Topology-Independent Mesh Deformation Using Curvenets

A Maya Implementation of Profile-Curve Articulation

By

Osher Shechter

MSc Computer Animation and Visual Effects
at Bournemouth University, UK

August 2026

Abstract

Character rigging commonly depends on deformation weights created for a specific polygon mesh, making it difficult to reproduce the same deformation on a model with different topology. This project presents a Maya implementation of Curvenets - networks of artist-authored curves placed on a model's surface - which provide a deformation structure independent of the underlying polygon connectivity. The curves are embedded into each target mesh by detecting where they cross polygon edges and using those crossings to divide the surface into corresponding regions. The Curvenet can then be bound to a standard joint hierarchy, with editable weights controlling how its nodes respond to skeletal motion. The mesh follows the articulated Curvenet through a deformation method that combines regional rigid motion with smooth propagation across the surface. The complete workflow was implemented as a C++ Maya plug-in supported by Python tools for drawing, binding, editing and transfer. Evaluation on three corresponding hand models demonstrated that a single authored Curvenet, skeleton and set of edited weights could be transferred between meshes with different polygon topology and used to produce corresponding poses without independently skinning each mesh. The resulting workflow demonstrates how a curve-based representation can separate deformation control from mesh connectivity and support consistent articulation across related models.

Contents

Page

1. Introduction

1.1 Context and motivation

Character deformation sits between modelling and animation. A modeller supplies a surface, a rigger constructs controls and deformation logic, and an animator expects that the surface will maintain a convincing form under motion. In a conventional Maya pipeline, smooth skinning associates every mesh vertex with one or more joints. Corrective blend shapes, pose-space deformations and hand-authored weight edits are then layered on top to address volume loss, joint collapse or characteristic shape changes. These techniques are well established, but their data is usually indexed by the vertices, edges or faces of one mesh. A change in topology can invalidate the correspondence on which the rig depends.

This is not only a technical inconvenience. Retopology is a normal production operation: a sculpt may be converted into an animation mesh, a hero asset may require several levels of detail, and the same character design may be represented by meshes made for different rendering or simulation requirements. If the deformation description is inseparable from vertex indices, much of the rigging work must be transferred heuristically or repeated. The problem becomes more visible when two meshes have the same shape but different edge flow. They appear equivalent to an artist, yet they are not equivalent to a vertex-indexed deformer.

Pixar's profile-curve method reframes this relationship (de Goes, Sheffer, & Panozzo, 2015). Instead of treating the polygon mesh as the primary rig representation, it places a network of curves on the character surface, referred to as a Curvenet. The Curvenet describes meaningful sections and deformation profiles at a level above individual polygons. Each profile curve is discretised around the same structure, enabling spatially corresponding points to define a shared logical structure rather than one tied to a particular mesh topology.

The practical aim of this project was therefore not simply to produce another curve deformer. It was to investigate the representation that makes curve-driven deformation transferable. The distinction is important. A nearest-curve falloff can move a mesh, but it does not construct shared nodes, embedded boundaries or corresponding surface regions. It remains dependent on geometric proximity and can behave differently when sampling density changes. A Curvenet system must preserve the authored graph, find how that graph crosses each mesh, and propagate motion without equating the logical node with one particular mesh vertex.

1.2 Research question and objectives

The research question is:

How can the core concepts of Pixar's Curvenet-based deformation system be implemented within Maya, and to what extent can topology-independent deformation be achieved?

The project addresses this question through five objectives:

1. create an artist-facing workflow for drawing a connected network of curves on an arbitrary Maya mesh.
2. build an internal Curvenet representation that is independent of Maya object names and target mesh vertex identifiers.
3. embed the curves by detecting crossings, splitting mesh topology and constructing ordered CutPaths in a half-edge mesh.
4. drive a target surface from a joint-bound Curvenet and transfer the logical rig to meshes with different topology.
5. evaluate correctness, deformation behaviour, usability and limitations on increasingly complex examples.

The intended outcome is a working research prototype rather than a reproduction of Pixar's complete production system. The paper includes a cut-cell discretisation, deformation-gradient constraints and subdivision-surface reconstruction. This implementation concentrates on the logical graph, cut-aware embedding, surface partitioning, transferable joint controls and harmonic positional deformation. That scope supports a clear evaluation: the project can test whether the same graph and region structure survive a topology change, while honestly identifying the quality gap between a positional prototype and a production deformation formulation.

1.3 Contributions

The work makes four connected contributions. First, it provides a surface-aware authoring tool that records explicit logical endpoint relationships. Second, it implements a half-edge cut pipeline that converts sampled profiles into mesh crossings, CutVertices, CutChains and physical boundary edges. Third, it separates logical Curvenet identity from physical mesh identity, allowing the same graph, skeleton and node weights to be attached to another mesh. Fourth, it supplies a Maya workflow around the research code: installation, a shelf launcher, drawing and connection stages, a sparse node-weight editor, transfer controls and visual region diagnostics.

The engineering contribution is as important as the final image. Many failures occurred where two individually reasonable operations met: a projected endpoint and a cut vertex were visually coincident but not topologically identical, a tiny numerical region was removed but the cleanup crossed a meaningful boundary, a curve preview looked smooth while its stored CutPath followed-

a different set of points. Resolving these cases required a consistent definition of authority. Authored metadata defines the logical graph, the cut mesh defines physical traversal, and validation checks connect the two.

2. Background and Related Work

2.1 Mesh-bound skeletal deformation

Linear blend skinning remains the standard baseline for interactive character rigs. A rest vertex x_i is transformed by a weighted combination of joint matrices:

$$x_i' = \sum_{j=1}^{m} w_{ij} T_j x_i, \qquad \sum_{j=1}^{m} w_{ij} = 1$$

The method is efficient and supported directly by Maya, but its weights w_{ij} are stored per mesh component. Transfer between non-corresponding topologies therefore requires a new mapping or new painting. Linear blending also creates familiar artefacts such as volume loss under twisting. Dual-quaternion skinning improves rotational blending and reduces the collapsing-joint artefact, but it does not remove the topology-dependent storage of the weights (Kavan, Collins, Žára, & O'Sullivan, 2007).

Corrective systems address deformation quality from another direction. Pose-space deformation associates sculpted corrections with positions in pose space (Lewis, Cordner, & Fong, 2000). It can produce high-quality, art-directable results, but corrections are again typically authored for a particular surface. This establishes the motivation for a sparse intermediate representation: if the artist's intent is stored on a transferable structure and the target mesh only receives a derived solve, fewer authored decisions are tied to one tessellation.

2.2 Space and curve-based deformation

Free-form deformation embeds an object in a lattice and deforms it by moving lattice control points (Sederberg & Parry, 1986). Its strength is independence from the object's local connectivity: points are evaluated in the surrounding parameterisation. However, a rectangular lattice is not naturally aligned to an articulated character surface, especially around branching shapes such as fingers.

Curve-based techniques offer a more compact and shape-aware interface. Wires deforms geometry using one-dimensional curve handles and radial influence functions (Singh & Fiume, 1998). Such approaches demonstrate that curves can be expressive rig controls, but a collection of independent curves is not automatically a surface partition or a transferable graph. The present project initially explored a comparable distance-based proof of concept. It confirmed that a C++ MPxDeformerNode could read a Maya curve, store a neutral state and move mesh points according to curve displacement. It was rejected as the final architecture because nearest-CV and

radius falloffs changed with curve sampling, did not create shared network nodes, and did not provide a cut-aware distinction between the two sides of a profile.

Diffusion curves provide a useful analogy for why a curve boundary matters. In image-space diffusion, a curve can carry different values on its two sides and the domain interpolates away from the discontinuity (Orzan et al., 2008). Pixar's Curvenet paper extends a related idea to character surfaces: profile curves are not merely attractors, they define constraints and discontinuities in a surface discretisation (de Goes et al., 2022). This distinction motivated the cut-mesh stage of the project.

2.3 Profile curves and Curvenets

De Goes et al. (2022) describe a Curvenet as a connected arrangement of cubic Bézier profile curves placed on a character. The curves encode a sparse rigging structure, including two-sided deformation behaviour, while remaining independent of the final polygon layout. The paper constructs a cut-aware surface discretisation and solves for deformation gradients before reconstructing a subdivision surface. Its examples show that a single curve setup can drive characters through retopology and level-of-detail changes.

Three concepts were treated as essential in this implementation. The first is graph identity: endpoints that an artist connects must represent the same Curvenet node. The second is physical embedding: a target mesh needs explicit cut paths that follow the profiles across its own faces. The third is target-independent control: deformation parameters belong to the Curvenet and are transferred through logical identifiers, not through source mesh vertex numbers.

Other elements were simplified. Maya NURBS curves are used as the artist-facing input, but profiles are sampled into polylines for robust intersection and cutting. The final deformer uses positional constraints and a harmonic graph solve instead of Pixar's full deformation-gradient formulation. The output remains the input polygon mesh rather than a newly reconstructed subdivision surface. Consequently, this project does not claim to reproduce Pixar’s complete production system. It investigates how the central Curvenet concept can be adapted into a working Maya prototype for curve authoring, mesh embedding, joint-driven articulation and transfer between different mesh topologies.

2.4 Half-edge meshes and cut topology

A conventional indexed polygon mesh stores vertices and faces efficiently, but many local traversal operations require repeated searching. A half-edge data structure represents each directed

side of an edge with references to its start and end vertices, face, opposite half-edge, and neighbouring half-edges. Directed-edge and half-edge representations are widely used for local mesh processing because adjacency is explicit (Campagna, Kobbelt, & Seidel, 1998; Botsch, Kobbelt, Pauly, Alliez, & Lévy, 2010).

In the project implementation, a HalfEdge stores startVertex, endVertex, next, twin and face indices. A face stores one incident half-edge from which its complete directed boundary can be traversed by repeatedly following next. Interior polygon edges are represented by two half-edges with reversed endpoints and reciprocal twin links. A boundary half-edge has no twin. A vertex additionally stores one outgoing half-edge as an entry point for adjacency queries. The implementation derives the previous member of a face loop when required rather than storing a separate previous field.

This project uses half-edges because cutting is fundamentally local and topological. Given a crossing on an edge, the system must find both incident faces, insert a vertex into both directed representations of the edge, reconnect the affected face loops and later determine which edges form barriers. When a profile passes through a polygon, two boundary vertices are joined by a new pair of directed half-edges, producing two closed face loops. The same representation supports region flood fill: neighbouring faces are reached through twin half-edges unless the shared edge belongs to an embedded Curvenet CutChain, the ordered physical chain associated with one logical profile.

The implementation was developed by translating the standard directed-edge records and traversal operations described by Campagna et al. (1998) and Botsch et al. (2010) into index-based C++ containers suitable for Maya data. Curvenet-specific operations were then added for grouped edge splitting, interior face division, ordered crossing storage and protected-boundary queries. This distinction is important: the references provide the connectivity model, while the crossing records, CutChain data and integration with Maya are adaptations made for this project rather than features of a generic half-edge implementation.

