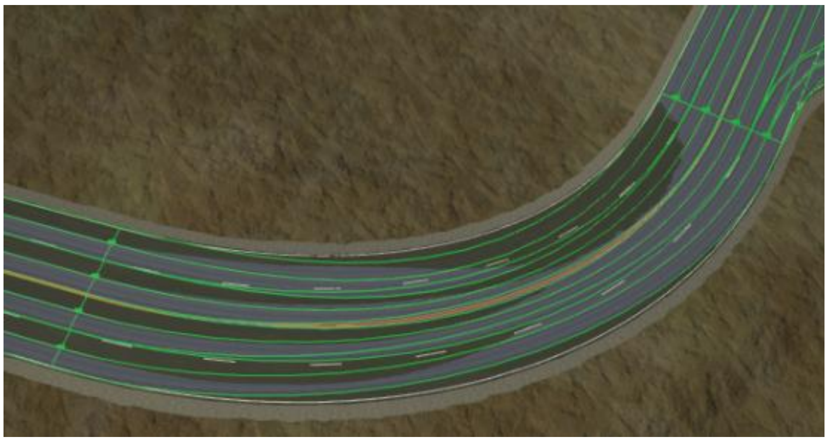
Computing Offset Curves for Cubic Splines

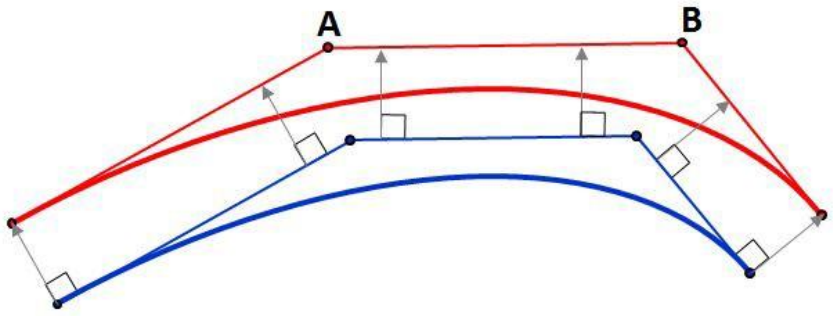
I recently had to come up with a way to create parallel curves from cubic Hermite splines, like railroad lanes. At first, I just displaced their start/end control points along their normal directions, while keeping the same start/end tangents. It worked fine for most cases because most splines in my use cases are not very curved.

But I soon discovered curves replicated in such a naive way often result in narrowing even with some mildly curved splines. So, I decided to bite the bullet and looked into what other intelligent people have already figured out on it by googling.


Narrowing from the naive approach

I quickly learned that an official term for such a curve is *offset curve* or *parallel curve*. The first approach I bumped into was to convert a curve into a poly-line, offset the poly-line, and then reconstruct it into a curve or curves: http://stackoverflow.com/a/3220819/900762. It sounded complicated and I wanted to find a more direct approach where I don’t need to go through the poly-line phase.

The next approach I found was offsetting its control polygon (http://math.stackexchange.com/a/467038 and http://brunoimbrizi.com/unbox/2015/03/offset-curve). You can easily convert cubic Hermite splines to Bézier curves as http://www.joshbarczak.com/blog/?p=730 explains. In a Bézier form, you get a control polygon from four control points. You can offset it as below and get an offset spline from it:



