

Overview of Computerized Font

Elliot Etches

December 10, 2021

1 Bitmap Fonts

When we first started putting words on electronic displays we really just needed a quick and dirty solution, enter - raster displays. These divided a screen into a grid of pixels, displaying images by illuminating a portion of them. Any time the image needed to change the pixels were scanned over a row at a time. Done quickly enough and the transition between images was imperceivable to the human eye.

The data structure that represented the specific one-to-one pixel value for a raster image came to known as a “bitmap.” The early CRT monitors were monochrome and therefore needed only one bit per pixel. As displays became better able to present color the amount of bits that needed to be assigned to each pixel increased to hold this new color data.

Bitmaps are essentially a matrix where each entry contains a certain integer which correlates to some color. The individual pixels to which this matrix is applied are discrete and identical in size. The discrete nature of the images presents a problem when continuous shapes, such as letters, are translated into the raster’s discrete form. This conversion process, known as “sampling”¹ presents a number of hurdles to overcome.

Sampling involves the loss of image information so you’re always going to run into some problems. The most notable of these goes by the technical name of “aliasing” and it’s responsible for the jagged edges on what are supposed to be smoothed edges. Now, with enough pixels at the right size you can conceal these imperfections. However, if you do something like enlarge the image you’ll start noticing them real quick. Well this might not be that big a deal if you’re just inputting data into a terminal it does become a problem if you try to do something a bit more graphically involved.

¹From what I briefly read, sampling seems like a super cool topic to investigate. It veered a bit too far away from geometry here, but I’d love to investigate it later.

2 Vector Fonts

Vector scalable graphics were developed out of the newly pioneered fields of computer assisted design and drafting. They combated the scaling issues by encoding images as some combination of vectors as opposed to bitmaps. This allowed the recreation of an image by following the steps outlined in the image file. The instructions themselves could be implemented at any image size so our scaling problem disappeared. Unfortunately, there's a considerable amount of computer overhead that comes with this compared to displaying raster images². This has become less of an issue as processors improved and vector formats are extremely ubiquitous today.

At time of writing we have two vector font standards: The joint Apple/Microsoft TrueType and and Adobe Type 1 fonts used in Postscript. Both make use of Bézier curves at different degrees. TrueType uses quadratic Bézier curves well Postscript uses cubic. It's possible to use higher degree Bézier curves to increase fidelity at the cost of additional computing overhead.

Side Note: Another really awesome use of these vector fonts is in the construction of East Asian characters which often take the form of logograms. These languages often contain a *lot* of individual characters (+100,000 in traditional Chinese / \sim 3500 in regular modern usage). This massive quantity of characters has often prevented modern bitmap based fonts from supporting them effectively. However many logograms are combinations of other logograms. Vector fonts are more easily able to reference previously defined characters and use them to construct new characters which greatly reduces the amount of data that needs to be stored.

2.1 General Bézier Curves

The initial construction of Bézier Curves is actually quite simple. Generally they take the form of

$$\sum_i^n \binom{n}{i} * (1-t)^{n-i} t^i * w_i$$

Which I initially found to be a little terrifying. Thankfully for typography purposes we really need to only concern ourselves with w . The rest of the variables are as follows:

- n is the degree of the curve. For the quadratic TrueType this is $n = 2$ well the cubic Type 1 has $n = 3$. Curves for $n = 1$ produce straight lines between two points. This quite useful for all the straight lines in

²The display of raster images can often be outsourced to the control unit for the display - so it's about as fast as you can get!

Latin letters A higher degree Bézier Curve can be used to represent Bézier Curves of all degrees lower than it. So we can always create straight lines with quadratic and cubic curves³.

- t is usually bounded between 0 and 1. As we move from 0 to 1 we map out all the points that comprise the curve. I like to think of it as time in the same way a recording of a curve being drawn can be paused at any moment.
- w are the weights of the curve and the number of them is equal to the degree of the curve. They each correspond to a particular Cartesian point and moving these points around is how we modify the shape of the curve.

