Bézier Spline Simplification Using Locally Integrated Error Metrics

Siqi Wang
New York University
New York City, NY, USA
sw4429@nyu.edu

Chenxi Liu
University of British Columbia
Vancouver, Canada
chenxil@cs.ubc.ca

Daniele Panozzo
New York University
New York City, NY, USA
panozzo@nyu.edu

Denis Zorin
New York University
New York City, NY, USA
dzorin@cs.nyu.edu

Alec Jacobson
University of Toronto, Adobe
Toronto, Canada
alecjacobson@adobe.com

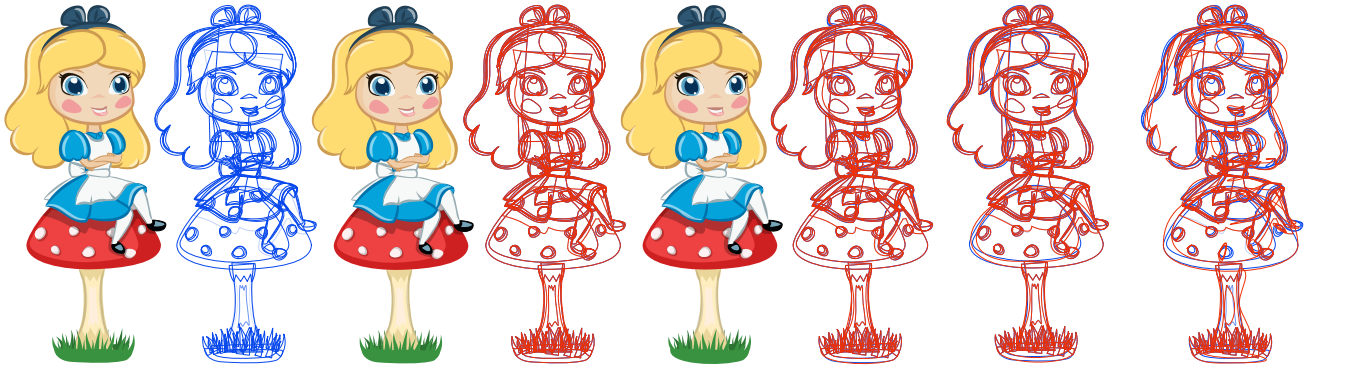


Figure 1: Left to right: Our method accepts as input collections of Bézier curves (left) and simplifies them *losslessly* to remove redundant control points. Our lossy extension produces much more simplified result without sacrificing visual quality. Simplifying using Adobe Illustrator and the method of Schneider [1990] (in Inkscape) produce worse results for the same number of curves.

ABSTRACT

Inspired by surface mesh simplification methods, we present a technique for reducing the number of Bézier curves in a vector graphics while maintaining high fidelity. We propose a curve-to-curve distance metric to repeatedly conduct local segment removal operations. By construction, we identify all possible *lossless* removal operations ensuring the smallest possible zero-error representation of a given design. Subsequent *lossy* operations are computed via local Gauss-Newton optimization and processed in a priority queue. We tested our method on the OpenClipArts dataset of 20,000 real-world vector graphics images and show significant improvements over representative previous methods. The generality of our method allows us to show results for curves with varying thickness and for vector graphics animations.

CCS CONCEPTS

• **Computing methodologies** → **Parametric curve and surface models**; *Physical simulation.*

KEYWORDS

Vector graphics, Bézier spline, simplification, animation

ACM Reference Format:
Siqi Wang, Chenxi Liu, Daniele Panozzo, Denis Zorin, and Alec Jacobson. 2023. Bézier Spline Simplification Using Locally Integrated Error Metrics . In *SIGGRAPH Asia 2023 Conference Papers (SA Conference Papers '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3610548.3618248

SA Conference Papers '23, December 12–15, 2023, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0315-7/23/12…$15.00
https://doi.org/10.1145/3610548.3618248

1 INTRODUCTION

Simplification is a core sub-routine found in any popular vector graphics editing tool (e.g., Inkscape, Adobe Illustrator, CorelDRAW). There are many ways that a design could end up with too many or too detailed curves: densely upsampling before applying a pointwise filter, adding anchors along curves but forgetting to delete them if they're never moved, scaling down a detailed design relative to its final display resolution, vectorizing raster art inefficiently, etc. With increasingly popular vector graphics *animation* formats (e.g., Lottie), dense vector graphics designed for static display or print may now expect to be served up to mobile devices at high framerates. Furthermore, vector graphics also function as path descriptions for CNC machines such as laser cutters, routers, and plotters. Overly dense designs cause fabrication defects or firmware failure.

In this paper, we demonstrate that existing simplification methods leave significant room for improvement. We are particularly interested at the high end of the simplification-accuracy curve: simplifying as much as possible while remaining exceptionally accurate to preserve the artist's intention. A critical unit test is recovering lossless simplification. For example, take a coarse spline, subdivide it repeatedly, and then try to simplify back to the original number of curves (see Fig. 2). Existing methods found in the literature and in commercial software fail this seemingly simple test.

We propose a Bézier spline simplification technique inspired by progressive surface mesh simplification methods. We conduct local segment removal operations in a priority queue. These operations are based on a measure of curve-curve distance which is carefully constructed to efficiently identify *lossless* removals as zero-cost operations. Once lossless removals are exhausted, subsequent lossy operations based on local Gauss-Newton optimization are conducted in a greedy manner (see Fig. 3).

To evaluate our method, we conduct a large-scale — first of its kind — benchmark comparison on a large dataset of vector graphics images. Our method consistently outperforms representative state-of-the-art methods. Our method is agnostic to the dimension of the input curves' embedding space: we show results appending varying stroke thickness as a 3rd coordinate. We also demonstrate simplifying across an entire animation, implicitly ensuring temporally coherent control point distribution and movement.

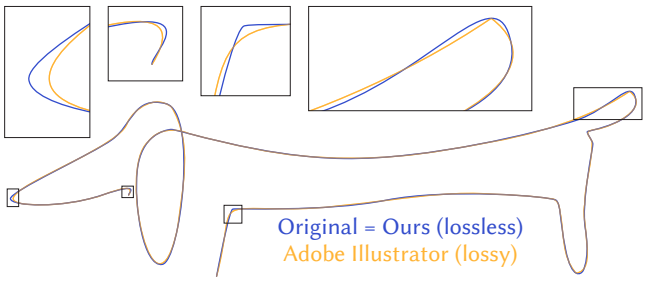


Figure 2: As a stress test, the blue curve is upsampled in place from 19 to 304 segments. Our method losslessly simplifies back to the original 19 segments. Adobe Illustrator's proprietary simplification to 19 segments produces noticeable error.

