

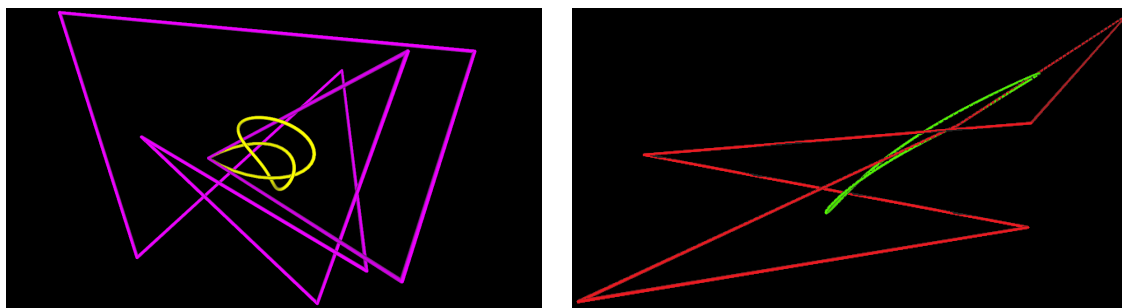
Computational Topology Counterexamples with 3D Visualization of Bézier Curves

J.Li* T.J.Peters† D. Marsh‡

November 22, 2011

Abstract

For applications in computing, Bézier curves are pervasive and are defined by a piecewise linear curve \mathcal{L} embedded in \mathbf{R}^3 to yield a smooth polynomial curve \mathcal{C} in \mathbf{R}^3 . It is of interest to understand which topological characteristics are common to \mathcal{L} and \mathcal{C} . Counterexamples are shown where \mathcal{L} is unknotted, while \mathcal{C} is knotted and where \mathcal{L} is equilateral and simple, while \mathcal{C} is self-intersecting. Curve visualizing software and numerical algorithms were central to rigorous proofs.



(a) Unknotted \mathcal{L} with knotted \mathcal{C}

(b) Equilateral, simple \mathcal{L} with self-intersecting \mathcal{C}

Figure 1: Counterexamples

1 Introduction

Computational topology lies properly within the broad scope of applied general topology by the introduction of arguments from general topology for rigorous proofs in computer science. For the computational geometric objects considered here, ideas from general topology, geometric topology and knot theory are complemented by numeric arguments in a novel integration of the ‘pure’ field of topology with the ‘applied’ focus of numerical analysis. Additionally, aspects of computer visualization and graphics are incorporated into the proofs.

*Department of Mathematics, University of Connecticut, Storrs, ji.li@uconn.edu

†Department of Computer Science and Engineering, University of Connecticut, Storrs, tpeters@cse.uconn.edu

‡This author was partially supported by NSF grants CCF 0429477, CMMI 1053077 and CNS 0923158. All statements here are the responsibility of the author, not of the National Science Foundation.

§Pratt and Whitney, East Hartford, CT, david.the.marsh@gmail.com

Attention is restricted to when \mathcal{C} is closed, implying that \mathcal{L} is also closed. As \mathcal{C} is created from \mathcal{L} , it is natural to ask which topological characteristics are shared by these two curves, particularly as the control polygon often serves as an approximation to the Bézier curve in many practical applications. The control polygon and the Bézier need not be ambient isotopic nor homeomorphic, where the counterexamples presented are new and expand upon related results, inclusive of new computational analyses to formally verify the claims made.

The counterexample of Bézier curve and its control polygon being homeomorphic, yet not ambient isotopic, is presented in Section 2. This counterexample was discovered by a computer visualization tool developed to study relationships between a Bézier curve and its control polygon. The images viewed served as catalysts to formal topological proofs, which included numerical analysis and geometric arguments.

For the second class, the visualization tool led to a conjecture, which was then refuted by a formal analysis. Many examples were viewed where the Bézier curve was simple, when its control polygon was equilateral and simple. While it is well-known that a Bézier curve can be self-intersecting even when its control polygon is simple, the conjecture was that the additional hypothesis of an equilateral control polygon resulted in both curves being simple. The visual evidence was suggestive, but misleading – underscoring the role of rigorous proof. Section 3 provides numerical algorithms to refute this conjecture by readily generated counterexamples.

1.1 Mathematical Definitions

The definitions presented are restricted to \mathbf{R}^3 , as sufficient for the purposes of this paper, but the interested reader can find appropriate generalizations in published literature.

Definition 1.1 *A knot [3] is a subspace of \mathbf{R}^3 which is homeomorphic to a circle.*

Homeomorphism is an equivalence relation over point sets and does not distinguish between different embeddings, while ambient isotopy is a stronger equivalence relation which is fundamental for classification of knots.

Definition 1.2 *Let X and Y be two subspaces of \mathbf{R}^3 . A continuous function*

$$H : \mathbf{R}^3 \times [0, 1] \rightarrow \mathbf{R}^3$$

*is an **ambient isotopy** between X and Y if H satisfies the following conditions:*

1. $H(\cdot, 0)$ is the identity,
2. $H(X, 1) = Y$, and
3. $\forall t \in [0, 1], H(\cdot, t)$ is a homeomorphism from \mathbf{R}^3 onto \mathbf{R}^3 .

*The sets X and Y are then said to be **ambient isotopic**.*

Definition 1.3 *Denote $\mathcal{C}(t)$ as the parameterized Bézier curve of degree n with control points $P_m \in \mathbf{R}^3$, defined by*

$$\mathcal{C}(t) = \sum_{i=0}^n B_{i,n}(t)P_i, t \in [0, 1]$$

where $B_{i,n}(t) = \binom{n}{i} t^i(1-t)^{n-i}$.

