GL-k curves: a family of polynomial curves for intuitive modelling

Andriamahenina Ramanantoanina · Kai Hormann

Abstract

Bézier curves are typically used for interactively designing polynomial curves, but it is well known that this becomes increasingly unintuitive as the degree of the curve grows. Gauss–Legendre (GL) curves have been proposed as an alternative for modelling high-degree polynomial curves. They come with the advantage of preserving a close relationship between the curve and its control polygon, but are costly to evaluate in their native form. We first show how to express GL curves in the Legendre basis. Building on this and taking advantage of the three-term recurrence relation of Legendre polynomials, we explore a linear-time algorithm for evaluating GL curves. We further introduce a more general family of GL-k curves, with GL curves corresponding to the case $k = 0$ and observe that GL-1 curves are often remarkably similar to cubic B-spline curves. Unlike Bézier curves, the start and the end of a GL-k curve are not tangent to the first and the last edge of the control polygon, respectively, and we present a strategy for restoring this property.

Citation Info

Journal
Computer Aided Geometric Design

Volume
127, June 2026

Article
102558, 12 pages

Note
Proceedings of GMP

DOI
10.1016/j.cagd.2026.102558

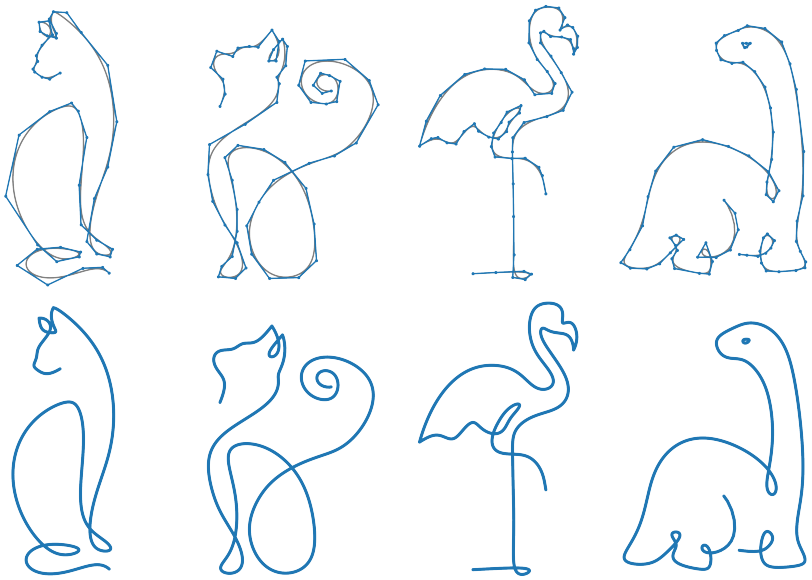

Figure 1: Examples of line art (bottom) and the corresponding control polygons (top), designed using GL-1 curves.

1 Introduction

For the purpose of curve design, a polynomial curve p is commonly expressed in Bézier form. However, as illustrated in Figure 2 (left), modelling high-degree curves using Bézier control polygons is often not very intuitive, because the relation between curve and control polygon weakens as the degree increases. Two general approaches have been developed to address this challenge. On the one hand, we can give up the concept of a single polynomial curve and work with piecewise polynomial curves instead, leading to the well-established use of splines (e.g., B-splines [15], κ-curves [20], and their G^2 alternatives [8]). On the other hand, we can give up one or more of the properties of the Bernstein basis (e.g., the non-negativity that gives rise to the *convex hull property* of Bézier curves) and adopt alternative polynomial bases. In this paper, we focus on the second option (Figure 1).

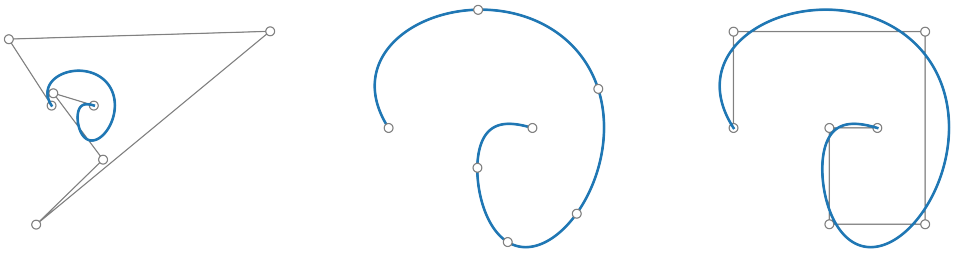

Figure 2: Example of the same polynomial curve of degree 6, represented as a Bézier curve (left), as an interpolatory curve (middle), and as a GL curve (right).

One such alternative basis are the Lagrange polynomials, which allow the curve to be expressed in terms of control points that are interpolated by p (see Figure 2, middle). However, high-degree Lagrange polynomials can suffer from significant oscillations near the endpoints, especially when uniformly spaced nodes are used, a behaviour known as Runge's phenomenon [17]. As a result, even small adjustments of the control points can create large and undesirable artefacts near the endpoints. One can attempt to reduce these oscillations using distance-based node distributions such as centripetal [11] or chordal parameterizations [6], but producing visually pleasing curves still requires a very skilled and experienced designer.

Another option is provided by the recently introduced Gauss–Legendre curves [9, 12, 13], which are defined by control polygons that are geometrically close to the curve, even for high degree (see Figure 2, right). A planar Gauss–Legendre (GL) curve is a polynomial curve $p:[-1,1]\to\mathbb{R}^2$ of degree n, defined by $n+1$ control points $p_0,\dots,p_n\in\mathbb{R}^2$, such that the tangent vectors $p'(\tau_i)$ at the roots $\tau_0,\dots,\tau_{n-1}$ of the degree-n Legendre polynomial P_n are parallel to the edge vectors $v_i=p_{i+1}-p_i$. A GL curve can be expressed as

$$p(t)=\sum_{i=0}^{n} p_i F_i^n(t), \qquad F_i^n(t)=G_{i-1}^n(t)-G_i^n(t), \quad i=0,\dots,n,$$

where

$$G_{-1}^n(t)=\frac{1}{2}, \qquad G_i^n(t)=\frac{\int_{-1}^{t}\ell_i(s)\,\mathrm{d}s}{\int_{-1}^{1}\ell_i(s)\,\mathrm{d}s}-\frac{1}{2}, \quad i=0,\dots,n-1, \qquad G_n^n(t)=-\frac{1}{2} \tag{1}$$

and the functions ℓ_i are the Lagrange polynomials

$$\ell_i(t)=\prod_{j=0,j\neq i}^{n-1}\frac{t-\tau_j}{\tau_i-\tau_j}, \quad i=0,\dots,n-1. \tag{2}$$

A natural property that is desired when designing with control polygons is that the curve p starts at p_0 and ends at p_n. This *endpoint interpolation property* is satisfied by GL curves. Indeed, since $G_i^n(-1)=-\frac{1}{2}$ for $i=0,\dots,n$, we find that $F_0^n(-1)=1$ and $F_i(-1)=0$ for $i=1,\dots,n$, hence $p(-1)=p_0$. Similarly, since $G_i^n(1)=\frac{1}{2}$ for $i=-1,\dots,n-1$, we deduce that $p(1)=p_n$. Moreover, one can verify that

$$p'(\tau_i)=\frac{v_i}{\int_{-1}^{1}\ell_i(s)\,\mathrm{d}s}, \quad i=0,\dots,n-1. \tag{3}$$

Note that unlike Bézier curves, the tangent vectors of p at $t=-1$ and $t=1$ are not parallel to v_0 and v_{n-1}, that is, GL curves do not have the *endpoint tangent property*. This property can be obtained by using an alternative quadrature-based polynomial basis [10].

