

Bezigon History and Derivation

Versions 4.2.0 and 2.2.0 · 16 October 2026

Purpose of this document

This records how each rule in Bezigon was arrived at, the mathematics behind it, what evidence it rests on, and where the library knowingly departs from Macromedia FreeHand or from Xara. It is written for whoever maintains the code next, who will want to know which parts are observed behaviour, which are derived, and which are decisions that could be revisited.

It covers the whole life of the library, from 2.1.1 to the current pair of versions, 4.2.0 (Corner, Smooth and Tangent nodes) and 2.2.0 (Corner and Smooth only). Nothing here is needed to use the library: the README and the *Technical Reference* cover that.

Where the rules came from

FreeHand has not shipped since 2005 and there is no published specification of its Bezigon tool. Every rule in this library was reached in one of four ways, and each rule below is marked accordingly.

Class	Meaning
Measured	Observed in FreeHand or Xara and read off the screen in pixels. The FreeHand shield and the Tangent handle lengths in the test suite come from this, as do the Xara cusp angles and the Xara two-node loop.
Observed	Seen in FreeHand or Xara and reproduced, without pixel measurement. Most of the Tangent rules are in this class.
Derived	Follows mathematically from another rule or from a requirement such as "points on a circle reproduce the circle". The derivations are given in full below.
Chosen	A decision made during development, where neither reference gave a useful answer, or where following one would have been less intuitive. These are the rules most worth revisiting, and they are listed together in <i>Where FreeHand was overridden</i> .

To be confirmed. The markings for versions up to 4.1.0 are a reconstruction from the development record. Those observations were made by Taras V and Vivek R, who should confirm or correct each marking before this document is relied on. The markings for 4.2.0 and 2.2.0 come from the decisions recorded in this round of work, and the evidence for each is given where it is described.

Derivations

Every handle is a fraction of its own segment

A node has two handles. The in-handle faces the previous node and the out-handle the next. Each automatic handle has length

$$h = f \cdot L$$

where L is the straight-line distance to the neighbour on that side, and f is the handle fraction, 0.39 by default. Because each side uses its own segment, the two handles of one node can differ in length: a node

close to one neighbour and far from the other gets a short handle toward the near one and a long handle toward the far one. This keeps every segment's shape independent of the scale of its neighbours.

So only the direction of each handle has to be decided. Every rule in this library decides a single unit vector $\mathbf{d}$, the direction of travel at the node on that side, and the handles are then

$$\mathbf{h}_{in} = -f L_{prev} \mathbf{d}, \quad \mathbf{h}_{out} = f L_{next} \mathbf{d}$$

A Smooth node uses the same $\mathbf{d}$ on both sides, which is what makes its two handles collinear. A Corner decides each side independently, which is what lets it turn sharply.

Why 0.39

The value is not arbitrary. It is the fraction that makes four Smooth nodes placed on a circle draw a near-perfect circle. (*Measured in FreeHand; derived here.*)

The quarter-circle constant

To approximate a quarter circle of radius R with one cubic Bezier, the four control points are placed at

$$P_0 = (R, 0), \quad P_1 = (R, kR), \quad P_2 = (kR, R), \quad P_3 = (0, R)$$

The constant k that puts the curve's midpoint exactly on the circle is the familiar one:

$$k = \frac{4}{3}(\sqrt{2} - 1) \approx 0.552285$$

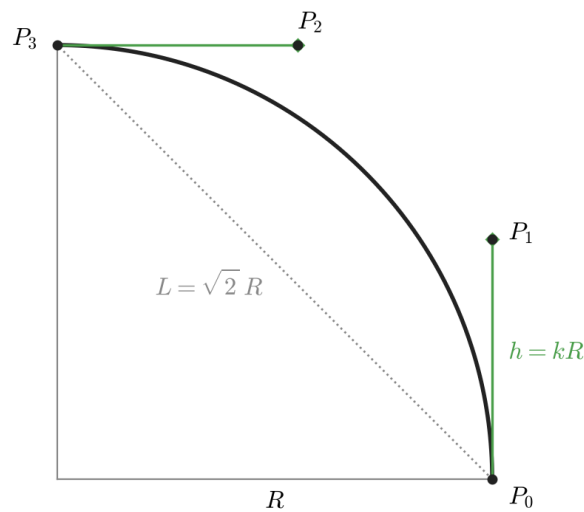


Figure 1. A quarter circle as one cubic Bezier. The handle is kR ; the only length an editor knows is the chord L .

From radius to chord

An interactive editor does not know R . It knows only where the user clicked, so the only length available is the chord between the two nodes:

$$h = \|P_1 - P_0\| = kR, \quad L = \|P_3 - P_0\| = \sqrt{2} R$$

Expressing the handle as a fraction of the chord, the radius cancels:

$$f = \frac{h}{L} = \frac{\frac{4}{3}(\sqrt{2} - 1) R}{\sqrt{2} R} = \frac{4}{3} \cdot \frac{2 - \sqrt{2}}{2} = \frac{4 - 2\sqrt{2}}{3} \approx 0.390524$$

Rounded to two places, 0.39. FreeHand's own handles measure at this value.