1.2 Related Literature

A Bézier curve and its control polygon may have substantial topological differences. It is well known that a Bézier curve and its control polygon are not necessarily homeomorphic [10]. Recently, it was shown that there exists an unknotted Bézier curve with a knotted control polygon [5]. A specific dual example has also been shown [12] of a knotted Bézier curve with an unknotted control polygon. However, the methodology was a visual construction without formal proof and is not easily generalized. Software, *Knot_Spline_Vis*, developed by the latter two authors, was used to find another example, where the methodology can more easily be generalized and *Knot_Spline_Vis* is publicly available [9]. Rigorous mathematical proofs of the claimed properties are provided to support the generalization process.

Previous work [11] in knot visualization provides a rich set of data for equilateral piecewise linear (PL) knots, where each edge is of the same length. The interface of *Knot_Spline_Vis* was designed to import this equilateral PL knot data and then generate the associated Bézier curves. This matured into an empirical study of dozens of cases, where all examples examined yielded simple Bézier curves for these simple equilateral control polygons. This raised the question of whether the presence of equilateral edges in the control polygon might be a sufficient additional hypothesis to ensure homeomorphic equivalence with the Bézier curve, as the previously cited examples [10] did not have equilateral control polygons. Further rigorous examination, supported by the numerical algorithms presented in Section 3, provided a negative answer.

2 Unknotted \mathcal{L} with knotted \mathcal{C}

In order to produce a knotted Bézier curve with an unknotted control polygon, we invoked *Knot_Spline_Vis* with an example (Figure 2) of an unknotted Bézier curve (Figure 2(b)), where the total curvature appeared to be larger than 4π . (The total curvature being larger than 4π is a necessary condition of knottedness.) We experimented on this example by

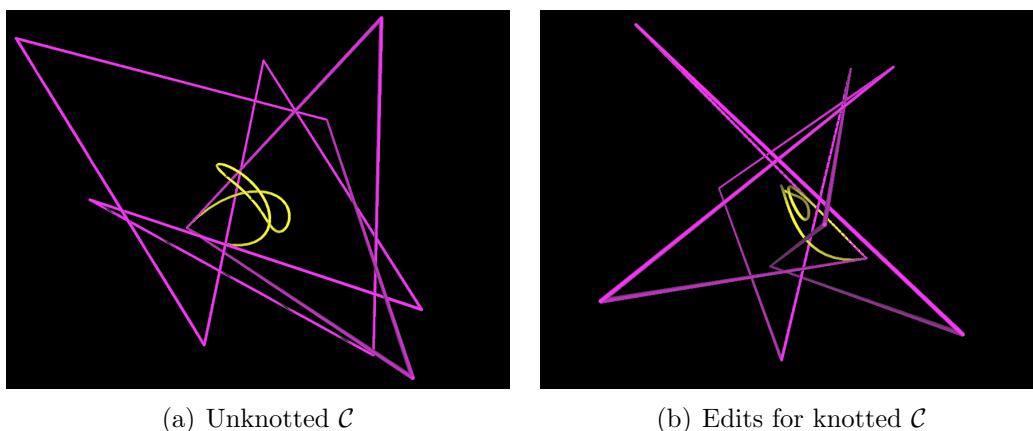


Figure 2: Visual experiments

moving control points to construct a Bézier curve that visually appeared to be knotted. We then moved control points to unknot the control polygon while keeping the Bézier curve knotted. In the end we obtained a Bézier curve and the control polygon (of degree 10)

(Figure 1(a)) defined by the control points $\{P_0, P_1, \dots, P_9, P_0\}$ listed below:

$$(-5.9, 4.7, -6.2), (10.3, -1.1, 8.9), (-2.6, -12.4, -6.3), (-10, 7, -0.3), (1.9, -12, -0.6), \\ (11.2, 7.5, -7.6), (-15.3, -1.7, -4.1), (-11.7, 20, 3.5), (17.9, -1.1, 2.9), (2.9, -13.7, 4.8), (-5.9, 4.7, -6.2).$$

The 3D visualization offers only suggestive evidence that the above Bézier curve is knotted while the control polygon is unknotted. Rigorous mathematical proofs of these properties are given below. Generally, proving knottedness or unknottedness is difficult [7], but are accessible for the counterexample generated, as given by the numerical methods described in Section 2.1.

2.1 Proofs of the Bézier curve being knotted

We shall prove that the Bézier curve is a trefoil¹. The first step is to orthogonally project the curve onto x - y plane. It is then shown that there are three self-intersections in the projection and these intersections are alternating crossings in 3D. Since projection preserve self-intersections, this curve can have no more than 3 self-intersections, but these self-intersections in x - y plane are shown to have pre-images that are 3D crossings, so the original curve has no self-intersections.

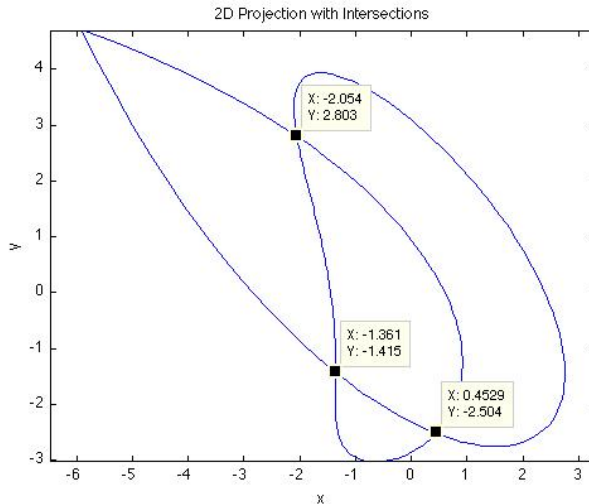


Figure 3: The 2D projection of the knot

Since the 2D curve in Figure 3 has degree 10, we use a numerical method implemented by MATLAB function ‘fminsearch’ to find the parameters where the curve intersects with itself. The numerical codes are attached in Appendix A.1. The data in finding these parameters are attached in Appendix A.3. The pairs of parameters of the self-intersections are listed below:

$$[0.0306, 0.5573], [0.1573, 0.9244], [0.3731, 0.9493].$$

¹A trefoil is the knot with three alternating crossings[8].