1.1 Contributions

The evaluation of a GL curve involves the computation of definite integrals in (1), and a naive algorithm requires $O(n^3)$ operations. In Section 2, we show how to express GL curves in the Legendre basis. We then take advantage of this expression to develop a linear-time evaluation algorithm in Section 3. In Section 4, inspired by [16], we extend the construction of GL curves to derive a whole family of GL-k curves (for $k\in\mathbb{N}_0$), and we prove some properties of these curves in Section 5. Finally, in Section 6, we develop a technique that allows for manipulating the curve's tangents at the endpoints and restores the endpoint tangent property of GL-k curves.

2 GL curves in the Legendre basis

While the polynomials G_i^n in (1) can be evaluated by means of quadrature rules, this becomes computationally expensive as the degree n grows. We propose instead to express GL curves in the Legendre basis. This is motivated by the fact that, contrary to Lagrange polynomials, simple formulas for the integrals of the Legendre polynomials P_i are known [2, Eq. (12.23)], namely

$$\int_{-1}^{t} P_i(s)\,\mathrm{d}s = \frac{P_{i+1}(s)-P_{i-1}(s)}{2i+1}\bigg|_{-1}^{t} = \frac{P_{i+1}(t)-P_{i-1}(t)}{2i+1} + \delta_{0,i}, \quad i \geq 0, \tag{4}$$

where $\delta_{i,j}$ is the Kronecker delta and following the convention that $P_{-1}(t) = 0$. Hence, it suffices to express the Lagrange polynomials ℓ_i in (2) in terms of Legendre polynomials and to then derive the coefficients of p in that basis.

Proposition 1. *Let $i, n, \in \mathbb{N}_0$ with $0 \leq i < n$ and consider the Lagrange polynomial ℓ_i in (2). Then,*

$$\ell_i(t) = \sum_{j=0}^{n-1} c_j P_j(t), \qquad c_j = \frac{(2j+1)P_j(\tau_i)}{\sum_{k=0}^{n-1}(2k+1)P_k(\tau_i)^2}, \quad j = 0, \ldots, n-1,$$

where $\tau_0, \ldots, \tau_{n-1}$ are the roots of P_n.

Proof. First, recall [7] that we can express $\ell_i(t)$ as

$$\ell_i(t) = \frac{P_n(t)}{(t-\tau_i)P_n'(\tau_i)}, \quad i = 0, \ldots, n-1. \tag{5}$$

Next, consider the Christoffel–Darboux kernel $K_{n-1}(t,s)$ [18, Eq. (1.12)], defined in terms of the Legendre polynomials,

$$K_{n-1}(t,s) = \frac{1}{2}\sum_{j=0}^{n-1}(2j+1)P_j(t)P_j(s). \tag{6}$$

The Christoffel–Darboux formula [1, Eq. (22.12.1)] then states that

$$K_{n-1}(t,s) = \frac{2(n-1)+1}{2} \cdot \frac{a_{n-1}}{a_n} \cdot \frac{P_{n-1}(s)P_n(t) - P_n(s)P_{n-1}(t)}{t-s},$$

where $a_k = \frac{1}{2^k}\binom{2k}{k}$ is the leading coefficient of P_k. Since τ_i is a root of P_n, we have

$$K_{n-1}(t,\tau_i) = \lambda_n \frac{P_n(t)}{t-\tau_i}, \qquad K_{n-1}(\tau_i,\tau_i) = \lambda_n P_n'(\tau_i), \qquad \lambda_n = \frac{2(n-1)+1}{2} \cdot \frac{a_{n-1}}{a_n} P_{n-1}(\tau_i), \tag{7}$$

and we finally deduce from (5) and (7) that $\ell_i(t)$ is given by the ratio

$$\ell_i(t) = \frac{K_{n-1}(t,\tau_i)}{K_{n-1}(\tau_i,\tau_i)}. \tag{8}$$

The statement then follows from (6). □

Using Proposition 1 and Equations (1) and (4), we can now express G_i^n in the Legendre basis.

Proposition 2. *Let $i, n, \in \mathbb{N}_0$ with $0 \leq i < n$. Then, $G_i^n(t)$ in (1) can be written as*

$$G_i^n(t) = \frac{1}{2}\left(\sum_{j=0}^{n-2}\big(P_{j-1}(\tau_i) - P_{j+1}(\tau_i)\big)P_j(t) + \sum_{j=n-1}^{n} P_{j-1}(\tau_i)P_j(t)\right).$$

Proof. We first notice from (1) and (8) that

$$G_i^n(t) + \frac{1}{2} = \frac{\int_{-1}^{t}\ell_i(s)\,\mathrm{d}s}{\int_{-1}^{1}\ell_i(s)\,\mathrm{d}s} = \frac{\int_{-1}^{t}K_{n-1}(s,\tau_i)\,\mathrm{d}s}{\int_{-1}^{1}K_{n-1}(s,\tau_i)\,\mathrm{d}s}.$$

It further follows from (4) and (6) that

$$\begin{aligned}
2\int_{-1}^{t} K_{n-1}(s,\tau_i)\,\mathrm{d}s &= \sum_{j=0}^{n-1}(2j+1)P_j(\tau_i)\int_{-1}^{t} P_j(s)\,\mathrm{d}s \\
&= 1+\sum_{j=0}^{n-1} P_j(\tau_i)\big(P_{j+1}(t)-P_{j-1}(t)\big) \\
&= 1+\sum_{j=1}^{n} P_{j-1}(\tau_i)P_j(t)-\sum_{j=-1}^{n-2} P_{j+1}(\tau_i)P_j(t) \\
&= 1-\tau_i+\sum_{j=1}^{n-2}\big(P_{j-1}(\tau_i)-P_{j+1}(\tau_i)\big)P_j(t)+\sum_{j=n-1}^{n} P_{j-1}(\tau_i)P_j(t)
\end{aligned}$$

and similarly

$$\begin{aligned}
2\int_{-1}^{1} K_{n-1}(s,\tau_i)\,\mathrm{d}s &= 1-\tau_i+\sum_{j=1}^{n-2}\big(P_{j-1}(\tau_i)-P_{j+1}(\tau_i)\big)P_j(1)+\sum_{j=n-1}^{n} P_{j-1}(\tau_i)P_j(1) \\
&= 2-\sum_{j=0}^{1} P_{j+1}(\tau_i)+\sum_{j=2}^{n-2}\big(P_{j-1}(\tau_i)-P_{j+1}(\tau_i)\big)+\sum_{j=n-1}^{n} P_{j-1}(\tau_i) \\
&= 2.
\end{aligned}$$

Hence, we deduce that

$$G_i^n(t)+\frac{1}{2}=\frac{1}{2}\left(1+\sum_{j=0}^{n-2}\big(P_{j-1}(\tau_i)-P_{j+1}(\tau_i)\big)P_j(t)+\sum_{j=n-1}^{n} P_{j-1}(\tau_i)P_j(t)\right).$$

□

Finally, it remains to define the coefficients

$$g_{i,j}=\frac{1}{2}\big(P_{j-1}(\tau_i)-P_{j+1}(\tau_i)\big),\quad j=0,\ldots,n-2,\qquad g_{i,n-1}=\frac{1}{2}P_{n-2}(\tau_i),\qquad g_{i,n}=\frac{1}{2}P_{n-1}(\tau_i)$$

for $i=0,\ldots,n-1$ and

$$g_{-1,j}=\frac{1}{2}\delta_{0,j},\qquad g_{n,j}=-\frac{1}{2}\delta_{0,j},\qquad j=0,\ldots,n,$$

so that the curve p can then be expressed in the Legendre basis as

$$p(t)=\sum_{j=0}^{n} c_j P_j(t),\qquad c_j=\sum_{i=0}^{n}(g_{i-1,j}-g_{i,j})p_i. \tag{9}$$

3 Linear-time evaluation

Given a GL curve p with control points $\boldsymbol{p}=(p_0,\ldots,p_n)^T$, the alternative representation in (9) is nothing other than a change from the GL basis $\boldsymbol{F}^n=(F_0^n,\ldots,F_n^n)^T$ to the Legendre basis $\boldsymbol{P}^n=(P_0,\ldots,P_n)^T$, and the transformation matrix of this basis change is the matrix $M_n\in\mathbb{R}^{(n+1)\times(n+1)}$ with entries $m_{j,i}=g_{i-1,j}-g_{i,j}$ for $j,i=0,\ldots,n$. That is, instead of writing the curve as $p=\boldsymbol{p}^T\boldsymbol{F}^n$, we express it instead as $p=\boldsymbol{c}^T\boldsymbol{P}^n$, where $\boldsymbol{c}=(c_0,\ldots,c_n)^T$ are the curve's Legendre coefficients, which can be written as $\boldsymbol{c}=M_n\boldsymbol{p}$.