With this general form out of the way we can move onto the much more digestible quadratic and cubic forms ($n = 2, n = 3$)

2.2 The Bézier Curves We're Interested In

The weights (w) that were mentioned previously are really important to what we're doing. So instead of unceremoniously referring to them as w_1, w_2, \dots we're going to give them the much more fitting letters A, B, C, \dots

The quadratic curve takes the form:

$$A(1-t)^2 + 2Bt(1-t) + Ct^2$$

Now, this will only give us one number and we need two to plot it on a Cartesian plane. We end up using the same function but separate it into x and y components. Remember A, B, \dots are all coordinate points so they each have an x and y component.

$$\begin{cases} x & A_x(1-t)^2 + 2B_x t(1-t) + C_x t^2 \\ y & A_y(1-t)^2 + 2B_y t(1-t) + C_y t^2 \end{cases}$$

Matrices!: Now I've never taken a linear algebra class but the later assignments have demonstrated to me that representing coordinate geometry using matrices often makes things surprisingly easy. Under this vein of reasoning I was pleasantly surprised to find a matrix representation! Our previous cubic function can be presented as:

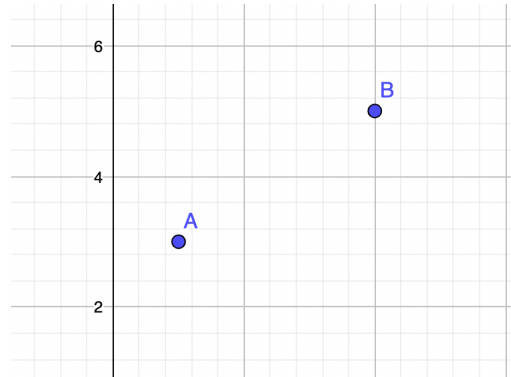
$$(1 \quad t \quad t^2) * \begin{pmatrix} 1 & 0 & 0 \\ -2 & -2 & 0 \\ 1 & -2 & 1 \end{pmatrix} * \begin{pmatrix} A \\ B \\ C \end{pmatrix}$$

³This also means that cubic curves can represent quadratic curves without losing any information. So Postscript fonts can display TrueType fonts without loss of fidelity, but not the other way around.





2.3 De Casteljau's Algorithm - Curves in GeoGebra

All these equations are lovely and all, but it would be fantastic if we could actually construct these curves in GeoGebra. Thankfully, we can leverage De Casteljau's Algorithm to do just that

We begin with two points



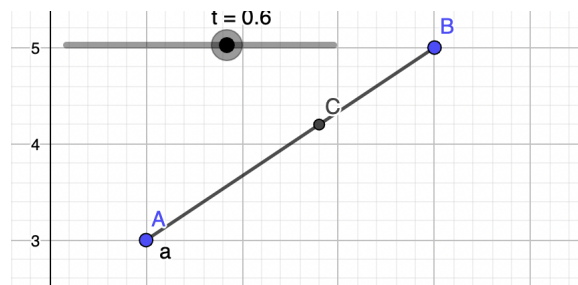
We then can create a parametric equation a connecting these two points.

| | | |
|---|---|---|
|  | $A = (1, 3)$ |  |
|  | $B = (4, 5)$ | \vdots |
|  | $a = \text{Curve}(x(A) + (x(B) - x(A)) v, y(A) - (y(A) - y(B)) v, v, 0, 1)$ $\rightarrow \left. \begin{array}{l} x = 1 + (4 - 1) v \\ y = 3 - (3 - 5) v \end{array} \right\} 0 \leq v \leq 1$ | \vdots |

