Seiler's Interpolation for Evaluating Polynomial Curves

Cem Yuksel
cem@cemyuksel.com
Cyber Radiance
USA

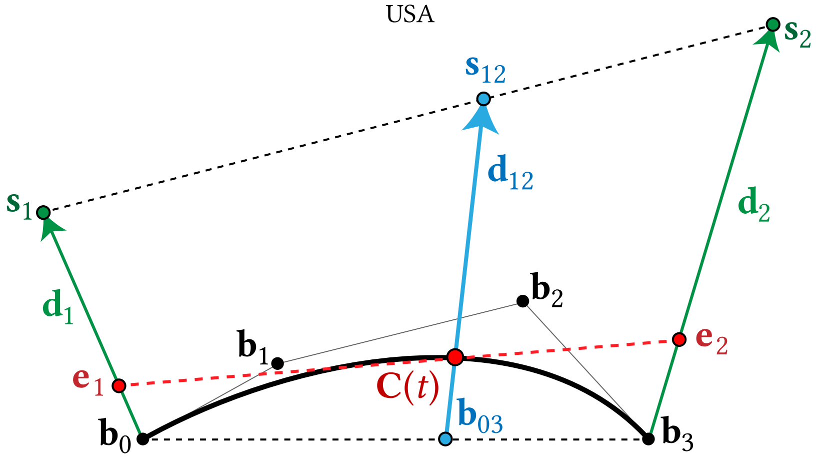

Figure 1: *An illustration of Seiler's interpolation, evaluating a cubic Bézier curve* $\mathbf{C}(t)$ *at* $t = 0.6$ *using only 3 linear interpolations that combine the end points* $\mathbf{b}_0$ *and* $\mathbf{b}_3$ *with Seiler points* $\mathbf{s}_1$ *and* $\mathbf{s}_2$*, which can be precomputed using the internal Bézier control points* $\mathbf{b}_1$ *and* $\mathbf{b}_2$*.*

ABSTRACT

Seiler's interpolation allows evaluating polynomial curves, such as Bézier curves, with a small number of linear interpolations. It is particularly effective with hardware linear interpolation used in GPU texture filtering. We compare it to the popular alternatives, such as de Casteljau's algorithm, and present how it extends to higher-degree polynomials.

ACM Reference Format:
Cem Yuksel. 2024. Seiler's Interpolation for Evaluating Polynomial Curves. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH Talks '24), July 27 - August 01, 2024.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3641233.3664331

1 INTRODUCTION

Polynomial curves are everywhere in computer graphics. They are often evaluated using Bézier control points, but they can also be evaluated using polynomial coefficients. Another common alternative is de Casteljau's algorithm, which purely relies on linear interpolations and can be particularly advantageous when some of these linear interpolations can be performed in hardware (such as cubic texture filtering on the GPU).

We present Seiler's interpolation, which can evaluate a cubic Bézier curve with only 3 linear interpolations (i.e. *lerps*), as opposed to 6 needed by de Casteljau's algorithm. The original form of Seiler's interpolation was presented by Larry Seiler and his colleagues [Lin et al. 2021] for slightly modifying the GPU hardware to support hardware-accelerated higher-order texture filtering. We discuss Seiler's interpolation in a more general context and compare it against other methods for evaluating cubic Bézier curves. We include its original form using *difference terms*, a pure lerp version, and an alternative ordering of computation to reduce the number of micro-operations using offsets. We primarily concentrate on cubic Bézier curves, as they are the most common ones in graphics, but we also include formulations of quadratic curves and higher-degree polynomials than cubics.

