

DC-Focus: A Hyperbolic Focus+Context Formalism for Scalable Feature-Model Visualization

¹Muhammad Nura Malami, ¹Muhammad Garba, ¹Abubakar Ahmad Aliero, ¹Afar Aminu, ¹Musa Muhammad Lawal, ¹Saratu Ibrahim Mungadi, ²Bashar Bin Usman

¹Department of Computer Science, Faculty of Computing, Abdullahi Fodio University of Science and Technology, Aliero, Nigeria

²Department of Computer Science, Faculty of Engineering and Computer Sciences, Cosmopolitan University, Abuja, Nigeria

*Corresponding authors' email: Nuragwn44@gmail.com

ABSTRACT

Feature models in industrial-scale Software Product Lines (SPL) routinely exceed a thousand features, yet the dominant visual metaphor for exploring them remains the Euclidean tree, whose readability degrades sharply as the number of nodes grows. Existing hyperbolic tree browsers address the general rendering problem but do not natively capture feature-model semantics — typed nodes (root/branch/child), a dependency relation separate from the tree structure, and cross-tree constraints — which is the specific gap this paper addresses. This paper presents a formal focus+context visualization model that embeds a feature model as a labelled graph in the Poincare disk and defines a Mobius-transformation-based re-centring operator to bring any selected feature into visual focus while preserving the surrounding structural context. We define the model as a seven-tuple over feature nodes, directed relationships, a type function, a root, a dependency relation, visualization attributes, and cross-tree constraints, and we specify five supporting algorithms for node insertion, focus navigation, node update, node deletion, and ancestry tracing. We show that the focus transformation is an isometry of the disk and therefore preserves the topological structure of the underlying feature model under re-centring. The model underlies a working browser-based prototype; this paper's contribution is the formal model, the isometry proof, and the algorithmic specification, and we do not report quantitative scalability results here. A worked numerical example illustrates the transformation on a small feature tree, and we discuss the architectural realization of the model as a four-layer, browser-based system. The formalism is intended as a reusable, implementation-independent foundation for feature-model visualization; a systematic empirical evaluation of its scalability across feature models of varying size is identified as future work (Section 7).

Received: 18 Sept 2026

Accepted: 30 Sept. 2026

Published: 06 Oct. 2026

Keywords:

Software Product Line Engineering, Variability management, feature model visualization, Focus+context, Hyperbolic geometry, Poincare disk, Mobius transformation

INTRODUCTION

Software Product Line Engineering (SPLE) organizes families of related software products around a shared set of core assets, with variability among family members captured in a feature model (Pohl & Metzger, 2018; Quinton et al., 2020). As product lines mature, feature models grow from illustrative examples of a few dozen features to industrial artifacts comprising hundreds or thousands of features connected by complex dependency relationships (Diaz et al., 2022; Lindohf et al., 2021). Visualizing and navigating models at this scale is a longstanding open problem: conventional tree-based renderings, still the dominant metaphor in widely used tools, suffer from graphical overload once the number of visible nodes exceeds a few hundred, forcing users to scroll, collapse subtrees, or rely on textual search rather than direct visual exploration (Rabiser et al., 2018).

A separate line of work in information visualization has addressed analogous scale problems through focus+context techniques, in which the region of current interest is rendered at high detail while the remaining structure is compressed but kept visible, preserving the user's sense of location within the whole (Furnas, 1986). Hyperbolic and Poincare-disk-based layouts are a well-established instance of this idea for tree and

graph data, since the exponential growth of hyperbolic space allows an exponentially larger neighbourhood to be displayed at a fixed visual radius than is possible in Euclidean space (Lamping et al., 1995). Despite this, focus+context visualization has seen limited adoption within variability-management tooling specifically, where feature-model semantics (root/branch/child typing, dependency edges, genealogy) impose additional structure that a generic hyperbolic-tree layout does not natively capture.

