(Lossless) Split and Merge for Bezier Curves

Jialin Lu luxxxlucy@gmail.com 2025-07-20

The editor (in its preliminary version) is available at luxxlucy/houjing, see `crates/houjing-main`

Note that this is the second blog in the ***Bezielogue*** *series on bezier curves.*

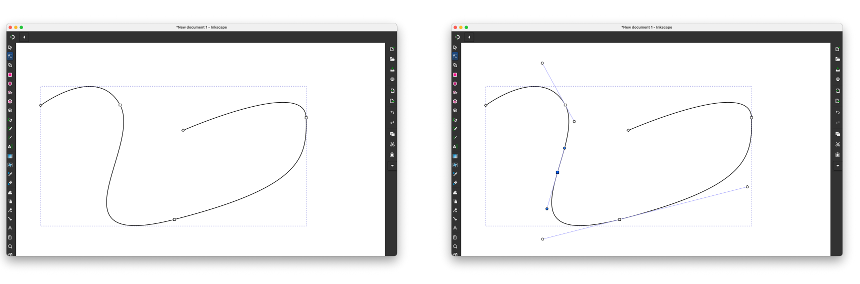
TL;DR We introduce two fundamental operations **split** and **merge** that a vector graphics editor should support for manipulating the control points (nodes) of Bézier curves.

Introduction

Popular vector graphics applications like Adobe Illustrator, Inkscape, and Figma all provide curve cutting and joining tools that rely on these operations. However, merging is typically not implemented in a lossless way but rather handled by general-purpose simplification algorithms. More on this in the future.

Users of vector graphics editors routinely need to **split** and **merge** curve segments.

Given an already-created path (a consecutive sequence of Bézier segments), split enables a user to create a new control point on a curve, making it possible to further manipulate the curve. In Inkscape, this is done by selecting the `Node` tool and double-clicking on a curve.



Create a new path using the **Path** tool — Split: double-click to insert a new control point (**Node** tool)

Figure 1:

Split operation in Inkscape: inserting a new control point

Throughout this blog, we focus on cubic Bézier curves. Linear and quadratic Bézier curves are trivial and can be pre-processed into cubic ones, while higher-order curves are beyond our scope.

Merge is the reverse operation of split. Sometimes a user creates a new point, forgets about it, and ultimately makes no changes to it. Or data imported from external sources may naturally contain duplicated unnecessary points. Ideally, cleanup is needed—either deliberately triggered by the user or implicitly performed during export. These unnecessary points should be removed, effectively merging two or more Bézier segments back into one.

Note that split operations are always lossless—the path before and after splitting remains identical. However, merge operations are not always lossless.

In Inkscape, merge can be achieved either by explicitly deleting a point using the **Node tool** or by applying the **Simplify** tool. In both cases, the merge operation is not guaranteed to be lossless. Point deletion and simplification are both destructive processes that do not aim to preserve the original curve shape. After all, users may simply want to delete a point or simplify the curve without any expectation of lossless behavior. For point deletion, if the curve can be merged losslessly, such deletion will indeed maintain the shape. However, for simplification, even when lossless merge is possible, the general-purpose simplification algorithm will still fail to recognize such merge opportunities.

In this blog, we focus on the lossless case where merge operations can recover the original curve exactly within numerical precision.

Basic Formulation

We introduce the basic formulation of Bézier curves before diving into split and merge operations.

Cubic Bézier curves are defined by control points p_0, p_1, p_2, p_3:

$$B(t) = (1-t)^3 p_0 + 3(1-t)^2 t p_1 + 3(1-t)t^2 p_2 + t^3 p_3$$

The two operations are defined as follows:

1. **Split**: Given curve $B(t)$ with control points p_0, p_1, p_2, p_3 and parameter $t_s \in [0,1]$, produce curves with control points a_0, a_1, a_2, a_3 and b_0, b_1, b_2, b_3 such that:
   - Left curve represents $B(t)$ for $t \in [0, t_s]$
   - Right curve represents $B(t)$ for $t \in [t_s, 1]$
2. **Merge**: Given adjacent curves with control points a_0, a_1, a_2, a_3 and b_0, b_1, b_2, b_3, reconstruct original control points p_0, p_1, p_2, p_3.

Split: De Casteljau Construction

Paul de Casteljau developed this algorithm in 1959 while working at Citroën, though it wasn't published until 1975. It predates Bézier's work by several years.

De Casteljau's algorithm provides an elegant geometric method for splitting Bézier curves. Rather than manipulating the algebraic form, it uses repeated linear interpolation to find the split point and new control points.