2 BACKGROUND

Let $\mathbf{C}(t)$ represent a polynomial parametric curve of degree d with a parameter value $t \in [0, 1]$. Given Bézier control points $\mathbf{b}_i$, where $i \in \{0, 1, \cdots, d\}$, we can evaluate it using Bernstein polynomials. For a cubic curve, we can write

$$\mathbf{C}(t) = (1-t)^3\,\mathbf{b}_0 + 3(1-t)^2 t\,\mathbf{b}_1 + 3(1-t)t^2\,\mathbf{b}_2 + t^3\,\mathbf{b}_3\ . \quad (1)$$

A computationally more efficient alternative is its polynomial form after its coefficients $\mathbf{p}_i$ are pre-computed. For a cubic curve, we can write $\mathbf{p}_0 = \mathbf{b}_0$, $\mathbf{p}_1 = 3(\mathbf{b}_1 - \mathbf{b}_0)$, $\mathbf{p}_2 = 3(\mathbf{b}_0 - 2\mathbf{b}_1 + \mathbf{b}_2)$, and $\mathbf{p}_3 = -\mathbf{b}_0 + 3\mathbf{b}_1 - 3\mathbf{b}_2 + \mathbf{b}_3$, resulting

$$\mathbf{C}(t) = t^3\,\mathbf{p}_3 + t^2\,\mathbf{p}_2 + t\,\mathbf{p}_1 + \mathbf{p}_0\ . \quad (2)$$

de Casteljau's algorithm uses linear interpolation functions $\mathbf{L}(\mathbf{a}, \mathbf{b}, t) = (1-t)\,\mathbf{a} + t\,\mathbf{b}$. For a cubic curve, first calculates three intermediate points $\mathbf{b}_{01} = \mathbf{L}(\mathbf{b}_0, \mathbf{b}_1, t)$, $\mathbf{b}_{12} = \mathbf{L}(\mathbf{b}_1, \mathbf{b}_2, t)$, and $\mathbf{b}_{23} = \mathbf{L}(\mathbf{b}_2, \mathbf{b}_3, t)$. Then, it interpolates them as $\mathbf{b}_{02} = \mathbf{L}(\mathbf{b}_{01}, \mathbf{b}_{12}, t)$ and $\mathbf{b}_{13} = \mathbf{L}(\mathbf{b}_{12}, \mathbf{b}_{23}, t)$. Finally, the sixth interpolation forms the results $\mathbf{C}(t) = \mathbf{L}(\mathbf{b}_{02}, \mathbf{b}_{13}, t)$.

Table 1: *Multiplications M and additions A needed for evaluating cubic polynomial curves with different formulations in n dimensions.*

	Polynomial	Bernstein Poly.	de Casteljau	**Seiler** (diff. terms)	**Seiler** (pure lerp)	**Seiler** (offsets)
All micro-ops	$\boldsymbol{n(3M+3A)}$	$n(4M+3A)+6M$	$n(12M+6A)$	$n(5M+3A)+M$	$n(6M+3A)+M$	$n(4M+3A)+M$
After initial lerps	—	—	$n(6M+3A)$	$\boldsymbol{n(M+A)+M}$	$n(2M+A)+M$	$n(2M+A)$

3 SEILER'S INTERPOLATION FOR CUBICS

The original form of the Seiler's interpolation [Lin et al. 2021] for cubics uses a lerp of the endpoints $\mathbf{b}_{03} = \mathbf{L}(\mathbf{b}_0, \mathbf{b}_3, t)$ and a lerp of two *difference terms*, $\mathbf{d}_1$ and $\mathbf{d}_2$, using $\mathbf{d}_{12} = \mathbf{L}(\mathbf{d}_1, \mathbf{d}_2, t)$, where

$$\mathbf{d}_1 = 3(\mathbf{b}_1 - \mathbf{b}_0) - (\mathbf{b}_3 - \mathbf{b}_0) \tag{3}$$

$$\mathbf{d}_2 = 3(\mathbf{b}_2 - \mathbf{b}_3) - (\mathbf{b}_0 - \mathbf{b}_3) \ . \tag{4}$$

The final position on the curve is evaluated by adding a portion of the interpolated difference terms to the interpolated endpoints, such that $\mathbf{C}(t) = \mathbf{b}_{03} + (1-t)t\,\mathbf{d}_{12}$. This is illustrated in Figure 1.