The choice has a cost. Splitting an edge or face requires updating several linked records without leaving stale identifiers. Near-coincident crossings can create duplicate or zero-area topology. Automated tests therefore cover local invariants such as valid face loops, opposite-half-edge consistency, ordered CutChains and region barriers, rather than relying only on a successful Maya viewport result.

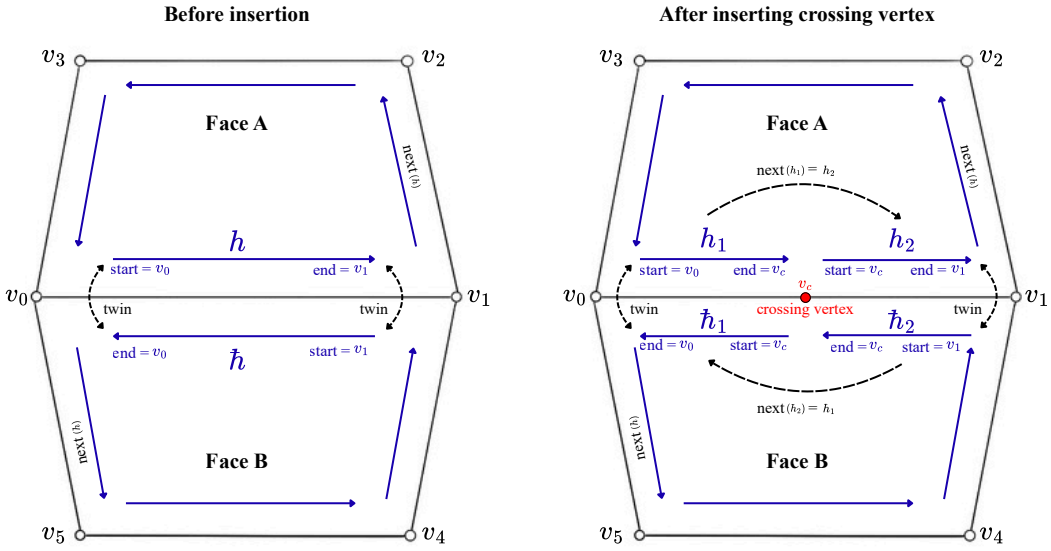


Figure 1. Half-edge structure before and after a profile crossing

2.5 Harmonic propagation and rigid motion

Laplacian surface methods express a vertex in relation to its neighbours. If constrained vertices are moved and unconstrained vertices minimise a discrete membrane energy, the resulting displacement changes smoothly over the graph (Sorkine et al., 2004). Harmonic coordinates use a related principle to propagate control weights through a volume or surface and have been applied to character articulation (Joshi, Meyer, DeRose, Green, & Sanocki, 2007). The present implementation uses a simpler graph-harmonic positional solve over the cut mesh.

For displacement field $\mathbf{u}$, the uniform graph energy is:

$$E(\mathbf{u}) = \frac{1}{2} \sum_{(i,j)\in E} w_{ij} \|\mathbf{u}_i - \mathbf{u}_j\|^2, \quad w_{ij} = 1 \text{ in the current prototype}$$

Embedded CutChain vertices receive prescribed displacements from the current profile curves. The remaining vertices are iteratively relaxed toward the average of their neighbours. A naive displacement solve misinterprets a global rigid rotation as non-rigid bending. To avoid this, the implementation first estimates a best-fit rigid transform between neutral and current curve samples using Horn's quaternion form of absolute orientation, closely related to Kabsch's least-squares rotation solution (Horn, 1987; Kabsch, 1976). Residual curve motion is solved harmonically after the rigid component has been removed. This improves pose reset and reduces the volume collapse seen in early tests, although it is still not an as-rigid-as-possible or deformation-gradient reconstruction.

The implementation combines ideas from several published methods. Sorkine et al. (2004) and Joshi et al. (2007) motivated the use of a discrete Laplacian or harmonic field to propagate motion

from constrained points to the surrounding surface. The connectivity required for this calculation is obtained from the graph already represented by the half-edge mesh. Horn's closed-form alignment method (1987) is used to estimate the common rigid rotation and translation before the remaining deformation is propagated. These references provide the mathematical basis of the method, while decisions such as the number of relaxation iterations, the placement of constraints, the organisation of cached data and integration with Maya's evaluation system were developed specifically for this implementation.

3. Requirements and System Design

3.1 Requirements derived from the research question

The research question leads to requirements at three levels. At the representation level, a Curvenet must be a graph of profiles and shared nodes whose identifiers remain stable across meshes. At the geometric level, each target must produce a valid embedded path and surface partition. At the workflow level, an artist must be able to draw, bind, inspect, refine and transfer the rig without manually editing C++ data.

The functional requirements were therefore:

- accept multiple connected profile curves on a selected Maya mesh.
- preserve explicit endpoint connections and stable logical curve identifiers.
- sample curves, detect edge crossings and order them along each profile.
- split crossed mesh edges and faces into a valid cut mesh.
- form shared logical nodes even when physical embedded vertices differ.
- identify Curvenet edges and surface regions separated by those edges.
- bind Curvenet nodes and curves to a selected joint set.
- deform the mesh from the moving Curvenet.
- transfer the Curvenet, skeleton and edited weights to another mesh.
- expose concise validation and diagnostics in Maya.

Non-functional requirements included avoiding hard-coded mesh names, preserving interactive authoring, preventing stale generated scene objects, providing automated C++ tests, and maintaining a workflow that could be installed from a Maya shelf. Performance was treated as a goal rather than a guaranteed production target because complete embedding can modify thousands of polygons and the initial implementation prioritised correctness and traceability.

3.2 Layered architecture

The system is divided into four layers. The Python authoring layer creates Maya curves, node spheres and endpoint metadata. The plugin input layer samples those curves and converts the selected Maya mesh into an internal half-edge mesh. The embedding layer detects crossings, splits topology, builds CutChains, Curvenet nodes, edges and faces. The deformation layer evaluates joint-driven profile motion and propagates it across the mesh.

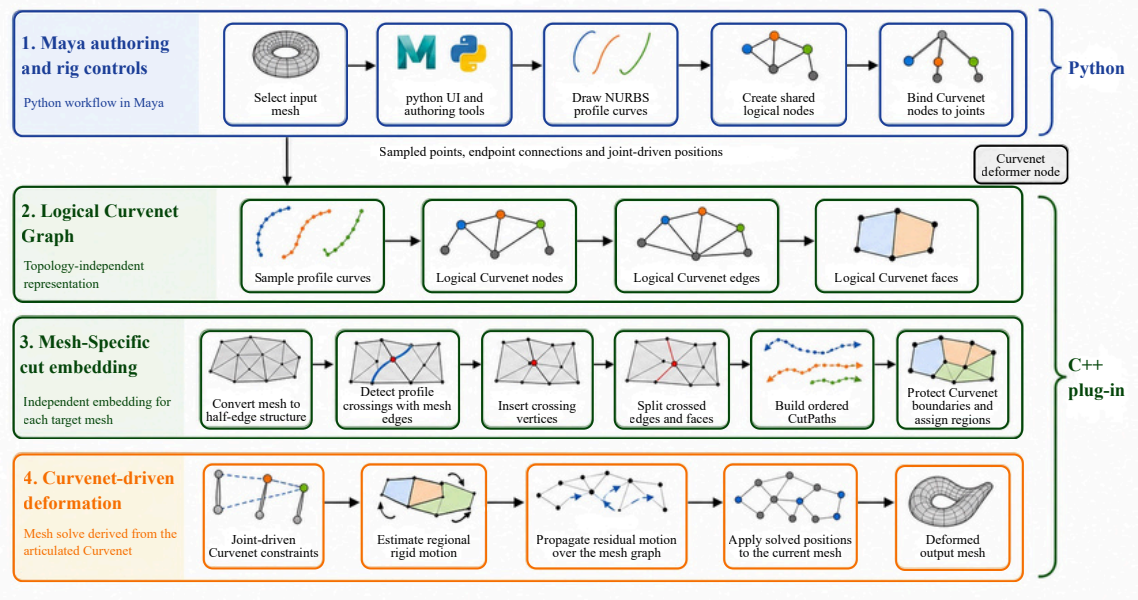


Figure 2. Curvenet system architecture

This separation prevents Maya scene objects from becoming the only representation of the Curvenet data. During authoring, spherical controls are displayed in Maya's 3D viewport to represent logical Curvenet nodes. These controls allow the artist to view and select the nodes, but they are not vertices of the target mesh. Similarly, the projected NURBS curves carry the artist-authored profiles into the plug-in, but they are resampled before intersection detection and cutting. The internal half-edge mesh is a temporary representation that can be reconstructed from the target mesh. Correspondence between different meshes is instead maintained through stable logical identifiers and explicit metadata describing which curve endpoints share the same Curvenet node.

3.3 Core data structures

ProfileCurveInput stores a logical curve identifier, sampled points and endpoint connection information. A CutCrossing records the curve, segment, mesh half-edge, face and three-dimensional intersection position. Once an edge is split, a CutVertex records the new mesh vertex and its position along the curve. A CutChain stores the ordered embedded points and half-edges produced for one profile.

The final CurvenetCutResult aggregates the cut mesh, embedded vertex identifiers, embedded half-edge identifiers, Curvenet faces, shared nodes and a curve-ID-to-CutChain map. The result retains enough information for validation and later deformation without relying on the order in which Maya happens to return scene connections.

Logical nodes and physical nodes are deliberately distinct. A logical node is identified by authored endpoint metadata. A physical node is the mesh vertex or sampled cut location produced on one target. One logical node can therefore be represented by different physical vertices on different

meshes. In difficult cases, two endpoints on the same mesh can remain physically separate after cutting yet still be unified in the logical graph. Treating those as contradictory would reintroduce mesh dependence. Instead, the data model records both facts.

3.4 Coverage modes and topological expectations

Two coverage interpretations are supported. An authored-face Curvenet occupies a local portion of a surface and only closed cycles inside the network are treated as regions. A full-surface Curvenet wraps and partitions the complete closed surface. The UI can resolve the mode automatically from metadata, while an advanced override remains available for debugging.

For a connected Curvenet graph embedded over a complete closed surface, the expected number of surface regions follows Euler's relation. If V is the number of logical nodes and E is the number of logical edges, the expected number of regions F is:

$$F = 2 - V + E$$

For example, the full-surface Curvenet created for the cylinder test contained 14 logical nodes and 28 logical edges. Euler's relation therefore predicted 16 surface regions. This expected value provided a useful topology check because the colour preview could still appear plausible when two regions had been incorrectly joined. For a local authored-face Curvenet, only the bounded cycles created within the network are counted. When the logical graph is connected, their expected number is given by the cycle rank $E - V + 1$.