What the rounding costs

On a quarter arc of radius 100 pixels the largest distance between the cubic and a true circle is:

Fraction	Largest error on a 100 px quarter circle
exact, 0.390524	0.027 px
0.39 (the default)	0.039 px
1/3 (Xara)	4.289 px

Neither of the first two is visible. Both figures were recomputed for this document by sampling the curve.

Where the rule stops being exact

The ratio above is exact only for a 90-degree arc. For an arc of angle θ the same derivation gives

$$f(\theta) = \frac{\frac{4}{3}\tan\frac{\theta}{4}}{2\sin\frac{\theta}{2}}$$

Arc angle	Exact fraction
30°	0.339
45°	0.347
60°	0.357
90°	0.391
120°	0.444
135°	0.482
150°	0.530
180°	0.667

The library uses 0.39 for every segment whatever its angle. This is a deliberate simplification inherited from FreeHand: one number keeps the rule simple, and the error only becomes noticeable on arcs far from a right angle. Making the fraction depend on the local angle would touch nothing but `compute_handles`. (*Chosen.*)

Xara's value

Xara uses about one third. This was measured twice: a Xara cusp handle is 0.33 of its chord, and in the Xara two-node loop described below the four handles measure 0.328 to 0.339 of the chord. On the 100-pixel quarter circle one third is off by 4.3 pixels, more than a hundred times the error of 0.39, and visible as a flattening. The library therefore keeps FreeHand's 0.39 even where it takes Xara's rules elsewhere. It is exposed as `Options::handleFraction`.

The Smooth node: the circle through three points

A Smooth node's direction **t** is the tangent, at the node, of the circle through the previous node, the node and the next node. (*Derived; consistent with the FreeHand measurements.*)

Why that circle

The requirement is that Smooth nodes placed on any circle draw that circle. Three consecutive nodes on a circle determine that circle uniquely, so the only heading that satisfies the requirement for every circle is the tangent of the circle through the node and its two neighbours. Combined with the 0.39 fraction, four Smooth nodes on a square's corners then give the near-perfect circle of Figure 1.

A closed form, by inversion

Let P be the node, A the previous node and N the next. Invert the plane about P :

$$X' = P + \frac{X - P}{\|X - P\|^2}$$

Inversion maps a circle through P to a straight line that does not pass through P , and that line is parallel to the circle's tangent at P . The line passes through the images A' and N' , so the tangent is parallel to $N' - A'$. Writing $\mathbf{u}$ for the unit vector from P to a neighbour and d for its distance:

$$\mathbf{t} \propto \frac{\mathbf{u}_{next}}{d_{next}} - \frac{\mathbf{u}_{prev}}{d_{prev}}$$

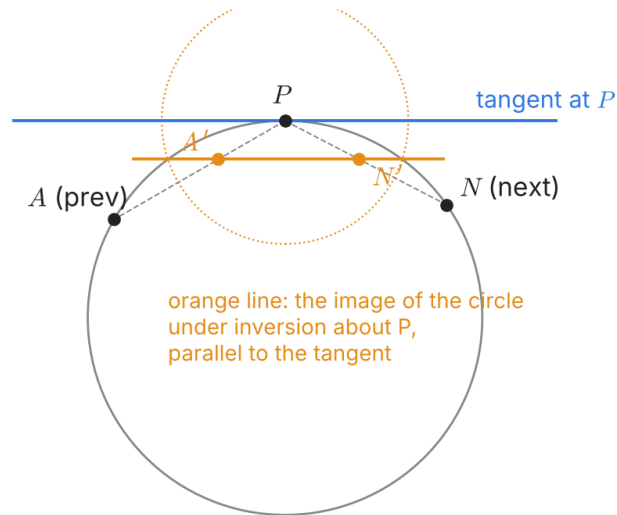


Figure 2. Inverting about P turns the circle through A , P and N into the orange line, which is parallel to the tangent at P .

The formula has three useful properties, each of which the code relies on:

- **Equal distances.** When the two neighbours are equally far away it reduces to the direction from the previous node to the next:

$$d_{prev} = d_{next} = d \Rightarrow \mathbf{t} \propto \frac{\mathbf{u}_{next} - \mathbf{u}_{prev}}{d} = \frac{N - A}{d^2}$$

- **A missing neighbour drops out.** At an end of an open path only one term remains, so the end node points straight along its one chord. No special case is needed.
- **It fails only when it should.** The vector is zero only when A and N are the same point (an out-and-back), where there is no circle at all. That case is the subject of the two-node loop and coincident-node rules below.

The code computes exactly this sum (circle_heading in 4.2.0, circle_tangent in 2.2.0), and a dragged handle on a Smooth node overrides it with the dragged direction.

The Corner: continuing a Smooth neighbour's arc

A plain Corner beside a Smooth node continues that node's curve. (*Measured: the FreeHand shield, in which every corner continues its smooth neighbours' arcs. The rule itself was reconstructed in 2.1.1.*)

The derivation is the tangent-chord angle theorem. On a circular arc from P to Q , the tangents at the two ends make equal angles with the chord, on opposite sides of it: each is the mirror image of the other in the chord's perpendicular bisector. Expressed with the chord's unit vector $\mathbf{c}$, the direction at Q is the direction at P reflected across the chord line:

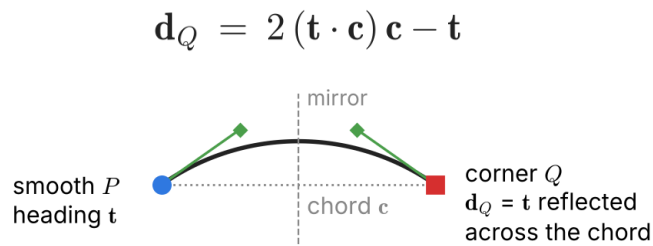


Figure 3. The corner Q continues the smooth node's arc by reflecting P 's heading across the chord.