To achieve a *pure lerp* form, we define two *Seiler points* $\mathbf{s}_1 = \mathbf{d}_1 + \mathbf{b}_0$ and $\mathbf{s}_2 = \mathbf{d}_2 + \mathbf{b}_3$. The second lerp is performed using these Seiler points as $\mathbf{s}_{12} = \mathbf{L}(\mathbf{s}_1, \mathbf{s}_2, t)$. Then, the final step turns into a third lerp, such that $\mathbf{C}(t) = \mathbf{L}\big(\mathbf{b}_{03}, \mathbf{s}_{12}, (1-t)t\big)$.

For reducing the number of operations, we can first compute *offset points* $\mathbf{e}_1 = \mathbf{b}_0 + (1-t)t\mathbf{d}_1$ and $\mathbf{e}_2 = \mathbf{b}_3 + (1-t)t\mathbf{d}_2$. Then, the point on the curve becomes a lerp of these offset points $\mathbf{C}(t) = \mathbf{L}(\mathbf{e}_1, \mathbf{e}_2, t)$, as shown in Figure 1. We can also evaluate offset points with $\mathbf{e}_1 = \mathbf{L}\big(\mathbf{b}_0, \mathbf{s}_1, (1-t)t\big)$ and $\mathbf{e}_2 = \mathbf{L}\big(\mathbf{b}_3, \mathbf{s}_2, (1-t)t\big)$, forming another alternative that only uses lerps.

Table 1 lists the number of micro-ops needed for different methods. The polynomial form has the fewest number of micro-ops and it can also benefit from *fused-multiply-add* operations. On the other hand, when implemented on the GPU, interpolation-based methods can utilize the texture filtering hardware to perform the initial lerps in hardware while reading the coefficients from a texture. After these initial lerps, Seiler's interpolation involves fewer operations than the others. Also, the initial lerps of Seiler's interpolation can be performed in parallel, which can provide savings in computation time even when performing lerps in hardware is not an option.

Piecewise cubic curves with m pieces require storing $4m$ coefficients in polynomial form, but the other methods can store $3m+1$ points, as the positions of the endpoints are common (i.e. $\mathbf{b}_0$ for a piece matches $\mathbf{b}_3$ of the previous piece). Consecutive pieces do not necessarily share the same Seiler points where they join, even when the curve has C^2 continuity. Therefore, splitting Bézier curves results in two different Seiler points on either side of the split.

One exception is cubic Catmull-Rom splines with uniform parameterization [Yuksel et al. 2011], where the first Seiler point ($\mathbf{s}_1$) of a piece matches the second Seiler point of the previous piece ($\mathbf{s}_2$). Thus, the difference terms of consecutive pieces also match. Therefore, such curves can be stored in Seiler's interpolation form using only $2m+2$ points. With other parameterizations of Catmull-Rom curves, the directions of the difference terms match where pieces join, but not their magnitudes, which vary by the ratios of parameter lengths of the consecutive pieces.

4 GENERAL POLYNOMIAL CURVES

In addition to cubics, Lin et al. [2021] also includes a formulation for quadratics, though with a different notation than ours. A quadratic Bézier has a single difference term $\mathbf{d}_1$, corresponding to two Seiler points $\mathbf{s}_1 = \mathbf{b}_0 + \mathbf{d}_1$ and $\mathbf{s}_2 = \mathbf{b}_2 + \mathbf{d}_1$.

For polynomial curves with degree d, Seiler's interpolation can be written in a recursive form, such that

$$\mathbf{C}(t) = \mathbf{L}(\mathbf{b}_0, \mathbf{b}_d, t) + (1-t)t\,\mathbf{D}_1(t) \ , \quad \text{where} \tag{5}$$

$$\mathbf{D}_i(t) = \begin{cases} \mathbf{0} \ , & \text{if } 2i = d+1 \\ \mathbf{d}_i \ , & \text{if } 2i = d \\ \mathbf{L}(\mathbf{d}_i, \mathbf{d}_{d-i}, t) + (1-t)t\,\mathbf{D}_{i+1}(t) \ , & \text{otherwise.} \end{cases} \tag{6}$$