Clearly, this conversion requires $O(n^2)$ operations, but it needs to be computed only once, whenever the number of control points changes. Afterwards, if the user interactively moves some control point p_i by some displacement vector Δ to $p_i+\Delta$, we keep $\boldsymbol{c}$ up-to-date by simply adding $\boldsymbol{m}_i\Delta^T$ to $\boldsymbol{c}$, where $\boldsymbol{m}_i$ denotes the i-th column of M_n, which can be done in $O(n)$ time.

The major advantage of the Legendre representation in (9) is that it can be used to compute $p(t)$ in $O(n)$ time for any $t\in[-1,1]$. To this end, recall that the Legendre polynomials obey the three-term recurrence relation [1, Eq. (8.5.3)],

$$P_0(t)=1,\qquad P_1(t)=t,\qquad P_i(t)=\alpha_i\,t\,P_{i-1}(t)-\beta_i P_{i-2}(t),\quad i\geq 2,$$

Algorithm 1 GLEVAL$(c_0,\dots,c_n,t)$

$r_2 \leftarrow 1$
$r_1 \leftarrow t$
$p \leftarrow c_0 + r_1 \cdot c_1$
for $j = 2,\dots,n$ **do**
  $r_j \leftarrow \alpha_j \cdot t \cdot r_1 - \beta_j \cdot r_2$
  $p \leftarrow p + r_j \cdot c_j$
  $r_2 \leftarrow r_1$
  $r_1 \leftarrow r_j$
return p

where the constants $\alpha_i = \frac{2i-1}{i}$ and $\beta_i = \frac{i-1}{i}$ for $i \geq 2$ can be precomputed and hard-coded or read from a file. We can then evaluate p with Algorithm 1 in linear time. For curves in $\mathbb{R}^2$, this algorithm requires $8n-4$ floating point operations. Hence, it performs almost as well as the most efficient polynomial evaluation algorithms, for example, evaluation in barycentric form, which requires $7n+9$ operations [4].

4 GL-*k* curves

Modelling with GL curves is intuitive, but the curves have the tendency to "overshoot" the control polygon by quite some margin (see Figure 2, right), which is caused by the fact that the GL basis functions $\boldsymbol{F}^n$ have rather big negative values (see Figure 4, left). The motivation for our work lies in alleviating this effect.

For closed curves over $[0,2\pi]$, Ramanantoanina and Hormann [16] explore a trigonometric analogue of GL curves and modify them as follows. While trigonometric GL curves have tangent vectors at the equidistant nodes $\psi_i = \frac{2i+1}{n+1}\pi$ that are parallel to the edge vectors $v_i = p_{i+1} - p_i$, the modified curves have tangent vectors at the dual nodes $\phi_i = \frac{2i}{n+1}\pi$ that are parallel to the *averages* of two successive edge vectors, $v_i^1 = (v_{i-1} + v_i)/2 = (p_i + p_{i+1})/2 - (p_{i-1} + p_i)/2$. Our goal now is to develop and extend a similar modification for polynomial GL curves, which gives rise to the family of GL-k curves, where $k \in \mathbb{N}_0$.

Given the $n+1$ control points $p_0,\dots,p_n \in \mathbb{R}^2$, we recursively define the points

$$p_0^k = p_0^{k-1}, \qquad p_i^k = \frac{p_{i-1}^{k-1} + p_i^{k-1}}{2}, \quad i = 1,\dots,n+k-1, \qquad p_{n+k}^k = p_{n+k-1}^{k-1},$$

for $k = 1,2,\dots$, where $p_i^0 = p_i$ for $i = 0,\dots,n$. Clearly, this averaging process can be written more compactly as $\boldsymbol{p}^k = A_k \boldsymbol{p}^{k-1}$, where

$$\boldsymbol{p}^k = (p_0^k,\dots,p_{n+k}^k)^T, \qquad A_k = \frac{1}{2}\begin{pmatrix} 2 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 0 \\ & \ddots & \ddots & \\ 0 & \cdots & 1 & 1 \\ 0 & \cdots & 0 & 2 \end{pmatrix} \in \mathbb{R}^{(n+k+1)\times(n+k)} \tag{10}$$

and $\boldsymbol{p}^0 = \boldsymbol{p}$. Now let p^k be the GL curve defined by $\boldsymbol{p}^k = (p_0^k,\dots,p_{n+k}^k)^T$, that is, $p^k = (\boldsymbol{p}^k)^T \boldsymbol{F}^{n+k}$. It is clear that the tangents of p^k at the roots of P_{n+k} are parallel to the edge vectors $v_i^k = p_{i+1}^k - p_i^k$ for $i = 0,\dots,n+k-1$, and we observe that the overshooting effect diminishes as k grows (see Figure 3, second row). This is essentially what we were aiming for, but p^k is a curve of degree $n+k$, even though it depends only on the $n+1$ control points $\boldsymbol{p}$. The core idea of GL-k curves now is to approximate the GL curve p^k with a polynomial curve of degree n that is as close as possible to p^k and interpolates the same endpoints (see Figure 3, third row).

Definition 1. Let $\mathcal{R}_k : \Pi_{n+k} \to \Pi_n$ be the *reduction operator* that projects any polynomial f of degree $n+k$ to the polynomial g of degree n that is closest to f in the L^2-norm, subject to the endpoint constraints $g(-1) = f(-1)$ and $g(1) = f(1)$. The *GL-k curve* c^k is then defined as $c^k = \mathcal{R}_k[p^k]$.

The next lemma reveals that the operator $\mathcal{R}_k$ has a rather simple representation matrix with respect to the Legendre basis, which can be combined nicely with the Legendre representation of GL curves in (9).