With both handles 0.39 of the same chord, the segment is then symmetric about the perpendicular bisector, which the tests check. The construction is exact for a true circular arc. Where the smooth node's heading does not come from a circle through Q (a dragged handle, for example) it is a reasonable continuation rather than an exact one.

The rule is deliberately limited to **Smooth** neighbours. A Corner does not follow a Tangent, even one whose handle the user pulled out, nor a handle dragged on another Corner (see 4.2.0: *a Corner ignores a pulled Tangent handle*). When there is nothing to follow, the handle lies along the chord.

The cusp: chords only

A cusp is a Corner converted from a curved node. It follows nothing: each handle lies along its own chord, so the node is a point with a straight approach on each side. (*Measured in Xara: see the teardrop table under 4.1.0.*)

The Tangent: continuing the opposite chord

FreeHand's *connector point* exists to join a straight edge to a curve without a kink. (*Observed and measured in FreeHand.*) If the edge from A to P is straight, its direction at P is simply $P - A$. For the curve leaving P to have no kink, its first control point must lie on that same line. So each handle of a Tangent continues the chord on the other side of the node:

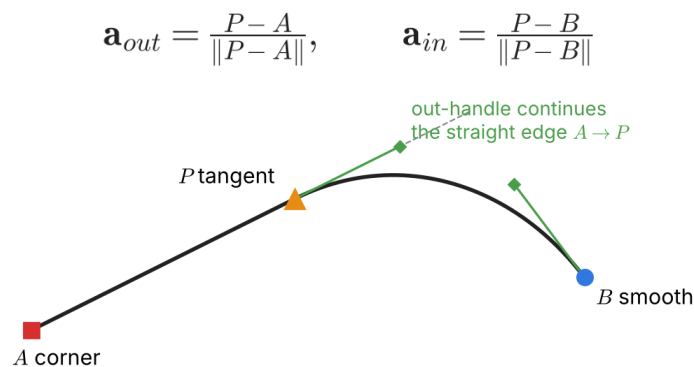


Figure 4. The tangent P joins the straight edge from A to the curve toward B without a kink.

Three consequences follow, all of which match FreeHand:

- Only the handle's length is free; its direction is fixed by the geometry. `handle_axis` returns that direction.
- When both handles are present they are not collinear (unless the three nodes are), so a Tangent between two curves makes a notch.
- At an end of an open path there is no chord on the other side, so the handle points along its own chord.

A side has a handle only if the segment on that side is already curved by the neighbour (the neighbour is Smooth, or has a dragged handle facing this node), or the user pulled one out. A handle on a straight segment would bend it, and the whole point of a Tangent is to keep a straight edge straight. So between two plain corners a Tangent is a polygon corner until a handle is pulled.

The modifier close

Closing a path is a second click on the start node, so the start node is made by two clicks. The rule is that each of its handles behaves as if its own click had ended an open path. (*Observed in FreeHand: closing on a smooth start with the modifier makes a teardrop.*)

- Its **in-handle** belongs to the closing click. A modifier click at an end gives a straight handle, so the in-handle reaches straight back to the last node, whatever that node is doing.
- Its **out-handle** belongs to the first click. If that was a plain click, the ordinary Corner rule applies. If it was a modifier click, the node was curved and is now a cusp, so the out-handle lies along its chord too.

In 2.1.1 this was expressed with a `was_smooth` flag set by the close. Since 4.2.0 it is simply the cusp flag: a modifier close turns a curved start node into a cusp, exactly as converting it to a Corner would.

The two-node loop

When a path of exactly two nodes is closed, each node has the other on both sides. The circle formula then gives a zero vector: the previous and next nodes are the same point and no circle passes through three points. The rule for this one case is that a Smooth node takes the circle with the chord as its diameter, so its heading is square to the chord. (*Measured in Xara; see 4.2.0.*)

With both handles square to the chord and 0.39 of it, each of the two segments is a cubic with control points (0,0), (0,h), (L,h), (L,0) in chord coordinates. Its furthest point from the chord is at the midpoint:

$$y(\frac{1}{2}) = \frac{3}{8}h + \frac{3}{8}h = \frac{3}{4}h = \frac{3}{4} \cdot 0.39 L = 0.2925 L$$

So a plain-closed pair of Smooth nodes draws an oval 0.585 of the chord across. A true circle would need each handle to be two thirds of the chord (the 180-degree entry in the table above); the oval is what the single 0.39 fraction gives.