4. Curvenet Authoring and Logical Representation

4.1 Surface-aware drawing

The artist begins by selecting a polygon mesh and activating the Curvenet drawing context. Clicks are ray-cast into the mesh so node spheres are placed on the actual surface rather than at an arbitrary screen depth. Two clicks create one profile segment. Clicking near an existing node reuses that node, which permits branches and loops without requiring exact component selection.

Early versions exposed several problems that became more visible on the hand than on the cylinder. A screen-space tolerance could snap to the wrong nearby finger. A world-space tolerance that worked for a large cylinder produced oversized handles on a small hand. Feature snapping to every mesh vertex made authoring topology-dependent, contradicting the purpose of the tool. The final behaviour scales node size and reuse tolerance from the selected mesh bounds, performs surface hit testing, and limits feature snapping to an optional mode for genuine hard features such as a rim.

The authoring curves are shown as thick display curves so the network can be understood while drawing. This display is intentionally separate from the eventual cut path. It solves a usability problem - thin projected curves can disappear behind the mesh - without claiming that a shortest mesh-edge route is the authored profile. Several attempted display strategies were rejected. Drawing the shortest path through mesh vertices made profiles inherit the existing topology. Camera-projected smoothing produced view-dependent loops. Automatically generated Bézier previews could diverge from the stored sampled curve and left stale expression errors in Maya. The adopted display uses the sampled authored profile and is treated as a visual aid, not a second geometric representation.

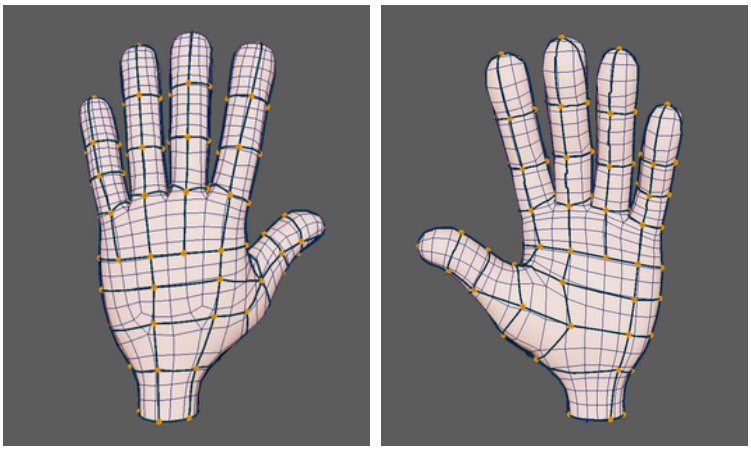


Figure 3. Front and back view of the complete hand Curvenet during authoring.

4.2 Sampling and projection

Maya NURBS curves are convenient for artists but inconvenient as direct cut primitives. The implementation therefore converts each curve into an ordered polyline in two stages. First, MFnNurbsCurve is evaluated at 200 uniformly spaced values over its knot domain. These dense parameter samples form a lookup polyline that approximates the continuous curve. They are not used directly as the final cutting resolution because equal parameter intervals do not generally correspond to equal distances along a NURBS curve.

Second, the dense polyline is resampled at approximately uniform arc-length intervals. If the control-polygon length is L_c, the mean target-mesh edge length is h_m, and the selected density multiplier is $\rho = 5$, the final number of samples is:

$$n = \max\left(2, \left\lceil \rho \frac{L_c}{h_m} \right\rceil\right)$$

Cumulative lengths are computed along the dense polyline. Final samples are placed at target distances $d_k = kL/(n-1)$, where L is the dense polyline length, by interpolating within the segment containing each target distance. This makes the cutting resolution responsive to both profile scale and target-mesh scale while avoiding clusters caused by NURBS parameterisation. Once the neutral state has been captured, the sample count is retained during posing so a moving profile continues to correspond to the same ordered constraint sequence.

The Curvenet method uses sampled profile curves to construct a cut-aware surface representation (de Goes et al., 2022). This project follows that general principle, while the specific two-stage procedure was developed for the Maya implementation. A dense lookup polyline first approximates the NURBS curve, after which mesh-scale arc-length resampling produces the points used for intersection detection and cutting.

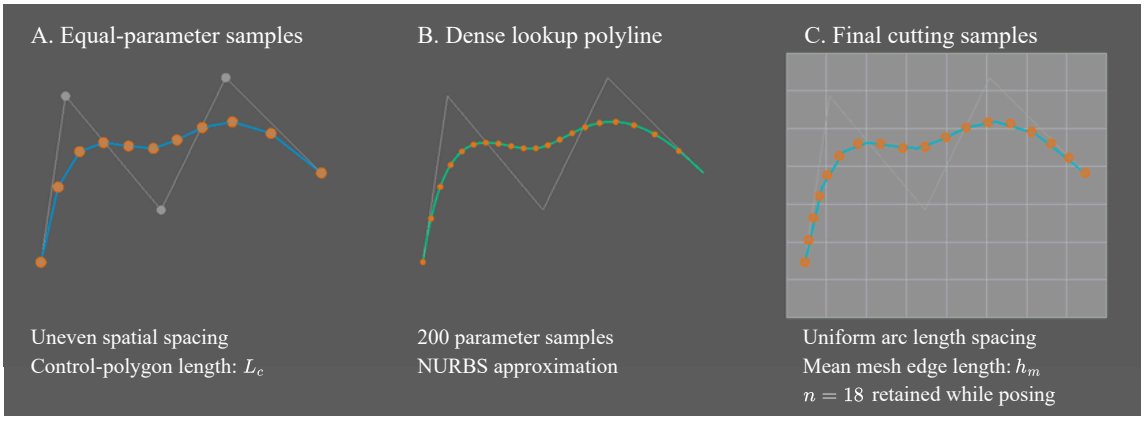


Figure 4. Profile sampling stages - from an artist-facing NURBS profile to cutting samples.
Equal parameter intervals on a NURBS curve do not necessarily produce equal spatial distances. A dense lookup polyline is therefore constructed first and subsequently resampled at approximately uniform arc-length intervals according to the scale of the target mesh. The resulting neutral sample count is retained during deformation.

The sampled points are projected onto the target mesh. Projection is performed separately for every target, because two meshes that occupy the same conceptual shape can have different polygon planes. The target curve keeps the source logical profile ID and endpoint metadata. Its control points are target-specific physical data.

A useful distinction emerged during development between authored curves and projected curves. The authored group preserves editable input and may pass slightly inside a curved object between clicks. The projected group is the plugin input and lies on the target surface. Keeping both supports non-destructive editing and transfer, but only the projected curves participate in embedding. The UI hides groups that are not needed for the current task to reduce scene clutter.

4.3 Explicit endpoint connectivity

At first, shared Curvenet nodes were detected only when two CutChains ended at the same mesh vertex. This worked on carefully procedural cylinder profiles but failed for drawn curves. Two endpoints that an artist snapped together could project through different samples and produce different cut vertices. Conversely, two unrelated profiles might pass very close in space without being connected.

The solution was to make connectivity explicit at authoring time. Every endpoint stores the logical node it references. When projected curves are connected to the plugin, this metadata is read alongside the sampled positions. The cut operation is completed first, because only then are the target-specific embedded vertices known. Explicit endpoint connections are then applied to the result, unifying endpoints into shared logical nodes even if the cutting stage generated different physical vertex IDs.

This order is essential. Applying logical merges before cutting would force the cutter to pretend that two physical locations are one mesh vertex, potentially corrupting face loops. Applying them afterwards preserves a valid physical embedding and a correct logical graph. Generated Curvenet controls are positioned from the authored sampled endpoints rather than the arbitrary representative cut vertex, so the visible controls align with the spheres the artist placed.

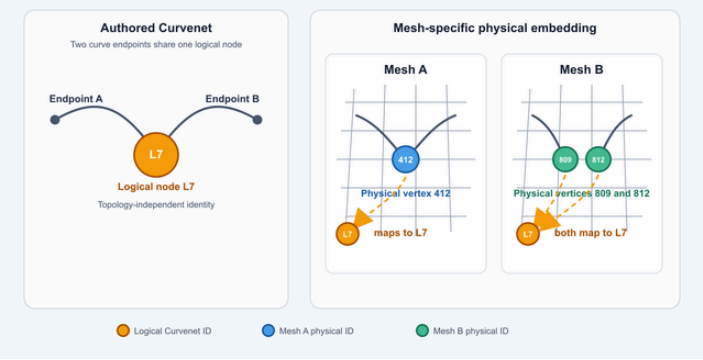


Figure 5. Logical and physical Curvenet node identity. Two authored endpoints share the topology-independent logical node L7. On Mesh A, both endpoints coincide at physical vertex 412. On Mesh B, the same logical connection is represented by physical vertices 809 and 812. Although the embedded vertex identifiers differ, all are mapped back to L7.

4.4 Graph construction and validation

After connections are resolved, each profile becomes a Curvenet edge between two logical nodes. Duplicate edge checks prevent repeated connections from changing the graph. Adjacency lists support cycle analysis, face-boundary construction and UI queries. The graph is validated independently from the cut mesh: all referenced logical nodes must exist, all profile IDs must be unique, and endpoint records must resolve to a valid node.

This independent validation caught a class of failures that visual inspection missed. For example, the cutter could report success while returning zero shared nodes because input connection metadata was absent. Conversely, the correct node count could be present while one CutChain lacked physical half-edges. The Maya output therefore reports both shared and physical node counts where relevant, followed by edge, region and mapped-face totals.

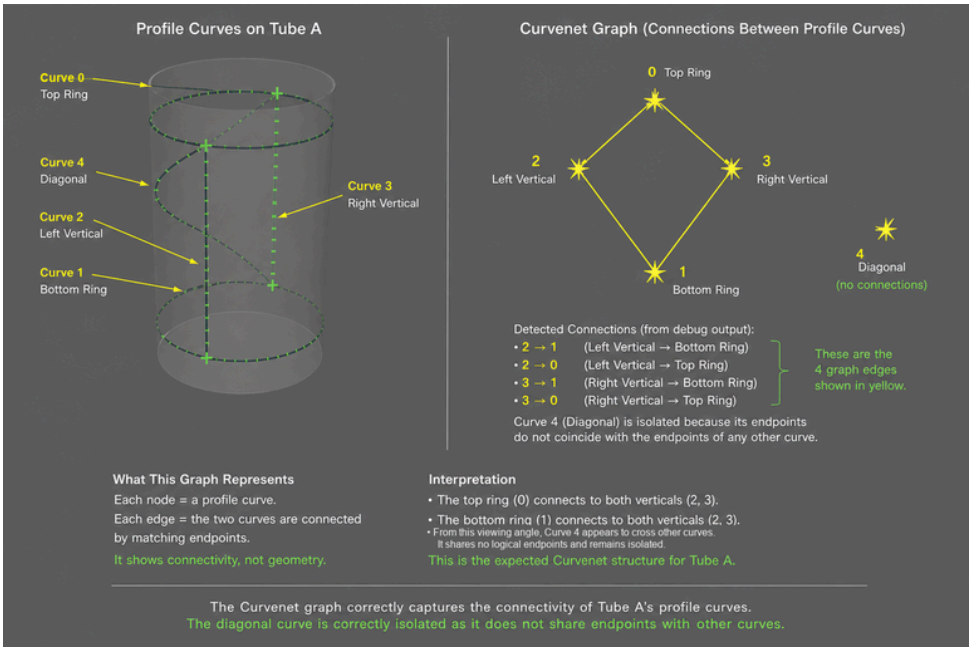


Figure 6. Logical graph used for an early Tube A validation.