Next we prove that these 2D intersections are projections from three alternating crossings in 3D. The above parameters are substituted into the 3D Bézier curve (Definition 1.3) to get pairs of points (numerical codes for this calculation is in Appendix A.2):

$$\begin{aligned} & [(-2.0539, 2.8001, -2.6929), (-2.0530, 2.7987, -2.0143)], \\ & [(0.4376, -2.5212, -0.0576), (-0.4364, -2.5206, -0.5547)], \\ & [(-1.3613, -1.4239, -2.2944), (-1.3624, -1.4232, -1.9067)]. \end{aligned}$$

The alternating crossings follow from comparing the z-coordinates in each pair.

2.2 Proofs of the control polygon being unknotted

To prove that the control polygon $P = (P_0, P_1, \dots, P_9, P_0)$ is an unknot, it is necessary to show that it is simple and unknotted. The non-self-intersection of the control polygon follows from straightforward calculations using the control points of P to show that each pair of segments of P does not intersect.

For the unknottedness, we derive an unknotted PL curve (Figure 4(f)) from P using pushes [4] (Figure 4). The simple closed PL curve in Figure 4(f) is unknotted since any PL knot with less than six segment is unknotted.

Now we need to prove that the pushes indicated in Figure 4 induce ambient isotopies, so that P is ambient isotopic to the PL curve after pushes (Figure 4(f)) and hence unknotted. It is sufficient to show that the pushes do not cause intersections [1]. (This is first visually verified by looking at the corresponding 3D graphics.)

Consider Figure 4(a), where P_3 is pushed to P'_3 (the middle point² of $\overrightarrow{P_2P_4}$). To show the push does not cause any intersections, it is sufficient to show that any segments other than $\overrightarrow{P_2P_3}$ and $\overrightarrow{P_3P_4}$ of P do not intersect the triangle $\triangle P_2P_3P_4$ or the triangle interior.

Parameterize each segment by:

$$\overrightarrow{P_iP_{i+1}} : P_i + (P_{i+1} - P_i)t, \quad t \in [0, 1]$$

for $i = 0, 1, \dots, 9$ and let $P_{10} = P_0$. Then the points given by $P_3 + a(P_2 - P_3) + b(P_4 - P_3)$ for $a, b \geq 0$ and $a + b \leq 1$ are on the $\triangle P_2P_3P_4$ and contained in its interior. Hence $P_i + (P_{i+1} - P_i)t$ intersects $\triangle P_2P_3P_4$ or its interior if and only if

$$P_i + (P_{i+1} - P_i)t = P_3 + a(P_2 - P_3) + b(P_4 - P_3) \quad (1)$$

has a solution for some $t \in [0, 1]$ and $a, b \geq 0$ and $a + b \leq 1$.

For each $i = 0, 1, \dots, 9$, we solve Equation 1 (a system of 3 linear equations) for a, b and t with the above constraints. Calculations show there is no solution for each system, and hence $P_i + (P_{i+1} - P_i)t$ does not intersect $\triangle P_2P_3P_4$ or its interior for each $i = 0, 1, \dots, 9$. Thus it follows that the push does not cause any intersections.

Similar computations verify that the other pushes do not cause any intersections, thus establishing the ambient isotopy.

²It is not necessary to push P_3 to the middle point. Any point along $\overrightarrow{P_2P_4}$ would suffice.