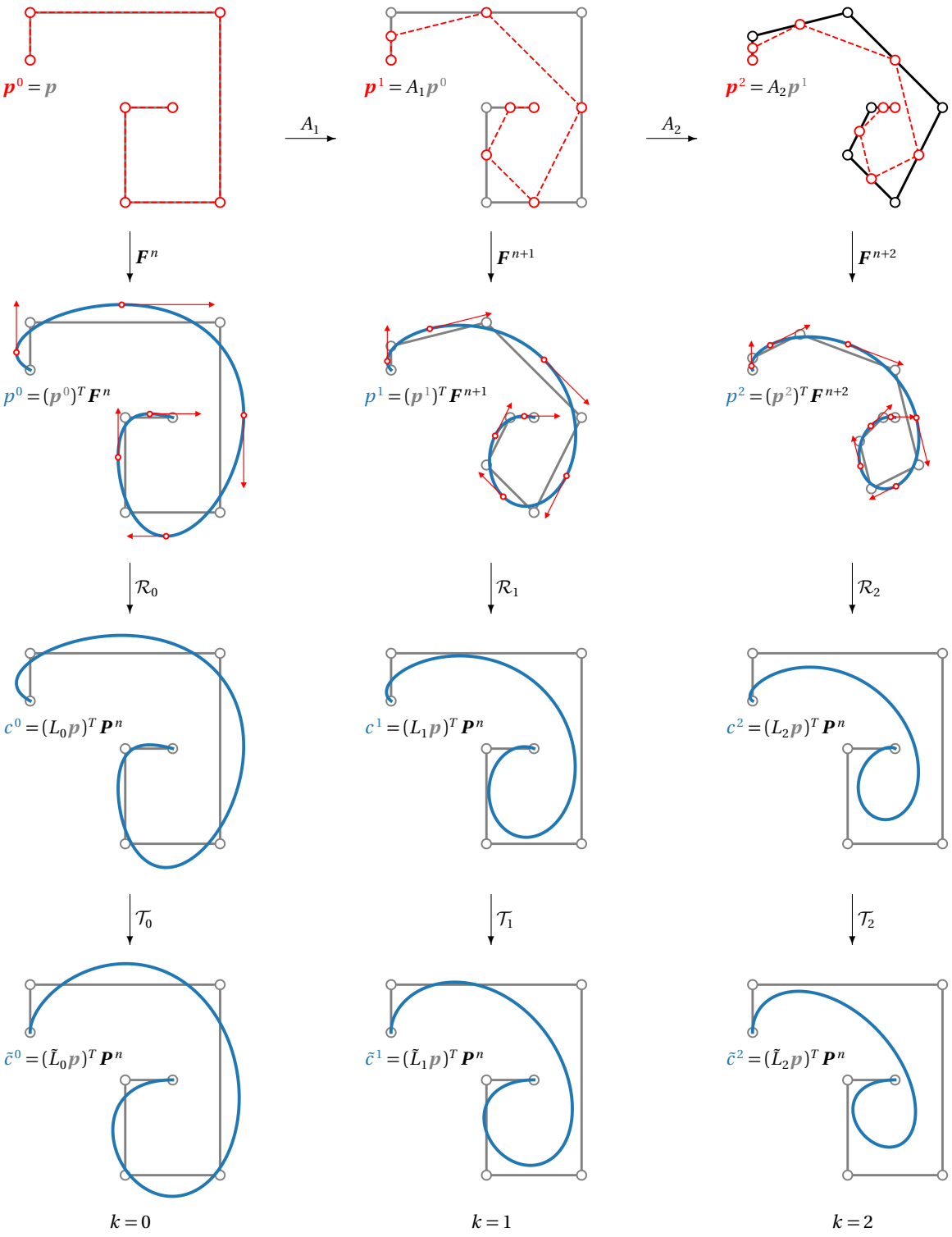

Figure 3: Overview of the proposed construction of GL-k curves. Given a control polygon $\boldsymbol{p}$, we smooth it k times (top, from left to right) by averaging neighbouring points. The GL curve p^k of degree $n+k$, defined by the resulting control polygon $\boldsymbol{p}^k$ (second row), is then projected to the space of degree-n polynomials with the same endpoints, using the reduction operator $\mathcal{R}_k$, to give the GL-k curve c^k (third row). Finally, the modified GL-k curve $\tilde{c}^k$ (last row) can be generated by using the tangent operator $\mathcal{T}_k$ that finds the degree-n curve with the endpoint tangent property that is closest to c^k.

Lemma 3. *For any $k \in \mathbb{N}$, let $m = n + k$ and let $f \in \Pi_m$ by a degree-m polynomial with Legendre coefficients $\boldsymbol{f} = (f_0, \dots, f_m)^T$. Then the Legendre coefficients $\boldsymbol{g} = (g_0, \dots, g_n)^T$ of the degree-n polynomial $g = \mathcal{R}_k[f] \in \Pi_n$ are*

$$g_i = f_i + \sum_{j=1}^{k} s_{ij} f_{n+j}, \quad i = 0, \dots, n,$$

where

$$s_{ij} = \frac{4i+2}{n+1} \cdot \begin{cases} 0, & \text{if } n+i+j \text{ is odd,} \\ \frac{1}{n}, & \text{if } n+i+j \text{ is even and } j \text{ is odd,} \\ \frac{1}{n+2}, & \text{if } n+i+j \text{ is even and } j \text{ is even.} \end{cases} \tag{11}$$

Hence, the representation matrix of $\mathcal{R}_k$ with respect to the Legendre basis is

$$R_k = [\, I_{n+1} \,|\, S \,] \in \mathbb{R}^{(n+1)\times(n+1+k)}, \tag{12}$$

where I_{n+1} is the identity matrix of size $n+1$ and the coefficients of $S \in \mathbb{R}^{(n+1)\times k}$ are the $s_{i,j}$ in (11) for $i = 0, \dots, n$ and $j = 1, \dots, k$.

Proof. Let $\tilde{\boldsymbol{f}} = (f_0, \dots, f_n)^T$ be the first $n+1$ Legendre coefficients of f. It is then well-known that $\tilde{f} \in \Pi_n$ with $\tilde{f}(t) = \sum_{i=0}^{n} f_i P_i(t)$ is the best L^2-approximation of f in Π_n [3]. Therefore, instead of minimizing $\|g - f\|_2$ to find g as the closest degree-n polynomial to f in the L^2-norm, we can determine g by minimizing $\|g - \tilde{f}\|_2$, subject to the endpoint constraints. By the linearity of the L^2-inner product,

$$\|g - \tilde{f}\|_2^2 = \langle g - \tilde{f}, g - \tilde{f} \rangle_2 = \sum_{i=0}^{n} \sum_{j=0}^{n} (g_i - f_i)(g_j - f_j) \langle P_i, P_j \rangle_2,$$

and since Legendre polynomials are orthogonal with respect to this inner product and satisfy [2, Eq. (12.47)]

$$\langle P_i, P_j \rangle_2 = \int_{-1}^{1} P_i(s) P_j(s) \, \mathrm{d}s = \frac{2}{2i+1} \delta_{i,j},$$

we have

$$\|g - \tilde{f}\|_2^2 = \sum_{i=0}^{n} (g_i - f_i)^2 \frac{2}{2i+1} = (\boldsymbol{g} - \tilde{\boldsymbol{f}})^T W (\boldsymbol{g} - \tilde{\boldsymbol{f}}),$$

where $W \in \mathbb{R}^{(n+1)\times(n+1)}$ is the diagonal matrix

$$W = \operatorname{diag}(w_0, \dots, w_n), \qquad w_i = \frac{2}{2i+1}. \tag{13}$$

Since $P_i(-1) = (-1)^i$ and $P_i(1) = 1$ for any $i \geq 0$, we can write the endpoint constraints compactly as $C_{n+1} \boldsymbol{g} = C_{m+1} \boldsymbol{f}$, where

$$C_k = \begin{pmatrix} 1 & -1 & \cdots & (-1)^{k-1} \\ 1 & 1 & \cdots & 1 \end{pmatrix} \in \mathbb{R}^{2\times k}, \quad k \in \mathbb{N}.$$

The Legendre coefficients $\boldsymbol{g}$ of the constrained L^2-minimizer $g = \mathcal{R}_k[f]$ are then given as the stationary point of the Lagrangian function

$$\mathcal{L}(\boldsymbol{g}, \lambda) = (\boldsymbol{g} - \tilde{\boldsymbol{f}})^T W (\boldsymbol{g} - \tilde{\boldsymbol{f}}) + \lambda^T (C_{n+1} \boldsymbol{g} - C_{m+1} \boldsymbol{f})$$

with Lagrangian multiplier $\lambda \in \mathbb{R}^2$. Setting the partial derivatives of $\mathcal{L}$ with respect to $\boldsymbol{g}$ and λ to zero, we get the two equations

$$2W(\boldsymbol{g} - \tilde{\boldsymbol{f}}) + C_{n+1}^T \lambda = 0, \qquad C_{n+1} \boldsymbol{g} - C_{m+1} \boldsymbol{f} = 0$$

and solving them in $\boldsymbol{g}$ and λ, we find that

$$\boldsymbol{g} = \tilde{\boldsymbol{f}} + W^{-1} C_{n+1}^T \big(C_{n+1} W^{-1} C_{n+1}^T \big)^{-1} (C_{m+1} \boldsymbol{f} - C_{n+1} \tilde{\boldsymbol{f}}).$$