5. Embedding and Cut-Mesh Construction

5.1 Converting the Maya mesh

The selected Maya mesh is converted into a compact half-edge representation. Each polygon contributes a directed loop. Opposite directions are paired through their vertex endpoints, and boundary edges retain a missing opposite. The conversion preserves original vertex and face identifiers where possible so diagnostics can refer back to Maya components.

Initial tests verified that every face loop closes, opposite half-edges reverse the same undirected edge, and neighbouring faces are found correctly. These tests are foundational: later cutting code assumes that traversing *next* around a face and *opposite* across an edge is reliable. A non-manifold or inconsistent input is therefore outside the validated scope and should be rejected or preprocessed in future versions.

The conversion can be expressed as four operations: create one vertex record for every Maya vertex, create a cyclic half-edge loop for every polygon, index each directed endpoint pair, then assign twins by looking up the reversed pair. This separates Maya API access from later topology editing. Once conversion has finished, crossing and splitting code operates on the internal integer-indexed records rather than repeatedly querying Maya's polygon iterators.

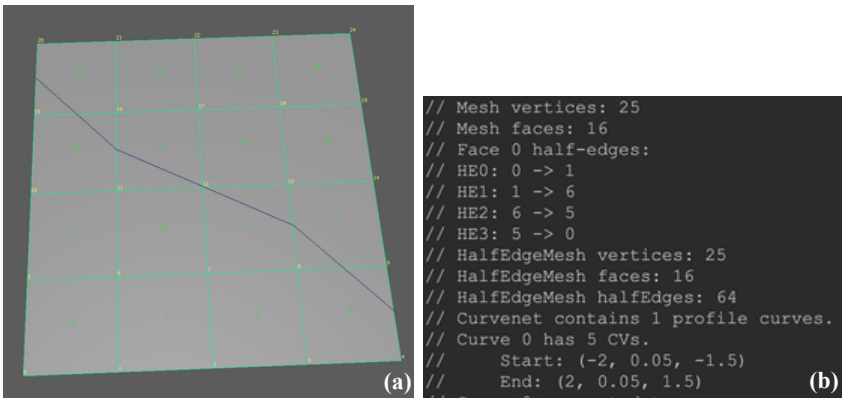


Figure 7. Validation of the half-edge mesh conversion.
(a) Numbered mesh faces and vertices used to inspect the converted topology.
(b) Console output confirming that the Maya mesh was converted into a half-edge representation containing 25 vertices, 16 faces and 64 half-edges, together with the imported profile-curve data.

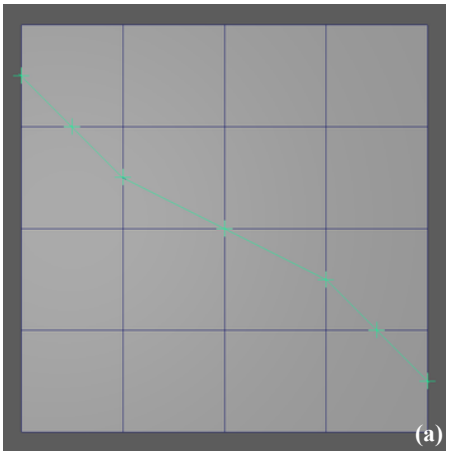
5.2 Curve-edge crossing detection

Each sampled curve segment is tested against candidate mesh edges in three dimensions. The closest points between segments are computed, and an intersection is accepted when the separation is below a tolerance and both parameters lie within their finite segments. A crossing stores curve segment s, local parameter t, face, half-edge and position. Its continuous order along the sampled profile is:

$$u = s + t$$

Sorting by u provides a stable curve order even when crossings were discovered in face order. Near-duplicate crossings on a shared mesh vertex or near the end of adjacent segments are consolidated. The tolerance is scaled relative to local mesh and profile length rather than using one fixed world-space value.

On a planar quad, a profile that enters through one edge and leaves through the opposite edge should produce one interval and split the face into two polygons. This simple case became a diagnostic model for hand failures. Extra triangles indicated that crossings were being duplicated, snapped to an unintended adjacent edge, or connected by a physical path that did not match the sampled profile.




```
// Crossings found: 7
// Crossing: curve 0, curve segment 0, face 12, half-edge 51, position (-2, 0, -1.5)
// Crossing: curve 0, curve segment 3, face 8, half-edge 34, position (-1.5, 0, -1)
// Crossing: curve 0, curve segment 6, face 8, half-edge 33, position (-1, 0, -0.502294)
// Crossing: curve 0, curve segment 12, face 5, half-edge 21, position (0, 0, 0)
// Crossing: curve 0, curve segment 18, face 6, half-edge 25, position (1, 0, 0.502294)
// Crossing: curve 0, curve segment 21, face 3, half-edge 14, position (1.5, 0, 1)
// Crossing: curve 0, curve segment 24, face 3, half-edge 13, position (2, 0, 1.5)
```

(b)

Figure 8. Validation of curve-crossing detection.
(a) Debug visualisation of the seven intersections detected between the profile curve and mesh edges.
(b) Console records identifying the curve segment, crossed face, corresponding half-edge and three-dimensional position of each intersection.

5.3 Edge splitting and CutVertices

Accepted crossings are grouped by undirected mesh edge. Multiple crossings on one edge are sorted by edge parameter before modification. The edge is then split in order, producing new vertices and half-edge segments on both incident faces. Processing the group as a whole avoids invalidating later crossing locations when the first split changes the topology.

A CutVertex records both the mesh vertex ID and the curve coordinates that created it. Reusing an existing endpoint is allowed when the crossing is within a strict local tolerance, but indiscriminate nearest-vertex snapping is avoided. That earlier approach made the hand network drift onto the target edge flow and could collapse distinct crossings.

5.4 Face intervals and CutPaths

Once edge crossings are materialised as vertices, consecutive crossings along a profile are paired into face intervals. A valid interval must have a face through which the curve passes and endpoints on the boundary of that face. The face is split by inserting two directed half-edges between the interval endpoints, producing two closed loops with consistent face ownership.

The ordered sequence of CutVertices and newly inserted half-edges forms a CutChain. The chain is the physical representation of one logical profile edge. It is used in three later stages: as a protected barrier during surface flood fill, as a correspondence between current profile samples and constraint vertices, and as validation evidence that a curve has actually been embedded.

Curves sometimes begin or end inside a polygon rather than on an existing mesh edge. Endpoint connections are therefore processed after ordinary cutting. A connected endpoint may refer to another curve endpoint or to a parameter on a target profile. The system resolves the target chain and inserts or associates a physical node where possible. Failures report the source curve, target curve, target parameter and chain endpoints rather than emitting per-segment debug spam.

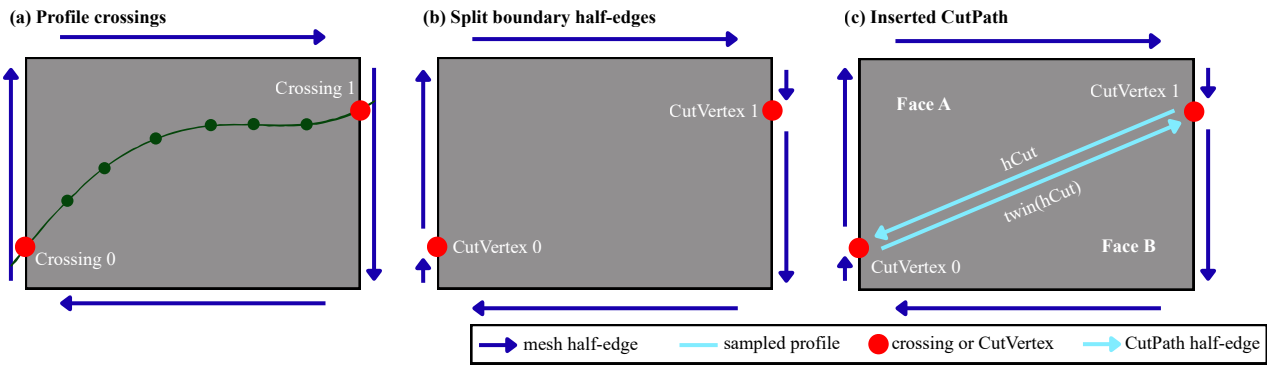


Figure 9. Construction of a CutPath through a crossed mesh face.
(a) A sampled profile segment enters and exits the face through two boundary half-edges.
(b) The crossed boundary half-edges are split by inserting two CutVertices.
(c) A reciprocal CutPath half-edge pair connects the CutVertices and separates the original polygon into two face loops. Arrows indicate the direction of each half-edge.

5.5 Multiple profiles and intersection order

Cutting one profile changes the mesh on which later profiles operate. A fresh intersection computed against the original mesh is no longer sufficient after earlier splits. The implementation rebuilds or updates path information against the current half-edge state, while retaining logical curve coordinates. Duplicate and near-coincident intersections are checked before a new split is accepted.

The most difficult cases occur where several profiles share an endpoint, cross close to an existing vertex, or run almost along an existing mesh edge. These are common on a hand because the network follows joints and finger boundaries while the mesh already contains anatomical edge loops. Robustness depends on preserving order: crossings must remain monotonic along the profile, splits on one mesh edge must remain monotonic along that edge, and the final half-edge chain must connect the expected endpoints.

An embedding validator compares each CutChain with these invariants. It reports mismatched half-edge counts, invalid half-edges and endpoint mismatches. During development this turned silent corruption into actionable failure messages. It also prevented a misleading state in which Maya displayed Curvenet cutting: SUCCESS even though the later graph contained no usable physical edges.

5.6 Surface regions and protected barriers

After all profiles have been embedded, surface regions are extracted by flood filling faces across ordinary mesh edges. An edge in a CutChain is a barrier: traversal must not cross it. Each connected component becomes a CurvenetFace containing its mesh face IDs and an ordered logical boundary where available.

Numerical slivers complicate this stage. When a transferred CutPath is almost coincident with an existing target edge, the two can enclose a tiny one- or two-polygon region. Early cleanup merged very small components with a neighbour based on area alone. This repaired counts in some examples but could merge through the authored boundary, causing colour to leak between two intended Curvenet faces. The corrected rule protects embedded Curvenet edges: cleanup may consolidate numerical fragments only through non-barrier adjacency.