The scalability problem addressed here is in fact three distinct issues that prior work often conflates. The first is *visualization scalability* — whether a layout remains readable and navigable as node count grows; generic hyperbolic browsers (Lamping et al., 1995) address this at the level of arbitrary trees. The second is *preservation of feature-model semantics* — whether the visualization retains node typing, dependency edges, and cross-tree constraints as first-class, queryable structure rather than opaque tree geometry; this is not addressed by generic hyperbolic browsers, and is the primary gap DC-Focus targets. The third is *interactive editing* — whether the model supports structural modification (insertion, update, deletion) without invalidating the layout or the genealogy trace; this is addressed here through the algorithms of Section 4. Concretely, the scalability claims in this paper

are claims about *computational complexity* of the model's operations (Section 4) and about the *metric property* that produces the focus+context visual effect (Section 3.3–3.4); they are not, in this paper, claims about measured rendering performance or user navigation efficiency at scale, which we identify as an empirical question for future work (Section 7). This paper formalizes a focus+context visualization model purpose-built for feature models. The contribution is threefold. First, we give a formal seven-tuple definition of a feature model annotated with the structure needed for scalable visualization: node types, dependency relations, visualization attributes, and cross-tree constraints. Second, we define a node-embedding and focus-transformation pair, grounded in the Poincare disk model of hyperbolic geometry, and show that re-centring the view on a newly focused node is an isometry, so no structural distortion is introduced by navigation. Third, we specify five algorithms, covering node creation, focus navigation, node modification, node deletion, and ancestry tracing, that together define the operational semantics of the model. A worked numerical example and an accompanying four-layer browser-based architecture illustrate how the formalism can be realized in a deployable system.

Concretely, the paper's testable contributions are: (i) a formal seven-tuple model of a feature model annotated for focus+context visualization, with explicit domains and codomains for each component (Section 3.1); (ii) a proof that the focus transformation T_f is a hyperbolic isometry, hence structure-preserving under re-centring (Section 3.4); and (iii) complete algorithmic specifications, including preconditions, postconditions, error handling, and worst-case time complexity, for all five operations (Section 4). Empirical validation of rendering performance and navigation efficiency at industrial scale is explicitly out of scope for this paper and is proposed as future work.

The remainder of this paper is organized as follows. Section 2 situates the work relative to prior focus+context and hyperbolic visualization techniques. Section 3 presents the formal model. Section 4 specifies the algorithms. Section 5 describes a reference architecture. Section 6 works through a small illustrative example. Section 7 discusses the properties and limitations of the formalism and directions for future work, and Section 8 concludes.

Related Work

Focus+context visualization has a substantial history in information visualization, motivated by the general problem of displaying more information than fits legibly on a single screen while retaining orientation within the larger structure (Furnas, 1986). Hyperbolic tree browsers were among the earliest instances of this idea applied to tree-structured data: by embedding a tree in the Poincare disk, a node's neighbourhood can be rendered at large visual scale near the centre of the disk while distant nodes are compressed toward the boundary, and re-centring the display on a different node is achieved through a conformal (angle-preserving) transformation of the disk rather than a re-layout of the tree (Lamping et al., 1995). This property, that navigation is a transformation of coordinates rather than a recomputation of structure, is what allows such browsers to remain responsive as the number of nodes grows.

Within variability management specifically, existing tool support has generally relied on Euclidean, FODA-style tree renderings, augmented in some tools by subtree collapsing or textual search to cope with scale. Commercial and open-source feature-modelling environments provide rich editing and analysis functionality but do not natively provide a

focus+context visual metaphor, and emerging browser-based feature-model editors have addressed collapsing and search-based navigation rather than a geometric focus transformation (Heß et al., 2025). The formalism presented here is intended to close this gap by adapting the Poincare-disk focus+context approach to the specific semantic structure of a feature model, namely typed nodes (root, branch, child), directed dependency relations, and node-level metadata required for genealogy traceability.

MATERIALS AND METHODS

Formal Model

A formal representation is adopted here for the same reasons formal models are used elsewhere in software engineering: it makes the semantics of the visualization and its operations precise and independent of any particular implementation, it supports correctness and complexity analysis of the associated algorithms, and it provides a stable foundation for extension. The model represents a feature model as a labelled, typed graph enriched with the geometric and semantic information needed to support focus+context visualization.

Structure of the Model

Let the DC-Focus be represented by the seven-tuple

$$\text{DC-Focus} = (N, E, T, R, D, A, C) \quad (1)$$

where N denotes the finite set of feature nodes; E denotes the set of directed relationships connecting feature nodes; T is the feature type function; R is the root feature of the product line; D denotes dependency relationships among features; A denotes visualization attributes assigned to each node; and C denotes the set of cross-tree constraints. The underlying variability model can equivalently be expressed as the graph $M = (N, E)$, $N = \{n_1, n_2, \dots, n_m\}$, $E = \{e_1, e_2, \dots, e_k\}$ (2), (3) Each node in N corresponds to one feature or variation point, and each directed edge in E represents a parent-child relationship between two nodes.