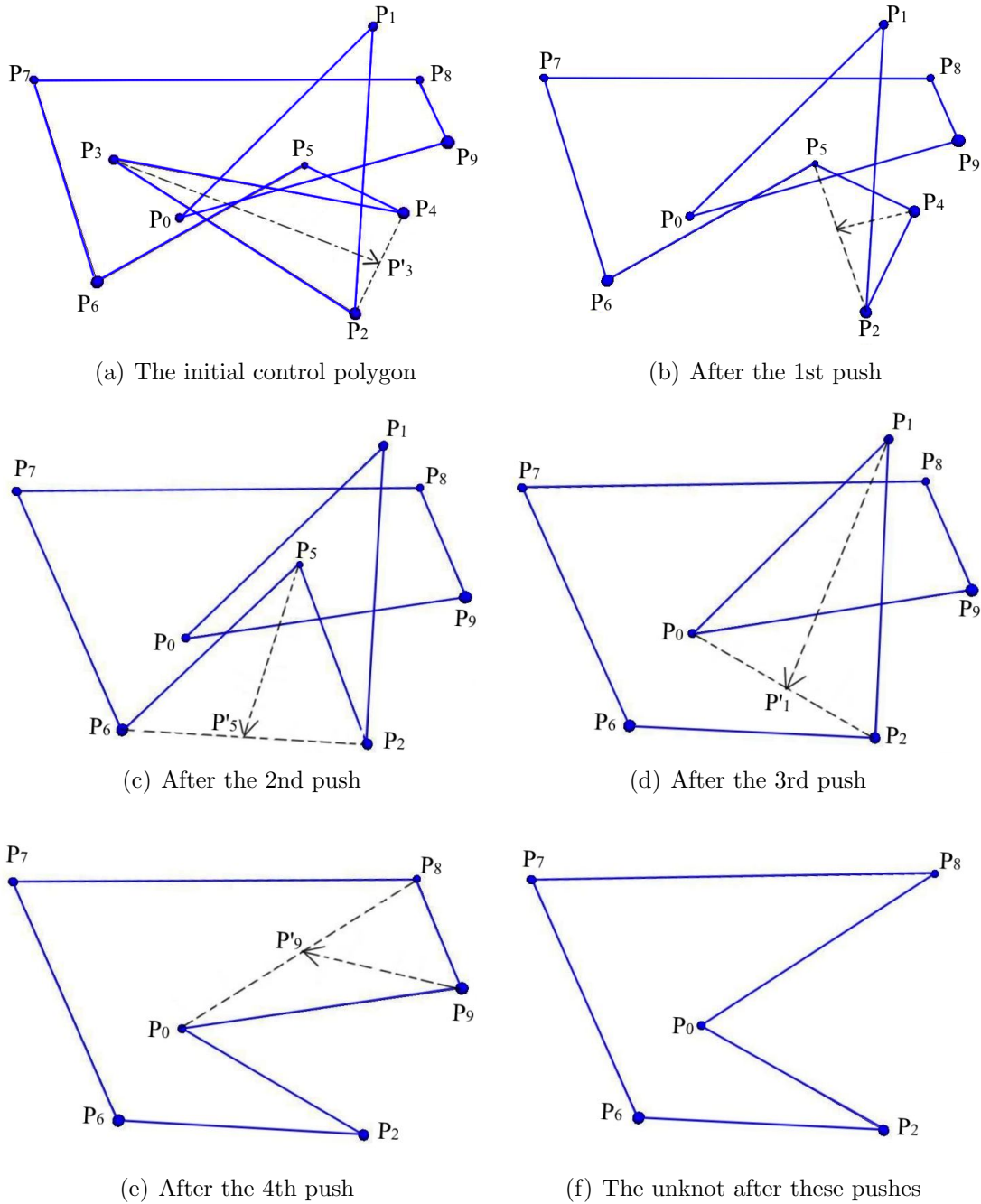


Figure 4: Resultant polygons after pushes that induce isotopies

3 Equilateral, simple \mathcal{L} with self-intersecting \mathcal{C}

Somewhat surprisingly, the many cases of simple³ equilateral control polygons that we viewed with our knot visualization tool all had Bézier curves that were simple, where some examples are shown in Figure 5. These experiments prompted the question: Do all equilateral control polygons define simple Bézier curves? We use numerical methods to obtain counterexamples.

³Throughout this section, all control polygons are simple.

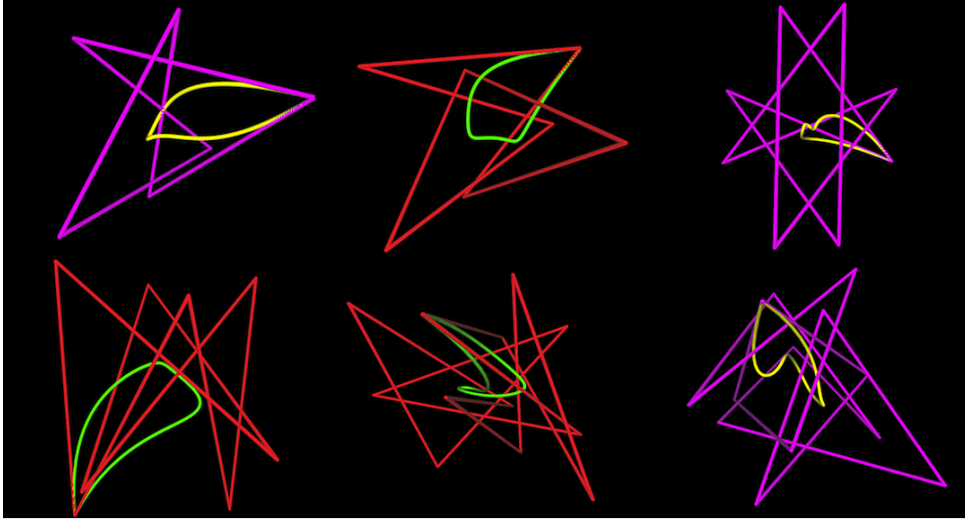


Figure 5: Simple Bézier curves with equilateral control polygons

3.1 Intuitive overview

The idea to obtain a closed equilateral polygon is to first assign $P_0 = (0, 0, 0)$. We then take $\{q_0, q_1, \dots, q_{n-1}\}$ (Equation 4) from the unit sphere so that

$$P_1 = P_0 + q_0, \quad P_2 = P_1 + q_1, \quad \dots, \quad P_n = P_{n-1} + q_{n-1}.$$

Ensuring that the polygon is closed is treated in Section 3.4 item (3).

We consider a sufficient and necessary condition for a Bézier curve being self-intersecting (Equation 3). Since we want the equilateral polygon to define a Bézier curve that is self-intersecting, we not only select $\{q_0, q_1, \dots, q_{n-1}\}$ from the unit sphere as above, but also select them such that Equation 3 is zero for some parameters s and t . Consequently the set of control points generated determines a closed equilateral control polygon and a self-intersecting Bézier curve.