Graph-colouring is used for the preview. The colour itself has no deformation meaning, it is assigned so adjacent physical regions are visually distinct. A larger palette reduces accidental repetition, but topological counts remain the authoritative validation. The preview is generated as a separate mesh because per-face materials on the live deformed input caused expensive Maya updates and made it harder to distinguish the neutral source from diagnostic output.

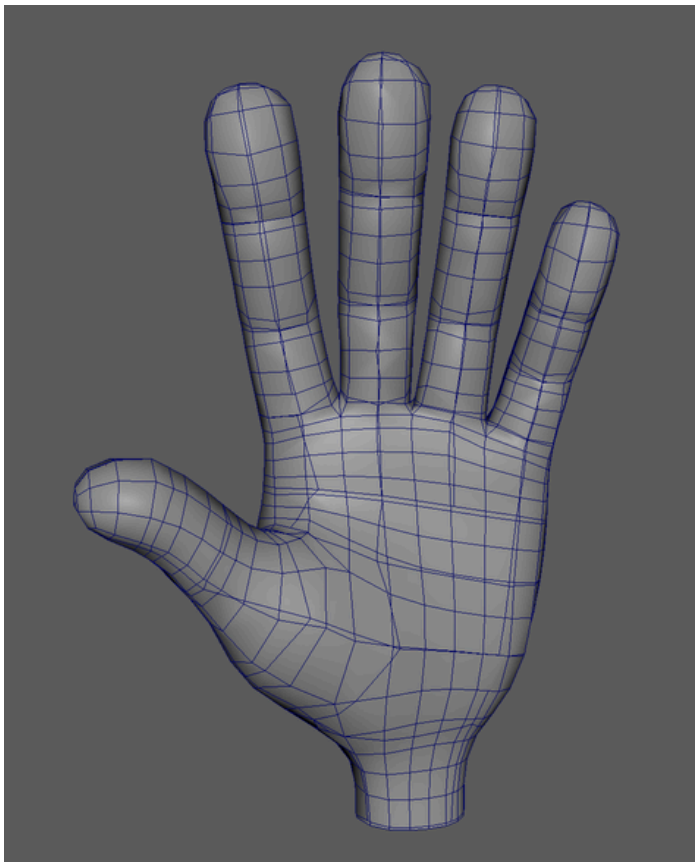


Figure 10. Cut-aware mesh generated from profile curves.

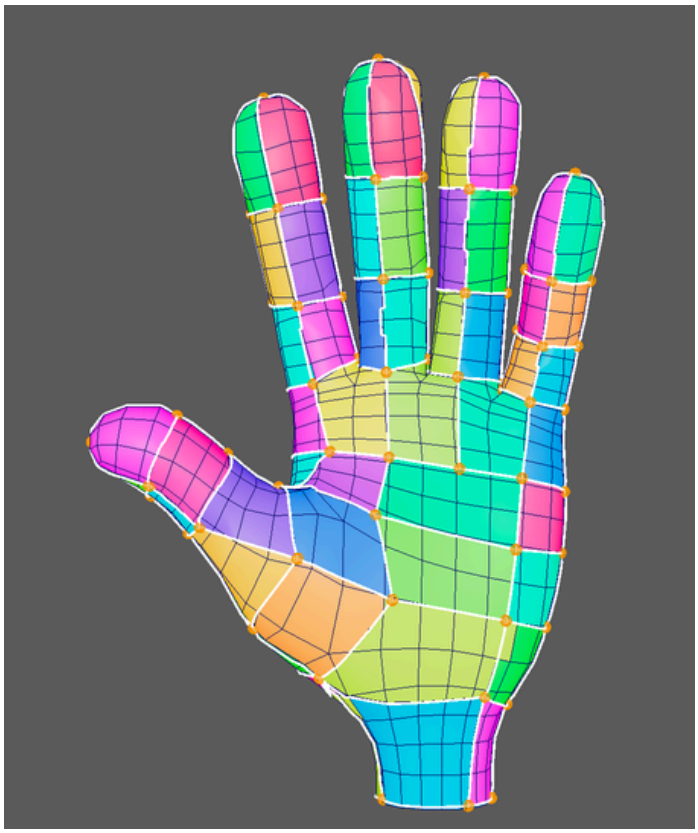


Figure 11. Region preview in which embedded Curvenet edges act as flood-fill barriers.

6. Cross-Topology Curvenet Transfer

6.1 What is transferred

Transfer does not copy source mesh components. It copies the authored graph and recreates physical data for the target. The transferable package contains logical profile IDs, logical endpoint IDs, source curve samples in a source-local frame, Curvenet node weights, the selected joint hierarchy and optional region correspondence information. The target receives newly projected curves, newly generated CutPaths and newly mapped surface regions.

This design is the central answer to the research question. If source vertex 417 were copied to target vertex 417, the result would only work for matching topology. Instead, logical edge 23 remains edge 23, while its target CutChain may contain a completely different number of vertices. Logical node 7 remains node 7, while its physical position is found from the target projection.

6.2 Mesh-local transfer frame

Meshes may be translated, rotated or scaled differently in a Maya scene. Source curves and skeletons are therefore converted through mesh-local coordinates. A source world point $\mathbf{p}_s$ is mapped into source object space and then into target world space:

$$\mathbf{p}_t = \mathbf{M}_t \, \mathbf{M}_s^{-1} \, \mathbf{p}_s$$

where $\mathbf{M}_s$ and $\mathbf{M}_t$ are the source and target world matrices. This corrected an early Tube B issue in which the transferred Curvenet group had the target transform but its preview geometry remained around Tube A. Pivots can remain at a source-looking location without affecting the evaluated world positions, but the generated preview now inherits the correct target frame to avoid confusing interaction.

6.3 Reprojection and endpoint metadata

Transformed source samples provide an initial target-space curve. Each point is then projected to the target surface. Source endpoint relationships are copied by logical ID, not rediscovered by proximity. This is particularly important between fingers: points on two adjacent fingers may be closer in Euclidean distance than points on opposite sides of the intended loop.

The projected target curves can have a different number of usable surface samples from the source. Weight transfer therefore evaluates source weights by normalised curve position rather than assuming equal control-vertex counts. Endpoint metadata is recreated on target curves before they

are connected to the plugin. Missing metadata is treated as an error, because proceeding without it can reduce the logical node count and invalidate all downstream correspondence.

6.4 Skeleton and weight transfer

The source joint hierarchy is duplicated and mapped through the same source-to-target frame. Corresponding target joints are constrained to follow the source pose for side-by-side demonstrations. The artist selects which source joints influence the Curvenet.

Logical node weights are stored on Curvenet nodes. A transferred node finds its source by logical node ID and copies the weight vector to the corresponding target joint list. Curve control-vertex weights are derived from endpoint node weights and can also preserve user edits through normalised resampling. Consequently, the second mesh does not require painting every polygon vertex. The authored weighting work remains on the sparse Curvenet.

6.5 Region correspondence

Matching graph counts are necessary but not always sufficient to prove that region 12 on one mesh corresponds to the intended region on another. The implementation can store source-region triangle samples in logical or local space and use them to seed target partitions. The target flood fill remains bounded by its own embedded Curvenet edges. This improves transfer on meshes whose tessellation produces different small components around nearly coincident boundaries.

The most stable validation combines three checks: the target graph has the same logical nodes and edges, the full-surface region count matches the Euler expectation, and every target polygon belongs to exactly one region. A visual colour comparison then acts as a human-readable confirmation rather than the only evidence.

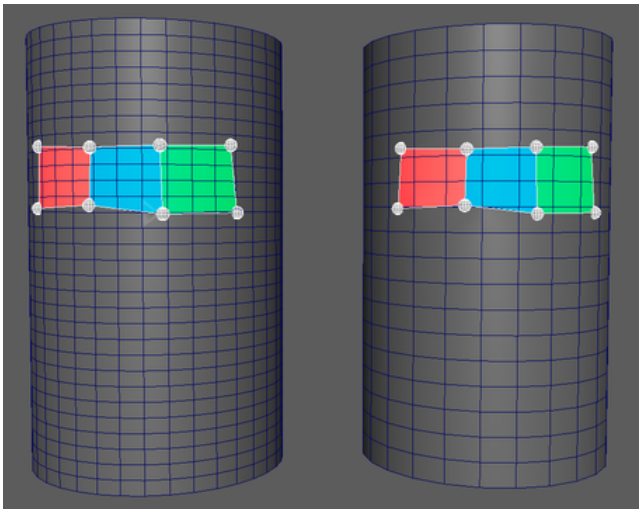


Figure 12. The same logical Curvenet embedded on two cylinders with different polygon topology.

7. Joint Binding and Surface Deformation

7.1 Binding the Curvenet, not the mesh

The polygon mesh is never directly skinned to the demonstration skeleton. Joints influence the Curvenet controls and projected curves. The C++ deformer reads the moving profiles and computes a new surface from the embedded constraints. This is the source of topology independence in the deformation stage: a target mesh needs a valid embedding, but it does not need the source mesh's per-vertex skin weights.

Automatic initial weights are generated from the closest joint segment rather than the closest joint point. For a node position $\mathbf{x}$ and a bone segment from $\mathbf{a}$ to $\mathbf{b}$, the clamped projection parameter is:

$$t = \operatorname{clamp}\left(\frac{(\mathbf{x}-\mathbf{a})\cdot(\mathbf{b}-\mathbf{a})}{\|\mathbf{b}-\mathbf{a}\|^2}, 0, 1\right)$$

The two segment joints receive weights $1-t$ and t. This creates smoother initial transitions than assigning every node rigidly to its nearest joint. This process provides an initial set of weights rather than interpreting the semantic structure of the model. The weight editor therefore allows the artist to correct inappropriate joint influences on individual Curvenet nodes.

7.2 Sparse logical-node weight editor

Traditional Maya weight painting operates on mesh vertices. That would defeat the transfer objective, so the custom editor exposes the Curvenet's logical node spheres. The user poses the skeleton, selects a joint as the active influence, selects one or more node spheres and changes their weight. Marker size is scaled to the mesh and the black Curvenet can be hidden to reduce visual clutter.

Editing only the endpoint control vertices caused breaks in incident curves. The final rule treats a profile as the interpolation between its two logical endpoints. If endpoint weight vectors are $\mathbf{w}_0$ and $\mathbf{w}_1$, a curve control point at normalised arc position t receives:

$$\mathbf{w}(t) = (1-t)\mathbf{w}_0 + t\mathbf{w}_1$$

All joint components are interpolated and renormalised. Thus changing node 1 affects the entire incident curve, with zero additional effect at node 0 and a smooth increase toward node 1. This prevents intermediate CVs from retaining inconsistent weights after an edit and preserves continuity between connected profiles.