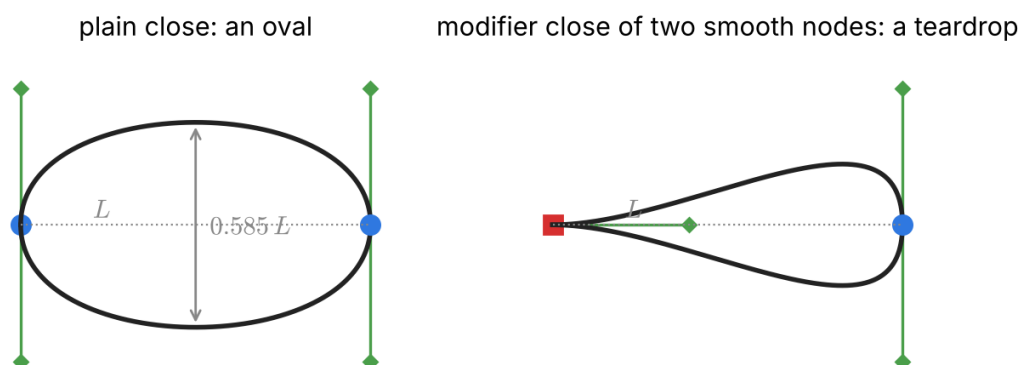


Figure 5. Two Smooth nodes closed plainly draw an oval; closed with the modifier, the start node becomes a cusp and the shape is a teardrop.

Xara's two-node loop was measured from its screenshot. Its handles are square to the chord to within one degree, they are 0.33 of the chord, and its oval is 0.502 of the chord across. The same construction with Xara's one-third fraction predicts $2 \times 0.75 \times 1/3 = 0.500$, so Xara is doing exactly this, with its own fraction. Bezigon's oval is fuller because it uses FreeHand's 0.39.

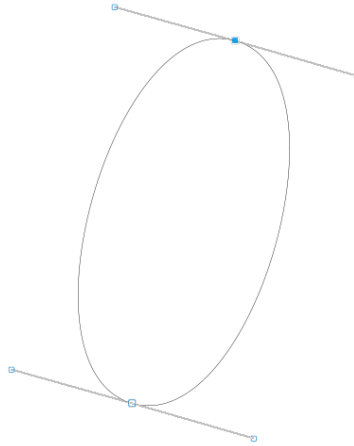


Figure 6. The Xara two-node loop that settled the rule (your screenshot).

The closing logic then applies on top of this, which is the "middle ground" chosen in 4.2.0: a plain close gives the oval, and a modifier close makes the start node a cusp, which lies along the chord and turns the oval into a teardrop. Two Corners give a doubled straight line either way.

Coincident nodes

Outside a two-node loop, two nodes either side of a node may still sit on one point: a path that goes out and comes back, or an interior node dragged exactly onto another. Then nothing special happens. The middle node has no circle, so its handles lie along their chords, and the two coincident nodes simply coexist until they are moved apart. (*Observed: your screenshots of node 4 dragged onto node 2, Figure 7.*) A neighbour sitting exactly on a node counts as absent for that side.

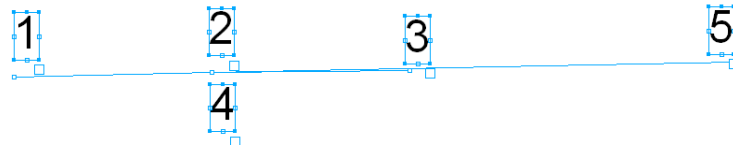


Figure 7. Node 4 dragged onto node 2: both nodes stay, and nothing else changes (your screenshot).

Straightness on a sharp node

Since 4.0.0 every side of a sharp node carries a handle, lying along its chord when it has nothing to follow. A handle lying on the chord leaves the segment straight, so the drawn shape is unchanged. To let an exporter write such a segment as a line, `segment_is_straight` tests both handles of a segment against its chord c of length L :

$$|\mathbf{h} \times \mathbf{c}| \leq \varepsilon L, \quad -\varepsilon L \leq \mathbf{h} \cdot \mathbf{c} \leq L$$

The first condition says the handle has no sideways component. The second says it points along the chord, not backwards, and does not reach past the far end; a control point beyond the far node would make the cubic overshoot its end even though it stays on the line.

Tolerances that scale

The circle formula adds terms of size $1/d$, so its result shrinks as the drawing grows. 2.1.1 compared it with a fixed threshold of 10^{-9} . For three nodes in a straight line, a distance L apart, the vector has length $2/L$, so

$$\|\mathbf{v}\| = \frac{2}{L} < 10^{-9} \iff L > 2 \times 10^9$$

Beyond about two billion units 2.1.1 would wrongly decide that a Smooth node had no circle and draw it straight. 4.x, and 2.2.0 since this round, scale the threshold by the same quantities:

$$\|\mathbf{v}\| < \varepsilon \left(\frac{1}{d_{next}} + \frac{1}{d_{prev}} \right), \quad \varepsilon = 10^{-9}$$

The comparison is then independent of the size of the drawing. The tests check Smooth nodes at scales from 10^{-6} to 10^{12} , and confirmed that the old fixed threshold fails at 3×10^9 . (This round's first prediction, that nearly straight runs at ordinary scales would trip the old code, was wrong; the real limit is this one.) (*Derived.*)

Locality

A node's handles depend on its own neighbours' positions and, for a Corner, on a neighbour's heading, which in turn depends on that neighbour's neighbours. So moving a node, or changing its type, can change handles at most two nodes away, and never beyond. A randomised test checks this on every run. It is what makes the history-free design (below) cheap: the whole path can be recomputed after any edit, and the result is the same as if only the affected nodes had been. (*Derived.*)

How the library developed

Version 2.1.1: two node types

The starting point. Corner and Smooth only, in about 220 lines, with these rules: the Smooth circle heading; a Corner reflecting a Smooth neighbour's arc; a Corner pointing straight at anything else; and the modifier close, recorded with a was_smooth flag on the start node. Every sharp node already carried handles along its chords.

