Bezier clipping

Asked 16 years, 3 months ago Modified 2 years, 3 months ago

Viewed 11k times

16

I'm trying to find/make an algorithm to compute the intersection (a new filled object) of two arbitrary filled 2D objects. The objects are defined using either lines or cubic beziers and may have holes or self-intersect. I'm aware of several existing algorithms doing the same with polygons, listed here. However, I'd like to support beziers without subdividing them into polygons, and the output should have roughly the same control points as the input in areas where there are no intersections.

This is for an interactive program to do some CSG but the clipping doesn't need to be real-time. I've searched for a while but haven't found good starting points.

graphics geometry 2d bezier csg

Share

Improve this question

Follow

edited Sep 20, 2008 at 23:15

Nils Pipenbrinck

86.2k ● 33 ● 155 ● 223

asked Sep 20, 2008 at 20:56

jjrv

4,335 ● 3 ● 43 ● 55

4 Answers

Sorted by: Highest score (default)

12

I found the following publication to be the best of information regarding Bezier Clipping:

T. W. Sederberg, BYU, Computer Aided Geometric Design Course Notes

Chapter 7 that talks about Curve Intersection is available online. It outlines 4 different approaches to find intersections and describes Bezier Clipping in detail:

https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=1000&context=facpub

Share Improve this answer

Follow

edited Sep 5, 2022 at 19:36

David Jones

5,119 ● 4 ● 35 ● 47

answered Jun 9, 2010 at 11:35

lehni

136 ● 1 ● 2

2 The link is outdated. – Jeff T. Feb 25, 2019 at 4:11

Newer link: scholarsarchive.byu.edu/cgi/… – Pranav Totla Feb 1, 2021 at 5:14

6

I know I'm at risk of being redundant, but I was investigating the same issue and found a solution that I'd

read in academic papers but hadn't found a working solution for.

You can rewrite the bezier curves as a set of two bi-variate cubic equations like this:

- $\Delta x = ax_1*t_1\text{^}3 + bx_1*t_1\text{^}2 + cx_1*t_1 + dx_1 - ax_2*t_2\text{^}3 - bx_2*t_2\text{^}2 - cx_2*t_2 - dx_2$
- $\Delta y = ay_1*t_1\text{^}3 + by_1*t_1\text{^}2 + cy_1*t_1 + dy_1 - ay_2*t_2\text{^}3 - by_2*t_2\text{^}2 - cy_2*t_2 - dy_2$

Obviously, the curves intersect for values of (t_1,t_2) where $\Delta x = \Delta y = 0$. Unfortunately, it's complicated by the fact that it is difficult to find roots in two dimensions, and approximate approaches tend to (as another writer put it) blow up.

But if you're using integers or rational numbers for your control points, then you can use **Groebner bases** to rewrite your bi-variate order-3 polynomials into a (possibly-up-to-order-9-thus-your-nine-possible-intersections) monovariate polynomial. After that you just need to find your roots (for, say t_2) in one dimension, and plug your results back into one of your original equations to find the other dimension.

Burchburger has a layman-friendly introduction to Groebner Bases called "**Gröbner Bases: A Short Introduction for Systems Theorists**" that was very helpful for me. Google it. The other paper that was helpful was one called "**Fast, precise flattening of cubic Bézier path and offset curves**" by TF Hain, which has lots of

utility equations for bezier curves, including how to find the polynomial coefficients for the x and y equations.

As for whether the Bezier clipping will help with this particular method, I doubt it, but bezier clipping is a method for narrowing down where intersections might be, not for finding a final (though possibly approximate) answer of where it is. A lot of time with this method will be spent in finding the mono-variate equation, and that task doesn't get any easier with clipping. Finding the roots is by comparison trivial.

However, one of the advantages of this method is that it doesn't depend on recursively subdividing the curve, and the problem becomes a simple one-dimensional root-finding problem, which is not easy, but well documented. The major disadvantage is that computing Groebner bases is costly and becomes very unwieldy if you're dealing with floating point polynomials or using higher order Bezier curves.

If you want some finished code in Haskell to find the intersections, let me know.

answered Feb 4, 2010 at 22:44

Adam
61 ● 1 ● 1

I wrote code to do this a long, long time ago. The project I was working on defined 2D objects using piecewise

4

Bezier boundaries that were generated as PostScipt paths.

The approach I used was:

Let curves p, q, be defined by Bezier control points. Do they intersect?

Compute the bounding boxes of the control points. If they don't overlap, then the two curves don't intersect. Otherwise:

p.x(t), p.y(t), q.x(u), q.y(u) are cubic polynomials on 0 <= t,u <= 1.0. The distance squared (p.x - q.x) ** 2 + (p.y - q.y) ** 2 is a polynomial on (t,u). Use Newton-Raphson to try and solve that for zero. Discard any solutions outside 0 <= t,u <= 1.0

N-R may or may not converge. The curves might not intersect, or N-R can just blow up when the two curves are nearly parallel. So cut off N-R if it's not converging after after some arbitrary number of iterations. This can be a fairly small number.

If N-R doesn't converge on a solution, split one curve (say, p) into two curves pa, pb at t = 0.5. This is easy, it's just computing midpoints, as the linked article shows. Then recursively test (q, pa) and (q, pb) for intersections. (Note that in the next layer of recursion that q has become p, so that p and q are alternately split on each ply of the recursion.)

Most of the recursive calls return quickly because the bounding boxes are non-overlapping.

You will have to cut off the recursion at some arbitrary depth, to handle weird cases where the two curves are parallel and don't quite touch, but the distance between them is arbitrarily small -- perhaps only 1 ULP of difference.

When you do find an intersection, you're not done, because cubic curves can have multiple crossings. So you have to split each curve at the intersecting point, and recursively check for more interections between (pa, qa), (pa, qb), (pb, qa), (pb, qb).

Share Improve this answer Follow

answered Jan 10, 2009 at 18:54

Die in Sente

9,907 ● 4 ● 37 ● 44

1

There are a number of academic research papers on doing bezier clipping:

http://www.andrew.cmu.edu/user/sowen/abstracts/Se306.html

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.6669

http://www.dm.unibo.it/~casciola/html/research_rr.html

I recommend the interval methods because as you describe, you don't have to divide down to polygons, and

you can get guaranteed results as well as define your own arbitrary precision for the resultset. For more information on interval rendering, you may also refer to http://www.sunfishstudio.com

Share Improve this answer

Follow

answered Sep 20, 2008 at 21:03

devinmoore

2,756 ● 3 ● 19 ● 14