7.3 CutChain constraints

Every embedded point in a CutChain stores a profile curve ID, segment ID and local segment parameter. During deformation, the current position of that constraint is interpolated from the current profile samples. Multiple curve constraints may meet at one cut vertex, their prescribed displacements are averaged after logical endpoint consistency has been applied.

This relationship ensures that the cut path and moving profile remain connected. It also explains why the accuracy of embedding matters to deformation: if a CutPath crosses the wrong target edge, the constraint is applied to the wrong physical part of the surface even though the logical curve is correct.

7.4 Rigid component and harmonic residual

The neutral and current curve samples are first compared to estimate a best-fit rigid transformation $(\mathbf{R}, \mathbf{t})$. The implementation accumulates a covariance matrix from centred sample pairs and finds the dominant quaternion of Horn's symmetric matrix. Translation aligns the centroids. If the fit error is within a small scale-relative threshold, the transformation is accepted as a common rigid component, otherwise the solve uses identity for that component to avoid imposing one rotation on a genuinely articulated pose.

For constraint point c, the residual displacement is:

$$\mathbf{d}_c = \mathbf{q}_c - (\mathbf{R}\mathbf{p}_c + \mathbf{t})$$

where $\mathbf{p}_c$ is neutral and $\mathbf{q}_c$ is current. Constrained cut vertices receive $\mathbf{d}_c$. Unconstrained vertices iteratively satisfy the graph Laplace equation:

$$\mathbf{u}_i \leftarrow \frac{1}{|N(i)|} \sum_{j \in N(i)} \mathbf{u}_j$$

while constraint values remain fixed. The final point combines rigid motion and residual deformation. The solver state is cleared on every evaluation, which fixes the earlier problem where resetting a joint to zero left a visibly curved tube.

In code, the half-edge mesh is first converted into an undirected vertex-neighbour graph. Every embedded CutChain point retains a curve identifier, sampled segment identifier and local segment parameter, allowing its neutral and current positions to be reconstructed consistently. Residual displacements at vertices with one or more constraints are averaged and held fixed. Other vertices are initialised from nearby constrained vertices and then updated in two buffers: each iteration reads the previous displacement field and writes the mean of neighbouring values into the next field. Swapping the buffers implements a Jacobi-style relaxation without making the answer depend on vertex traversal order. The current prototype uses a bounded iteration count rather than assembling and factorising a sparse matrix.

This implementation is an adaptation rather than a direct transcription of one source. Laplacian editing provides the discrete smoothness principle (Sorkine et al., 2004), harmonic coordinates establish its relevance to character control (Joshi et al., 2007), and Horn's method supplies rigid alignment. Maya's MPxDeformerNode contract determines when neutral data, cached topology and current curve samples are read (Autodesk, 2025a). The final algorithm combines those elements to meet the project's requirements and was refined through neutral-reset, translation, rotation and articulated-cylinder tests.

This is a smooth positional deformation, but it does not explicitly preserve local rotations or volume. Large finger curls can therefore compress the surface between sparse constraints. Pixar's system solves for deformation gradients on cut cells and reconstructs a subdivision surface, which provides a stronger model of local shape behaviour (de Goes et al., 2022). The current result should be understood as a functional topology-independent prototype and a basis for that future stage.

7.5 Performance considerations

Embedding is intentionally cached after topology is captured. Joint evaluation should reuse CutChains, adjacency and neutral samples rather than repeating intersections and face splitting. The harmonic solve nevertheless visits the cut-mesh graph for each evaluation, and a complete hand with 187 profiles is substantially slower than a cylinder. Maya viewport display, skin clusters on many projected curve control points and diagnostic curves add further dependency-graph cost.

The UI includes optional performance controls for demonstrations, such as hiding the region preview, authoring groups and weight markers. Maya Cached Playback may run out of its assigned memory when complex scenes are animated, disabling or reducing cached playback avoids confusing an application cache limit with a Curvenet algorithm failure. Future work should assemble a sparse linear system once and reuse a factorisation, move more curve-weight evaluation into compact plugin data, and update only vertices affected by changed constraints.

8. Maya Workflow and Engineering

8.1 C++ and Python responsibilities

C++ is used for mesh conversion, crossing storage, cutting, graph construction, region mapping and deformation. These operations are data-heavy and benefit from explicit types and automated unit tests. Python is used for the Maya-facing workflow: drawing contexts, node spheres, scene grouping, NURBS projection, joint selection, skinCluster setup, weight editing, skeleton duplication, transfer and UI.

Implementation did not follow from the Curvenet paper alone. Research publications established the representation and numerical principles, geometry-processing references supplied standard connectivity operations, and Maya documentation supplied the host API contracts. Small synthetic meshes were then used to translate those descriptions into testable C++ operations before the components were combined in Maya. This progression, from published method, to isolated data-structure operation, to DCC integration, was particularly important for half-edge mutation and harmonic propagation, where a correct equation does not by itself define memory ownership, update order, numerical tolerances or dependency-graph behaviour.

This boundary also supports debugging. Python can inspect scene connections and rebuild authoring objects without recompiling the plugin. The C++ layer receives flattened, validated data rather than querying arbitrary scene hierarchies throughout the cutter.

8.2 User workflow

The installed tool creates a **Curvenet** Maya shelf with a launcher button. The main window presents the stages in one vertical workflow:

- select a mesh and start or continue drawing
- finish drawing and connect the projected Curvenet to the plugin
- select the mesh and desired joints, then bind the Curvenet
- optionally refine logical-node weights
- select the source mesh followed by a target mesh and transfer the complete rig

The colour-region preview is available for checking whether the Curvenet has divided the mesh into the expected regions, but artists do not need to use it during the standard workflow. The interface provides short instructions explaining what must be selected before each action. All operations use the mesh currently selected by the artist rather than relying on predefined object names.

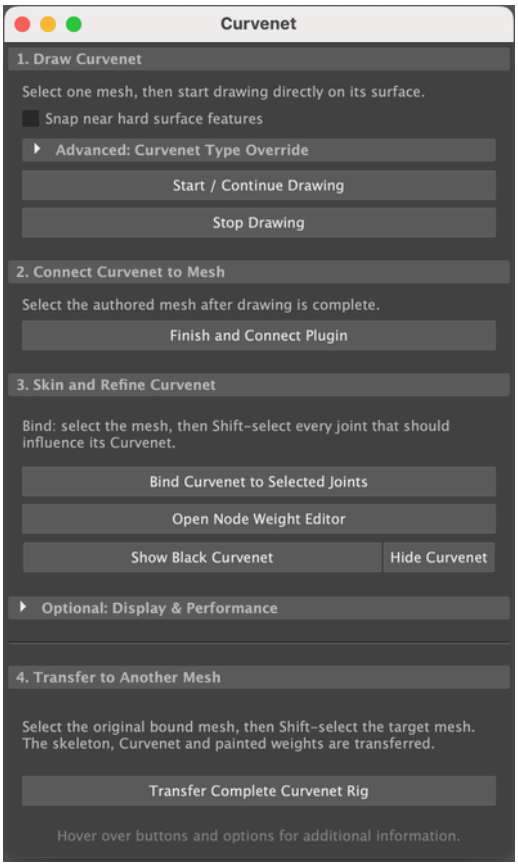


Figure 13. Final Maya workflow interface.

8.3 Managing generated scene objects

The tool creates several groups for different parts of the workflow. These include the original authoring controls, the curves projected onto the mesh, the generated Curvenet display and the optional colour-region preview. Each group is named after the mesh or deformer that created it, allowing multiple Curvenet rigs to exist in the same Maya scene without interfering with one another.

Generated displays are controlled through the tool rather than by deleting them manually. Maya may recreate a deleted display when the deformer is evaluated again, such as when the mesh becomes visible or its display mode changes. The tool therefore stores whether the generated Curvenet should be shown and uses this setting to manage its visibility consistently.

8.4 Diagnostics and testing

Console output is deliberately concise in normal use:

- Curvenet cutting: SUCCESS/FAILED;
- shared and physical Curvenet nodes
- surface-tracked CutPaths
- coverage mode
- edges, faces, mapped faces and unmapped faces
- a detailed reason only when a stage fails

Temporary per-segment, per-endpoint and per-control messages were removed after debugging. The current C++ suite contains 155 tests covering geometry utilities, half-edge conversion, crossings, splitting, CutPaths, shared-node logic, edge and face construction, region barriers, transfer correspondence and harmonic deformation. Unit tests are not a substitute for Maya validation, but they protect the local invariants that are hard to infer from one viewport image.

8.5 Portability

The plugin is built with CMake against the installed Maya SDK. Maya binary plugins are platform-specific: macOS produces a .bundle, Linux a .so, and Windows a .mll. The Python installer can create the shelf on each platform, but the C++ binary must be compiled for the matching Maya and compiler ABI.

macOS is the fully validated interactive platform for this submission. The Linux investigation achieved compilation with GCC 11, all automated tests, and a complete Maya-standalone pipeline. However, interactive Maya on the tested university RHEL workstation repeatedly crashed or stalled during the workflow, despite the standalone pass. The behaviour was not resolved within the project deadline, and Windows was not tested. This is reported as a deployment limitation rather than evidence that the geometry algorithms are platform-independent in production.

9. Evaluation

9.1 Evaluation strategy

Evaluation was organised as a progression rather than one final demonstration. Local unit tests established data-structure invariants. Procedural cylinders tested known graph counts and topology changes in a controlled shape. Drawn cylinder networks tested authoring and explicit connections. Complete hands tested branching anatomy, dense Curvenets, joint binding and transfer. A third triangulated hand acted as a stress test for tessellation and numerical region cleanup.

The principal measures were:

- logical node, edge and expected face counts
- physical node and CutPath completion
- mapped and unmapped polygon counts
- neutral-pose restoration and shared-pose behaviour
- visual separation of adjacent Curvenet regions
- transfer without target mesh vertex painting
- qualitative deformation smoothness and responsiveness

9.2 Unit and integration tests

The final local test suite reported 155 passing tests. Important examples include segment intersection order, multiple crossings on one edge, neighbouring-face traversal, edge split consistency, face division into closed loops, ordered CutChains, explicit shared endpoints, CurvenetEdge construction, Euler-consistent region extraction, protected barriers, uniform translation, rigid rotation and neutral reset of the harmonic solver.

Tests were written to check the underlying behaviour rather than a particular Maya scene. This was particularly important for near-coincident boundaries. The relevant tests create synthetic regions separated by a protected CutPath and verify that cleanup cannot merge across it. The purpose is to express the topological rule independently of the hand mesh that revealed the failure.

9.3 Cylinder experiments

The first Curvenet drawn on Tube A contained eight shared logical nodes, ten edges and three bounded faces. This confirmed that curves sharing the same authored endpoint were recognised as

connected, even when their physical embedding produced different vertices on the mesh.