Three details of 2.1.1 mattered later:

- compute_handles was tolerant of a caller who set the close state to *modifier* by hand without calling close_path: it inferred that the start node must be a corner. This hid the inconsistency rather than preventing it (see *The close state is sealed*).
- A Smooth node whose circle failed (an out-and-back) fell through to the Corner branch and copied a Smooth neighbour's arc, if it had one. This quirk was the only behavioural difference left between 2.1.1 and 4.1.0 on Corner and Smooth paths.
- drag_handle took a single node and did not recompute, so neighbouring corners went stale while a handle was dragged. The 2.1.1 test suite had a property asserting the opposite, which failed 27,495 checks; it was corrected to check what the API actually guaranteed.

Its demo closed a path by dragging the last node onto the first, reading the modifier on release.

Versions 3.x: the Tangent node

The Tangent node was added as a first-class third type, which meant rewriting rather than patching. The design was made **history-free**: the shape depends only on node types, positions and dragged handles, never on the order in which edits happened. One analysis pass works out everything the handles depend on, and a second writes the handles. A mutation test confirmed that the two halves of the analysis can run in either order with identical results.

Each intermediate version fixed something found by drawing shapes and comparing them with FreeHand and Xara:

Version	Change	Why
3.2.0	History-free redesign; born_curved; an origin-dependent toggle	The toggle was requested explicitly. The toggle chosen then (a node drawn sharp toggles sharp/smooth, a node drawn curved toggles smooth/cusp) was attributed to Xara; 4.2.0 found that attribution was wrong.
3.2.1	A cusp freezes both its sides	A cusp was collapsing when a neighbouring Tangent's handle was pulled
3.2.2	Out-and-back ellipse rule	A path leaving a node and returning to the same point had no defined heading
3.2.3	Comment and naming pass	No behaviour change
3.2.4	The start node of a modifier-closed path can be converted	Pressing S on it did nothing
3.3.0	close_path sets the start node's type; effective_type removed	Simpler: the node's type is what you see
3.4.0	Cusp memory added	An attempt to make the smooth-to-cusp toggle return to its previous shape

The FreeHand experiments behind the Tangent rules are preserved one test per experiment, in the order they were tried, in the test suite's *FreeHand experiments* group. They include converting one, two and all nodes of a closed four-node shape and of an open path, and converting all four nodes of a circle; see *How FreeHand and Xara were used*.

Version 4.0.0: handles on sharp nodes

2.1.1 gave every corner handles lying along its chords. The 3.x rewrite had dropped them, which was then the only remaining difference between the two engines: 312,620 handles out of 1,302,460. They were restored, for three reasons: any node becomes directly editable without being converted first; the change is invisible, because a control point on the chord leaves the segment straight; and it closed the gap with 2.1.1 completely. segment_is_straight was added at the same time, because straightness was no longer expressed as "no handle".

Version 4.1.0: the cusp lies on its chords

The cusp memory added in 3.4.0 was wrong. Three Xara screenshots settled it: a node made smooth and then turned back into a cusp lands on its chords, and does not recover whatever it was before. The memory was removed, along with five fields on Node and two helper functions, and the library got smaller. The measurement, from a Xara teardrop with a cusp at its point, is the cosine between each handle and the chord to its neighbour:

	Handle against chord
Xara's cusp	+0.9995
version 4.0.0	-0.5064
version 4.1.0	+1.0000

A Tangent converted straight to a Corner was made the exception, keeping its own handles. A Tangent's handles belong to the node, placed by its own rule; a Smooth node's handles belong to the curve passing through it, and have nothing to say once the curve stops.

On paths of Corner and Smooth nodes only, 4.1.0 matched 2.1.1 exactly: 1,302,460 handles over 200,000 random paths, with no differences.

The restart: auditing 4.1.0 against 2.1.1

This round of work began by restoring both code bases and comparing them on the same random paths, ignoring Tangents. They agreed everywhere except where a node's two neighbours coincide. Reading the code found three further differences that random paths could not reach: the caller-error tolerance in 2.1.1, the stale neighbours during a 2.1.1 handle drag, and the fixed tolerance. The API also differed: 4.1.0 had `make_node`, `set_node_type` and a path-level `drag_handle`; 2.1.1 had none of them.

The goal set at this point was that both versions should behave identically on Corner and Smooth nodes, with the code kept simple, readable and free of special cases, and integration into PhotoLine (a C++ application) made as easy as possible.

Versions 4.2.0 and 2.2.0

Every decision in this round was discussed and agreed before any code changed. They are listed in the order they were made.