3.2 Sufficient and necessary condition for self-intersection

The following equation [2] has been shown to provide conditions for self-intersection of a Bézier curve

$$S(s, t) = \frac{1}{n} \frac{\mathcal{C}(1-s) - \mathcal{C}(t)}{(1-s) - t}. \quad (2)$$

with the domain $D = \{(s, t) : s + t < 1, s, t \geq 0, (s, t) \neq (0, 0)\}$. A Bézier curve defined by $\mathcal{C}(t)$ is self-intersecting if and only if there exist t_1 and t_2 such that $t_1 \neq t_2$ by $\mathcal{C}(t_1) = \mathcal{C}(t_2)$. Equivalently, there exist s and t in the domain D such that $S(s, t) = 0$.

The paper [2] derives the following equation⁴ [2, Equation (6)] such that it can be directly used for algorithms and code to determine the existence of the self-intersections.

⁴One needs to write out the [2, Equation (6)] to obtain Equation 3.

$$S(s, t) = \frac{1}{n} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1-i} \binom{n-1-i}{j} s^{n-1-i-j} (1-s)^j \sum_{k=0}^i \binom{i}{k} (1-t)^{i-k} t^k q_{j+k}, \quad (3)$$

where

$$q_i = P_{i+1} - P_i \quad \text{for } i = \{0, 1, \dots, n-1\}. \quad (4)$$

3.3 A representative counterexample generated

While only one counterexample is presented, the numerical algorithm implemented can be used to find many such examples. The control points that determine an equilateral simple \mathcal{L} and a self-intersecting \mathcal{C} (as Figure 1(b) shows) are given below, as generated by the algorithm described in detail in Section 3.4.

$$(0, 0, 0), (0.0305, 0.0810, 0.9962), (-0.2074, -0.2671, 1.9030), \\ (-0.1792, -0.3402, 0.9063), (0.0189, 0.0782, 0.0185), (0.1557, 0.2329, -0.9600).$$

That \mathcal{L} is simple can be verified by considering all pairwise intersections of the segments of this control polygon. The self-intersection of the Bézier curve occurs at a point that is numerically approximated as

$$[s, t] = [0.2969, 0.0633]$$

where correspondingly,

$$S(s, t) = (-0.0003861, -0.000097, 0.0001462) \approx (0, 0, 0).$$

The error occurs because of numerical round off on s, t and the control points.

3.4 The numerical method for generating counterexamples

The numerical codes and data used are provided in Appendix B. Given a control polygon, we can determine whether a self-intersection of the Bézier curve occurs by determining whether S (Equation 3) has a root in D . If we consider S as a function $S(s, t, q)$ where $q = \{q_i\}_{i=0}^{n-1}$, then finding a self-intersecting Bézier curve with an equilateral control polygon is equivalent to finding s, t and q such that the following are satisfied:

1. $S(s, t, q) = 0$ where $s, t \in D$;
2. $\|q_i\| = r$ for each $i \in \{0, 1, \dots, n-1\}$, where $\|\cdot\|$ is the Euclidean norm;
3. $\sum_{i=0}^{n-1} q_i = \sum_{i=0}^{n-1} P_{i+1} - P_i = 0$ since P needs to be closed.

We assume $r = 1$ without loss of generality since the value of r can be adjusted by scaling. Throughout the provided codes, n is always the degree of the Bézier curve. The Appendix B.1 gives the code for function S , where $[s, t]$ is labelled as u and q as $[a, b, c]$.

The function SF (Appendix B.2) takes parameters for s, t and q and outputs a floating point value. It is designed to satisfy that the value is zero if and only if the above three conditions are satisfied simultaneously.

Precisely since q should be taken from the unit sphere, SF assigns a, b, c values given by

$$\begin{aligned} a &= \sin(\phi)\cos(\theta); \\ b &= \sin(\phi)\sin(\theta); \\ c &= \cos(\phi), \end{aligned}$$

where ϕ and θ represent input parameters. In order to satisfy the Condition (3) above, q_{n-1} is set equal to $-\sum_{i=0}^{n-2} q_i$. But in this way, $\|q_{n-1}\|$ may fail to be equal to 1. So we include the function F (Appendix B.2) to determine whether $\|q_{n-1}\| = 1$. The function is designed to be $F = \|q_{n-1}\| - 1$ such that $\|q_{n-1}\| = 1$ if and only if $F = 0$.

Symbolically,

$$SF = \|S\| + \|F\|, \tag{5}$$

where S is given by Equation 3 and $F = \|q_{n-1}\| - 1$. Having the above three conditions satisfied simultaneously is equivalent to finding input values such that $SF = 0$.

Since $SF \geq 0$, the minimum of SF is 0. The function $SFminimizer$ (Appendix B.3) uses $fminsearch$ (a numerical method integrated in MATLAB) to search for the minimum of SF , while returning this minimum and the corresponding values of s, t and q . The user supplied initial guesses for s, t and q greatly influence the results, we assign $SFminimizer$ randomized initial values M times so that we get M different minimums for different initial values. But no matter which initial values we use, as long as we can get “a” minimum of 0, then we get the equilateral control polygon which determines a self-intersecting Bézier curve.

The data of finding the counterexample of Section 3.3 is included in Appendix B.4.

4 Visualizing Bézier curves & their control polygons

To study the knot types of a control polygon and its Bézier curve, a knot visualization tool was developed, called *Knot_Spline_Vis*. The tool *Knot_Spline_Vis* takes a PL control polygon as input and displays both the PL curve and the associated Bézier curve. For these studies, the input was always restricted to be a PL curve of known knot type. The functions of *Knot_Spline_Vis* were designed to permit interactive studies of the topological relationships between a Bézier curve and its control polygon. The intent is to use these examples to stimulate mathematical conjectures as a prelude to formal proofs.

Some of the standard graphical manipulation capabilities provided are illustrated in the following Figure 6, where the rotation capabilities have been particularly useful to develop visual insights into the occurrence of self-intersections and crossings as fundamental for the study of knots. An editing window allows the user to change the coordinates of the control polygons, as shown in Figure 7. The software is freely available for download at the site www.cse.uconn.edu/~tpeters.