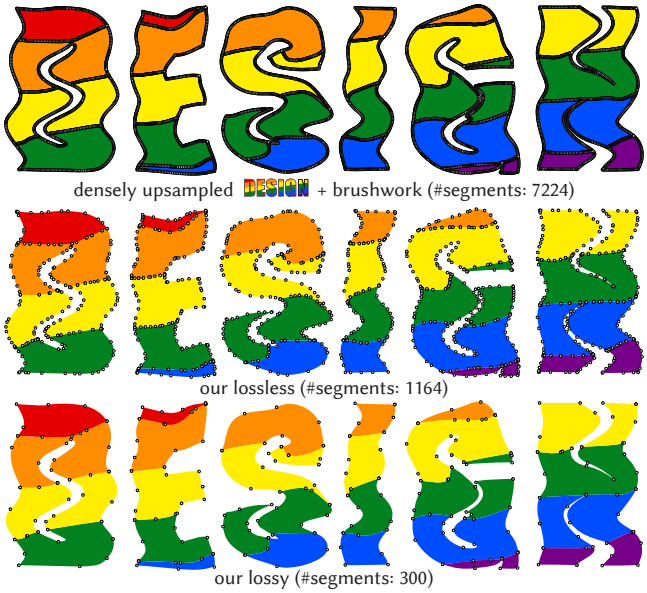


Figure 3: Simplification is a core subroutine in an editing setting. Densely upsampling this input allows an artist's Warp brushwork in Illustrator to apply a pointwise deformation over the design. Our lossless simplification removes geometrically unnecessary segments, and further lossy simplification produces a coarse, yet visually accurate design.

2 RELATED WORK

The literature on simplification for polylines is vast beginning with the error-bound method of Douglas and Peucker [1973], which still enjoys practical use. We focus our attention on methods for Bézier curve simplification and related problems on smooth curves.

Sampling-and-refitting. Schneider [1990] proposes a thorough and widely-used G^1-continuous solution to the general problem of fitting a cubic Bézier spline to a digitized curve, which can also be applied to Bézier curve simplification, as is used in the *Path Simplify* tool of Inkscape. Similar algorithms include [Chang and Yan 1998; Masood and Sarfraz 2009; Sarfraz and Masood 2007; Sarfraz and Razzak 2002] based on recursive segment subdivision, characteristic points detection, and least-square fitting or other curve approximation approaches. Kolesnikov [2010] introduces the inflection points with relaxed constraints of tangent continuity. Shao and Zhou [1996] proposes a global least-squares optimization fitting method with critical points identification to fit the input data points using block coordinate descend to do non-linear optimization. This global optimization technique struggles to find a global minimum for long complicated chains. Mokhtarian et al. [2005] fits the digitized contours through curvature scale space techniques. van Goethem et al. [2013] takes polygon as input and

allows topology-preserving Bézier curves fitting with Voronoi diagrams. The software Potrace can also transform bitmaps into vector graphics but takes a faster and simpler approach that only generates a subset of all possible Bézier curves. These methods are mostly aimed at fitting digitized curves or boundaries of bitmap images. However, in our setting, sampling and refitting the input introduces errors, cannot guarantee consistent tangents at endpoints, and cannot provide a lossless solution. Besides, the method has to pick a point on the closed loop to serve as the overlapping start and end points of the chain, which cannot be moved, and the selection is not optimized. These papers experimented on a small set of representative examples but were not tested on a large dataset. From the comparison of the latest paper [Masood and Sarfraz 2009] among them, the out-performance over [Schneider 1990] is insignificant. Thus, we use the widely-implemented method [Schneider 1990] as a representative comparison in our benchmark.

Cubic Bézier fitting. De Boor et al. [1987] describes an interpolation scheme that matches position, tangents, and curvature at the endpoints and proves convexity preservation and $O(n^6)$ error scaling; but the solution is not guaranteed to exist for all input data. Penner [2019] presents three criteria – fitting curvature at endpoints, least squares orthogonal distance fitting, and the fitting center of mass, with the former two relevant to our problem. However, the presented method for orthogonal distance fitting requires an initial setting of the parameters and is very slow. Levien [2009] provides a highly efficient and accurate solution to cubic Bézier curve fitting by building an error metric to match the signed area of a cubic Bézier curve, which involves solving a quartic-polynomial system, and making careful decisions in the subdivision. However, this method sometimes generates cusps as one of the solutions. We included a comparison of the reference implementation of [Levien 2009] (Kurbo) with our method in Section 4.

B-spline knot removal. Lyche and Mørken [1987] proposes a knot removal algorithm for parametric B-spline curves, which aims to reduce the number of polynomial segments in a curve without exceeding a given error bound. Jupp [1978] and Loach and Wathen [1991] keep the number of knots fixed but optimize their positions. Kang et al. [2015] solves a sparse optimization problem followed by additional fitting to determine the number and positions of knots. Dierckx [1995] reviews a variety of methods for fitting curves, including variable knots, choosing number and initial knot placement, adding smoothing terms, and convexity preservation constraints. Some of these methods use a Gauss-Newton optimization akin to ours to minimize their respective distance metrics. Dung and Tjahjowidodo [2017] presents a new strategy for fitting any forms of the curve by B-spline functions via a local algorithm by first splitting the data with bisection and then optimizing via the non-linear least-squares technique.

Vector graphics animation. Dalstein et al. [2015] allows features of a connected drawing to have topological changes via keyframes. Liu et al. [2014] provides a solution to animate an SVG with linear blend skinning interactively via least-squares fitting the control points of the input splines (whose density is defined by the user). Some 2D animation software, such as Cartoon Animation 5, supports the vector graphics format, but none pays attention to simplifying the SVG time series and keeping temporal coherency. In the animation software that has the functionality of adding spring bones and Free-Form Deformation (FFD), our work can also act as an adjoint tool to remove the redundant control points, add control points in the necessary regions, and export a smooth and clean SVG sequence. To maintain temporal coherency, a lot of work in mesh animation uses principle component analysis (PCA) or clustered principal component analysis (CPCA) to achieve animation compression, e.g. [Alexa and Müller 2000; Karni and Gotsman 2004; Lengyel 1999; Sattler et al. 2005; Váša and Skala 2009]. None of these methods can export a simplified sequence of vector graphic frames.

3 METHOD

Our method takes as input a vector graphics image and outputs a simplified vector graphics image. We operate directly on the underlying cubic Bézier splines representing the paths of standard vector graphics formats (e.g., `.svg` or `.ai` files). Consistent with common practice of vector graphics software (Inkscape, Illustrator, CorelDRAW) during editing, we homogenize all paths (e.g., circles, arcs, ellipses, quadratic Béziers) into cubic Bézier splines. These paths may define strokes, fill boundaries, clip-mask path boundaries, and other stylings. Our method is agnostic to stylings, which are reassociated with the simplified output paths to render the final image. We'll use the following terminology in this paper:

- **segment**: a single cubic Bézier curve,
- **chain**: G^1 continuous sequence of one or more segments,
- **component**: C^0 continuous sequence of one or more chains,

and unless context demands, Bézier always refers to cubic Bézier.

Thus, the *input* to our method is a collection of N segments in $\mathbb{R}^d$ and a target number of output segments K. The *output* is a collection of K segments in $\mathbb{R}^d$ whose components closely match corresponding input components.