For a cubic curve with control points p_0, p_1, p_2, p_3, splitting at parameter t proceeds in three levels:

Level 1: Linear interpolation between adjacent control points

$$q_0 = p_0 + t(p_1 - p_0) = (1-t)p_0 + tp_1$$
$$q_1 = p_1 + t(p_2 - p_1) = (1-t)p_1 + tp_2$$
$$q_2 = p_2 + t(p_3 - p_2) = (1-t)p_2 + tp_3$$

Level 2: Interpolate between the q points

$$r_0 = q_0 + t(q_1 - q_0) = (1-t)q_0 + tq_1$$
$$r_1 = q_1 + t(q_2 - q_1) = (1-t)q_1 + tq_2$$

Level 3: Find the split point

$$s = r_0 + t(r_1 - r_0) = (1-t)r_0 + tr_1$$

The split produces two cubic curves:
- **Left curve**: $a_0 = p_0, a_1 = q_0, a_2 = r_0, a_3 = s$
- **Right curve**: $b_0 = s, b_1 = r_1, b_2 = q_2, b_3 = p_3$

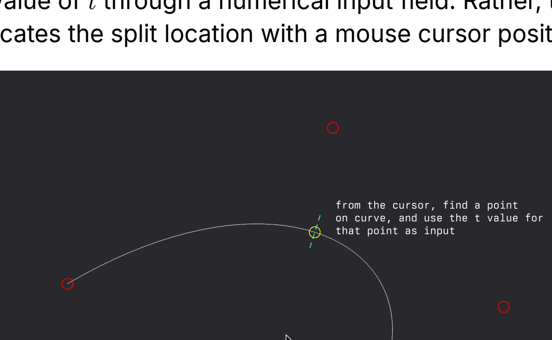


Figure 2:

De Casteljau construction showing the geometric process of splitting a cubic Bézier curve at parameter t. The algorithm constructs intermediate points through successive linear interpolations. (Modified from SFU CMPT361 lecture slides, see link)

This construction guarantees that both resulting curves are valid cubic Bézier curves that together represent the original curve exactly.

It is worth noting that in a real editor, splitting does not proceed by the user providing a value of t through a numerical input field. Rather, the user typically indicates the split location with a mouse cursor position.

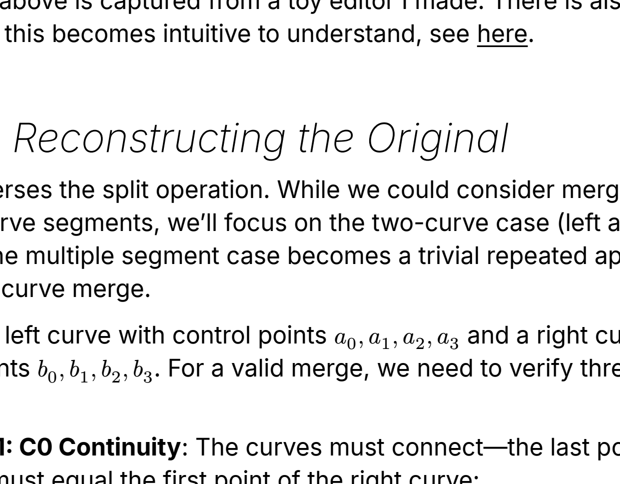


Figure 3:

In an actual split session, the user indicates intent through cursor position. The editor finds the closest point on the curve and uses its corresponding t value to perform the split.

The figure above is captured from a toy editor I made. There is also a live GIF so that this becomes intuitive to understand, see here.

Merge: Reconstructing the Original

Merge reverses the split operation. While we could consider merging multiple curve segments, we'll focus on the two-curve case (left and right curves). The multiple segment case becomes a trivial repeated application of the two-curve merge.

Consider a left curve with control points a_0, a_1, a_2, a_3 and a right curve with control points b_0, b_1, b_2, b_3. For a valid merge, we need to verify three conditions:

An implementation detail: we need to determine the correct order of the two curves. A simple approach is to test both directions—if C0 continuity fails in both cases, the curves cannot be merged.

Condition 1: C0 Continuity: The curves must connect—the last point of the left curve must equal the first point of the right curve:

$$a_3 = b_0$$

Condition 2: Direction Check: Once we confirm the curves are connected, we verify that vectors $a_3 - a_2$ and $b_1 - b_0$ point in the same direction:

$$\text{angle}(a_3 - a_2) = \text{angle}(b_1 - b_0)$$

If this fails, the curves cannot be merged losslessly.