A larger Curvenet was then drawn around the complete cylinder, including the side and both caps. It contained 14 logical nodes and 28 edges, producing the expected 16 surface regions.

The same Curvenet was transferred to Tube B, which had different polygon subdivisions. Both cylinders produced the same 16 Curvenet regions despite their different mesh connectivity. The coloured regions in Figure 14 show that the transferred Curvenet preserved the same logical division of the surface.

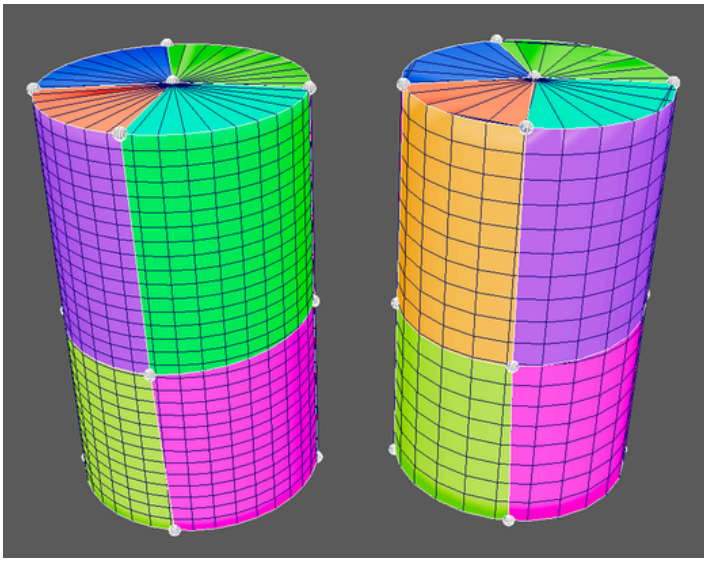


Figure 14. Region validation on two cylinder meshes. Equal Curvenet counts coexist with different polygon counts.

Joint rotation produced the same overall bend on both cylinders. Returning the joints to their neutral values restored the original shapes. Separating the rigid motion from the smoothing process also prevented the end caps from becoming distorted during bending.

9.4 Complete hand experiments

The complete hand network contains 96 logical nodes and 187 profile edges. For full-surface coverage, the expected region count is: $F = 2 - 96 + 187 = 93$

The network includes longitudinal profiles along fingers, rings around joints and profiles across the palm and wrist. This is much more demanding than the cylinders: fingers are close in space, profiles meet at high-valence palm nodes, and existing edge loops often lie almost on the transferred curves.

The completed Curvenet rig was transferred to two additional hand meshes with different polygon layouts. At the final validation checkpoint, all three meshes reported 96 shared nodes, 187 edges,

93 full-surface regions and no unmapped faces. The region colours confirmed that the finger sections, palm panels and wrist sections remained separated across all three meshes.

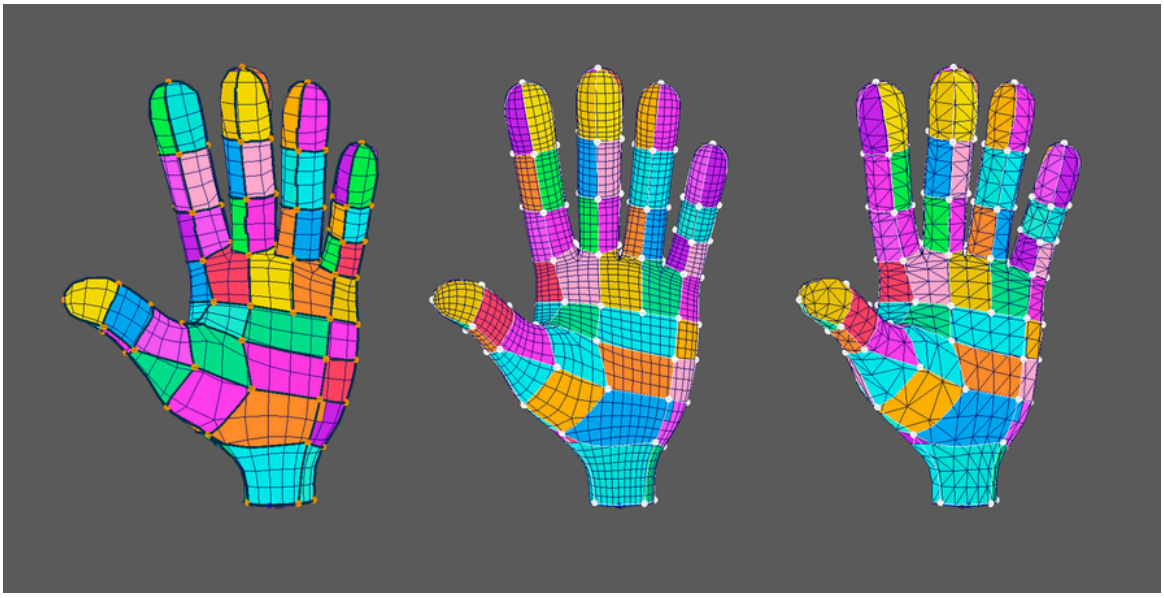


Figure 15. Front region validation on the three hand topologies.

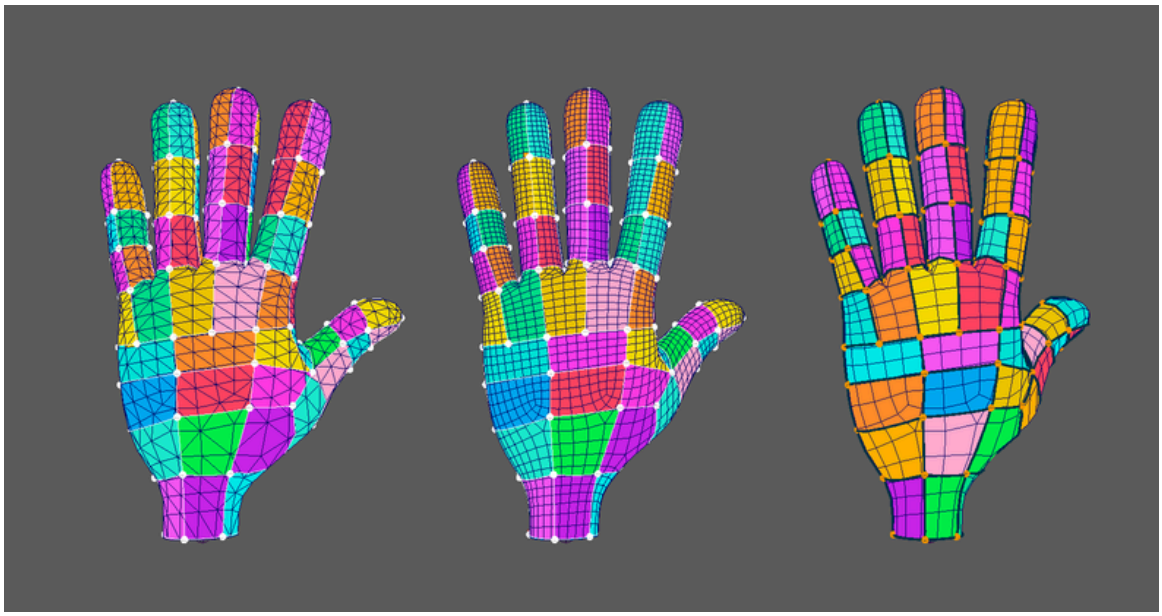


Figure 16. Back region validation on the three hand topologies.

The same skeleton animation drove all three Curvenets, and the edited node weights were transferred through their logical identifiers. The target meshes therefore reproduced the same overall pose without rebuilding or repainting the rig. The node-weight editor was used to remove unwanted joint influences from individual Curvenet nodes, with each correction blended along the connected curves to maintain continuity.

9.5 Performance and usability

The cylinder workflow is practical for iterative testing. The complete hand is slower: initial connection and region-preview generation can take minutes, and posed evaluation becomes laggy when the full preview, skeleton, many curve CVs and Maya caching are visible. This limited the number of repeat trials that could be performed close to the deadline.

Usability improved markedly over development. Surface clicking replaced manual curve creation, adaptive sphere size made the tool usable on hands and cylinders, explicit endpoint reuse enabled loops, thick display curves prevented blind drawing, the shelf and single-window UI replaced a collection of script snippets, and the node-weight editor reduced refinement from thousands of mesh vertices to 96 logical controls. Remaining issues include visual clutter in dense poses, difficult selection among joints and nodes, long rebuilds, and insufficient progress feedback during expensive C++ evaluation.

9.6 Comparison with the proof of concept

The early proof of concept deformed vertices by distance to one moving curve. It was quick and smooth for small motions, but the deformation field depended directly on target vertex positions and a chosen radius. It had no graph, no distinction between two sides of a curve, no transferable region correspondence and no mechanism for connected profile nodes.

The final prototype is more complex and slower, but it answers the research question more directly. The same logical Curvenet can be embedded independently on two meshes, weights are authored on sparse logical controls, and the surface solve derives from target-specific CutChains. The comparison demonstrates why the additional half-edge and CutPath machinery is not incidental overhead. It is the mechanism that converts a curve interface into a topology-independent representation.

10. Critical Discussion

10.1 What worked

The strongest result is the separation of logical and physical representation. Explicit endpoint metadata solved a fundamental ambiguity that proximity and shared vertex IDs could not solve. It allowed drawn profiles to form a consistent graph on meshes whose cuts generated different identifiers. This design also made skeleton and weight transfer straightforward: logical IDs are stable keys while target embeddings remain free to differ.

The half-edge representation proved appropriate for the investigation. It supported ordered local modifications, physical edge barriers and direct topology validation. The stored CutPaths, half-edges and region relationships developed early in the project were not discarded when deformation was added, they became the link between the current Curvenet samples and mesh constraints.

Euler and cycle-rank checks were effective research tools. They converted an intuitive statement - “these colours look like separate panels” - into a falsifiable expectation. On the cylinders and hands, equal logical counts and complete coverage provided evidence beyond visual similarity.

The sparse weight workflow also supports the topology-independent goal. The user edits 96 logical nodes rather than separate weight maps for meshes containing different numbers of vertices. Curve weights are reconstructed from endpoint weights, avoiding broken middle segments. The same edits can be transferred to corresponding nodes on another mesh.

10.2 What did not work, and why

Several early approaches helped define what the final system needed to preserve. The first difficulty was displaying and projecting the authored curves. Following the nearest mesh edges made the curves dependent on the target topology, while projection from the camera could send a curve around the wrong side of a convex shape. Smoother Bézier previews were also tested, but their displayed shape did not always match the CutPath stored by the plugin. This was misleading because the artist could see one boundary while the mesh was divided along another. The final workflow therefore keeps the displayed curve and its surface-tracked CutPath consistent.