Overview. Our method works in a greedy manner analogous to edge-collapse decimators for triangle mesh simplification in computer graphics [Garland and Heckbert 1997; Hoppe 1996]. We define local "collapse" operations which remove a single segment and adjust neighboring control points on the component. We define a non-negative (and sometimes zero) cost function to measure the loss incurred by each operation. All possible operations are placed into a priority queue and processed in a greedy manner. When a collapse operation is conducted, operations involving neighboring segments become out of date and are replaced in the queue with operations referencing the newly repositioned control points. All local operations are constructed to have $O(1)$ complexity so that total processing is asymptotically efficient: $O(N \log N)$. Let us now consider the definitions of operations and cost functions.

3.1 Distance between two segments

A fundamental expression in the following section will be a measure of distance between parametric spans of two Bézier segments.

There are many ways to measure distance between two curves. Hausdorff distance is difficult to compute precisely and sensitive to outliers [Agarwal 2007]. Chamfer distance or integrated closest point is a good choice perceptually, but does not have a closed form expression and is sensitive to (near) overlaps and self-intersections.

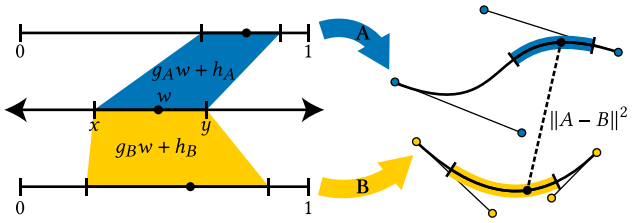


Figure 4: We measure the distance between partial parametric spans of two Bézier segments as an integral over a shared parametric domain that is linearly warped into each segment's Bézier parameter space.

Meanwhile, Fréchet distance leverages that both curves may be parameterized over the real line segment [0,1]. Fréchet distance considers *all possible* re-parameterizations of the two curves. Imagine taking the max distance between your left fingertip as it runs over one curve and your right fingertip as it runs over the other. Fréchet distance takes the minimum across *all* possible parameterizations of each curve, or all possible speeds that our fingertips move relative to each other. Precise computation of Fréchet distance is somewhat tractable [Rote 2007] but has similar outlier issues as Hausdorff: ultimately the measure is determined by a single pair of points [Agarwal 2007].

Similar to Fréchet, we propose considering all possible pairs of piecewise-linear reparametrizations of each curve with parameter locations co-located at each segment boundary, but unlike Hausdorff or Fréchet we take the integrated squared distance along the curves. That is, the Bézier parameterization of each segment is only affected by a linear mapping. We will show this is both friendly to computation and a good model of perceived distance, as indicated by our qualitative results. Additionally, the distance is zero if and only if a lossless merge is possible and smoothly increases away from this case, enabling gradient-based minimization.

Consider a Bézier segment $\mathbf{A} : [0,1] \to \mathbb{R}^d$ defined[1] by control points $\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4 \in \mathbb{R}^d$. Similarly, define another curve $\mathbf{B} : [0,1] \to \mathbb{R}^d$ with control points $\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4 \in \mathbb{R}^d$. The parametric domains of $\mathbf{A}$ and $\mathbf{B}$ are both [0,1], but we will integrate distance over an arbitrary parametric subdomain:

$$\int_x^y \|\mathbf{A}(g_A w + h_A) - \mathbf{B}(g_B w + h_B)\|^2 \, dw, \tag{1}$$

where various parameters appear to control the measure: x, y control the stretch of a shared parametric domain from which the variable of integration w spans, g_A, h_A and g_B, h_B are parameters which define an *affine* map from this shared domain to the Bézier parameter domains of curves $\mathbf{A}$ and $\mathbf{B}$, respectively.

3.2 Distance between two chains

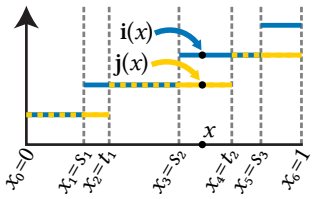
Building on segment-segment distance, we now consider distance between two chains. Consider a chain of n segments: $\mathcal{A} : [0,1] \to \mathbb{R}^d$ defined by control points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{3n+1}\}$ and a set of $n+1$ parameter locations $s_0, s_1, \ldots, s_n \in [0,1]$ where $s_0 = 0$ and $s_n = 1$:

[1] $\mathbf{A}(u) = (1-u)^3 \, \mathbf{a}_1 + (1-u)^2 u \, \mathbf{a}_2 + (1-u) u^2 \, \mathbf{a}_3 + u^3 \, \mathbf{a}_4$.

$$\mathcal{A}(x) = \begin{cases} \mathbf{A}^1\left(\frac{x-s_0}{s_1-s_0}\right) & \text{if } x \le s_1 \\ \mathbf{A}^2\left(\frac{x-s_1}{s_2-s_1}\right) & \text{else if } x \le s_2 \\ \vdots & \\ \mathbf{A}^i\left(\frac{x-s_{i-1}}{s_i-s_{i-1}}\right) & \text{else if } x \le s_i \\ \vdots & \\ \mathbf{A}^n\left(\frac{x-s_{n-1}}{s_n-s_{n-1}}\right) & \text{else if } x \le s_n, \end{cases} \tag{2}$$

where points $\mathbf{p}_{3i-2}, \mathbf{p}_{3i-1}, \mathbf{p}_{3i}, \mathbf{p}_{3i+1}$ control each segment $\mathbf{A}^i$.

Similarly, define a chain $\mathcal{B} : [0,1] \to \mathbb{R}^d$ with $n-1$ segments with control points $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_{3n-2}\}$ and parameter locations t_j.



For each of these, let us define piecewise-constant indexing functions:

$$\mathbf{i} : [0,1] \to \{1, \ldots, n\}, \text{ where } \mathbf{i}(x) = i \text{ if } s_{i-1} < x \le s_i, \text{ and} \tag{3}$$

$$\mathbf{j} : [0,1] \to \{1, \ldots, n-1\}, \text{ where } \mathbf{j}(x) = j \text{ if } t_{j-1} < x \le t_j. \tag{4}$$

Without loss of generality, assume the parameter locations s_i and t_j are disjoint (outside of endpoints) so they cut up the domain [0,1] into $2n-2$ intervals. Since s_i and t_j may be in any order relative to each other, we introduce a joint sequence notation:

$$\{x_0, x_1, \ldots, x_{2n-2}\} = \{0, \text{sort}(s_1, \ldots, s_{n-1}, t_1, \ldots, t_{n-2}), 1\}, \tag{5}$$

while this simplifies notation, it obscures that each parameter location x_k *depends* (when it comes to derivatives) on some s_i or t_j; this will be essential for correct gradient computation later.