Since $\tilde{\boldsymbol{f}} = [\, I_{n+1} \,|\, \boldsymbol{0}_k \,] \boldsymbol{f}$, where $\boldsymbol{0}_k$ is the zero matrix of size $(n+1) \times k$, and $C_{m+1} = [\, C_{n+1} \,|\, C_k' \,]$, where $C_k' = \operatorname{diag}((-1)^{n+1}, 1) \cdot C_k$, we can write $\boldsymbol{g}$ as

$$\boldsymbol{g} = [\, I_{n+1} \,|\, \boldsymbol{0}_k \,] \boldsymbol{f} + V \big([\, C_{n+1} \,|\, C_k' \,] - C_{n+1} [\, I_{n+1} \,|\, \boldsymbol{0}_k \,] \big) \boldsymbol{f} = [\, I_{n+1} \,|\, W^{-1} V \,] \boldsymbol{f},$$

Algorithm 2 LEGENDREREDUCE$(k, \boldsymbol{f})$

$m \leftarrow \text{length}(\boldsymbol{f})-1$
$n \leftarrow m-k$
$\sigma_e, \sigma_o \leftarrow 0$
for $j = 1,\dots,k$ **do**
  if $n+j$ is even **then**
    $\sigma_e \leftarrow \sigma_e + f_{n+j}$
  else
    $\sigma_o \leftarrow \sigma_o + f_{n+j}$
if n is even **then**
  $\sigma_e \leftarrow \sigma_e/(n+1)/(n+2)$
  $\sigma_o \leftarrow \sigma_o/n/(n+1)$
else
  $\sigma_e \leftarrow \sigma_e/n/(n+1)$
  $\sigma_o \leftarrow \sigma_o/(n+1)/(n+2)$
for $i = 0,\dots,n$ **do**
  if i is even **then**
    $g_i \leftarrow f_i + (4i+2)\sigma_e$
  else
    $g_i \leftarrow f_i + (4i+2)\sigma_o$
return $\boldsymbol{g}$

where $V = C_{n+1}^T \left(C_{n+1} W^{-1} C_{n+1}^T\right)^{-1} C_k' \in \mathbb{R}^{(n+1)\times k}$. To find the coefficients of V, note that

$$W^{-1} = \operatorname{diag}\left(\frac{1}{w_0},\dots,\frac{1}{w_n}\right),$$

hence

$$C_{n+1} W^{-1} C_{n+1}^T = \begin{pmatrix} \sum_{i=0}^n \frac{2i+1}{2}(-1)^{2i} & \sum_{i=0}^n \frac{2i+1}{2}(-1)^i \\ \sum_{i=0}^n \frac{2i+1}{2}(-1)^i & \sum_{i=0}^n \frac{2i+1}{2} \end{pmatrix} = \frac{n+1}{2}\begin{pmatrix} n+1 & (-1)^n \\ (-1)^n & n+1 \end{pmatrix}$$

and

$$\left(C_{n+1} W^{-1} C_{n+1}^T\right)^{-1} = \frac{2}{n(n+1)(n+2)}\begin{pmatrix} n+1 & (-1)^{n+1} \\ (-1)^{n+1} & n+1 \end{pmatrix}.$$

Multiplying from the left with C_{n+1}^T and from the right with C_k' gives

$$V = \frac{4}{n(n+1)(n+2)}\begin{pmatrix} \boldsymbol{v}_0 \\ \vdots \\ \boldsymbol{v}_n \end{pmatrix}$$

with row vectors

$$\boldsymbol{v}_i = \begin{cases} (n+2,0,n+2,0,\dots), & \text{if } n+i \text{ is odd,} \\ (0,n,0,n,\dots), & \text{if } n+i \text{ is even,} \end{cases}$$

which means that the coefficients $v_{i,j}$ of V for $i = 0,\dots,n$ and $j = 1,\dots,k$ are

$$v_{i,j} = \frac{4}{n+1}\begin{cases} 0, & \text{if } n+i+j \text{ is odd,} \\ \frac{1}{n}, & \text{if } n+i+j \text{ is even and } j \text{ is odd,} \\ \frac{1}{n+2}, & \text{if } n+i+j \text{ is even and } j \text{ is even.} \end{cases}$$

The statement then follows by noting the $s_{i,j}$ in (11) are the coefficients of $S = W^{-1}V$. □

Note that the reduction operator $\mathcal{R}_k$ with respect to the Legendre basis can be implemented with $O(n+k)$ instructions, as shown in Algorithm 2.

Let us summarize the construction of GL-k curves (see Figure 3). Given the $n+1$ control points $\boldsymbol{p} = (p_0,\dots,p_n)^T$, we average these points k times to get the points $\boldsymbol{p}^k = A_k \cdots A_1 \boldsymbol{p}$. We then consider the GL curve p^{n+k} for these points, with coefficients $M_{n+k}\boldsymbol{p}^k$ in the Legendre basis. We finally degree-reduce the

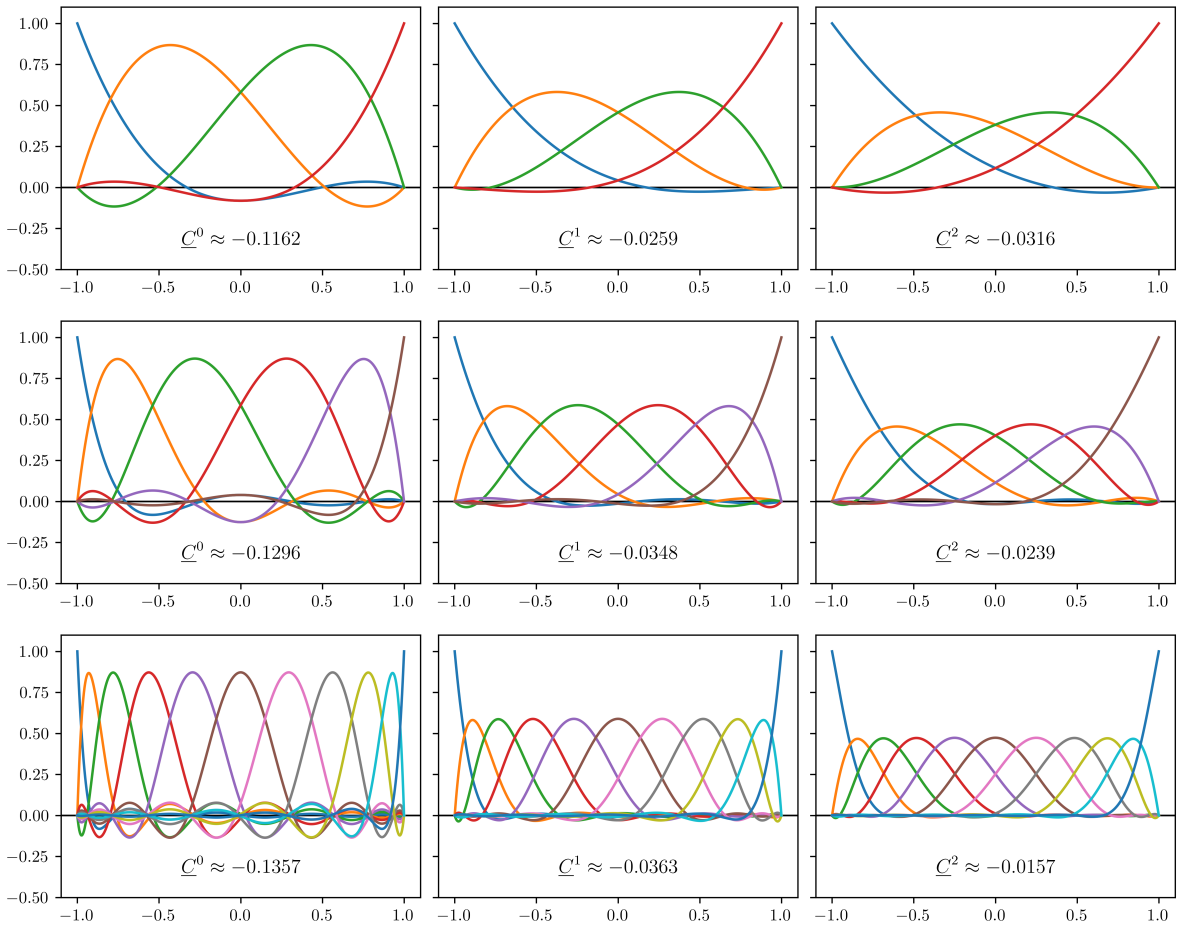

Figure 4: Basis functions $C_0^k, \ldots, C_n^k$ of GL-k curves for degrees $n = 3, 5, 10$ (top to bottom) and $k = 0, 1, 2$ (left to right).