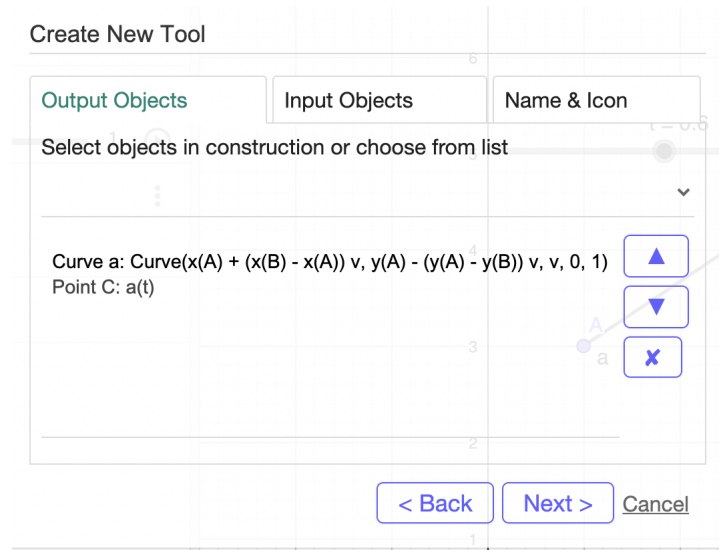
We can then create a slider t that travels along a .

| | | |
|----------------------------------|--|---|
| <input type="radio"/> | A = (1, 3) | |
| <input type="radio"/> | B = (4, 5) | ⋮ |
| <input type="radio"/> | $a = \text{Curve}(x(A) + (x(B) - x(A)) v, y(A) - (y(A) - y(B)) v, v, 0, 1)$ $\rightarrow \left. \begin{array}{l} x = 1 + (4 - 1) v \\ y = 3 - (3 - 5) v \end{array} \right\} 0 \leq v \leq 1$ | ⋮ |
| <input checked="" type="radio"/> | t = 0.6 0 1 | ⋮ |
| <input type="radio"/> | C = a(t) $\rightarrow (2.8, 4.2)$ | ⋮ |

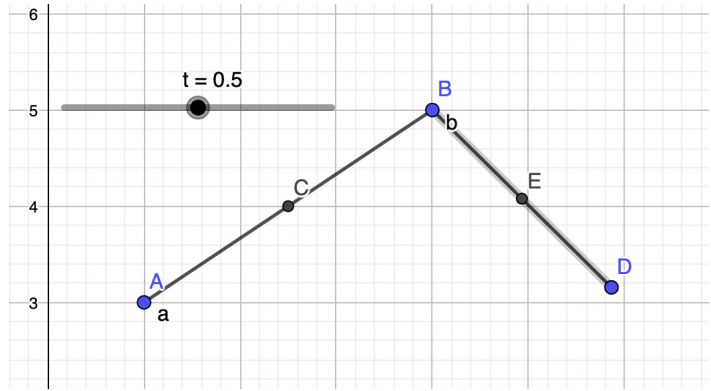
At this point, if we trace the path of C we would have a Bézier Curve of degree one which, as mentioned previously, is linear.



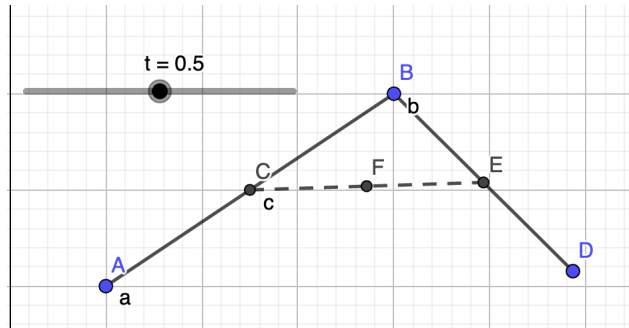
This construction will be the basis for everything else. We can then make our lives easier by making a function that takes A, B and t as inputs and produces C as our output.