Equipped with a way to refer to these intervals in order, we may define a distance between these chains as a sum over intervals:

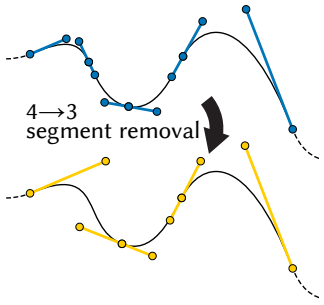
$$\mathrm{E}(\mathbf{P}, \{s_i\}, \mathbf{Q}, \{t_j\}) = \sum_{k=1}^{2n-2} \omega_k \int_{x_{k-1}}^{x_k} \left\| \mathbf{A}^{\mathbf{i}(x)}\left(\frac{x - s_{\mathbf{i}(x)-1}}{s_{\mathbf{i}(x)} - s_{\mathbf{i}(x)-1}}\right) - \mathbf{B}^{\mathbf{j}(x)}\left(\frac{x - t_{\mathbf{j}(x)-1}}{t_{\mathbf{j}(x)} - t_{\mathbf{j}(x)-1}}\right) \right\|^2 dx, \tag{6}$$

where — despite the indexing hellscape — we observe that each term in the summation is an integral of the form in Eq. 1. The ω_k term *weighs* the impact of the kth integral on the total. One choice would be to use the arc-length of the corresponding pieces of the segments of $\mathcal{A}$ and $\mathcal{B}$. This will be problematic in our setting because only the longer curve $\mathcal{A}$ is known in advance and *approximating* arc-lengths of Bézier curves relies on computationally expensive operations. Instead, we propose to use $\omega_k = 1/(s_i - s_{i-1}) + 1/(t_j - t_{j-1})$, which can be seen as a first-order approximation of arc-length (i.e., assumes segments are straight) *and* discourages intervals from disappearing by inversely proportionally growing in scale. We draw special attention that ω_k therefore *depends* on some s_i and t_j values.

3.3 Segment removal operation

Analogous to an edge-collapse operation for triangle meshes, we now define a local segment remove operation for a chain of length n,

defined by control points $\mathbf{P} = \{\mathbf{p}_1, \ldots, \mathbf{p}_{3n+1}\}$. For now, let us assume the input to this subroutine are n segments forming a chain within a possibly longer chain continuing at either end. The output are $n-1$ segments defined by control points $\mathbf{Q} = \{\mathbf{q}_1, \ldots, \mathbf{q}_{3n-2}\}$ and a measured *cost* of accepting this solution:



$$\text{cost} = \min_{\{s_i\},\mathbf{Q},\{t_j\}} E(\mathbf{P}, \{s_i\}, \mathbf{Q}, \{t_j\}). \tag{7}$$

3.3.1 General case ($n \geq 2$). The cost in Eq. 7 is a minimization of a continuous function of the unknowns $\{s_i\}$, $\mathbf{Q}$, $\{t_j\}$. We will optimize this via a modified Gauss-Newton method.

By appending constraints to this opimization, we ensure C^0 and G^1 continuity at the endpoints (where the chain joins neighboring segments) and at internal interpolated control points in the output (for $n > 2$). Namely, we maintain C^0 continuity with:

$$\mathbf{q}_1 = \mathbf{p}_1 \quad \text{and} \quad \mathbf{q}_{3n-2} = \mathbf{p}_{3n+1} \tag{8}$$

and G^1 continuity with:

$$\exists\, \alpha > 0 \mid \mathbf{q}_2 - \mathbf{q}_1 = \alpha\,(\mathbf{p}_2 - \mathbf{p}_1)\,, \tag{9}$$

$$\exists\, \beta > 0 \mid \mathbf{q}_{3n-3} - \mathbf{q}_{3n-2} = \beta\,(\mathbf{p}_{3n} - \mathbf{p}_{3n+1})\,, \tag{10}$$

$$\exists\, \gamma_j > 0 \mid \mathbf{q}_{3j} - \mathbf{q}_{3j+1} = \gamma_j\,(\mathbf{q}_{3j+1} - \mathbf{q}_{3j+2})\ \forall\, j = 1, \ldots, n-2. \tag{11}$$

The constraints in Eq. 8 can be substituted into Eqs. 9 & 10, revealing they are linear in $\alpha, \mathbf{q}_2$ and $\beta, \mathbf{q}_{3n-3}$, respectively. These are easily enforced via the null-space method: introducing α and β as variables.

That is, we build a matrix $\mathbf{N} \in \mathbb{R}^{d(3n-2)\times(3d(n-2)+2)}$ from relevant entries of $\mathbf{P}$ so that:

$$\mathbf{Q} = \mathbf{N}\,\mathbf{y} + \tilde{\mathbf{Q}} \text{ such that Eqs. 8-10 hold for any } \mathbf{y} \in \mathbb{R}^{3d(n-2)+2} \tag{12}$$

where $\mathbf{y}$ collects remaining free variables in $\mathbf{Q}$ and α, β.

Meanwhile, the constraints in Eq. 11 are *bi*-linear in the γ_js and $\mathbf{Q}$, respectively. Treating the s, γ, and t variables as *known*, then the cost function is a quadratic function in $\mathbf{Q}$. This implies it has a closed-form solution $\mathbf{Q}^\star$, which can be expressed as a non-linear function $f : \mathbb{R}^{3n-5} \to \mathbb{R}^{d(3n-2)}$ of γ, s, and t variables:

$$\mathbf{Q}^\star = f(\{s_i\}, \{\gamma_j\}, \{t_j\}). \tag{13}$$

We can now substitute $\mathbf{Q}^\star$ from Eq. 13 into the optimization problem in Eq. 7 to get a non-linear optimization problem over $\{s_i\}, \{\gamma_j\}, \{t_j\}$:

$$\min_{\{s_i\},\{\gamma_j\},\{t_j\}} E(\mathbf{P}, \{s_i\}, f(\{s_i\}, \{\gamma_j\}, \{t_j\}), \{t_j\}). \tag{14}$$

While we could apply simple gradient descent, we see significant improvements leveraging the sum-of-squares nature of the distance in Eq. 6 to use Gauss-Newton method.

Because the integrands in Eq. 1 are sixth-order polynomials over the integration variable (squared norm of a cubic polynomial), we may immediately apply sixth-order accurate quadrature rules to *exactly* evaluate these integrals for any input parameters.

$$\sum \int \|\ldots\|^2 \quad \xrightarrow{\text{exact quadrature}} \quad \sum \sum \|\ldots\|^2 \;=\; \sum \|\ldots\|^2$$

Our sum of integrals is now a simple sum of squared function evaluations: suitable for discrete Gauss-Newton method.

All that remains is to apply chain-rule and differentiate each term with respect to the free variables. Our implementation uses complex-step numerical differentiation [Lyness and Moler 1967] to easily (without hand-coding any derivatives), accurately (up to machine precision), and efficiently (with little more cost than a few forward evaluations) compute necessary Jacobian matrices. For small n (in our cases $n \leq 4$), this choice is appropriate. For much larger n (e.g., $n = O(N)$), sophisticated automatic-differentiation tools such as [Agarwal et al. 2022] may start to see performance improvements.

Given a Gauss-Newton search direction we conduct a backtracking line-search [Boyd and Vandenberghe 2006] to find a suitable step length, while constraining α, β, γ parameters to stay positive. For our most common case of $n = 4$, we observe convergence typically in ≈10 iterations. For fixed n, this entire *local* optimization is $O(1)$ with respect to the complete input vector graphics image of N segments.