The pure lerp formulation replaces the additions above with lerp using Seiler points $\mathbf{s}_i = \mathbf{s}_{i-1} + \mathbf{d}_i$ and $\mathbf{s}_{d-i} = \mathbf{s}_{d-i+1} + \mathbf{d}_{d-i}$ for $i \in \{1, \cdots, \lfloor d/2 \rfloor\}$, where $\mathbf{s}_0 = \mathbf{b}_0$ and $\mathbf{s}_d = \mathbf{b}_d$. The difference terms can be computed from the Bézier control points $\mathbf{b}_i$, using

$$\mathbf{d}_1 = d(\mathbf{b}_1 - \mathbf{b}_0) - (\mathbf{b}_d - \mathbf{b}_0) \tag{7}$$

$$\mathbf{d}_{d-1} = d(\mathbf{b}_{d-1} - \mathbf{b}_d) - (\mathbf{b}_0 - \mathbf{b}_d) \tag{8}$$

$$\begin{aligned} \mathbf{d}_2 = {} & \binom{d}{2}(\mathbf{b}_2 - \mathbf{b}_1) - \binom{d-2}{2}(\mathbf{b}_1 - \mathbf{b}_0) \\ & - (d-3)(\mathbf{b}_{d-1} - \mathbf{b}_d) - 3(\mathbf{b}_{d-1} - \mathbf{b}_1) \end{aligned} \tag{9}$$

$$\begin{aligned} \mathbf{d}_{d-2} = {} & \binom{d}{2}(\mathbf{b}_{d-2} - \mathbf{b}_{d-1}) - \binom{d-2}{2}(\mathbf{b}_{d-1} - \mathbf{b}_d) \\ & - (d-3)(\mathbf{b}_1 - \mathbf{b}_0) - 3(\mathbf{b}_1 - \mathbf{b}_{d-1}) \ . \end{aligned} \tag{10}$$

These formulas are sufficient for generating the difference terms for polynomials up to (and including) degree 5 (i.e. $d \leq 5$).

For a polynomial curve of degree d with $d+1$ control points, Seiler's interpolation uses d lerps, $\lceil d/2 \rceil$ of which can be computed in parallel, as they interpolate disjoint pairs of difference terms ($\mathbf{d}_i$ and $\mathbf{d}_{d-i}$) or the endpoints ($\mathbf{b}_0$ and $\mathbf{b}_d$).

We can reduce the number of operations with offsets and a single lerp at the end, using $\mathbf{d}_0 = \mathbf{b}_0$, $\mathbf{d}_d = \mathbf{b}_d$, and

$$\mathbf{C}(t) = \mathbf{L}\big(\mathbf{E}_0^+(t), \mathbf{E}_d^-(t), t\big) \ , \quad \text{where} \tag{11}$$

$$\mathbf{E}_i^\pm(t) = \begin{cases} \mathbf{d}_i \ , & \text{if } d-1 \leq 2i \leq d+1 \\ \mathbf{d}_i + (1-t)t\,\mathbf{E}_{i\pm1}^\pm(t) \ , & \text{otherwise.} \end{cases} \tag{12}$$

5 CONCLUSION

We have presented Seiler's interpolation for evaluating polynomial curves with three variants, including quadratic, cubic, and higher-order polynomials, extending the original formulation of Lin et al. [2021]. Considering the broad use of polynomial curves in graphics, various applications might benefit from the alternative computation and storage benefits of these variants. Future work can investigate the difference term rules for higher-order polynomials ($d \geq 6$).

REFERENCES

Daqi Lin, Larry Seiler, and Cem Yuksel. 2021. Hardware Adaptive High-Order Interpolation for Real-Time Graphics. *Computer Graphics Forum (Proceedings of HPG 2021)* 40, 8 (2021), 1–16. https://doi.org/10.1111/cgf.14377

Cem Yuksel, Scott Schaefer, and John Keyser. 2011. Parameterization and Applications of Catmull-Rom Curves. *Computer Aided Design* 43, 7 (2011), 747–755. https://doi.org/10.1016/j.cad.2010.08.008