From de Casteljau's construction, we know:

This relationship emerges from the de Casteljau construction. The vectors $a_3 - a_2$ and $b_1 - b_0$ are scaled versions of the same geometric direction.

$$\frac{a_3 - a_2}{t} = \frac{b_1 - b_0}{1 - t} \tag{1}$$

where t is the original split parameter. We can solve for t:

$$t = \frac{\|a_3 - a_2\|}{\|a_3 - a_2\| + \|b_1 - b_0\|}$$

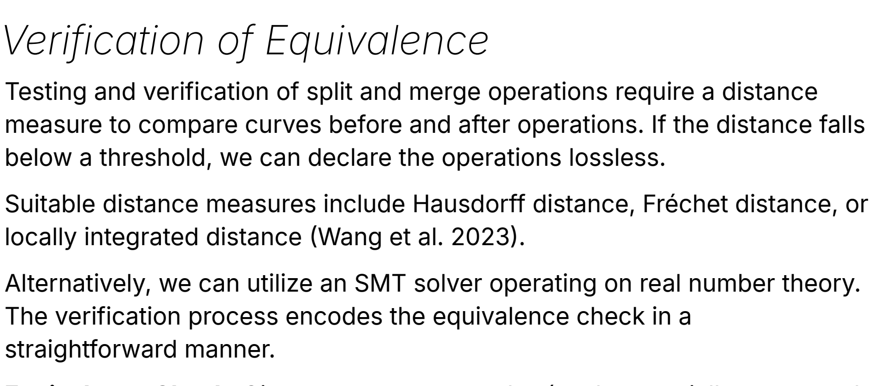
Condition 3: Intermediate Point Consistency: The intermediate construction point must match:

$$a_1 + \frac{a_2 - a_1}{t} = b_2 + \frac{b_1 - b_2}{1 - t}$$

Referring to Figure 2, we can compute the intermediate point Q_1 in two ways: from the left curve ($a_1 + \frac{a_2 - a_1}{t}$) or from the right curve ($b_2 + \frac{b_1 - b_2}{1-t}$). The condition is satisfied when these two calculations yield the same point (within numerical tolerance).

Once the three conditions are met, we can reconstruct the original control points:

$$p_0 = a_0$$
$$p_1 = a_0 + \frac{a_1 - a_0}{t}$$
$$p_2 = b_3 + \frac{b_2 - b_3}{1 - t}$$
$$p_3 = b_3$$



we can manually split a segment into multiple segments — apply merge, all these segments should recover to the original one

Figure 4:

A sanity-check: split and then merge

A live GIF can be found here.

Verification of Equivalence

Wang, Siqi, et al. "Bézier Spline Simplification Using Locally Integrated Error Metrics." SIGGRAPH Asia 2023 Conference Papers. 2023.

Testing and verification of split and merge operations require a distance measure to compare curves before and after operations. If the distance falls below a threshold, we can declare the operations lossless.

Suitable distance measures include Hausdorff distance, Fréchet distance, or locally integrated distance (Wang et al. 2023).

Alternatively, we can utilize an SMT solver operating on real number theory. The verification process encodes the equivalence check in a straightforward manner.

Equivalence Check: Given two curves A and B (each potentially composed of multiple segments), they are equivalent if and only if:
1. For all $t_1 \in [0,1]$, there exists $t_2 \in [0,1]$ such that $A(t_1) = B(t_2)$
2. For all $t_1 \in [0,1]$, there exists $t_2 \in [0,1]$ such that $B(t_1) = A(t_2)$