To initialize our optimization, we set $s_i = \sum_{k=1}^{i} \ell_k / \sum_{k=1}^{n} \ell_k$, where ℓ_i is the arc-length of the ith segment (approximated numerically). Then we pick the best $\{t_j\}$ initial values among all $n-1$ subsequences of $\{s_i\}$ and an additional uniform setting of $s_i = 1/(n-1)$. For small n, we observe that trying a few initial values often helps speed up convergence. Our specific heuristic ensures that we immediately identify input chains of $n-1$ straight segments where one segment can always be losslessly removed. Tangent length ratios α, β, γ are initialized to relative arc-lengths: again agreeing with constant speed straight segments.

3.3.2 Special lossless case $n = 2$. If $n = 2$ *and* we know that we are only interested in accepting the segment removal for numerically zero cost, then we can avoid the continuous optimization of the previous section and jump directly to the solution of the low-order polynomial root finding problem.

Key Observation: A lossless segment removal on a two-segment chain defined by control points $\mathbf{c}_1, \mathbf{c}_2, \mathbf{c}_3, \mathbf{c}_4 \in \mathbb{R}^d$ and $\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_4 \in \mathbb{R}^d$ exists *if and only if* there exists a single curve with control points $\mathbf{q}_1, \mathbf{q}_2, \mathbf{q}_3, \mathbf{q}_4 \in \mathbb{R}^d$ and a parameter value $t \in (0, 1)$ such that subdividing the $\mathbf{q}$ at t produces the segments $\mathbf{c}$ and $\mathbf{d}$.

The output segment $\mathbf{q}$ has constant third derivative, so to admit a lossless removal the inputs must have equal third derivatives that must be equal up to a scale factor. That scale factor can be written in terms of a parameter $t \in (0, 1)$ such that $\mathbf{c}$ and $\mathbf{d}$ segments are the result of splitting the output segment $\mathbf{q}$ at t:

$$-6\mathbf{q}_1 + 18\mathbf{q}_2 - 18\mathbf{q}_3 + 6\mathbf{q}_4 = \tag{15}$$

$$\frac{-6\mathbf{c}_1 + 18\mathbf{c}_2 - 18\mathbf{c}_3 + 6\mathbf{c}_4}{t^3} = \frac{-6\mathbf{d}_1 + 18\mathbf{d}_2 - 18\mathbf{d}_3 + 6\mathbf{d}_4}{(1-t)^3}, \tag{16}$$

$$\frac{\ddot{\mathbf{c}}}{t^3} = \frac{\ddot{\mathbf{d}}}{(1-t)^3}, \tag{17}$$

where we further assume that third-derivatives $\dddot{\mathbf{c}}, \dddot{\mathbf{d}} \in \mathbb{R}^d$ are non-degenerate (if they both are, then the segments could be merged as quadratic or linear Bézier curves).

Now, we can solve this third-order polynomial equation for t:

$$\underbrace{\frac{\dddot{\mathbf{d}} \cdot \dddot{\mathbf{c}}}{\left\|\dddot{\mathbf{d}}\right\|^2}}_{r}(1-t)^3 = t^3 \tag{18}$$

$$-(1+r)t^3 + 3rt^2 - 3rt + r = 0. \tag{19}$$

For $r > 0$ (implying $\dddot{\mathbf{c}}$ and $\dddot{\mathbf{d}}$ point in the same direction), this equation has a single real root:

$$t = \left(1 + r^{-1/3}\right)^{-1}. \tag{20}$$

Finally, if this lossless removal is possible, then we can recover the corresponding control points from our continuity assumptions:

$$\mathbf{q}_1 = \mathbf{c}_1 \tag{21}$$

$$\mathbf{q}_4 = \mathbf{c}_4 \tag{22}$$

$$\mathbf{q}_2 - \mathbf{q}_1 = \frac{\mathbf{c}_2 - \mathbf{c}_1}{t} \tag{23}$$

$$\mathbf{q}_4 - \mathbf{q}_3 = \frac{\mathbf{d}_4 - \mathbf{d}_3}{1-t}. \tag{24}$$

To determine if a lossless removal is actually possible, a practical approach is to compute t and $\mathbf{q}_i$ as above and then check if the integrated error between the original two segments and the new single segment (using Eq. 6) is (numerically close enough to) zero.

Assuming an infinite precision machine, a straightforward algorithm for lossless simplification of a sequence of n segments is to process all consecutive pairs of segments in a queue or stack. Upon each pop, if the pair is still valid and can be losslessly merged, then conduct it and insert any new consecutive pairs into the queue. The $2 \longrightarrow 1$ merge operation is local and $O(1)$ cost, and the full algorithm is $O(n)$. The algorithm is also guaranteed to find all possible lossless mergers, resulting in the most compact lossless representation.

In practice, floating point operations introduce errors and these errors may accumulate. In the worst case, we could merge pairs *in order* along a chain of N segments all stemming from a single ($K = 1$) curve, accumulating error N times over before the final merge. Instead, we can process operations in a queue, which accumulates the worst case error at a rate of $\log N$.

Our cubic root finding relies on floating point operations which are not error-free. We observe that in the range $t \in [10^{-3}, 1-10^{-3}]$ the numeric loss for theoretically lossless merges is very low $<10^{-16}$. However, this numeric loss grows toward larger values $\approx 10^{-5}$ as the solution t gets closer to 0 or 1. We apply one iteration of the Gauss-Newton descent on E , initialized at our $\mathbf{c}$ and t values to "brush off" the last bit of numerical error.

3.4 Greedy Priority Queue Processing

Equipped with our local operations, we split the input into chains at corners, determined by a threshold on incident tangents. Every pair of consecutive segments is initially placed in a priority queue based on its lossless $2 \longrightarrow 1$ operation (see Section 3.3.2). These are processed — pushing neighboring lossless $2 \longrightarrow 1$ operations onto the queue any time an operation is accepted — until the queue is empty or only K segments remain.

If more than K segments remain, then for every consecutive four segments we push a (potentially lossy) $4 \longrightarrow 3$ operation on the queue. For chains of length $n < 4$, we also push an appropriate $n \longrightarrow (n-1)$ operation. These are processed — again pushing appropriately updated neighboring operations upon each acceptance — until K segments remain.

We use a priority queue with pop-min, update-key, and insert operations with cost $O(\log n)$, keyed on the cost of each possible $4 \longrightarrow 3$ or $2 \longrightarrow 1$ operation. Computing each cost is $O(1)$ and there are $O(n)$ operations: filling the queue has thus a $O(n \log n)$ cost. Finding the cheapest operation is $O(\log n)$, executing it $O(1)$, and updating the cost of the neighboring elements is $O(\log n)$ (note that the number of neighbors is constant): overall, it is $O(\log n)$ total work per pop. Since we have $K \leq n$ operations to pop, the final time complexity is $O(K \log n + n \log n) = O(n \log n)$.

Finally, our algorithm assumes all corners are preidentified. We can also simplify *across corners* (cost permitting) by relaxing the G^1 constraints in Eq. 11 based on a user-provided threshold. To support loops: during the local segment removal optimization for closed loop chains of length $\leq n$, G^1 continuity constraints are appropriately modified so that the choice of endpoints is unbiased.

Table 1: Runtime performance across the examples in this paper.