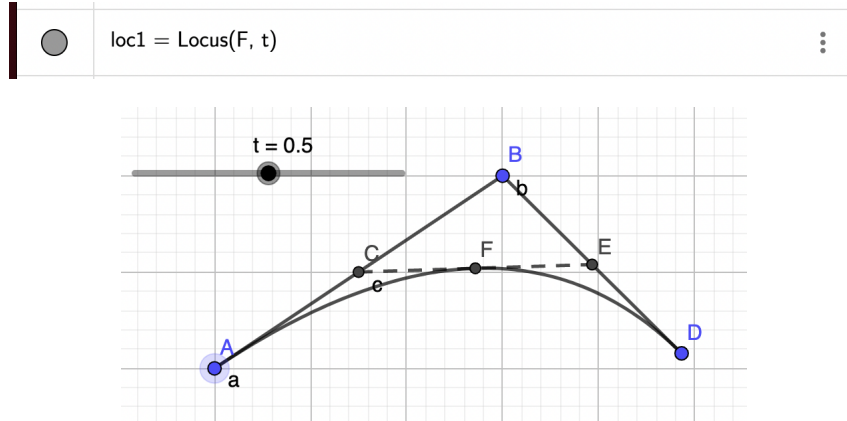
To move from linear curves to quadratic we'll need to add another point D and use our new function on B, D and t to create E .



We can then use our function on C, E and t to create F



We can then use the Locus function to trace the path of F as t moves from 0 to 1



We're left with our quadratic Bézier Curve!

Side Note: Unless you're outputting your curves to a planner or oscilloscope it's likely you won't be able to completely escape raster images. Most consumer monitors display images in a rasterized format. This brings us right back to the sampling problem mentioned earlier! The process of mitigating these sampling issues in vector fonts is called hinting and it is big business. Hinting algorithms are kept under extreme lock and key by Microsoft, Apple and Adobe. These proprietary hinting algorithms have been a big hurdle for free and open source implementations of the standards.

2.4 TrueTypes and Derivatives

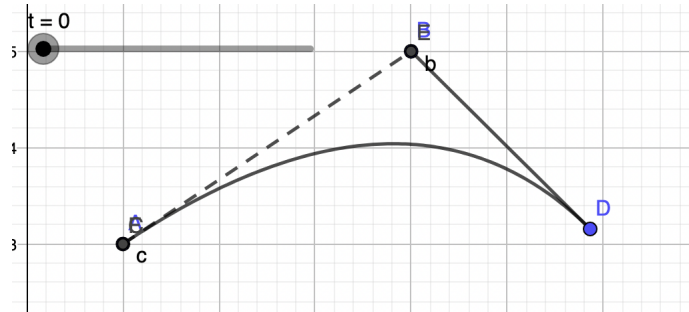
The first derivative of our quadratic Bézier Curve with respect to t is

$$2(1 - t)(B - A) + 2t(C - B)$$

In this state we can easily see that when $t = 0$ we get

$$2(1)(B - A) + 2(0)(C - B) = 2(B - A)$$

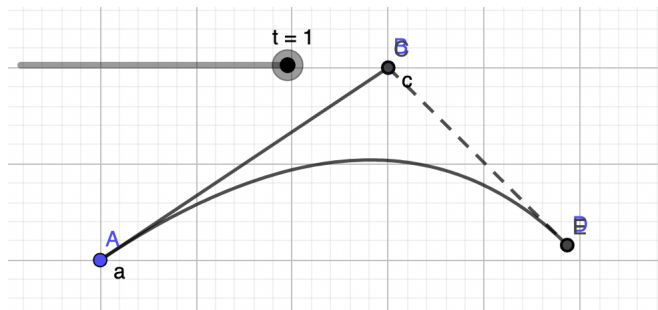
At $t = 0$ our curve is at B and we now know that the slope of our tangent line at B is $2(B - A)$. So we know that the tangent line of the A end of our curve must intersect with B .



Now let's check out the other end at C , when $t = 1$:

$$2(0)(B - A) + 2(1)(C - A) = 2(C - B)$$

So, similarly, our tangent line at the C end of the curve must also intersect with B .

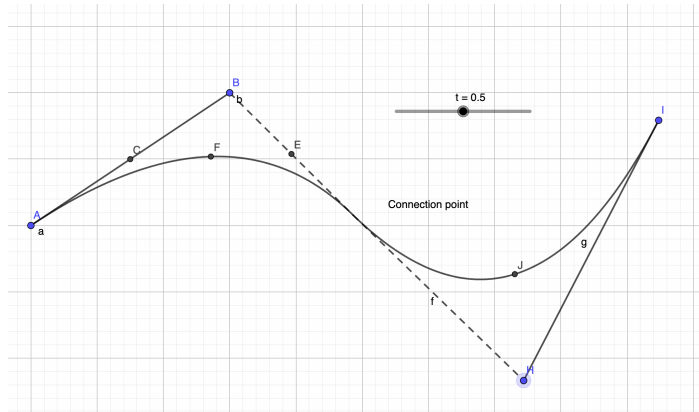


This may seem kind of silly to bring up now, but the Truetype documentation defines the curves in this way for a particular reason.