To make Eq. (1) precise, each component is defined with an explicit domain and codomain:

- $T : N \rightarrow \{\text{root}, \text{branch}, \text{child}\}$ — a total function assigning exactly one structural type to every node; exactly one node has $T(n) = \text{root}$.
- $R \in N$ — the distinguished node with $T(R) = \text{root}$; R is unique by construction (Section 4.1, step 2).
- $D = (N, \Delta)$, with $\delta : N \times N \rightarrow \Delta$ a partial function (δ is defined only for pairs of nodes that are actually related by a dependency; Section 3.2).
- $A : N \rightarrow (D^2 \times \mathbb{R}^+ \times \{\text{Orange}, \text{Purple}, \text{Blue}\})$ — a total function assigning each node a disk position $\phi(n) \in D^2$ (Section 3.3), a visual radius $r(n) \in \mathbb{R}^+$ (Section 4.2, step 3), and a display colour $c(n)$ (Section 3.5).
- $C \subseteq N \times N \times \{\text{Requires}, \text{Excludes}\}$ — the set of cross-tree constraints, realized in the current model via the same labelled relation used for D (Section 3.2). Feature-group cardinality (XOR/OR-groups), feature optionality, and mandatory-child semantics beyond the Requires/Excludes vocabulary are not modelled in the current version of DC-Focus; we identify their integration as an explicit direction for future work (Section 7) rather than claim coverage the model does not provide.

Dependency Representation

Dependencies between features are represented as

$$D = (N, \Delta), \quad \Delta = \{\text{Requires}, \text{Excludes}, \text{Uses}, \text{Depends}, \text{Extends}\} \quad (4)$$

For any two features n_i, n_j in N , the dependency relationship between them is given by $\delta(n_i, n_j) = dk$, where dk is an

element of Δ . Representing dependencies as a labelled relation over pairs of nodes, rather than embedding dependency logic in the tree edges themselves, allows dependency tracing and consistency verification to be performed independently of the visualization layer.

E and D are deliberately disjoint in role. $E \subseteq N \times N$ contains exactly the parent-child (tree) edges used for the hierarchical layout in Eq. (2)–(3) and consumed by TRACE_ANCESTRY (Section 4.5); it is what INSERT_NODE (Section 4.1) writes to. D, by contrast, is an auxiliary labelled relation over $N \times N$ that records semantic dependencies (Requires, Excludes, Uses, Depends, Extends) that are orthogonal to parenthood and contribute no edges to E. A pair (n_i, n_j) may be a parent-child pair in E, a dependency pair via δ , both, or neither; membership of a pair in D never implies membership in E, and vice versa. This separation is what allows dependency tracing and consistency verification (Section 3.2) to be performed independently of the tree layout and the focus transformation (Section 3.4), which act on E and $\phi(N)$ only.

Node Distance Metric

Node positions are embedded in the Poincare disk D^2 using a dotted-circle geometric layout. Writing $\phi(u)$ and $\phi(v)$ for the embeddings of two nodes u and v in D^2 , the hyperbolic distance between them is:

$$d(u, v) = 2 \cdot \tanh^{-1} (|\phi(u) - \phi(v)| / |1 - \bar{\phi}(u) \cdot \phi(v)|) \quad (5)$$

where $\bar{\phi}(u)$ denotes the complex conjugate of $\phi(u)$. A defining property of this metric is that equal Euclidean displacements correspond to much larger hyperbolic distances near the boundary of the disk than near its centre. Consequently, nodes positioned near the centre appear large and well separated (in focus), while nodes near the boundary appear small and compressed (context), which is what produces the focus+context visual effect without any explicit subtree collapsing or hiding step.