latter with the reduction operator $\mathcal{R}_k$ to get the Legendre coefficients $\boldsymbol{c}^k = L_k \boldsymbol{p}$ of the GL-k curve c^k, where the matrix $L_k = R_k M_{n+k} A_k \cdots A_1 \in \mathbb{R}^{(n+1)\times(n+1)}$ can be precomputed. As for GL curves, the initial conversion from $\boldsymbol{p}$ to $\boldsymbol{c}^k$ requires $O(n^2)$ operations, any subsequent change of some control point p_i leads to an $O(n)$ update of $\boldsymbol{c}^k$, and the actual evaluation of c^k happens in linear time with Algorithm 1. Note that the GL-0 curve c^0 is identical to the GL curve p^n, because $\mathcal{R}_0$ is just the identity.

5 Properties of GL-k curves

So far, we have considered GL-k curves in terms of the Legendre coefficients $\boldsymbol{c}^k$ with respect to the Legendre basis,

$$c^k(t) = (\boldsymbol{c}^k)^T \boldsymbol{P}^n = \sum_{i=0}^{n} c_i^k P_i(t),$$

but since $\boldsymbol{c}^k = L_k \boldsymbol{p}$, we can also express them in terms of the control points $\boldsymbol{p}$ with respect to the functions $\boldsymbol{C}^k = (C_0^k, \ldots, C_n^k)^T = (L_k)^T \boldsymbol{P}^n$ as

$$c^k(t) = \boldsymbol{p}^T \boldsymbol{C}^k = \sum_{i=0}^{n} p_i C_i^k(t).$$

We shall now show two important properties of the functions C_i^k.

Proposition 4. *The functions $C_0^k, \ldots, C_n^k$ are polynomials and form a basis of Π_n.*

Proof. It is clear that the functions $C_0^k, \ldots, C_n^k$ are polynomials of degree n, because they are linear combinations of the Legendre polynomials $P_0, \ldots, P_n$. It remains to show that $L_k = R_k M_{n+k} A_k \cdots A_1$ has full rank

and thus is a change-of-basis matrix. However, this follows from the facts that the averaging matrices A_k in (10) have full column rank, the reduction matrix R_k in (12) has full row rank, and the matrix M_{n+k} which describes the basis change from the GL basis to the Legendre basis has full rank [13]. □

Proposition 5. *The polynomials $C_0^k,\dots,C_n^k$ form a partition of unity.*

Proof. Using the notation of Proposition 4, we need to show that $\mathbf{1}_{n+1}^T \boldsymbol{C}^k = (\boldsymbol{C}^k)^T \mathbf{1}_{n+1} = 1$, where $\mathbf{1}_{n+1} = (1,\dots,1)^T \in \mathbb{R}^{n+1}$. Using the change-of-basis matrix L_k and observing that the entries in each row of A_i sum to one, that is, $A_i \mathbf{1}_{n+i} = \mathbf{1}_{n+i+1}$, we have

$$(\boldsymbol{C}^k)^T \mathbf{1}_{n+1} = (\boldsymbol{P}^n)^T R_k M_{n+k} A_k \cdots A_1 \mathbf{1}_{n+1} = (\boldsymbol{P}^n)^T R_k M_{n+k} \mathbf{1}_{n+k+1}.$$

Now, recall that M_{n+k} is the change-of-basis matrix from the GL basis $\boldsymbol{F}^{n+k}$ to the Legendre basis $\boldsymbol{P}^{n+k}$ and that the polynomials $\boldsymbol{F}^{n+k}$ form a partition of unity [13]. Therefore, $M_{n+k}\mathbf{1}_{n+k+1}$ is the vector of Legendre coefficients of the constant 1 function, but as this function is just the Legendre polynomial P_0, this coefficient vector is just $e_0^{n+k} = (1,0,\dots,0)^T \in \mathbb{R}^{n+k+1}$. By the structure of R_k in (12), we then have

$$(\boldsymbol{C}^k)^T \mathbf{1}_{n+1} = (\boldsymbol{P}^n)^T R_k e_0^{n+k} = (\boldsymbol{P}^n)^T e_0^n = P_0,$$

which proves the claim. □

It follows from these properties that all polynomial curves of degree n can be described as GL-k curves and that every GL-k curve is invariant under affine transformations of its control polygon, which are two key properties that GL-k curves share with Bézier curves. Unlike Bézier curves, GL-k curves do not have the *convex hull property*, because the basis functions C_i^k are *not* non-negative (see Figure 4). However, we observe (and further experiments reveal) that the lower bound $\underline{C}^k = \min\{C_i^k(t) : i = 0,\dots,n,\, t \in [-1,1]\} < 0$ seems to converge rather quickly to 0 as k increases (at least if $n \geq 2k$). In other words, the basis functions C_i^k are obtained by forming convex combinations of the GL basis of degree $n+k$ and then projecting them back to degree n. The convex combinations increase the lower bound, while the back projection partially counteracts this effect. In practice, for small values of k, the influence of the back projection remains rather limited. Furthermore, unlike GL curves, the existence of a derivative vector parallel to each edge of the control polygon at very specific nodes is no longer guaranteed for GL-k curves with $k > 0$.

6 Endpoint tangents

Another common property, which is key for intuitive design, is the endpoint interpolation property. In contrast to Bézier curves, GL-k curves do not have the endpoint tangent property, that is, the vectors $(c^k)'(-1)$ and $(c^k)'(1)$ are not necessarily parallel to the vectors $p_1 - p_0$ and $p_n - p_{n-1}$, respectively.