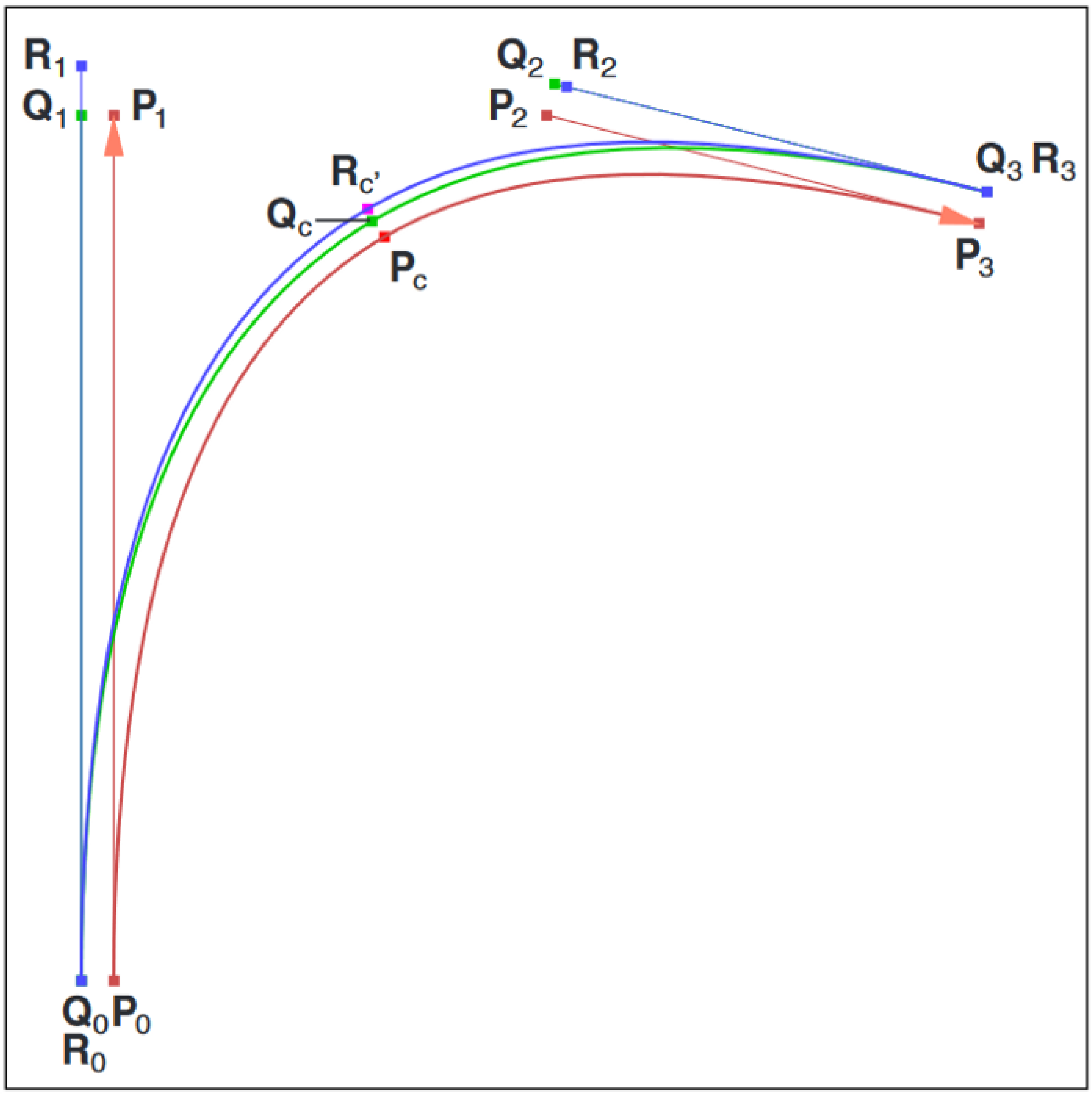
An image from http://math.stackexchange.com/a/467038

As both of the links above mention, the approach won’t work for curves with high curvatures and you need to split the original curve to multiple segments to get a better result in such cases. I didn’t think splitting / subdividing would be necessary for my use cases and decided to give the approach a try.

I had 3D splines, so I used the distance equation between skewed lines like http://2000clicks.com/mathhelp/GeometryPointsAndLines3D.aspx to get intersections between offsetted edges of the control polygon. My splines also have just one tangent for each control point. In other words, no separate incoming / outgoing tangents. So, once I calculate offset splines for neighboring segments, I set the tangent of the shared control point to an average of the end tangent of the first segment and the start tangent of the second segment.

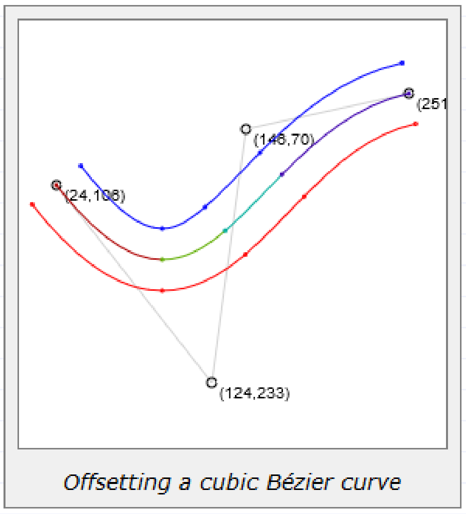
The approach provided parallel curves looking good enough for the most splines I have. But offset curves for some went bananas when their tangents are long and their control polygons self-intersect. Also, for some other cases, they kind of overcompensated and got wider than they should be. So, I introduced some tweaks like falling back to the naive way when internal angles of the control polygon are too acute and *lerp*ing between the results of the naive and this approach based on the cosines of the internal angles to avoid becoming too wide in some cases. There were still some rare cases this tweaked approach fails. For them, I could have applied splitting before this approach, but, for now, I decided to manually modify the original splines so that they can behave better with the approach without an automatic splitting. I think having a single tangent per control point (no separate incoming / outgoing tangents) also made some of these cases worse than it could be.