Where the out-and-back ellipse came from

Nobody could recall the origin of 4.1.0's out-and-back ellipse. The history recorded it as taken from Xara in 3.2.2; your Xara screenshot of a two-node loop (Figure 6) showed the shape. Two things were then noticed. The Xara shape is a *closed path of two nodes*, which 4.1.0 could not even build, because `close_path` required three. And the 4.1.0 demo could not reach the out-and-back case at all, because a click near an existing node grabs that node instead of placing a new one.

A conflict in the record was also found. A comment in the 4.1.0 tests says FreeHand draws an ellipse when you click A, modifier-click B, click C and drag C onto A. The 4.1.0 history says FreeHand was never observed doing this. The two cannot both be right, and the question is still open (see *Open questions*).

Two nodes are enough to close a path

You decided that a two-node vector shape should be possible, as in Xara, and that three was an artificial limit. `close_path` now accepts two nodes. A single node is a valid path but cannot be closed, since there is nothing to join it to. (*Taken from Xara.*)

With two nodes allowed, the two-node loop needs no new mechanism: each node has the other on both sides, which is exactly the out-and-back condition. And the closing logic already sits in front of the heading rule, because a modifier close turns the start node into a Corner (a cusp) before any heading is computed. So a plain close gives Xara's oval and a modifier close gives a teardrop. You chose this split explicitly, as the middle ground between Xara (which simply gives a smooth loop) and the library's own closing logic. (*Chosen.*)

Every other coincidence: nothing happens

The same zero vector also appears when an interior node is dragged onto another, or when an open path returns to its start. Applying the ellipse there would have meant one accidental overlap reshaping a large path. You decided to follow FreeHand: nothing happens, both nodes coexist until moved apart (Figure 7). Because the arithmetic cannot tell the two situations apart, the ellipse rule was narrowed structurally: it fires

only for a closed path of exactly two nodes. Everywhere else a Smooth node with no circle lies along its chords. (*Observed in FreeHand; the narrowing is Chosen.*)

This also removed 2.1.1's quirk of copying a Smooth neighbour's arc in this case. Welding (merging any node onto any other) was considered and dropped: it is not standard FreeHand behaviour and would have enlarged the library for little gain.

The close state is sealed

In 2.1.1 and 4.1.0 the close state was a public field that a caller could set directly, without calling `close_path`. If the start node was Smooth, the two versions then disagreed: 2.1.1 inferred the start must be a corner and drew the teardrop, 4.1.0 trusted the node and drew it round. You worked through the pentagon example (five modifier clicks, then a modifier close) to see why the two pictures differed. Rather than choosing between a forgiving library and a strict one, the inconsistency was made impossible: the close state can now only be read, and only `close_path` can set it, in the same step that changes the start node. Setting it any other way is a compile error. (*Chosen.*)

2.1.1 brought up to the same rules and API

- `make_node(x, y, smooth)` and a conversion function, `set_node_smooth`, matching 4.2.0's `set_node_type`.
- `close_path` returns whether it closed the path.
- `drag_handle` takes the path and an index and recomputes at once, so neighbouring corners follow a dragged handle live. The old signature could not do this, because a single node cannot reach its neighbours.
- The scaled tolerance, so the two versions agree at every scale, not just the tested ones.

Afterwards the two versions were compared on 300,000 random paths, including two-node loops, out-and-back cases, conversions and drags. Six paths in 100,000 differed in the ninth significant digit, because 2.2.0 divided where 4.2.0 multiplied by a reciprocal. Matching the order of arithmetic made the output byte-for-byte identical. That was the only change to algorithm code made because of a test; it changed rounding, not behaviour.

Closing gesture: from collide-and-click to drag-to-close

The first design was *collide and click*: drag the end node onto the start node, where it snapped exactly into place while in range; then click the start node, plain or with the modifier, to close. Live snapping was chosen over snapping on release so the user sees the result before committing. When you tried it in the demo, the extra click proved unintuitive. The gesture became simpler: releasing the end node on the start node closes the path, and the modifier held at release gives a modifier close. The end node is merged into the start node, since keeping it would leave a zero-length segment. Clicking the start node still closes as before. (*Chosen by testing in the demo.*)

The cusp rule corrected

Testing the 2.2.0 demo, you converted a node drawn as a Corner (between Smooth nodes) to Smooth and back. It returned to its original dip instead of becoming a cusp, which is what Xara does. In a two-node loop the conversion appeared to do nothing at all.

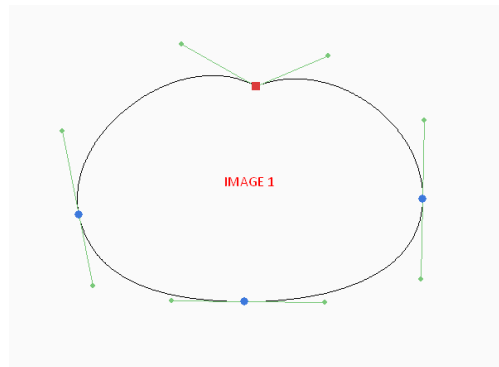


Figure 8. Your test: a node drawn as a corner between smooth nodes. It continues their arcs, giving the dip.