To restore this property, we can follow the steps in the proof of Lemma 3 to define the *tangent operator* $\mathcal{T}_k$ that projects the GL-k curve c^k to the *modified* GL-k curve $\tilde{c}^k = \mathcal{T}_k[c^k] = \tilde{\boldsymbol{c}}^k \boldsymbol{P}^n$ which minimizes $\|\tilde{c}^k - c^k\|_2$, subject to the endpoint interpolation and the endpoint tangent property. Since $P_i'(-1) = (-1)^{i+1} d_i$ and $P_i'(1) = d_i$, where $d_i = i(i+1)/2$ (see [14, Eq. (18.9.15)] and [19, Eq. (4.1.1)]) and recalling that $\boldsymbol{c}^k = L_k \boldsymbol{p}$, we can write the constraints compactly as $D\tilde{\boldsymbol{c}}^k = E\boldsymbol{p}$, where

$$D = \begin{pmatrix} 1 & -1 & \cdots & (-1)^n \\ 1 & 1 & \cdots & 1 \\ -d_0 & d_1 & \cdots & (-1)^{n+1} d_n \\ d_0 & d_1 & \cdots & d_n \end{pmatrix}, \qquad E = \begin{pmatrix} \begin{pmatrix} 1 & -1 & \cdots & (-1)^n \\ 1 & 1 & \cdots & 1 \end{pmatrix} L_k \\ \begin{pmatrix} -\eta_1 & \eta_1 & 0 & \cdots & 0 & 0 \\ 0 & 0 & 0 & \cdots & -\eta_2 & \eta_2 \end{pmatrix} \end{pmatrix}$$

with $\eta_1, \eta_2 > 0$ denoting the scaling factors between the endpoint tangents of $\tilde{c}^k$ and the vectors $p_1 - p_0$ and $p_n - p_{n-1}$. As in the proof of Lemma 3, one can show that the solution of this constrained optimization problem is

$$\tilde{\boldsymbol{c}}^k = \tilde{L}_k \boldsymbol{p}, \qquad \tilde{L}_k = L_k + W^{-1} D^T \left(D W^{-1} D^T\right)^{-1} (E - D L_k),$$

with W as in (13). The corresponding curve $\tilde{c}^k$ depends on the scaling factors η_1 and η_2, and the actual choice can be left to the user. Based on our experiments, a reasonable default value is to set $\eta_1 = \eta_2 = 1/\omega_0$ where ω_0 is the first Legendre weight $\omega_0 = \int_{-1}^1 \ell_0(s)\mathrm{d}s$, so that $(\tilde{c}^k)'(-1) = p'(\tau_0)$ and $(\tilde{c}^k)'(1) = p'(\tau_{n-1})$ (cf. (3)). We used this default value in Figure 3 (last row) and Figure 8.

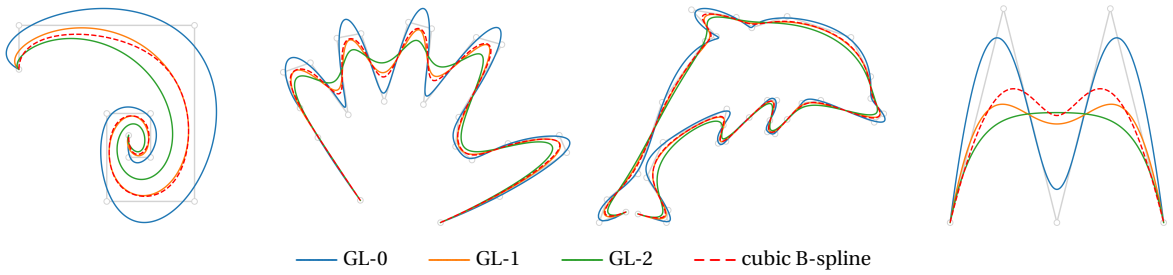

Figure 5: Comparison of GL-k curves for $k = 0, 1, 2$ and cubic B-spline curves with respect to uniform nodes.

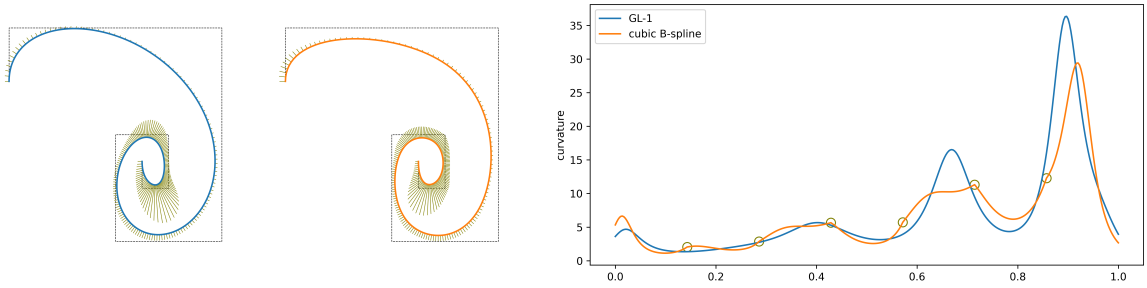

Figure 6: Curvature comb plots of a modified GL-1 (left) and a cubic B-spline curve (centre) with the same control polygon and plots of both curvature functions (right). The circles indicate where the curvature of the cubic B-spline curve is only C^0.

7 Results

We implemented GL-k (and modified GL-k) curves in `C++` on an Ubuntu laptop with an 8-core Intel i7-10510U CPU at 1.80 GHz and 16 GB of RAM. The code was compiled with *CMake* using the `-O2` optimization flag. As described above, each time the number of control points changes, we compute the $(n+1)\times(n+1)$ matrix L_k (or $\tilde{L}_k$) and the Legendre coefficients $\boldsymbol{c}^k$ (or $\tilde{\boldsymbol{c}}^k$) in a preprocessing step which is quadratic in n. During a modelling session, these coefficients are updated in linear time whenever the user moves a control point. For plotting the curve, we evaluate it at 1000 equidistant parameter values using Algorithm 1, which also has linear runtime. For comparison, cubic B-spline curves are evaluated using the *Eigen* module [5].

Figure 5 shows a comparison of GL-k and cubic B-spline curves. We observe that GL-1 curves typically behave and appear like cubic B-spline curves with uniformly distributed knots. As k increases, GL-k curves keep "shrinking" away from the control polygon (like Bézier curves, but much less so) and behave like higher-degree B-spline curves. In particular, they fail to capture the intended shape if the control polygon contains zig-zag sections (Figure 5, right). This is less of an issue with GL-0 curves, the classical GL curves. On the other hand, manipulating GL-0 curves can cause unintended artefacts as the basis functions are less local than the basis functions of GL-k curves for $k > 0$ (see Figure 4).

We further investigate the similarity between modified GL-1 and cubic B-spline curves by looking at their curvature profiles. As can be seen in Figure 6, the curvature of the GL-1 curve in this example is very similar to that of the cubic B-spline curve with the same control polygon, but with two advantages. On the one hand, it oscillates less and has fewer inflection points. On the other hand, it is continuously differentiable, while the curvature of the cubic B-spline curve is only C^0 at the knots.

Figure 7 reports the runtimes of our implementation. For this experiment, we generated 1000 random curves with $n+1$ control points, for $n = 3, \ldots, 19$, and evaluated them at 1000 equidistant parameter values. The plots confirm our expectations. On the one hand, the initial conversion from control points to Legendre coefficients is quadratic in n and grows with k. On the other hand, the evaluation of GL-k curves is linear in n and *independent of* k, while cubic B-splines can be evaluated essentially in constant time. We observe that evaluating GL-k curves in the Legendre basis with Algorithm 1 is more efficient than evaluating cubic B-spline curves for up to 10 control points, and even for 20 control points, it is less than twice as expensive.

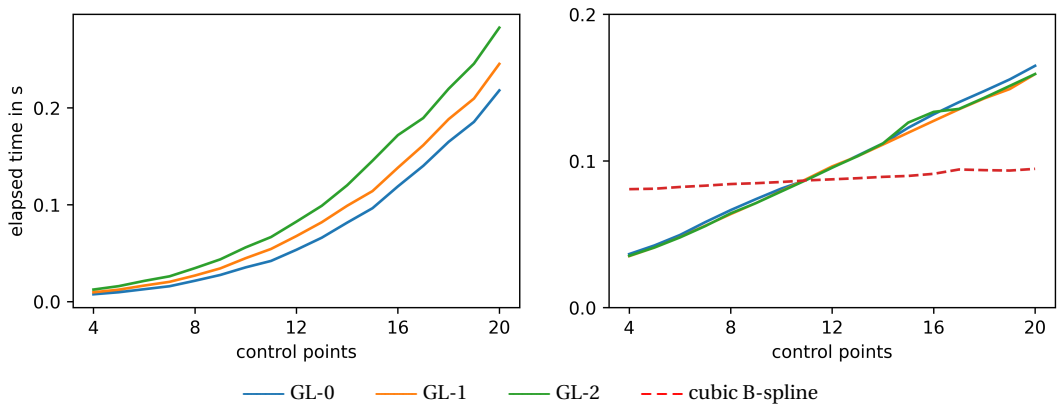

Figure 7: Runtime comparison for evaluating GL=0, GL-1, GL-2, and cubic B-spline curves. While the preprocessing step that converts control points to Legendre coefficients is quadratic in the number of control points (left), the actual evaluation of GL-k curves has linear runtime and is faster than the evaluation of cubic B-spline curves for up to 10 control points (right).

