Hair Tubes: Stylized Hair from Polygonal Meshes of Arbitrary Topology

Soorya Narayan J M
Animal Logic
Sydney, Australia
soorya.narayan@al.com.au

Figure 1: Different hairstyles generated from hair tubes.

ABSTRACT

In this paper, we describe a fast, topology-independent method to generate bundles of hair from a mesh defining the outward shape of the hair. This allows artists to focus on the outward appearance and create stylized painterly hairstyles. We describe a novel approach to parameterize a hair mesh using ideas from discrete differential geometry and offer simple controls to distribute hair within the volume of the mesh. We present real-world production examples of various hairstyles created using our proposed method.

CCS CONCEPTS

• **Computing methodologies** → **Mesh geometry models**.

KEYWORDS

Procedural Geometry, Hair Mesh, Hair Modeling, Discrete Differential Geometry

ACM Reference Format:

Soorya Narayan J M. 2023. Hair Tubes: Stylized Hair from Polygonal Meshes of Arbitrary Topology. In *SIGGRAPH Asia 2023 Technical Communications (SA Technical Communications '23), December 12–15, 2023, Sydney, NSW, Australia.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3610543.3626157

1 INTRODUCTION

Detailed hair models are an important part of a character's look. However, modeling individual strands of hair is a tedious process. A common practice is for artists to model hair styles as polygonal surfaces which are then used to automatically generate individual hair strands.

Over the years, numerous approaches have been explored to describe hair curves from higher-level representations. Early methods, as surveyed by [Ward et al. 2007], relied on procedural techniques to generate fine hair details. Although this allowed artists to focus on the overall look, approaches like patch and wisp based modeling had a drawback: hair models were indirectly specified through parameters. Not having an explicit silhouette mesh limited direct control over the outer shape, making it difficult to match a precise look. Having a silhouette mesh also has the benefit of simplifying some types of hair simulation workflows, since the polygon mesh can be tetrahedralized and simulated as a solid. There are existing workflows for converting polygon meshes to hair strands; however, they either impose topology restrictions on the input mesh [Yuksel et al. 2009] or involve converting the mesh to a volume [Ghoniem and Museth 2013].

In this paper, we introduce a new topology-independent approach to generate hair from polygonal meshes. Our aim is to offer an artist-friendly workflow that grants complete control over crafting stylized hairstyles while minimizing the technical demands placed on artists. Our technique leverages discrete differential geometry for mesh parameterization, which serves as the basis for generating the hair strands. We also propose a simple process to break down larger hair models into smaller bundles, giving artists more flexibility in styling hair.

2 HAIR TUBES WORKFLOW

In Figure 2, we provide an overview of the hair generation process. First, we compute a smooth, slowly changing parameterization of the input silhouette mesh, which will allow us to identify evenly spaced cross sections of the mesh (subsection 2.1). After computing the isolines from parameterization, an optional cleanup step