Focus Transformation

Bringing a node into focus is implemented as a Mobius transformation of the disk. Given a user-selected focus node f in N , the transformation T_f re-centres the disk at $\phi(f)$:

$$T_f(z) = (z - \phi(f)) / (1 - \bar{\phi}(f) \cdot z) \quad (6)$$

For every node n in N , the updated position after focusing on f is

$$\phi'(n) = T_f(\phi(n)) = (\phi(n) - \phi(f)) / (1 - \bar{\phi}(f) \cdot \phi(n))$$

T_f is a bijection from D^2 to D^2 mapping $\phi(f)$ exactly to the origin. It is an isometry of the hyperbolic plane: all hyperbolic distances $d(u, v)$ are preserved under the transformation for every pair of nodes u, v in N . A direct consequence is that focus navigation never distorts the topological structure of the feature model; only the visual coordinates change, and they change in a way that leaves all pairwise hyperbolic distances, and therefore the relative structure of the model, invariant.

Proof sketch. Transformations of the form of Eq. (6) are Blaschke factors: for any $a \in D^2$ with $|a| < 1$, the map $z \mapsto (z - a)/(1 - \bar{a}z)$ is a bijection of D^2 onto itself, and Blaschke factors satisfy the pseudo-hyperbolic identity

$$|T_f(u) - T_f(v)| / |1 - \overline{T_f(u)} \cdot T_f(v)| = |u - v| / |1 - \bar{\phi}(u) \cdot \phi(v)| \quad \text{for all } u, v \in D^2.$$

Since Eq. (5) defines $d(u, v)$ as $2 \cdot \tanh^{-1}$ of exactly the left-hand quantity evaluated at $\phi(u)$, $\phi(v)$, substituting $T_f(\phi(u))$, $T_f(\phi(v))$ for $\phi(u)$, $\phi(v)$ leaves d unchanged, i.e. $d(T_f(u), T_f(v)) = d(u, v)$. We verify this numerically for the example in Section 6, where $d(r, b)$ computed before and after T_f agrees to machine precision.

This isometry result guarantees only that pairwise hyperbolic distances among nodes are preserved under re-centring; it is a geometric, not a semantic, guarantee. It does not by itself

establish that every constraint in C (Section 3.1) remains satisfiable after a structural edit — constraint preservation under INSERT_NODE, UPDATE_NODE, and DELETE_NODE is addressed separately through the pre/postconditions of those algorithms (Section 4), not by T_f .

Node Colour Function

Each node is assigned a display colour $c(n)$ based on its type $\tau(n)$, giving users an immediate visual cue to a node's structural role without requiring them to read a label: $c(n)$ = Orange if $\tau(n)$ = root; Purple if $\tau(n)$ = branch; Blue if $\tau(n)$ = child

Algorithmic Specification

This section specifies the algorithms that operate on a DC-Focus feature model FM. Each algorithm takes FM as input and returns an updated FM, and where applicable a refreshed visual rendering derived from the transformation defined in Section 3.4. Following the reviewer's request, all five algorithms below are given as complete pseudocode with explicit preconditions, postconditions, error handling, and worst-case time complexity.

Node Insertion

INSERT_NODE(FM, name, type, value, description, parent_id)

Input : FM: current feature model; name, type, value, description: node attributes; parent_id

Output: Updated FM

- i. Precondition: FM is a well-formed DC-Focus instance (Eq. 1); if $\text{type} \neq \text{root}$, parent_id references an existing node in N .
- ii. Generate unique nodeID n_{new}
- iii. IF $\text{type} = \text{"root"}$ AND $\exists r \in N$ such that $\tau(r) = \text{root}$ THEN
- iv. RETURN error: "A root node already exists"
- v. END IF
- vi. IF $\text{type} \in \{\text{branch}, \text{child}\}$ AND $\text{parent_id} = \emptyset$ THEN
- vii. RETURN error: "Non-root node requires a valid parent"
- viii. END IF
- ix. Set $\tau(n_{\text{new}}) \leftarrow \text{type}$; $\rho(n_{\text{new}}) \leftarrow \text{parent_id}$; $\delta(n_{\text{new}}) \leftarrow \delta(\text{parent_id}) + 1$; $v(n_{\text{new}}) \leftarrow \text{value}$
- x. Add directed edge $(\text{parent_id}, n_{\text{new}})$ to E ; persist n_{new} and its attributes
- xi. Recompute $\phi(n)$ for all $n \in N$ // re-embed the updated tree
- xii. RETURN updated FM

Postcondition: $n_{\text{new}} \in N$ with $\tau(n_{\text{new}}) = \text{type}$; if $\text{type} = \text{root}$, $n_{\text{new}} = R$; E and ϕ are consistent with the new tree.