The control polygon is an initial PL approximation to its associated Bézier curve. Subdivision algorithms [10] produce additional control points to further refine the original control

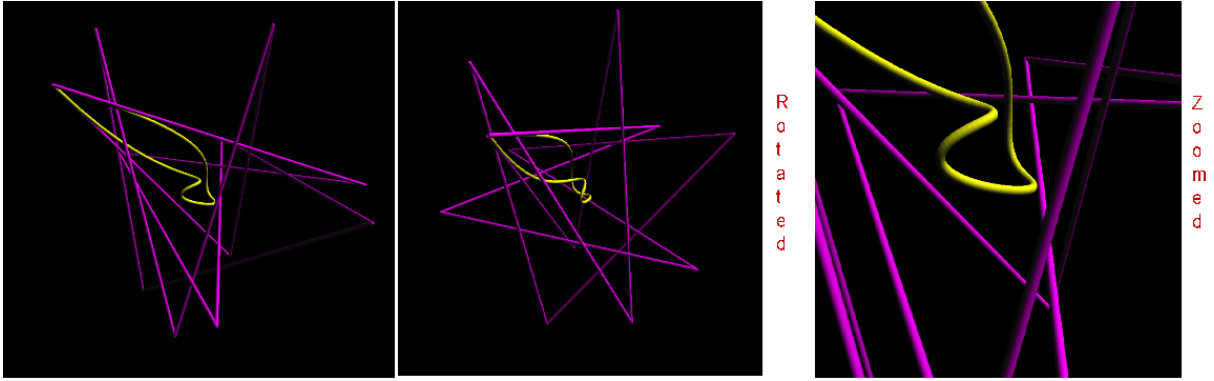
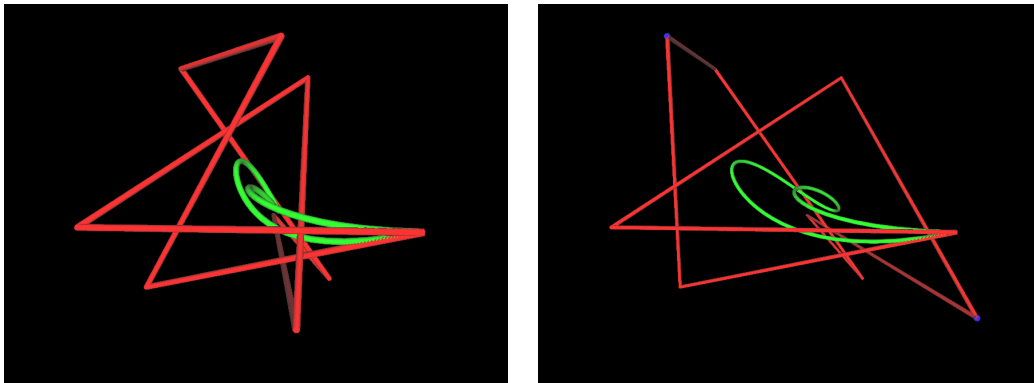


Figure 6: Rotating and scaling



(a) Initial display

(b) Some vertices moved

Figure 7: Moving control points

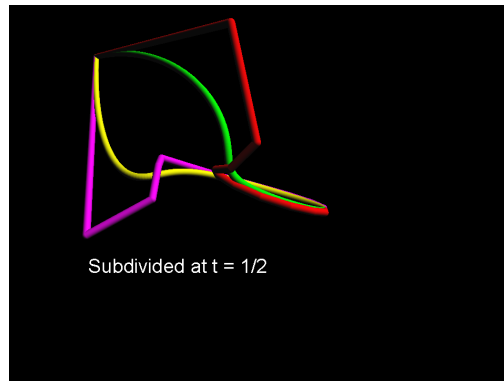


Figure 8: A subdivision of Figure 6

polygon, converging in distance to the Bézier curve. Figure 8 illustrates performance of a standard subdivision technique, the de Casteljau algorithm [6]. Users can specify a subdivision parameter. Figure 8 shows the case with parameter $\frac{1}{2}$.

5 Conclusions and Future Work

For Bézier curves, a counterexample is shown where the control polygon is unknotted, while the curve is knotted and an equilateral simple control polygon is shown to generate a self-intersecting Bézier curve. This discovery is aided by a spline visualization tool, *Knot_Spline_Vis* that allows easy generation of examples for rapid insight to support rigorous proofs. The development of *Knot_Spline_Vis* and the examples presented were motivated by the applied interest to develop scientific animations that would be synchronized with macromolecular simulations on high performance computing architectures. For these simulations, it is important for the biochemical user community to visually understand the complexity of curve embeddings within \mathbf{R}^3 , with particular attention to when the embedding might change. The tool *Knot_Spline_Vis* is a rich visual environment that facilitates software development by easy generation of extreme test data.

6 Web Posting of Supplemental Materials

Appendices listed below are posted on this webpage:

<http://www.math.uconn.edu/~jili/Supplemental-materials.pdf>

Appendix A: Numerical codes for knottedness of \mathcal{C} (Figure 1(a)):

- A.1 Codes for searching self-intersections in Figure 3;
- A.2 Codes for determining under or over crossings;
- A.3 Data for searching intersections in Figure 3.

Appendix B: Numerical codes for searching the Example of Figure 1(b):

- B.1 Codes for Equation 3;
- B.2 Codes for Equation 5;
- B.3 Codes for searching the roots of the system of Equations 3 and 5;
- B.4 Data for finding the example shown in Figure 1(b).