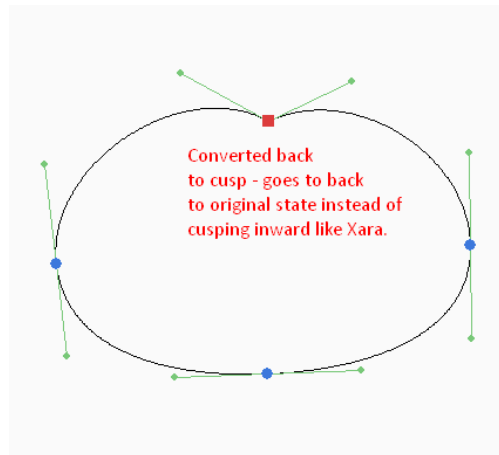


Figure 9. Converted to smooth and back to corner, it returned to the dip instead of cusping as Xara does.

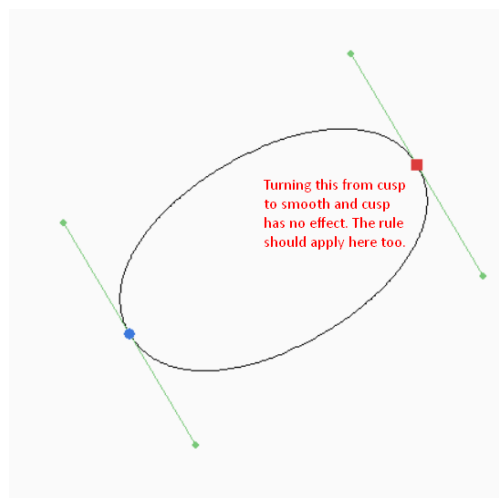


Figure 10. In a two-node loop, converting to smooth and back had no visible effect.

The cause was history: the library remembered how each node was drawn (born_curved) and only made a cusp of nodes drawn curved. This was the origin-dependent toggle introduced in 3.2.0 and attributed to Xara, and the comparison showed that attribution was wrong. The rule is now: **any curved node converted to a Corner becomes a cusp**, however it was drawn. A plain corner, which follows its neighbours' arcs, comes only from a plain click. born_curved was replaced by a cusp flag, set by conversion and by a modifier close, and the two-node case follows automatically: the cusp lies along the chord and the oval becomes a teardrop. (Observed in Xara.)

A Corner ignores a pulled Tangent handle

Testing 4.2.0, you placed a Tangent between two Corners (no handles, correctly) and pulled one of its handles out. The Corner beside it swung its own handle out to continue the new arc.

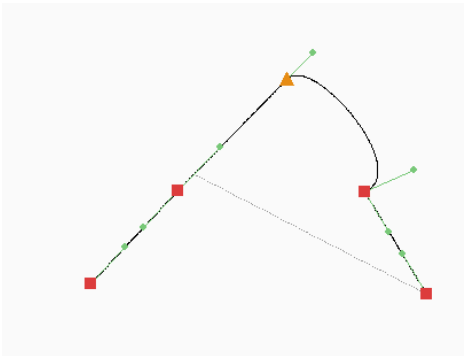


Figure 11. Your test: a Tangent between corners with one handle pulled; the corner beside it swung its handle out.

This was not a regression: running the same steps on the original 4.1.0 code gave exactly the same handle, and a 4.1.0 test required it. The Corner rule had continued any curved handle facing it, and a pulled Tangent handle counted. The record gives no FreeHand basis for it. The rule is now that a Corner continues only a **Smooth** neighbour's arc; a pulled handle bends only the Tangent's own end of the segment. The change affected only that case: three of 113 Tangent table entries, and no Corner or Smooth case. *(Chosen by testing in the demo.)*

Documentation and verification

The permutation tables were rebuilt for both versions and extended with three-node loops, polygons, two-node shapes, out-and-back paths and edit sequences. Every case was run against the original 4.1.0 code. The only differences are the four agreed changes: two-node loops, out-and-back with a Smooth node, the cusp rule, and the pulled Tangent handle.

How FreeHand and Xara were used

FreeHand: the primary reference

FreeHand defines what the tool should feel like, and its numbers are kept wherever it gives a clear answer. It was used in three ways.

The shield. A closed shape of alternating Corner and Smooth nodes was drawn in FreeHand and its node positions and handles read off the screen. It is the first example scene in the 4.2.0 demo and a test in the suite, and it is the evidence for the Corner reflection rule.

Tangent measurements. Tangent nodes were placed directly in FreeHand and their handle lengths read in pixels; the tests reproduce them within a few pixels:

Clicks	What FreeHand showed
PC, TN, MC, MC, closed	The tangent's handle goes toward the smooth node, continuing the edge from the corner. 258 px.
TN, TN, TN, MC, closed	Only the two tangents beside the smooth node have a handle (124 and 121 px); the one between two tangents has none.
TN, MC, TN, modifier close	The start node becomes a corner with handles straight at its neighbours (152 and 158 px). The last node's handle goes toward the smooth node (161 px); toward the start node there is none.

Clicks	What FreeHand showed
TN, MC, TN, MC, open (a wave)	The first tangent points at node 2; the middle one has two handles making a notch (134 and 128 px). Another drawing measured 146 px; ending on a tangent gave a handle back at the smooth node (110 px).

Conversion experiments. Nodes of a closed four-node shape (corners along the top, smooth along the bottom), of an open path and of a circle were converted to Tangents one at a time, two at a time and all together, and the directions noted. Each experiment is a test, in the order it was tried.