Example	*#curves before*	*#curves after*	*Runtime (s)*
Fig. 1	2,233	687	558
Fig. 2	304	19	1
Fig. 3	7,224	300	367
Fig. 7	2,280	400	432
Fig. 8	1,000	300	125
Fig. 9	1,130 × 100	(avg.) 89 × 100	1,349
Fig. 10 (1)	2,916	1,458	779
Fig. 10 (2)	5,001	1,500	1,595
Fig. 10 (3)	20,293	6,087	1,822
Fig. 10 (4)	16,628	6,651	2,094

4 EXPERIMENTS & DISCUSSION

We implemented our algorithm in MATLAB, using gptoolbox [Jacobson et al. 2021] for vector graphic support and basic geometry processing. We implemented our Gauss-Newton solver with backtracking line search ($\alpha = 0.3, \beta = 0.5$) [Boyd and Vandenberghe 2004], with stopping criteria: relative energy $< 10^{-15}$, relative gradient $< 10^{-5}$, or max 30 iterations. Our algorithm and implementation is *by construction* $O(N \log N)$, but beyond ensuring correct asymptotic performance, we did not optimize our method and leave a high-performance implementation as incremental future work. We report the runtime of our implementation for the examples in the paper in Table 1, measured on an Intel i7 processor clocked at 2.6 GHz. Instead, we focus on the superior quality of our results in terms of error and the generality of our approach with respect to input. Fig. 10 contains a gallery and our supplemental data contains a more thorough `.html` explorer of results.

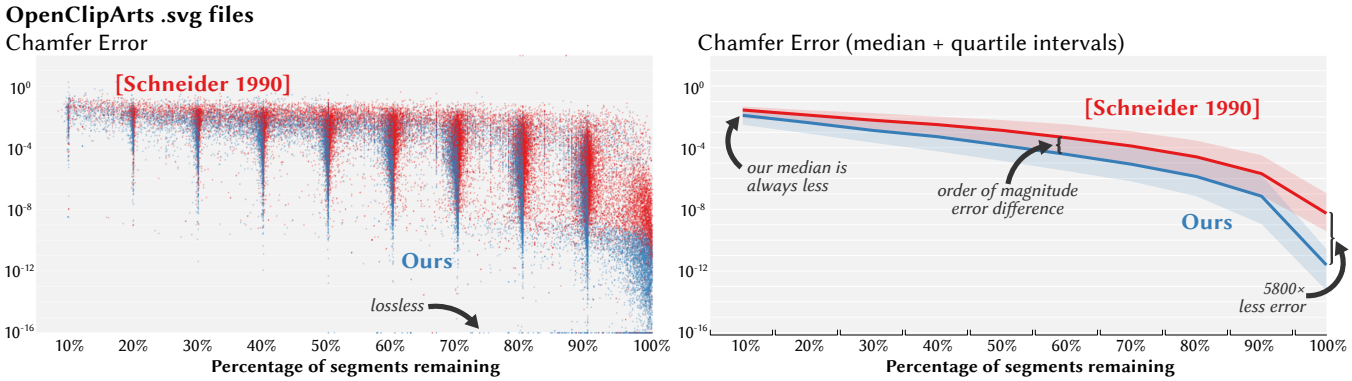


Figure 5: We conducted a large-scale benchmark of vector graphics simplification — as far as we know — the first of its kind. We compare our simplification method to [Schneider 1990] for a variety of target segment counts, recording the chamfer error to the original (left). Around 75% experience at least some lossless removals. On the right, we bin collected samples in deciles and report their medians and quartile intervals. Our method's median is always smaller though best improvements are observed for larger percentages. *For visualization clarity, we snap extremely small errors of our lossless results to axis bounds.*

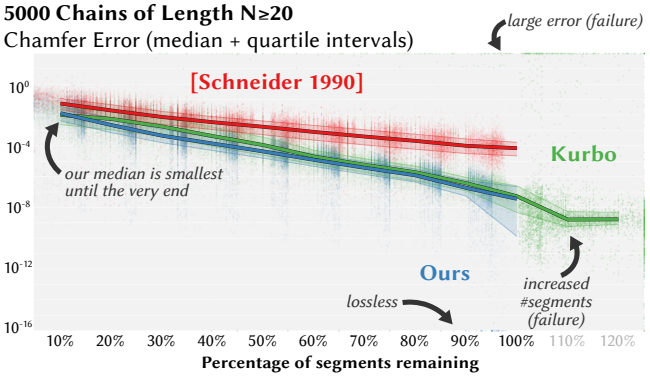


Figure 6: To include Kurbo [Levien 2009] in our comparisons, we build a smaller dataset of long, smooth chains. When Kurbo works, it works reasonably well: its median is similar to ours. However, it often creates extreme error results or even adds more segments instead of removing them. *We snap extremely large/small errors and increased segment counts to axis bounds.*

Large-Scale Evaluation. We conducted a large-scale quantitative evaluation on 17,944 in-the-wild .svgs composed of 55 million segments in total from OpenClipArts dataset [Hu et al. 2019] (≈2k models were excluded because our .svg loader does not support esoteric path commands). Our lossless simplification benefited 74.2% of the models. This suggests that most vector graphics found in the wild could save on storage and bandwidth without any noticeable change: motivating our lossless contributions. Our method demonstrates favorable quality for the same simplification amount compared to [Schneider 1990] (Fig. 5). For each model, we consider a series of K values: we run our lossless process until completion $K = K_{\text{lossless}}$, then applying our lossy process to reach $K = 90\%, 80\%, \ldots, 10\%$ of the original number of segments until no lossy removal is possible without exceeding a very large error value. We run [Schneider 1990] until achieving roughly the same K values. We measure bi-directional chamfer error between the input and output: densely, uniformly randomly sampling all curves of the input and taking the mean squared L^2 distance to the output curves, then *vice versa*, and averaging the two errors. Our method outperforms [Schneider 1990] across the entire range of the simplification amount: the median curve mostly staying around an order of magnitude lower, with a much large difference improvement in the 90-100% range.

We conducted a smaller quantitative comparison including Kurbo [Levien 2009] on 5,000 especially numerically smooth chains of length $N \geq 20$ from randomly sampled models of the OpenClipArts dataset. We constructed this dataset out of fairness to Kurbo, as it is more sensitive to slight tangent mismatches at perceptually smooth control points than [Schneider 1990] and our algorithm. Kurbo does not provide a direct way of controlling the output number of segments, so we do a 13-value sweep over its simplifier's tolerance parameter. Fig. 6 visualizes the collected data and demonstrates our method again consistently outperforming [Schneider 1990]. We also observe that *when Kurbo succeeds* it works fairly well, but it often fails in one of three different ways: (1) getting no results after an extremely long time (>1 hour), (2) producing excessively high error results due to explosion (curves far outside the input's bounding box), and (3) outputting *more* segments than the input (see types (2) & (3) in Fig. 6). We ran Kurbo $65,000 = 5,000 \cdot 13$ times, of which Kurbo failed — in one of these three ways — 34.4% of the time.