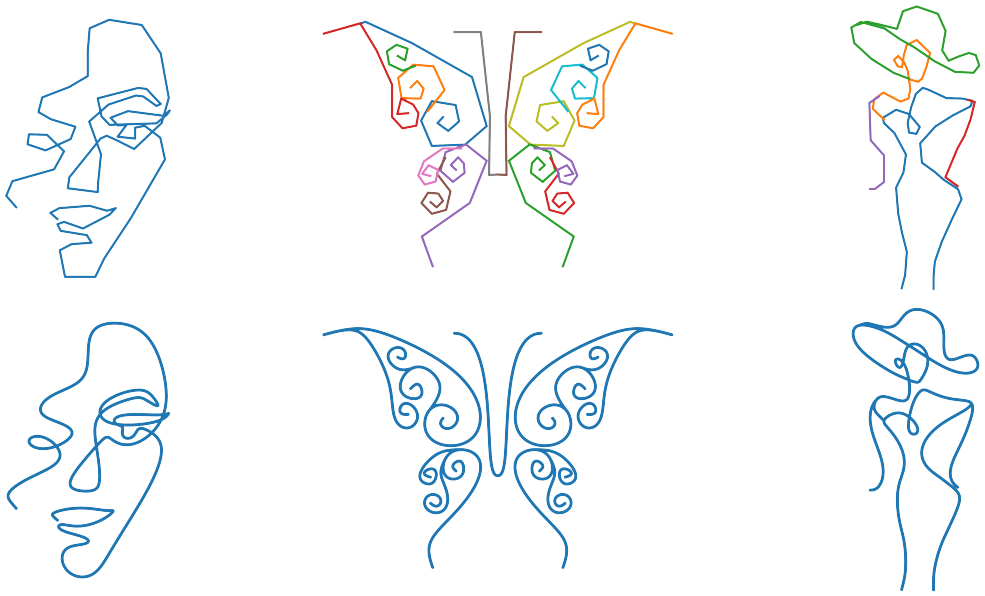

Figure 8: Examples of line art with modified GL-1 curves. The face stroke is generated using a single curve of degree 75. The butterfly and the lady are examples where several GL-1 curves were used to design a more complicated artwork.

8 Conclusion

GL-k curves constitute a novel family of polynomial curves that offer an intuitive tool for curve modelling. Unlike Bézier curves, the corresponding basis functions are not non-negative (see Figure 4), and consequently a GL-k curve is not guaranteed to remain within the convex hull of its control points. This is, however, a trade-off that allows the curve to follow the shape of its control polygon more closely, especially for high-degree curves.

Based on our experiments, our favourite curves are modified GL-1 curves, which offer the best design experience and can be used to quickly design single-stroke polynomial drawings (see Figure 8, left). Their intuitive manipulation is particularly useful for manual refinement. Instead of relying on a single high-degree curve, one may also generate more intricate artwork by using several curves (see Figure 8, middle and right). Thanks to the endpoint interpolation and the endpoint tangent property, it is easy to ensure C^0 or C^1 continuity at corners and branching points.

Acknowledgements

We thank the anonymous reviewers for their valuable comments and suggestions, which helped to improve this paper. This work was supported by the Swiss National Science Foundation (SNF) under project No. 188577.

References

[1] M. Abramowitz and I. A. Stegun, editors. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, volume 55 of *Applied Mathematics Series*. Dover Publications, New York, 1965. ISBN 978-0-486-61272-0.

[2] G. B. Arfken, H. J. Weber, and F. E. Harris. *Mathematical Methods for Physicists: A Comprehensive Guide*. Academic Press, Oxford, 7th edition, 2012. ISBN 978-0-12-384654-9.

[3] U. M. Ascher and C. Greif. Best approximation. In *A First Course in Numerical Methods*, Computational Science & Engineering, chapter 12, pages 365–382. Society for Industrial and Applied Mathematics, Philadelphia, 2011.

[4] C. Fuda, A. Ramanantoanina, and K. Hormann. A comprehensive comparison of algorithms for evaluating rational Bézier curves. *Dolomites Research Notes on Approximation*, 17(3):56–79, Sept. 2024. [PDF]

[5] G. Guennebaud, B. Jacob, et al. Eigen 3.4.0. `http://eigen.tuxfamily.org`, Aug. 2021. [Online; accessed 11-April-2026].

[6] P. J. Hartley and C. J. Judd. Parametrization of Bézier-type B-spline curves and surfaces. *Computer-Aided Design*, 10(2):130–134, Mar. 1978.

[7] P. Henrici. *Elements of Numerical Analysis*. John Wiley and Sons, New York, 1964. ISBN 978-0-471-37241-7.

[8] B. Jiang and R. Chen. G^2 interpolating spline with local maximum curvature. *ACM Transactions on Graphics*, 44(6):Article 229, 13 pages, Dec. 2025.

[9] S. H. Kim and H. P. Moon. Rectifying control polygon for planar Pythagorean hodograph curves. *Computer Aided Geometric Design*, 54:1–14, May 2017.

[10] S. H. Kim and H. P. Moon. Gauss–Lobatto polygon of Pythagorean hodograph curves. *Computer Aided Geometric Design*, 74:Article 101768, 20 pages, Oct. 2019.

[11] E. T. Y. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363–370, July–Aug. 1989.

[12] H. P. Moon, S. H. Kim, and S.-H. Kwon. Shape analysis of planar PH curves with the Gauss–Legendre control polygons. *Computer Aided Geometric Design*, 82:Article 101915, 18 pages, Oct. 2020.

[13] H. P. Moon, S. H. Kim, and S.-H. Kwon. Gauss–Legendre polynomial basis for the shape control of polynomial curves. *Applied Mathematics and Computation*, 451:Article 127995, 16 pages, Aug. 2023.

[14] F. W. J. Olver, D. W. Lozier, R. F. Boisvert, and C. W. Clark. *NIST Handbook of Mathematical Functions*. Cambridge University Press, New York, 2010. ISBN 978-0-521-19225-5.

[15] H. Prautzsch, W. Boehm, and M. Paluszny. *Bézier and B-spline techniques*. Mathematics and Visualization. Springer, Berlin, Heidelberg, 2002. ISBN 978-3-642-07842-2.

[16] A. Ramanantoanina and K. Hormann. Trigonometric tangent interpolating curves. In R. Chen, T. Ritschel, and E. Whiting, editors, *Pacific Graphics Conference Papers and Posters*, pages 1–8, Huangshan, China, Oct. 2024. [PDF]

[17] C. Runge. Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. *Zeitschrift für Mathematik und Physik*, 46:224–243, 1901.

[18] B. Simon. The Christoffel–Darboux kernel. In *Perspectives in Partial Differential Equations, Harmonic Analysis and Applications*, volume 79 of *Proceedings of Symposia in Pure Mathematics*, pages 295–336. American Mathematical Society, Providence, 2008.

[19] G. Szegő. *Orthogonal Polynomials*, volume 23 of *Colloquium Publications*. American Mathematical Society, Providence, 4th edition, 1975. ISBN 978-0-8218-1023-1.

[20] Z. Yan, S. Schiller, G. Wilensky, N. Carr, and S. Schaefer. κ-curves: Interpolation at local maximum curvature. *ACM Transactions on Graphics*, 36(4):Article 129, 7 pages, Aug. 2017.