We present a JavaScript library for creating interactive subdivision curves. Although subdivision curves are well-suited for use in interactive visualizations, the InfoVis community has used them sparingly. *SDCurve.js* is a lightweight, open-source, D3-compatible library that implements several common curve schemes, such as interpolating curves and B-Splines of any degree, in a subdivision framework that allows fast and easy curve interaction. We hope InfoVis researchers and designers will take advantage of it and contribute to its expansion.

1 INTRODUCTION

When implementing interactive visualizations, designers and developers often need to make use of curves. Curves can form the entire core of a visualization (e.g., [1]), or they may simply highlight certain aspects or link discrete elements together (e.g., [4, 7]). We focus on visualizations where it may be important to be able to interact directly with the curve, e.g., adjusting a lasso selection.

Bezier curves are a common choice for smooth, well-behaved curves. However, each control point affects all areas of the curve. This makes local control of the curve difficult, as adjusting one control point changes the entire shape of the curve. B-spline curves extend Bezier curves to provide local control, i.e., each point on the curve depends on a subset of the control points. However, they are ill-suited for some applications, as the control points are still unlikely to lie on the final curve, and interacting with the curve often must be done indirectly by manipulating these offset, often hidden, control points. Interpolating schemes, on the other hand, can be used to guarantee that a curve will pass through a set of control points. In practice, supporting multiple curve schemes through a unified framework focused on direct interaction is ideal. Designers can then work with the best scheme for their task.

A subdivision curve—a curve that is iteratively refined through linear operations on the control points—can create B-splines of any degree, but can also easily handle many interpolating schemes. This flexibility allows subdivision curves to be easily applied to many visualization needs. Despite these advantages, subdivision curves, popular in the graphics community, are rarely used in InfoVis. To make subdivision curves accessible to InfoVis researchers and designers, we contribute a fast, interactive JavaScript library that implements B-splines of any degree, including commonly used quadratic (Chaikin [3]) B-splines, and Dyn-Levin [5, 8] interpolating subdivision curves. Figure 1 shows how choosing different schemes leads to different curves. The curves can be created and animated in real time, are designed to be compatible with the popular D3 library [2], and alternate schemes can easily be added via open source development. This lightweight library, designed by InfoVis designers for InfoVis designers, will make it easier to create web-based visualizations where interactive curves are important.

2 SUBDIVISION CURVES

Subdivision converts an initial set of control points to a fine set of curve points by applying repeated linear operations. Linear masks define the instructions for this process; a mask \([m_0, m_1, \ldots, m_l]\) operates on consecutive “coarse” points \(P_0, P_{n+1}, \ldots, P_{n+l}\) and creates a new “fine” point \(\sum m_i P_{i+n}\). With smartly chosen linear masks, the result of each step creates a refined curve that is continuous and smooth, and after only 4 or 5 steps, the curves are visually pleasing.

Figure 2 illustrates this process with the Chaikin “corner-cutting” scheme [3], whose linear masks are \([\frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]\) and \([\frac{1}{4}, \frac{1}{2}, \frac{1}{4}]\). That is, for any two consecutive non-endpoints \(p_n\) and \(p_{n+1}\), the subdivision operation creates new points at \(\frac{3}{4}p_n + \frac{1}{4}p_{n+1}\) and \(\frac{1}{4}p_n + \frac{3}{4}p_{n+1}\). Special masks are designed to handle edgepoint cases; for Chaikin, each endpoint is carried forward to the next subdivision step, and the mask \([\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}]\) is used if \(p_n\) or \(p_{n+1}\) is an endpoint.

Applying different masks will result in different curves. The Chaikin scheme produces curves that are equivalent to quadratic B-splines with enough subdivisions. Quadratic B-splines have high local control and tend to get quite close to the control points; if a more gradual curve is needed, the degree of the B-spline can be increased. In our library, we implemented Stam’s [9] algorithm, a faster version of the Lane-Riesenfeld algorithm [6], which indirectly applies masks through an iterative smoothing process. This can produce B-splines of any degree. For very high-degree B-splines, this algorithm requires many smoothing steps. Our library can be expanded to include the direct mask calculation for any arbitrary B-spline degree if certain high-degree curves are often used.

The Dyn-Levin subdivision scheme [5] directly interpolates the control points using a linear mask that is 4 points wide. In the non-endpoint case, each point is carried forward to the next subdivision level (mask \([0, 1, 0, 0]\)), and a new point is created with the mask \([-1, 9, 9, -1]\). Endpoint conditions are used from Semmerud [8]. Unlike B-splines, this means that all control points lie on the curve.

Each curve type has different merits depending on the needs of the designer. Subdivision curves encapsulate the benefits of traditional B-splines while also easily supporting alternate schemes in

---

*e-mail: rapusch@ucalgary.ca*

†e-mail: charles.perin@ucalgary.ca

‡e-mail: sheelagh@ucalgary.ca

*SDCurve.js: A JavaScript Library for Interactive Subdivision Curves*

Richard Pusch‡

Charles Perin†

Sheelagh Carpendale‡

University of Calgary

University of Calgary

University of Calgary

Figure 1: Sample curves created using *SDCurve.js* illustrating how using different schemes leads to different curves for the same set of control points. The first four images show open curves and the rightmost image shows closed curves.
Our library also provides ways to measure distance along or near the curve. The `pointAt(u)` function returns the point of distance \( u \) (0 ≤ \( u \) ≤ 1) along the curve, where \( u = 0 \) is the beginning of the curve and \( u = 1 \) is the end of the curve. To accomplish this, we parameterize the curve by arclength in the background. We can also call the `getClosestPoint(point)` function to get the closest point on the curve to the provided point. This is helpful for snapping a moving mouse coordinate directly to the curve, for example.

### 3.3 Interactive Functionality

Manipulating Bezier or B-spline curves is typically done by moving control points. This can make fine control over the curve difficult, and control points are usually hidden in visualizations. Our library allows designers to specify any point on the curve and move it to a new location through the `moveCurve()` function. This is simply done by providing a point along the curve (for example, the location of a mouse click) and a movement vector to specify its new location. The curve will make the necessary changes to the nearest control points to achieve this effect. This allows the curve to be deformed by direct interaction, and no knowledge of the underlying control point structure is needed. The `moveCurve()` function optionally accepts a “width” argument, which specifies how many of the underlying control points are allowed to move. This supports influencing only a small area, causing sharp alterations to the curve, or a wide area for a more gradual, smooth transition. Figure 3 illustrates interacting with the curve with different “width” values.

### 4 Conclusion

Subdivision curves, popular in the graphics community, are rarely used in InfoVis. They offer great flexibility, can support standard curves, and are particularly well-suited to interacting directly with points on the curve instead of offset control points.

SDCurve.js is an open-source library available at https://github.com/rpusch/sdcurve.js/. It can be used “as is” for creating curves and is easily plugged into D3 visualizations. SDCurve.js will continue to evolve with novel algorithms, optimizations, and subdivision schemes. New schemes are easy to add and we hope the community of InfoVis developers and designers will contribute to expanding the library.

### Acknowledgements

Thanks to support from NSERC, AITF, and SMART Technologies.

### References


---

1. [https://github.com/mrdoob/three.js/](https://github.com/mrdoob/three.js/)