Operation Type Ablation. We conducted an ablation experiment on the Night Horsewoman poster in Fig. 10. We compared performing only a single operation independently against our algorithm that combines them. We observe that 4 → 3 only and

$3 \rightarrow 2$ only cannot simplify as much as our reference strategy, leading to higher errors. The $2 \rightarrow 1$ only strategy is more flexible and can simplify as much as our reference but with higher errors. Overall, this experiment confirms that using all operations leads to superior simplification results for the same quality.

Editing Software. Our method qualitatively out-performs standard software: Adobe Illustrator and Inkscape ([Schneider 1990]), as shown in Fig. 1 and Fig. 2. Additionally, it supports lossless simplification, which is not supported by these software. In Fig. 1, our lossless simplification down to 87.82% of the original segments. We further *lossily* simplify to 30.77% without perceptible visual change: at the same simplification level, Adobe Illustrator introduces noticeable visual differences over the entire image (especially on the mushroom) and Schneider [1990] significantly alters the image. Kurbo [Levien 2009] fails on this input.

Lossless simplification workflow. In Fig. 3, we emulate an artist's workflow with a black box pointwise brush tool. We densely upsample a graphic so the brush is applied directly to control points, then simplify to preserve only the necessary control points. The editing operation often leaves parts of the model with dense samples unmoved or moved in a locally affine way: this presents a significant opportunity for lossless simplification. We simplify down to 16.1% in this example. The artist can continue to repeat this workflow of upsampling, editing, lossless simplification for as many iterations as desired because no information is lost in the procedure, unlike in all previous methods. Once the artist is satisfied with the final design, the image can be processed by our lossy simplification into a much smaller file without a noticeable change to the design. This opens the door to a workflow similar to traditional photo editing, where lossless compression formats (`.png`) are used for editing, and lossy formats (`.jpg`) are used for distribution.

Manufacturing. Our simplification can benefit manufacturing methods (plotting, laser cutting) that take vector paths as input. We drew the boundary of Norway using a robotic plotting machine (Fig. 7) and observe that the vector path simplified by our algorithm has less ink bleeding compared to the original one.

Higher dimensions. Simplification is a core subroutine for not only 2D vector images but also vector data of higher dimensions. We demonstrate that our method can be easily extended to higher-dimension vector data, including vector images with stroke width and vector animation. The "John Hancock" signature in Fig. 8 consists of cubic Bézier curves whose control points contain an extra stroke width dimension. Our method yields a more compact image without losing noticeable accuracy. In Fig. 9, we densely upsampled the input `.svg` and applied an embedded mesh animation based on a full-space complementary dynamics [Zhang et al. 2020] and barycentric coordinates interpolation: this procedure is necessary to faithfully capture the deformation but introduces many redundant control points. If the animation is applied directly on the `.svg` without upsampling, the final animation is noticeably defective: e.g., the dolphin's blowhole is incorrectly appears outside of the poorly animated silhouette. With our method, we can efficiently simplify the upsampled sequence with temporal coherency by (1) applying truncated principal component analysis (PCA) on the high dimensional animation data similar to [Alexa and Müller 2000] and then conducting lossy simplification on the PCA vectors and then (2) applying lossless simplification on each frame individually as a post process to remove redundant control points. Our simplification result is qualitatively better than a baseline of simplifying every frame independently, which leads to undesired temporal popping artifacts (see supplemental video).

5 LIMITATIONS & FUTURE WORK

We focused exclusively on asymptotic performance and implemented our prototype in Matlab where scripting overhead is high. We expect that a high-performance implementation in C++ is a straightforward extension, there are also advances in parallelism for surface mesh simplification in parallel which could be adapted to our analogous setting [Karis 2022].

Our method does *not* conduct topological simplification; most notably, we simplify but never remove an entire input component. Thus, a valid input K should be greater than or equal to the number of input components. We conjecture that topological simplification requires more advanced perceptual metrics and leave for a future work which may benefit from our method as a subroutine. We also do not consider overdraw (we don't flatten or remove hidden paths). More perceptually accurate corner detection and persistence would aid all smooth simplification methods, including ours. Finally, we hope our emphasis on statistically meaningful large-scale testing inspires chasing down the remaining gains in vector graphics simplification.

ACKNOWLEDGMENTS

We thank Otman Benchekroun for preparing the rig animation for the dolphin example; Raph Levien for sharing implementation details of Kurbo; William Moses for the help with Enzyme; Dingyi Zhuang for video editing; all the artists for sharing the vector graphic images and anonymous reviewers for their helpful comments and suggestions. This work was partially supported by NSERC Discovery (RGPIN–2022–04680), the Ontario Early Research Award program, the Canada Research Chairs Program, a Sloan Research Fellowship, the DSI Catalyst Grant program, the National Science Foundation grants OAC-1835712 and NSF CHS-1908767, and gifts by Adobe Inc.

REFERENCES

Pankaj K. Agarwal. 2007. *Lecture 23: Hausdorff and Frechet distance.*

Sameer Agarwal, Keir Mierle, and The Ceres Solver Team. 2022. *Ceres Solver.* https://github.com/ceres-solver/ceres-solver

Marc Alexa and Wolfgang Müller. 2000. Representing animations by principal components. In *Computer Graphics Forum*, Vol. 19. Wiley Online Library, 411–418.

Stephen Boyd and Lieven Vandenberghe. 2006. *Convex Optimization.*

Stephen P Boyd and Lieven Vandenberghe. 2004. *Convex optimization.* Cambridge university press.

Hung-Hsin Chang and Hong Yan. 1998. Vectorization of hand-drawn image using piecewise cubic Bezier curves fitting. *Pattern recognition* 31, 11 (1998), 1747–1755.

Boris Dalstein, Rémi Ronfard, and Michiel Van De Panne. 2015. Vector graphics animation with time-varying topology. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

Carl De Boor, Klaus Höllig, and Malcolm Sabin. 1987. High accuracy geometric Hermite interpolation. *Computer Aided Geometric Design* 4, 4 (1987), 269–278.

Paul Dierckx. 1995. *Curve and surface fitting with splines.* Oxford University Press.

David H Douglas and Thomas K Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: the international journal for geographic information and geovisualization* 10, 2 (1973), 112–122.

Van Than Dung and Tegoeh Tjahjowidodo. 2017. A direct method to solve optimal knots of B-spline curves: An application for non-uniform B-spline curves fitting. *PloS one* 12, 3 (2017), e0173857.

Michael Garland and Paul S Heckbert. 1997. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. 209–216.

Hugues Hoppe. 1996. Progressive meshes. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. 99–108.

Yixin Hu, Teseo Schneider, Xifeng Gao, Qingnan Zhou, Alec Jacobson, Denis Zorin, and Daniele Panozzo. 2019. TriWild: robust triangulation with curve constraints. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–15.

Alec Jacobson et al. 2021. gptoolbox: Geometry Processing Toolbox. http://github.com/alecjacobson/gptoolbox.

David LB Jupp. 1978. Approximation to data by splines with free knots. *SIAM J. Numer. Anal.* 15, 2 (1978), 328–343.