As you can see, it wasn't totally satisfactory. It required some not-so-clean tweaks and still could not handle all cases. I dug some more in my spare time and found out some useful info and more approaches worth trying in the future. This archived PDF document http://web.archive.org/web/20061202151511/http://www.fho-emden.de/~hoffmann/bezier18122002.pdf explains getting Bézier offset curves by solving a system of three linear equations, starting from the page 34. It starts with the naive approach and compute how the resulting curve should be modified to satisfy the properties of an ideal offset curve by solving a system of linear equations with three unknowns. The approach also has some failure cases and the document explains a way to handle them by fixing one of the unknowns and just using two linear equations. It has sample code in Postscript.



Captured from http://web.archive.org/web/20061202151511/http://www.fho-emden.de/~hoffmann/bezier18122002.pdf

Many references tell you that, in general, one cannot express offset curves of a Bézier curve using another Bézier curve (or any polynomial parametric curve, for that matter, regardless of however higher-order you are willing to go). Page 28 of https://www.slideshare.net/Mark_Kilgard/22pathrenderalso says that an offset curve of an arbitrary cubic curve requires a 10th-order *algebraic* curve. A very nice and detailed primer on Bézier curves at https://pomax.github.io/bezierinfo/ has all sorts of useful info about them, with interactive and visual guides. Its section for offset curves explains another approach that involves splitting. With it, you "chop up a curve into 'safe' sub-curves (where safe means that all the control points are always on a single side of the baseline, and the midpoint of the curve at t=0.5 is roughly in the centre of the polygon defined by the curve coordinates) and then point-scale each sub-curve with respect to its scaling origin (which is the intersection of the point normals at the start and end points)." It has all the relevant codes in https://pomax.github.io/bezierjs/, so you can easily check its implementation details.



Captured from https://pomax.github.io/bezierinfo/#offsetting

The two alternatives mentioned above sound sensible and look not too hard to implement. I'm going to try them when the current approach of offsetting the control polygon becomes too fragile and cumbersome for my use cases.

This short research of mine was quite a fun and educational ride for me and demonstrated nuances and intricacies involved in a seemingly simple question of how you can get *parallel* versions of a spline curve. I hope it has be a fun and of help to you, too!

References

1. Cubic Hermite spline: https://en.wikipedia.org/wiki/Cubic_Hermite_spline
2. bezier path widening: http://stackoverflow.com/a/3220819/900762
3. Control points of offset bezier curve: http://math.stackexchange.com/a/467038
4. How to Draw an Offset Curve: http://brunoimbrizi.com/unbox/2015/03/offset-curve/
5. Splines Are Just Obfuscated Beziers: http://www.joshbarczak.com/blog/?p=730
6. Equation of Lines in 3D — distance between two skew lines, and closest points: http://2000clicks.com/mathhelp/GeometryPointsAndLines3D.aspx
7. Bézier Curves by Gernot Hoffmann: http://web.archive.org/web/20061202151511/http://www.fho-emden.de/~hoffmann/bezier18122002.pdf
8. CS 354 Vector Graphics & Path Rendering: https://www.slideshare.net/Mark_Kilgard/22pathrender
9. Algebraic curve: https://en.wikipedia.org/wiki/Algebraic_curve
10. A Primer on Bézier Curves: https://pomax.github.io/bezierinfo/
11. A Primer on Bézier Curves | Curve offsetting: https://pomax.github.io/bezierinfo/#offsetting
12. Bezier.js: https://pomax.github.io/bezierjs/
13. A StackOverflow entry at http://stackoverflow.com/questions/4148831/how-to-offset-a-cubic-bezier-curve has some more useful links.
14. Parallel curve: https://en.wikipedia.org/wiki/Parallel_curve
15. Offsets (a.k.a stroking) of Bézier curves: https://en.wikipedia.org/wiki/B%C3%A9zier_curve#Offsets_.28a.k.a._stroking.29_of_B.C3.A9zier_curves