Error handling: rejects a second root (step 2–3) and a non-root node without a valid parent (step 5–6); both are returned as explicit errors, not silently ignored.

Complexity: $O(|N|)$, dominated by the re-embedding of all nodes in step 10; the validity checks in steps 2 and 5 are $O(1)$ with a hash-indexed node table.

Focus+Context Navigation

FOCUS(FM, f)

Input : FM: current feature model; $f \in N$ user-selected node

Output: Updated visual rendering of FM, centred on f

Precondition: $f \in N$.

- i. Retrieve $\phi(f)$

- ii. FOR EACH $n \in N$: $\phi'(n) \leftarrow (\phi(n) - \phi(f)) / (1 - \phi(f) \cdot \phi(n))$ // re-centre via Eq. (6)
- iii. FOR EACH $n \in N$: $r(n) \leftarrow r_{\max} \cdot (1 - |\phi'(n)|^2)$ // visual radius
- iv. FOR EACH $n \in N$: assign $c(n)$ by type per Section 3.5
- v. Render all $n \in N$ at position $\phi'(n)$, radius $r(n)$, colour $c(n)$
- vi. Populate node-information panel with nodeID, $\tau(f)$, $\delta(f)$, $v(f)$, $\rho(f)$, TRACE_ANCESTRY(FM, f)
- vii. RETURN updated rendering

Postcondition: $\phi'(f) = 0$ (Section 3.4); for every $u, v \in N$, $d(\phi'(u), \phi'(v)) = d(\phi(u), \phi(v))$ by the isometry result of Section 3.4; E and D are unchanged.

Error handling: if $f \notin N$, FOCUS returns error “Unknown focus node” and leaves the current rendering unchanged.

Complexity: $O(|N|)$, one constant-time transformation per node across steps 2–4; TRACE_ANCESTRY in step 6 adds $O(\text{depth}(f))$ (Section 4.5).

Node Update

UPDATE_NODE(FM, n , name', value', description')

Input : FM: current feature model; $n \in N$: node to modify; name', value', description': new attribute values

Output: Updated FM

Precondition: $n \in N$.

- i. IF $n \notin N$ THEN
- ii. RETURN error: “Unknown node”
- iii. END IF
- iv. $v(n) \leftarrow \text{value}'$; $\text{description}(n) \leftarrow \text{description}'$; $\text{name}(n) \leftarrow \text{name}'$ // attribute-only update
- v. $\tau(n)$ and $\rho(n)$ are NOT modified by this operation // type and parent are immutable post-creation
- vi. IF a re-parenting of n is requested THEN
- vii. RETURN error: “Re-parenting is not permitted; $\rho(n)$ is immutable once set”
- viii. END IF
- ix. RETURN updated FM

Postcondition: $\text{name}(n)$, $\text{value}(n)$, $\text{description}(n)$ reflect the new values; $\tau(n)$, $\rho(n)$, and E are unchanged, so TRACE_ANCESTRY(FM, n) (Section 4.5) returns the same path as before the update.

Error handling: an update to an unknown node (step 1–3) or an attempted re-parenting (step 6–7) is rejected with an explicit error rather than silently applied or ignored, which is what keeps the ancestry path returned by TRACE_ANCESTRY stable once established.

Complexity: $O(1)$, a direct attribute write with a hash-indexed node table.

Node Deletion

DELETE_NODE(FM, n)

Input : FM: current feature model; $n \in N$: node to delete

Output: Updated FM

Precondition: $n \in N$.

- i. IF $n \notin N$ THEN
- ii. RETURN error: “Unknown node”
- iii. END IF
- iv. IF $\tau(n) = \text{root}$ THEN
- v. RETURN error: “The root node cannot be deleted”
- vi. END IF
- vii. IF $\exists m \in N$ such that $(n, m) \in E$ THEN // n has at least one child
- viii. RETURN error: “Only leaf nodes with no children may be deleted”
- ix. END IF

- x. IF $\exists (n_i, n_j) \in D$ such that $n \in \{n_i, n_j\}$ (n has an active dependent) THEN
- xi. RETURN error: “Node has active dependency relations; remove them first”
- xii. END IF
- xiii. Remove n from N ; remove $(\rho(n), n)$ from E ; persist deletion
- xiv. Recompute $\phi(n)$ for all remaining $n \in N$ // re-embed the updated tree
- xv. RETURN updated FM

Postcondition: $n \notin N$; for every remaining pair $u, v \in N$ with $u, v \neq n$, $(u, v) \in E$ iff it held before the deletion; the tree defined by (N, E) remains connected and rooted, since only childless, non-root, dependency-free nodes are removable.