Xara: the second reference

Xara was consulted where FreeHand's behaviour was unclear, unhelpful, or could not be observed. It settled:

- **The cusp.** Its handles lie on its chords (the teardrop measurement under 4.1.0), and since 4.2.0, any curved node converted to a corner becomes one.
- **Conversion discards dragged handles.** Confirmed against Xara.
- **The two-node loop.** Handles square to the chord, measured from the screenshot in Figure 6.

Xara's handle fraction of about one third was measured and deliberately not adopted.

The demo: the third reference

Some questions have no answer in either program, because they are about what feels intuitive. These were settled by drawing in the demo: the closing gesture, the cusp rule's failure, and the Corner following a pulled Tangent were all found this way in this round. A behaviour that is consistent with the rules but surprising in the hand is treated as a bug.

Where FreeHand was overridden

These are the places the library knowingly does something other than FreeHand, or something FreeHand has no equivalent for.

	FreeHand	Bezigon	Why
1. Sharp nodes carry handles	A plain corner shows no handles; it must be converted to be shaped.	Every corner carries handles lying along its chords. The drawn shape is identical.	Direct manipulation, and continuity with 2.1.1. A host can hide handles on straight segments. <i>(Chosen.)</i>
2. A converted Tangent shows its handle at once	Converting a corner to a Tangent beside a smooth neighbour leaves it without a handle until something else changes.	The handle appears at once, exactly as for a Tangent placed directly.	The delay looked like a redraw artefact, and reproducing it would need edit history. <i>(Chosen.)</i>
3. The cusp	No equivalent of converting a smooth node into a cusp.	Any curved node converted to a Corner is a cusp, its handles on its chords.	Requested as more intuitive. <i>(Taken from Xara.)</i>
4. Two-node loops	Not observed.	Two nodes can be closed: an oval with a plain close, a teardrop with the modifier.	Xara's shape, combined with the library's closing logic. <i>(Taken from Xara; the split is Chosen.)</i>

	FreeHand	Bezigon	Why
5. Four Tangents on a circle	Keeps the circle's own handles and shows four notches.	Conversion resets handles, and a Tangent has no automatic handle toward another Tangent, so the circle becomes a square until handles are pulled.	Follows from the conversion and Tangent rules without special cases. <i>(Chosen.)</i>
6. A Corner beside a pulled Tangent	Not measured.	The Corner keeps its handle on its chord.	A pulled handle should bend only its own end. <i>(Chosen by testing.)</i>
7. Closing by drag	Not part of the comparison.	The demo closes on releasing the end node on the start node, merging the two.	Fewer steps than collide-and-click. A host decision, not a library rule. <i>(Chosen by testing.)</i>

And the places where FreeHand was deliberately kept against Xara: the handle fraction (0.39, not one third), and coincident nodes (nothing happens, rather than a loop).

Evidence register

Every rule, its class, its evidence, and the version it arrived in.

Rule	Class	Evidence	Since
Handle is 0.39 of its own segment	Measured, Derived	FreeHand handles; quarter-circle derivation	2.1.1
Smooth: tangent of the circle through three nodes	Derived	Points on a circle reproduce it; inversion	2.1.1
End node points along its chord	Derived	Falls out of the circle formula	2.1.1
Corner continues a Smooth neighbour's arc	Measured	FreeHand shield; tangent-chord theorem	2.1.1
Corner with nothing to follow lies on its chord	Observed, Chosen	FreeHand draws straight; the handle is the override	2.1.1, restored 4.0.0
Modifier close: in-handle straight back	Observed	FreeHand teardrop	2.1.1
Cusp: handles on its chords	Measured	Xara teardrop, cosine +0.9995	4.1.0
Any curved node made a Corner is a cusp	Observed	Xara, compared in the 2.2.0 demo	4.2.0, 2.2.0
Conversion discards dragged handles	Observed	Xara	4.1.0
A Tangent keeps its handles as a cusp	Chosen	Its handles belong to the node	4.1.0
Tangent handles continue the opposite chord	Observed, Measured	FreeHand placements and experiments	3.x
Tangent handle only on a curved segment	Observed	FreeHand experiments	3.x
Converted Tangent shows its handle at once	Chosen	Override 2	3.2.0
History-free computation	Chosen	Design	3.2.0
Corner ignores a pulled Tangent handle	Chosen	Your test in the demo	4.2.0
Two nodes can be closed	Observed	Xara two-node loop	4.2.0, 2.2.0

Rule	Class	Evidence	Since
Two-node loop: square to the chord	Measured	Xara: handles within 1° of square, oval 0.502 of chord	3.2.2, narrowed 4.2.0
Two-node loop follows the closing logic	Chosen	Your decision	4.2.0, 2.2.0
Other coincidences: nothing happens	Observed	FreeHand behaviour as you described it (Figure 7)	4.2.0, 2.2.0
Close state set only by cclose_path	Chosen	Removes the caller error	4.2.0, 2.2.0
Tolerances scale with distance	Derived	Fails beyond 2×10^9 otherwise	3.x; 2.2.0
drag_handle recomputes the path	Chosen	Neighbours follow live	3.x; 2.2.0
Drag-to-close with modifier at release	Chosen	Your test in the demo	4.2.0, 2.2.0 demos