In SMT-LIB format, suppose A and B each have two curve segments: $A = \{\{p_0, p_1, p_2, p_3\}, \{p_4, p_5, p_6, p_7\}\}$ and $B = \{\{q_0, q_1, q_2, q_3\}, \{q_4, q_5, q_6, q_7\}\}$. We can write:

```
; define the bezier cubic function
(define-fun bezier_cubic ((t Real) (p0 Real) (p1 Real) (p2 Real) (p3
Real)) Real
    (+
        (* (* (* (- 1 t) (- 1 t)) (- 1 t)) p0)
        (* (* (* (* 3 t) (- 1 t)) (- 1 t)) p1)
        (* (* (* (* 3 t) t) (- 1 t)) p2)
        (* (* (* t t) t) p3)))

; defines the control points of A
(declare-const p0_x Real) (declare-const p0_y Real)
(declare-const p1_x Real) (declare-const p1_y Real)
(declare-const p2_x Real) (declare-const p2_y Real)
(declare-const p3_x Real) (declare-const p3_y Real)
(declare-const p4_x Real) (declare-const p4_y Real)
(declare-const p5_x Real) (declare-const p5_y Real)
(declare-const p6_x Real) (declare-const p6_y Real)
(declare-const p7_x Real) (declare-const p7_y Real)

; assign values the control points of A
(assert (and
    (= p0_x 0.0) (= p0_y 0.0)
    (= p1_x 1.0) (= p1_y 2.0)
    (= p2_x 2.0) (= p2_y 2.0)
    (= p3_x 3.0) (= p3_y 0.0)
    (= p4_x 3.0) (= p4_y 0.0)
    (= p5_x 2.0) (= p5_y 2.0)
    (= p6_x 1.0) (= p6_y 2.0)
    (= p7_x 0.0) (= p7_y 0.0)
))

; defines the control points of B
(declare-const q0_x Real) (declare-const q0_y Real)
(declare-const q1_x Real) (declare-const q1_y Real)
(declare-const q2_x Real) (declare-const q2_y Real)
(declare-const q3_x Real) (declare-const q3_y Real)
(declare-const q4_x Real) (declare-const q4_y Real)
(declare-const q5_x Real) (declare-const q5_y Real)
(declare-const q6_x Real) (declare-const q6_y Real)
(declare-const q7_x Real) (declare-const q7_y Real)

; assign values the control points of B
(assert (and
    (= q0_x 0.0) (= q0_y 0.0)
    (= q1_x 1.0) (= q1_y 2.0)
    (= q2_x 2.0) (= q2_y 2.0)
    (= q3_x 3.0) (= q3_y 0.0)
    (= q4_x 3.0) (= q4_y 0.0)
    (= q5_x 2.0) (= q5_y 2.0)
    (= q6_x 1.0) (= q6_y 2.0)
    (= q7_x 0.0) (= q7_y 0.0)
))

(assert (forall ((t1 Real))
    (=>
        (and (>= t1 0.0) (<= t1 1.0))
        (exists ((t2 Real))
            (and
                (>= t2 0.0) (<= t2 1.0)
                (let ((p_x (bezier_cubic t1 p0_x p1_x p2_x p3_x))
                      (p_y (bezier_cubic t1 p0_y p1_y p2_y p3_y)))
                    (or
                        (and
                            (= p_x (bezier_cubic t2 q0_x q1_x q2_x q3_x))
                            (= p_y (bezier_cubic t2 q0_y q1_y q2_y q3_y))
                        )
                        (and
                            (= p_x (bezier_cubic t2 q4_x q5_x q6_x q7_x))
                            (= p_y (bezier_cubic t2 q4_y q5_y q6_y q7_y))
                        )
                    )
                )
            )
        )
    )
))

(assert (forall ((t1 Real))
    (=>
        (and (>= t1 0.0) (<= t1 1.0))
        (exists ((t2 Real))
            (and
                (>= t2 0.0) (<= t2 1.0)
                (let ((p_x (bezier_cubic t1 p4_x p5_x p6_x p7_x))
                      (p_y (bezier_cubic t1 p4_y p5_y p6_y p7_y)))
                    (or
                        (and
                            (= p_x (bezier_cubic t2 q0_x q1_x q2_x q3_x))
                            (= p_y (bezier_cubic t2 q0_y q1_y q2_y q3_y))
                        )
                        (and
                            (= p_x (bezier_cubic t2 q4_x q5_x q6_x q7_x))
                            (= p_y (bezier_cubic t2 q4_y q5_y q6_y q7_y))
                        )
                    )
                )
            )
        )
    )
))

(assert (forall ((t1 Real))
    (=>
        (and (>= t1 0.0) (<= t1 1.0))
        (exists ((t2 Real))
            (and
                (>= t2 0.0) (<= t2 1.0)
                (let ((q_x (bezier_cubic t1 q0_x q1_x q2_x q3_x))
                      (q_y (bezier_cubic t1 q0_y q1_y q2_y q3_y)))
                    (or
                        (and
                            (= q_x (bezier_cubic t2 p0_x p1_x p2_x p3_x))
                            (= q_y (bezier_cubic t2 p0_y p1_y p2_y p3_y))
                        )
                        (and
                            (= q_x (bezier_cubic t2 p4_x p5_x p6_x p7_x))
                            (= q_y (bezier_cubic t2 p4_y p5_y p6_y p7_y))
                        )
                    )
                )
            )
        )
    )
))

(assert (forall ((t1 Real))
    (=>
        (and (>= t1 0.0) (<= t1 1.0))
        (exists ((t2 Real))
            (and
                (>= t2 0.0) (<= t2 1.0)
                (let ((q_x (bezier_cubic t1 q4_x q5_x q6_x q7_x))
                      (q_y (bezier_cubic t1 q4_y q5_y q6_y q7_y)))
                    (or
                        (and
                            (= q_x (bezier_cubic t2 p0_x p1_x p2_x p3_x))
                            (= q_y (bezier_cubic t2 p0_y p1_y p2_y p3_y))
                        )
                        (and
                            (= q_x (bezier_cubic t2 p4_x p5_x p6_x p7_x))
                            (= q_y (bezier_cubic t2 p4_y p5_y p6_y p7_y))
                        )
                    )
                )
            )
        )
    )
))

(check-sat)
```

While the SMT-LIB syntax is verbose, the underlying concept is straightforward and easily automated.