Error handling: deletion is rejected for an unknown node (1–3), the root (4–6), a node with children (7–9), or a node with unresolved dependency relations in D (10–12); each case returns a distinct, explicit error.

Complexity: $O(\text{deg}(n) + |N|)$: $O(\text{deg}(n))$ to check for children and dependents in steps 7 and 10 ($O(1)$ with an adjacency index), plus $O(|N|)$ for the re-embedding in step 14.

Ancestry Tracing

TRACE_ANCESTRY(FM, n)

Input : FM: current feature model; $n \in N$ query node

Output: Ordered list path = [root, ..., n]

Precondition: $n \in N$.

- i. path $\leftarrow$ empty list; current $\leftarrow n$
- ii. WHILE $\rho(\text{current}) \neq \emptyset$ DO path.prepend(current); current $\leftarrow \rho(\text{current})$
- iii. Path.prepend(current) // the root node
- iv. RETURN path

Because parent pointers are immutable once set (Section 4.3), TRACE_ANCESTRY runs in time proportional to the depth of n and always terminates at R , supporting the genealogy-traceability requirement independently of overall model size. Postcondition: $\text{path}[0] = R$ and $\text{path}[\text{last}] = n$; for every consecutive pair $(\text{path}[i], \text{path}[i+1])$, $(\text{path}[i], \text{path}[i+1]) \in E$. Error handling: if $n \notin N$, TRACE_ANCESTRY returns error “Unknown node” rather than an empty or partial path.

Complexity: $O(\text{depth}(n))$, since ρ is immutable per node (Section 4.3) and the loop in step 2 performs one constant-time hop per ancestor.

Reference Architecture

The formal model of Sections 3 and 4 is realized architecturally as a four-layer system separating presentation, application logic, visualization, and data management. The front end exposes four coordinated views over a single underlying model: a circle node browser implementing the focus+context rendering of Section 4.2; a node-creation view implementing INSERT_NODE; a node-management view implementing UPDATE_NODE and DELETE_NODE; and a node-information view implementing TRACE_ANCESTRY. Maintaining these as separate, purpose-built views rather than a single dense interface distributes the cognitive load of variability management across dedicated interaction spaces. The back end exposes the four algorithms of Section 4 as service endpoints consumed by the browser client, backed by a persistence layer that stores nodes, edges, dependency relations, and visualization attributes. The client retains only the current dotted-disk projection of the model in memory and requests a re-embedding from the server whenever the underlying structure changes (insertion or deletion), while a focus operation (Section 4.2) can be computed and rendered client-side, since it is a pure coordinate transformation that does not alter N or E . This division keeps focus navigation

responsive even for models with a very large number of nodes, since re-centring never requires a server round-trip.

Illustrative Example

Consider a small feature model with a root node r and two children a and b , positioned in the Poincare disk at $\varphi(r) = 0$, $\varphi(a) = 0.5$, and $\varphi(b) = -0.3$ (real-valued for simplicity). Focusing on node a applies the transformation $T_a(z) = (z - 0.5) / (1 - 0.5z)$. Substituting the three positions gives $T_a(\varphi(r)) = -0.5$, $T_a(\varphi(a)) = 0$ as required by construction, and $T_a(\varphi(b))$

$= (-0.3 - 0.5) / (1 - 0.5 \times -0.3) = -0.8 / 1.15 \approx -0.6957$. Node a moves to the origin and is rendered at maximum visual radius; the root moves outward to -0.5 and node b moves further outward still to approximately -0.6957 , reflecting the fact that b is not adjacent to the newly focused node.

Figure 1 shows the original and transformed layouts side by side. Table 1 reports the pairwise hyperbolic distances computed from Eq. (5) before and after the transformation, rather than merely stating that they are equal.

