j \\ \vdots \\ a_p \end{pmatrix} = \begin{pmatrix} y_0 \\ \vdots \\ y_k \\ \vdots \\ y_n \end{pmatrix} + \begin{pmatrix} \epsilon_0 \\ \vdots \\ \epsilon_k \\ \vdots \\ \epsilon_n \end{pmatrix}$$

$$\iff Au = y + \epsilon$$

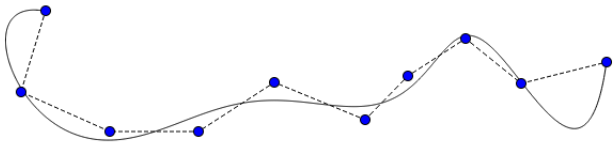
Now, we want to minimize the euclidian norm of errors ϵ.

$$\|\epsilon\|^2 = \|Au - y\|^2$$

We also need an important result which is :

$$({}^tAA)u = {}^tAy \iff u = ({}^tAA)^{-1}\,({}^tAy)$$

The following figure is the approximation of 10 points by a polynomial of degree 8 using least square method.



others

An extra button was added to the tools Widget: "Del Curve". In Edit mode, it deletes the curve of the selected Point. It was implemented by tracing the code of the other Edit buttons in the code architecture. There are two main steps: adding the button in the tool Widget and binding it to an action when the mouse is clicked, and modifying the scene and the drawing Widget in such a way than when this button is used, a whole curve will be deleted (a function to delete a selected curve already existed).

Prospects

We had a lot of ideas but some of them have not been made. It was because of time, or because we fear to modify some files. We do not master the architecture of the whole project. For example matstering

scene.h would allowed many things, like to save. Other ideas that were started but never finished were the following:

- A button Del All, which cleared the drawin Widget.
  The steps to follow where the same as Del Curve, but there was no pre-existing function that eliminated all curves, and we couldn't make it work. Same for a function that would permit to move an entire curve, or to resize it.

- Buttons in the Create Widget that let you choose a type of parametrization, if it applies to the selected curve.
  Instead of changing entirely the way Curves 2D worked, there was maybe a way to select 'hidden curves' (curves that were implemented but that were not added to the create Widget) when clicking these parametrization buttons. The same principle could be used to make already implemented curves that translated or rotated (by calling 'hidden' curves that depended on the *frame*).