References

- [1] J. W. Alexander and G. B. Briggs. On types of knotted curves. *Annals of Mathematics*, 28:562–586, 1926-1927.
- [2] L. E. Andersson, T. J. Peters, and N. F. Stewart. Selfintersection of composite curves and surfaces. *CAGD*, 15:507–527, 1998.
- [3] M. A. Armstrong. *Basic Topology*. Springer, New York, 1983.
- [4] R. H. Bing. *The Geometric Topology of 3-Manifolds*. American Mathematical Society, Providence, RI, 1983.
- [5] J. Bisceglia, T. J. Peters, J. A. Roulier, and C. H. Sequin. Unknots with highly knotted control polygons. *CAGD*, 28(3):212–214, 2011.
- [6] G. Farin. *Curves and Surfaces for Computer Aided Geometric Design*. Academic Press, San Diego, CA, 1990.
- [7] Joel Hass, Jeffrey C. Lagarias, and Nicholas Pippenger. The computational complexity of knot and link problems. *Journal of the ACM*, 46(2):185–221, 1999.
- [8] C. Livingston. *Knot Theory*, volume 24 of *Carus mathematical monographs*. Mathematical Association of America, Washington, DC, 1993.
- [9] T. J. Peters and D. Marsh. Personal home page of T. J. Peters. <http://www.cse.uconn.edu/>.
- [10] L. Piegl and W. Tiller. *The NURBS Book*. Springer, New York, 2nd edition, 1997.
- [11] Robert Scharein. The knotplot site. <http://www.knotplot.com/>.
- [12] Carlo H. Sequin. Spline knots and their control polygons with differing knottedness. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-152.html>.

A Code: knottedness of Bézier curve of Figure 1(a)

A.1 Code for self-intersections of Figure 3

```
%The function C2d(t) defines the projection of the Bézier curve.
function [value] = C2d(t)
n=10;
P=zeros(2,11);
P(1,:)= [-5.9, 10.3, -2.6, -10, 1.9, 11.2, -15.3, -11.7, 17.9, 2.9, -5.9];
P(2,:)= [4.7, -1.1, -12.4, 7, -12, 7.5, -1.7, 20, -1.1, -13.7, 4.7];

sum=0;
for i=0:n
    sum = sum + nchoosek(n,i) * t^i * (1 - t)^(n - i) * P(1, i + 1);
end
v1 = sum;

sum=0;
for i=0:n
    sum = sum + nchoosek(n,i) * t^i * (1 - t)^(n - i) * P(2, i + 1);
end
v2 = sum;

value = [v1, v2];

% Use the function 'fminsearch' to find the minimums of fnS(x), the zero minimums and
parameters where the zeros occur are what we look for.

function [value] = fnS(x)
value = norm(C2d(x(1)) - C2d(x(2)), 2);

function [value] = Smin(s0,t0)
Max=optimset('MaxFunEvals',1e+19);
comb=@(x)fnS(x);
u=[s0,t0];
[xval, fval] = fminsearch(comb, u, Max)
```

A.2 Code for determining crossings

```
%Below is the function of the Bézier curve in 3D (Definition 1.3).
function [value] = C(t)
n=10;
P=zeros(3,11);
P(1,:)= [-5.9, 10.3, -2.6, -10, 1.9, 11.2, -15.3, -11.7, 17.9, 2.9, -5.9];
```

```

P(2,:)= [4.7, -1.1, -12.4, 7, -12, 7.5, -1.7, 20, -1.1, -13.7, 4.7];
P(3,:)= [-6.2, 8.9, -6.3, -0.3, -0.6, -7.6, -4.1, 3.5, 2.9, 4.8, -6.2];

sum=0;
for i=0:n
    sum = sum + nchoosek(n,i) * t^i * (1 - t)^(n - i) * P(1, i + 1);
end
v1 = sum;

sum=0;
for i=0:n
    sum = sum + nchoosek(n,i) * t^i * (1 - t)^(n - i) * P(2, i + 1);
end
v2 = sum;

sum=0;
for i=0:n
    sum = sum + nchoosek(n,i) * t^i * (1 - t)^(n - i) * P(3, i + 1);
end
v3 = sum;

value = [v1, v2, v3];

```

A.3 Data for searching intersections in Figure 3

```

% The Matlab commands and corresponding results:
% The initial input (0.03, 0.55) is figured out by the observation and calculations on self-
intersections in Figure 3. Similarly for the others below.
Smin(0.03,0.55)
xval = 0.0306 0.5573
fval = 2.9567e-04

Smin(0.15,0.92)
xval = 0.1573 0.9244
fval = 1.5848e-04

Smin(0.37,0.95)
xval = 0.3731 0.9493
fval = 1.4637e-04

```

B Code: generating counterexamples like Figure 1(b)

B.1 Code for Equation 3

```
function [value] = S(u,a,b,c)
sum=0; subsum1=0; subsum2=0;
for i=0:n-1
for j=0:n-1-i
% The use of adding 1 to the vectors a, b and c is required by MATLAB.
for k=0:i subsum2 = subsum2 + nchoosek(i, k) * (1 - u(2))^(i - k) * u(2)^k * a(j + k + 1);
end
subsum1 = subsum1 + subsum2 * nchoosek(n - 1 - i, j) * u(1)^(n - 1 - i - j) * (1 - u(1))^j;
end
sum = sum + subsum1;
end
v1 = (1/n) * sum;

sum=0; subsum1=0; subsum2=0;
for i=0:n-1
for j=0:n-1-i
for k=0: i
subsum2 = subsum2 + nchoosek(i, k) * (1 - u(2))^(i - k) * u(2)^k * b(j + k + 1);
end
subsum1 = subsum1 + subsum2 * nchoosek(n - 1 - i, j) * u(1)^(n - 1 - i - j) * (1 - u(1))^j;
end
sum = sum + subsum1;
end
v2 = (1/n) * sum;

sum=0; subsum1=0; subsum2=0;
for i=0:n-1
for j=0:n-1-i
for k=0:i subsum2 = subsum2 + nchoosek(i, k) * (1 - u(2))^(i - k) * u(2)^k * c(j + k + 1);
end
subsum1 = subsum1 + subsum2 * nchoosek(n - 1 - i, j) * u(1)^(n - 1 - i - j) * (1 - u(1))^j;
end
sum = sum + subsum1;
end
v3 = (1/n) * sum;
value = [v1, v2, v3];
```