A second problem concerned shared Curvenet nodes. Curves that meet at the same authored control should remain logically connected, even when cutting places their endpoints at different physical vertices on a target mesh. Attempting to identify these connections only from the cut vertices caused valid connections to be lost. Merging nearby mesh vertices was not a safe alternative because it could damage the mesh structure. The final system instead stores the

authored connection separately and restores it after each curve has been embedded into the target mesh.

Region cleanup required a similar distinction between geometric size and logical meaning. Early versions removed very small regions according to area alone. However, a narrow region may be small because a Curvenet boundary lies close to an existing mesh edge, not because the region is unnecessary. Removing it could merge two areas that the artist intended to keep separate. The final cleanup process therefore treats embedded Curvenet edges as protected boundaries that must not be crossed.

The deformation stage exposed a different limitation. Harmonic interpolation produces smooth motion between the constrained Curvenet curves, but smoothness alone does not guarantee that the original volume and local shape will be preserved. This is particularly noticeable when a large unconstrained area lies between a small number of controls. Separating rigid motion from the harmonic solve improved the result by preserving overall translation and rotation. However, the method still does not model detailed local deformation such as volume preservation or contact between nearby parts of a mesh.

10.3 Extent of topology independence

In this project, topology independence means that the same Curvenet rig can be applied to corresponding meshes with different polygon layouts and resolutions. The logical nodes, profile curves, skeleton influences and edited weights do not depend on specific mesh vertices. Instead, each target mesh creates its own physical embedding of the shared Curvenet. This allows the meshes to follow the same joint-driven deformation without skinning their vertices directly. It does not imply that one Curvenet can be transferred unchanged to a completely unrelated shape.

10.4 Relation to Pixar's method

The project implements the paper's central representational insight but simplifies the deformation mathematics. Both systems use profile curves as a sparse, topology-independent interface and require a cut-aware relationship between curves and surface. Both distinguish the reusable rig from the target discretisation.

Pixar's method goes further by assigning richer two-sided data, solving deformation gradients on cut cells and reconstructing a smooth subdivision surface (de Goes et al., 2022). The present system constrains cut vertices positionally and diffuses residual displacements on a polygon graph. It therefore cannot be evaluated as a reproduction of Pixar's final deformation quality. Its contribution is a Maya-based investigation of the preceding infrastructure and a working end-to-

end transfer prototype.

10.5 Evaluation limitations

The evaluation used cylinder tests and three corresponding hand meshes with different polygon layouts. These examples test changes in mesh connectivity and resolution, but they do not represent every type of character or surface. Further evaluation would be needed on meshes with different proportions, levels of detail and polygon structures.

Performance was assessed through the responsiveness observed while using the tool in Maya. Formal timing tests were not completed, so the results may be affected by the workstation, viewport settings and Maya's evaluation behaviour. Separate measurements of curve projection, mesh cutting, region construction and deformation would provide a clearer performance analysis.

Region separation was checked using colour previews together with reported node, edge, region and unmapped-face counts. These checks were useful for finding incorrect boundaries, although an automated comparison of corresponding regions across meshes would provide stronger evidence.

Most interactive development and validation took place in Maya on macOS. The plugin compiled and passed its automated and standalone pipeline tests on Linux, but the complete interactive Maya workflow was not stable on the tested university workstation. The current deployment claim is therefore limited to the macOS environment used for the final demonstration.

11. Conclusion and Future Work

This project investigated how the Curvenet method presented by de Goes et al. (2022) could be developed as a working deformation tool in Maya. The completed prototype allows an artist to draw a network of curves on a mesh, bind its control nodes to a skeleton, adjust their joint weights and transfer the resulting rig to corresponding meshes with different polygon topology.

The main technical contribution is the separation between the logical Curvenet and its physical embedding. Logical node identifiers, curve connections and joint weights remain consistent across meshes, while the half-edge system creates a separate cut representation for each target. This allows the same authored structure to define corresponding surface regions without depending on matching vertex or face identifiers. Cylinder and hand experiments demonstrated this process on meshes with different polygon layouts. The transferred meshes followed the same skeleton animation without requiring their polygon vertices to be skinned or painted individually.

The project also demonstrated a complete connection between Curvenet authoring, surface cutting and deformation. CutPaths embed the profile curves into the target mesh and divide it into regions. The Curvenet controls are bound to joints, and their movement is propagated across the mesh using harmonic interpolation. Separating the overall rigid motion from the smoothing stage improved stability and allowed the mesh to return correctly to its neutral pose. The node-weight editor also provided a practical way to correct unwanted joint influences while preserving continuity along connected curves.

The current deformation remains most suitable as a research prototype. Harmonic interpolation produces smooth motion, but it does not explicitly preserve volume or local rigidity. Dense Curvenets can also be slow to evaluate interactively in Maya. Future development could use a pre-factorised sparse solver and a deformation method based on local transformations or deformation gradients. More robust geometric predicates would improve cutting when Curvenet boundaries lie very close to existing mesh edges. A custom viewport display would also reduce scene complexity by drawing the Curvenet without creating large numbers of separate Maya objects.

Further work is also required for distribution beyond the tested macOS environment. The plugin should be built, packaged and interactively validated for supported versions of Maya on macOS, Linux and Windows. Additional evaluation on a wider range of corresponding character meshes would provide stronger evidence of its reliability.

The project shows that topology-independent deformation does not require topology to be ignored. Instead, the artist’s controls and connections are stored independently, while the mesh-specific topology needed for deformation is reconstructed for each target. The resulting prototype provides a working Maya implementation of this principle and a foundation for further development of Curvenet-based rigging tools.

References

Autodesk. (2025a). MPxDeformerNode class reference. Maya Developer Help. https://help.autodesk.com/cloudhelp/2025/ENU/MAYA-API-REF/cpp_ref/class_m_px_deformer_node.html

Autodesk. (2025b). MFnNurbsCurve class reference. Maya Developer Help. https://help.autodesk.com/cloudhelp/2025/ENU/MAYA-API-REF/cpp_ref/class_m_fn_nurbs_curve.html

Autodesk. (2025c). MFnMesh class reference. Maya Developer Help. https://help.autodesk.com/cloudhelp/2025/ENU/MAYA-API-REF/cpp_ref/class_m_fn_mesh.html

Botsch, M., Kobbelt, L., Pauly, M., Alliez, P., & Lévy, B. (2010). Polygon mesh processing. A K Peters/CRC Press.

Campagna, S., Kobbelt, L., & Seidel, H.-P. (1998). Directed edges—A scalable representation for triangle meshes. Journal of Graphics Tools, 3(4), 1–12. https://doi.org/10.1080/10867651.1998.10487494

Catmull, E., & Clark, J. (1978). Recursively generated B-spline surfaces on arbitrary topological meshes. Computer-Aided Design, 10(6), 350–355. https://doi.org/10.1016/0010-4485(78)90110-0

de Goes, F., Sheffler, W., & Fleischer, K. (2022). Character articulation through profile curves. ACM Transactions on Graphics, 41(4), Article 139. https://doi.org/10.1145/3528223.3530060

Horn, B. K. P. (1987). Closed-form solution of absolute orientation using unit quaternions. Journal of the Optical Society of America A, 4(4), 629–642. https://doi.org/10.1364/JOSAA.4.000629

Joshi, P., Meyer, M., DeRose, T., Green, B., & Sanocki, T. (2007). Harmonic coordinates for character articulation. ACM Transactions on Graphics, 26(3), Article 71. https://doi.org/10.1145/1276377.1276466

Kabsch, W. (1976). A solution for the best rotation to relate two sets of vectors. Acta Crystallographica Section A, 32(5), 922–923. https://doi.org/10.1107/S0567739476001873

Kavan, L., Collins, S., Žára, J., & O'Sullivan, C. (2007). Skinning with dual quaternions. In Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games (pp. 39–46). Association for Computing Machinery. https://doi.org/10.1145/1230100.1230107

References

Lewis, J. P., Cordner, M., & Fong, N. (2000). Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In Proceedings of SIGGRAPH 2000 (pp. 165–172). Association for Computing Machinery. https://doi.org/10.1145/344779.344862

Meyer, M., Desbrun, M., Schröder, P., & Barr, A. H. (2003). Discrete differential-geometry operators for triangulated 2-manifolds. In H.-C. Hege & K. Polthier (Eds.), Visualization and mathematics III (pp. 35–57). Springer. https://doi.org/10.1007/978-3-662-05105-4_2

Orzan, A., Bousseau, A., Winnemöller, H., Barla, P., Thollot, J., & Salesin, D. (2008). Diffusion curves: A vector representation for smooth-shaded images. ACM Transactions on Graphics, 27(3), Article 92. https://doi.org/10.1145/1360612.1360691

Sederberg, T. W., & Parry, S. R. (1986). Free-form deformation of solid geometric models. Computer Graphics, 20(4), 151–160. https://doi.org/10.1145/15886.15903

Singh, K., & Fiume, E. (1998). Wires: A geometric deformation technique. In Proceedings of SIGGRAPH 1998 (pp. 405–414). Association for Computing Machinery. https://doi.org/10.1145/280814.280946

Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., & Seidel, H.-P. (2004). Laplacian surface editing. In Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing (pp. 175–184). Association for Computing Machinery. https://doi.org/10.1145/1057432.1057456

Generative AI declaration:

OpenAI was used to support code troubleshooting and refinement, to improve the structure and clarity of project documentation, and to assist with the creation of the plugin shelf icon. All generated material was reviewed, adapted and verified before inclusion in the final project.

Appendix A. User Workflow

1. Install the module by dragging *drag_to_maya.py* into Maya.
2. Open the **Curvenet** shelf and launch the tool.
3. Select one mesh and choose **Start / Continue Drawing**.
4. Draw connected profile segments; reuse existing spheres to form shared nodes.
5. Choose **Finish & Connect Plugin**.
6. Select the mesh followed by the joints that should influence its Curvenet.
7. Choose **Bind Curvenet to Selected Joints**.
8. Pose the skeleton and use the node-weight editor if refinement is required.
9. Select the source mesh and then a corresponding target mesh.
10. Choose **Transfer Complete Curvenet Rig**.

* **Processing time:** These operations may take several minutes for a dense Curvenet. In the final hand tests, processing took approximately three minutes on the development computer. Maya may appear temporarily unresponsive while the Curvenet is projected, embedded and evaluated. Allow the operation to finish before interacting with the scene or closing Maya.

Appendix B. Principal Validation Outputs

Full-surface cylinder pair

- Shared Curvenet nodes: 14
- Curvenet edges: 28
- Curvenet faces: 16
- Tube A mapped mesh faces: 1,394
- Tube B mapped mesh faces: 635

Complete principal hand pair at stable validation

- Shared Curvenet nodes: 96
- Curvenet edges: 187
- Expected and observed full-surface faces: 93
- Unmapped mesh faces: 0

Automated tests

- Tests passed: 155
- Tests failed: 0