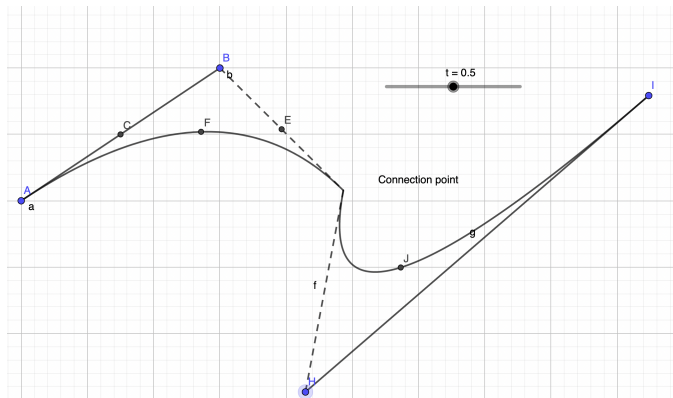
2.5 Bézier Spines - Connecting Our Curves

These Béziers are great if we want to make a single smooth looking curve. However, quite a few characters are formed from more than just a single curve which presents a bit of a problem. Thankfully we can readily solve this problem by just adding *more curves*. When we connect two or more Bézier Curves together we get a Bézier Spine.

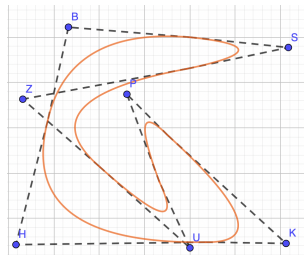
Now for the twenty cent question: "How do we connect these curves?" Well it relies on the tangent fact we mentioned earlier. If we want to connect to curves A, B, C and D, E, F at the the points C and D we need B, C, D , and E to be collinear. Since we're connecting the points at C and D we can safely assume $C = D$ so we can get away with demanding B, C, E be collinear.



So, going off the derivatives we covered previously we're looking for a situation where $2(C - B) = 2(B - A)$. If we don't hold this as a requirement the curves will still "connect" but they'll look jagged, losing the smoothness that makes them so interesting.



Using these features we can begin to construct our own characters. I present a hideous⁴ "G" constructed from the GeoGebra steps covered earlier:



Of course real type faces, unless heavily inspired by a certain 60s aesthetic, make ample use of linear equations to connect their Bézier curves. Regressing back to the straight lines seemed inappropriate in this context however!

⁴I don't think I'd make it as a graphic designer.

3 Conclusions

I hope this brief outline of Raster and Vector fonts was at least somewhat informative. Bézier curves are way more fascinating than I first thought and the field of typography includes a ton of interesting tangents to explore. It has everything from the historical roots of the typefaces to incredibly advanced sampling algorithms used today. There seems to be much more to develop and improve upon. This goes especially for the design of certain East Asian character sets as well as the development of better free and open source vector rendering engines.

References

- [1] Pomax, “A Primer on Bézier Curves”
- [2] James D. Foley, Foley van Dam, Steven K. Van Dam, Steven K. Feiner, John F. Hughes, J. Hughes “Computer Graphics: Principles and Practice”
- [3] Apple, “Digitizing Letterform Designs”, *TrueType Reference Manual*
- [4] TYPE*chimérique, “TrueType Outlines”
- [5] X. Li, J. Xue, “Complex Quadratic Bézier Curve on Unit Circle”
- [6] Steve Klassen, “GeoGebra Bézier Curve Construction”
- [7] Turksvids, “GeoGebra Parametric Equations - Using The Curve For a Line Segment”