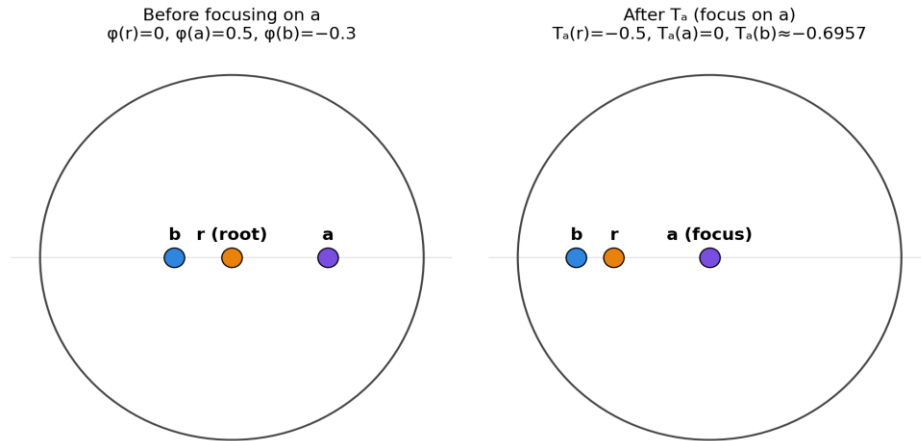


Figure 1: Node Positions in the Poincare Disk before and after Focusing on Node a . Focusing re-Centres a to the Origin; r and b move Outward, with b (Non-adjacent to a) Moving Furthest

Node pair	d before focusing	d after focusing (T_a)
r, a	1.098612	1.098612
r, b	0.619039	0.619039
a, b	1.717651	1.717651

Table 1: Hyperbolic distances (Eq. 5) among the three nodes before and after focusing on a , computed to six decimal places. All three distances are invariant under T_a to machine precision, confirming numerically that re-centring preserves the structural relationship among nodes even though their Euclidean coordinates on screen have changed

These figures are illustrative of the transformation's correctness, not of scalability: a three-node example demonstrates that Eq. (6) is distance-preserving, but it does not by itself establish readability, rendering performance, or navigation efficiency for feature models with hundreds or thousands of nodes. We address this limitation explicitly in Section 7.

Discussion

The formalism presented here isolates the geometric and algorithmic core of a focus+context feature-model visualization from any particular implementation, which is the property that allows Section 6's example to be checked independently of a running system. Two limitations are worth noting. First, the model as specified treats dependency relations (Section 3.2) as a labelled relation available for querying and display, but does not itself define an anomaly-detection or constraint-satisfaction procedure over D and C ; integrating solver-backed analysis (e.g., detecting dead features, false-optional features, or void models) over this same formal structure is a natural extension. Second, the algorithms of Section 4 assume a single active editor; extending `INSERT_NODE`, `UPDATE_NODE`, and `DELETE_NODE` with conflict resolution for concurrent edits

would be required to support real-time collaborative use of the model. Both extensions can be layered onto the seven-tuple of Section 3.1 without altering the core distance metric or focus transformation, which is one of the intended benefits of separating the formal model from its algorithmic operations.

As noted in Sections 1 and 6, this paper does not report an experimental scalability evaluation; the contribution is the formal model, the isometry proof, and the algorithmic specification. We make this limitation explicit rather than imply validated scale claims. Beyond the concurrent-editing and constraint-analysis extensions above, three further limitations bear directly on scalability and are identified here as concrete future work:

- Numerical precision near the disk boundary. As $|\varphi(n)| \rightarrow 1$, the denominator $|1 - \bar{\varphi}(f) \cdot \varphi(n)|$ in Eq. (5)–(6) approaches its lower bound and $\tanh^{-1}$ approaches a singularity; double-precision floating point loses accuracy for deeply nested nodes pushed near the boundary. A production implementation should clamp $|\varphi(n)|$ below a fixed bound (e.g., $1 - \epsilon$) or adopt arbitrary-precision arithmetic for very deep trees; this trade-off has not been characterized empirically here.
- Label overlap. The dotted-circle layout in Section 3.3 does not itself resolve label collisions among nodes crowded near the boundary at large N . Force-directed label placement or greedy occlusion-avoidance layered on top of φ is a plausible mitigation, left unimplemented and unevaluated in this paper.
- Performance at industrial scale. Section 4 gives worst-case time complexity for each algorithm ($O(|N|)$ for `INSERT_NODE`, `FOCUS`, and `DELETE_NODE`; $O(1)$ for

UPDATE_NODE; $O(\text{depth}(n))$ for TRACE_ANCESTRY), but asymptotic complexity is not the same as measured rendering latency, memory consumption, or perceived responsiveness for feature models with thousands of nodes and rich dependency structure. Establishing these figures against baseline tools (e.g., MUSA, DyMMer 2.0, FeatureIDE, variability.dev) requires an implementation-backed benchmark, which we identify as the immediate next step for this line of work rather than claim here.