B.2 Code for Equation 5

```
function [value] = SF(x,n)
q=zeros(3,n); p=zeros(3,n+1);
a=zeros(1,n);b=zeros(1,n);c=zeros(1,n);
alpha = zeros(1, 2 * n - 2);
for i = 1 : 2 * n - 2
alpha(i)=x(i);
end
for i=1:n-1
a(i) = sin(alpha(i)) * cos(alpha(n - 1 + i));
b(i) = sin(alpha(i)) * sin(alpha(n - 1 + i));
c(i) = cos(alpha(i));
q(:,i)=[a(i),b(i),c(i)];
end

a(n)=0; b(n)=0; c(n)=0;
for i=1:n-1
a(n)=a(n)-a(i); b(n)=b(n)-b(i); c(n)=c(n)-c(i);
q(:,n)=[a(n),b(n),c(n)];
end

u=zeros(1,2); u = [x(2 * n - 1), x(2 * n)];
value=abs(F(alpha,n))+norm(S(u,a,b,c),2);

for i=1:n
for j=1:i;
p(:,i+1)=p(:,i+1)+q(:,j);
end
end
p

function [value]=F(alpha,n)
for i=1:n-1
a(i) = sin(alpha(i)) * cos(alpha(n - 1 + i)); b(i) = sin(alpha(i)) * sin(alpha(n - 1 + i)); c(i) =
cos(alpha(i));
end

a(n)=0; b(n)=0; c(n)=0;
for i=1:n-1
a(n)=a(n)-a(i); b(n)=b(n)-b(i); c(n)=c(n)-c(i);
end
value = abs(a(n)2 + b(n)2 + c(n)2 - 1);
```

B.3 Codes for solving the system of Equations 3 and 5

```
function [value] = SFminimizer(n,M)
Max=optimset(MaxFunEvals;1e+9);
comb=@(x)SF(x,n);
xval=zeros(2*n,M);
fval=zeros(1,M);

for i=1:M
phi=unifrnd(0,pi,1,n-1); theta=unifrnd(0,2*pi,1,n-1);
s0=unifrnd(0,1); t0=unifrnd(0,1-s0);
x0=[phi,theta,s0,t0];
[xval(:,i), fval(i)] = fminsearch(comb, x0, Max);
%make sure  $s + t = xval(2 * n - 1, i) + xval(2 * n, i) < 1$  so that  $s, t \in D$ .
if  $xval(2 * n - 1, i) + xval(2 * n, i) > 1$  or  $xval(2 * n - 1, i) + xval(2 * n, i) = 1$ 
fval(i)=1;
else
end
end

%Display the minimums and corresponding  $s, t$  and  $q$ .
xval fval
%Returns the optimal one.
[C, I] = min(fval);
value=[C,I];
```

B.4 Data for finding the example shown in Figure 1(b)

The counterexample (Section 3.3) uses the second column below for the input parameters.
SFminimizer(6,10)

xval =

Columns 1 through 10

2.1907	0.0867	1.0279	2.4529	0.8294	1.3029	1.0958	0.8502	1.2136	0.3308
2.1572	0.4353	1.9303	0.3841	-0.2788	0.3006	-0.2769	2.8797	3.0782	3.1699
0.0079	3.2225	3.2107	1.8739	3.1482	3.6489	1.3900	0.5959	0.1228	1.8107
0.5496	2.6633	0.2000	1.1955	0.5784	2.0316	1.6273	2.0583	3.1341	1.3276
1.8511	2.9336	1.0424	2.6615	2.3439	1.2207	2.4580	4.3850	0.1877	-0.1146
5.2200	1.2107	7.0598	3.4826	0.3696	0.5633	5.1432	4.2546	3.5523	5.5103
2.7090	4.1128	4.1941	2.3445	2.4291	2.0589	0.4295	2.9467	1.0576	5.9281
0.1954	2.0119	2.2644	7.0575	2.1168	2.3027	1.9744	-0.2264	2.5206	2.4902
0.5312	7.4240	2.2923	5.1205	1.2132	4.0211	3.4493	1.6729	1.7930	5.7145
1.3429	0.8465	1.4447	4.4661	4.0120	1.8860	0.0947	3.9222	5.1409	1.2114
0.5200	0.2969	0.7942	0.7334	0.2149	0.6042	1.3269	0.9360	0.1185	0.0320
0.0054	0.0633	0.2813	0.2543	0.6229	0.4588	0.0165	0.1667	0.5815	0.5970

```
fval =  
Columns 1 through 7  
0.0000 0.0000 1.0000 0.2180 0.0001 1.0000 1.0000  
Columns 8 through 10  
1.0000 0.0001 0.0000
```

```
% The corresponding function values of  $S$  and  $F$  are given:  
Svalue =  $1.0e - 03$ *  
-0.3861 -0.0970 0.1462
```

```
Fvalue =  $2.2329e - 05$ 
```