Hongmei Kang, Falai Chen, Yusheng Li, Jiansong Deng, and Zhouwang Yang. 2015. Knot calculation for spline fitting via sparse optimization. *Computer-Aided Design* 58 (2015), 179–188.

Brian Karis. 2022. *The Journey to Nanite.*

Zachi Karni and Craig Gotsman. 2004. Compression of soft-body animation sequences. *Computers & Graphics* 28, 1 (2004), 25–34.

Alexander Kolesnikov. 2010. Approximation of digitized curves with cubic Bézier splines. In *2010 IEEE International Conference on Image Processing*. IEEE, 4285–4288.

Jerome Edward Lengyel. 1999. Compression of time-dependent geometry. In *Proceedings of the 1999 symposium on Interactive 3D graphics*. 89–95.

Raphael Linus Levien. 2009. *From spiral to spline: Optimal techniques in interactive curve design.* University of California, Berkeley.

Songrun Liu, Alec Jacobson, and Yotam Gingold. 2014. Skinning Cubic Bézier Splines and Catmull-Clark Subdivision Surfaces. *ACM Transactions on Graphics (TOG)* 33, 6 (2014), 9 pages.

PD Loach and AJ Wathen. 1991. On the best least squares approximation of continuous functions using linear splines with free knots. *IMA journal of numerical analysis* 11, 3 (1991), 393–409.

Tom Lyche and Knut Mørken. 1987. Knot removal for parametric B-spline curves and surfaces. *Computer Aided Geometric Design* 4, 3 (1987), 217–230.

J. N. Lyness and C. B. Moler. 1967. Numerical Differentiation of Analytic Functions. *SIAM J. Numer. Anal.* 4, 2 (1967), 202–210.

Asif Masood and Muhammad Sarfraz. 2009. Capturing outlines of 2D objects with Bézier cubic approximation. *Image and Vision Computing* 27, 6 (2009), 704–712.

Farzin Mokhtarian, Yoke Khim Ung, and Zhitao Wang. 2005. Automatic fitting of digitised contours at multiple scales through the curvature scale space technique. *Computers & Graphics* 29, 6 (2005), 961–971.

Alvin Penner. 2019. Fitting a cubic Bezier to a parametric function. *The College Mathematics Journal* 50, 3 (2019), 185–196.

Günter Rote. 2007. Computing the Fréchet distance between piecewise smooth curves. *Comput. Geom.* 37, 3 (2007), 162–174. https://doi.org/10.1016/j.comgeo.2005.01.004

Muhammad Sarfraz and Asif Masood. 2007. Capturing outlines of planar images using Bézier cubics. *Computers & Graphics* 31, 5 (2007), 719–729.

Muhammad Sarfraz and MFA Razzak. 2002. An algorithm for automatic capturing of the font outlines. *Computers & Graphics* 26, 5 (2002), 795–804.

Mirko Sattler, Ralf Sarlette, and Reinhard Klein. 2005. Simple and efficient compression of animation sequences. In *Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*. 209–217.

Philip J Schneider. 1990. An algorithm for automatically fitting digitized curves. *Graphics gems* 1 (1990), 612–626.

Lejun Shao and Hao Zhou. 1996. Curve fitting with Bezier cubics. *Graphical models and image processing* 58, 3 (1996), 223–232.

Arthur van Goethem, Wouter Meulemans, Andreas Reimer, Herman Haverkort, and Bettina Speckmann. 2013. Topologically safe curved schematization. *The Cartographic Journal* 50, 3 (2013), 276–285.

Libor Váša and Václav Skala. 2009. Cobra: Compression of the basis for pca represented animations. In *Computer Graphics Forum*, Vol. 28. Wiley Online Library, 1529–1540.

Jiayi Eris Zhang, Seungbae Bang, David IW Levin, and Alec Jacobson. 2020. Complementary dynamics. *arXiv preprint arXiv:2009.02462* (2020).

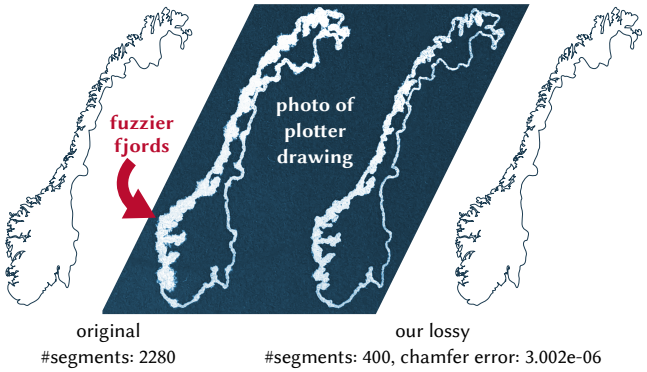


Figure 7: Attempting to draw a dense spline with a robotic plotting machine results more ink bleeding than plotting its simplification.

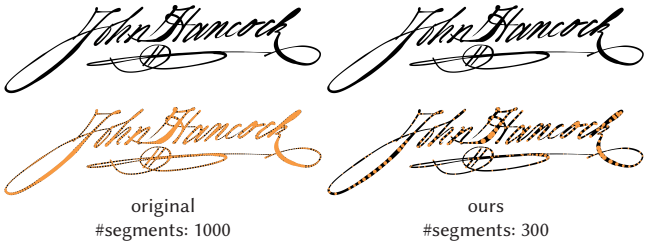


Figure 8: We are able to simplify a "3D" curve by treating the stroke width as an extra dimension. The bottom row visualizes stroke radii (3rd dimension) as an orange circle at each interpolated control point. We show that the hand-written "John Hancock" with 1000 segments can be represented by 300 curves without losing noticeable accuracy.

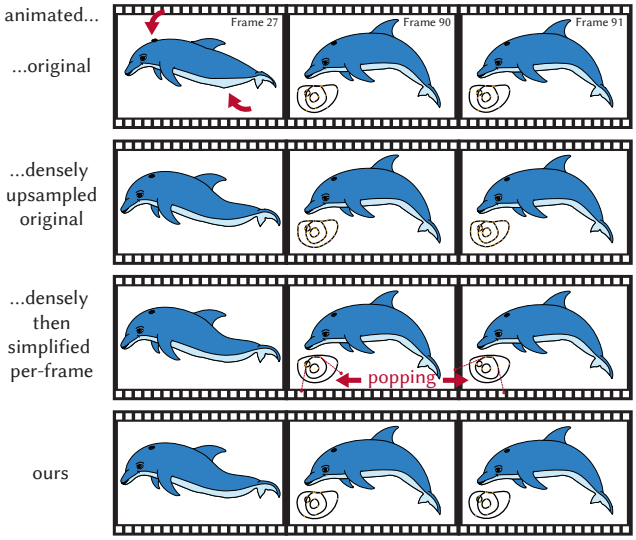


Figure 9: Naively applying animations to vector graphics either ends up with artifacts or too many segments. Simplifying per-frame produces popping artifacts, while our method is temporally coherent.

Figure 10: Our simplification results on stock images from the Internet. Our algorithm losslessly simplified the images to around 75-90% and managed to simplified the images further to 40% or less than 40% of its original number of segments without noticeable visual difference.