Future work therefore includes: a full implementation instrumented for the benchmarks above; a controlled usability study of focus+context navigation against conventional tree-based tooling; and validation using industrial-scale feature models (hundreds to thousands of features) to establish whether the theoretical $O(|N|)$ and $O(\text{depth}(n))$ bounds of Section 4 translate into practically acceptable latency.

CONCLUSION

This paper presented a formal focus+context visualization model for feature models in Software Product Line Engineering, defined over a seven-tuple structure, a Poincaré-disk node embedding, and a Möbius-transformation-based focus operator shown to preserve hyperbolic distances under re-centring. Five algorithms, each specified with preconditions, postconditions, error handling, and worst-case complexity, define the operational semantics of the model, and a four-layer browser-based architecture illustrates how the formalism can be realized as a deployable system.

These are the properties we have demonstrated: the seven-tuple's components have explicit domains and codomains (Section 3.1); the focus transformation is provably a hyperbolic isometry (Section 3.4, verified numerically in Section 6); and the five algorithms are fully specified with worst-case complexity bounds (Section 4). We have not established, and do not claim, that the resulting visualization is empirically more scalable, readable, or efficient to navigate than existing tree-based or hyperbolic-tree tooling — that requires the implementation-backed benchmark identified in Section 7. By formalizing the geometry and operations independently of a specific tool, the model is intended to serve as a reusable foundation for scalable variability-model visualization, with empirical validation of that scalability left as the immediate next step.

REFERENCES

Diaz, O., Montalvillo, L., Medeiros, R., Azanza, M., & Fogdal, T. (2022). Visualizing the customization endeavor in

product-based-evolving software product lines: A case of action design research. *Empirical Software Engineering*, 27(3), Article 75. <https://doi.org/10.1007/s10664-021-10101-6>

Furnas, G. W. (1986). Generalized fisheye views. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 16–23). Association for Computing Machinery. <https://doi.org/10.1145/22627.22342>

Heß, T., Ostheimer, L., Betz, T., Karrer, S., Schmidt, T. J., Coquet, P., Semmler, S., & Thüm, T. (2025). variability.dev: Towards an online toolbox for feature modeling. In *Proceedings of the 6th International Workshop on Languages for Modelling Variability (MODEVAR 2024)*. arXiv preprint arXiv:2506.09845.

<https://doi.org/10.48550/arXiv.2506.09845>

Lamping, J., Rao, R., & Piroli, P. (1995). A focus+context technique based on hyperbolic geometry for visualizing large hierarchies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 401–408). ACM/Addison-Wesley.

<https://doi.org/10.1145/223904.223956>

Lindohf, R., Krüger, J., Herzog, E., & Berger, T. (2021). Software product-line evaluation in the large. *Empirical Software Engineering*, 26, Article 26 (pp. 1–41).

<https://doi.org/10.1007/s10664-020-09913-9>

Pohl, K., & Metzger, A. (2018). *Software product lines. In The essence of software engineering* (pp. 185–201). Springer.

https://doi.org/10.1007/978-3-319-73897-0_11

Quinton, C., Vierhauser, M., Rabiser, R., Baresi, L., Grünbacher, P., & Schuhmayer, C. (2020). Evolution in dynamic software product lines. *Journal of Software: Evolution and Process*, 32(6–7), e2293.

<https://doi.org/10.1002/smr.2293>

Rabiser, D., Prähofer, H., Grünbacher, P., Petruzelka, M., Eder, K., Angerer, F., Kromoser, M., & Grimmer, A. (2018). Multi-purpose, multi-level feature modeling of large-scale industrial software systems. *Software and Systems Modeling*, 17(3), 913–938. <https://doi.org/10.1007/s10270-016-0564-7>