SA Technical Communications '23, December 12–15, 2023, Sydney, NSW, Australia

ACM ISBN 979-8-4007-0314-0/23/12…$15.00
https://doi.org/10.1145/3610543.3626157

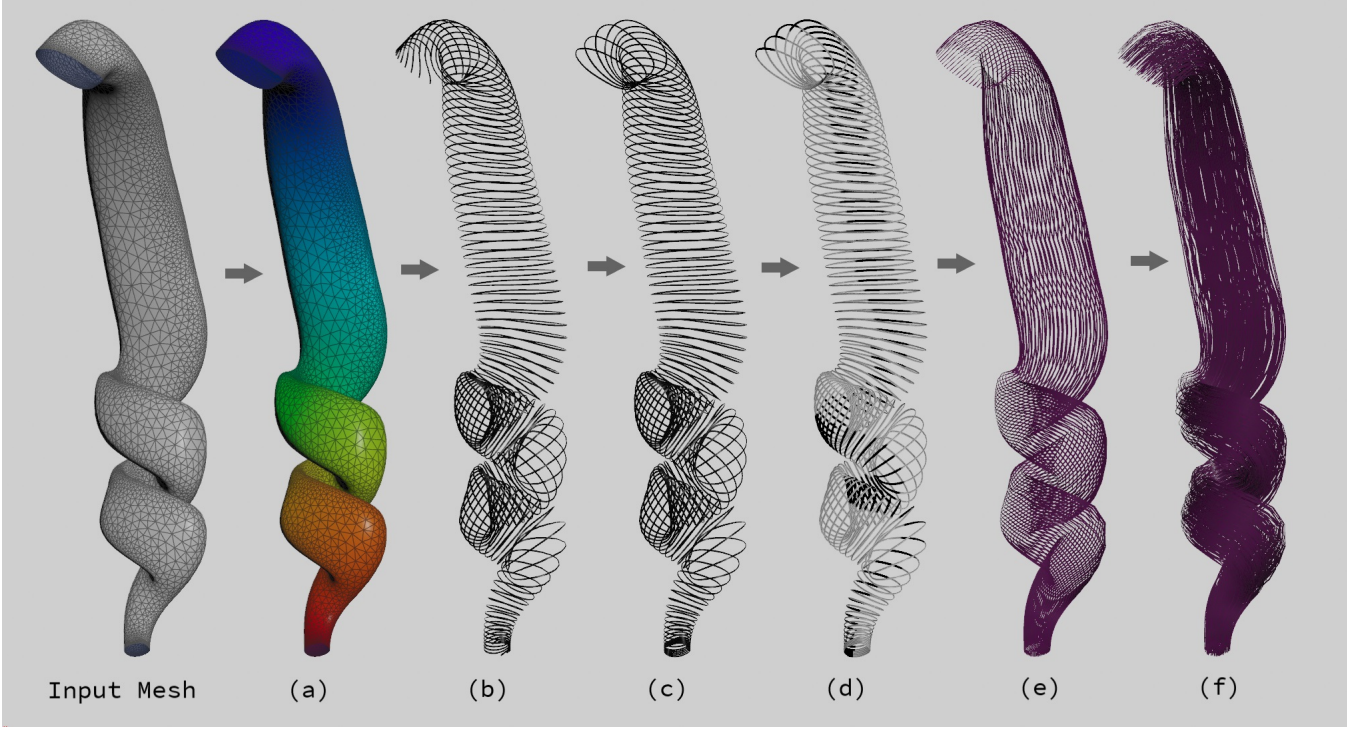

Figure 2: Breakdown of our procedural method to generate hair. (a) Smooth 1-d mesh parameterization. (b) Traced isolines. (c) Optional cleanup of isolines. (d) Finding the corresponding points in isolines. (e) Outer hair generation. (f) Inner hair generation.

ensures satisfactory results for challenging input geometry (subsection 2.2). We then create the outer hairs by re-sampling, sorting, and connecting points on the isolines. To generate internal hairs and prevent tangles, we interpolate between the outer hairs. We provide user controls to customize the hair distribution to attain the desired visual appearance (subsection 2.4). We also briefly describe how we generate attributes to style hair and additional tools to work with larger hair meshes (section 3).

2.1 Mesh Parameterization

An important objective of the algorithm is to generate hair for any input mesh topology. To accomplish this, we need a parameterization of the mesh that allows us to compute evenly spaced cross sections of the mesh perpendicular to its medial axis. The parameterization should be:

(1) smooth, so that the isolines are smooth.
(2) slowly changing, to obtain evenly spaced isolines.
(3) aware of the underlying geometry, so you can smoothly interpolate from one end to the other.

For our initial attempt, we tried calculating geodesic distances by isolating one end loop of the tube and computing distances from this loop to every other point on the mesh. However, as it doesn't account for the curvature of the mesh, this can generate skewed isolines and unsatisfactory results in general (see Figure 3).

To obtain a more natural set of isolines that follows the flow of our mesh, we turned to ideas from geometry processing and graph

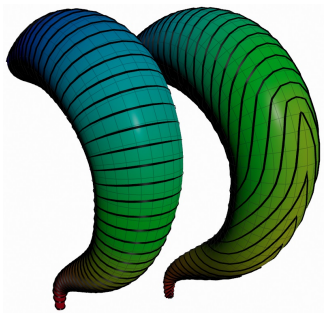

Figure 3: Left: Fiedler Vector, Right: Geodesic distance.

theory. Harmonic functions, which are solutions to the Laplace equation, are commonly used in cage deformation for boundary interpolation. As demonstrated by [DeRose and Meyer 2006], these functions satisfy all the specified criteria. Isenburg et al. [2005] introduced a parameterization technique based on spectral graph theory to computing a vertex ordering for a mesh.

The work of Levy et al. [2006] unifies these concepts by constructing a function basis over an arbitrary topology using the Laplacian and spectral graph theory. They also show how the natural ordering derived from Isenburg et al. [2005] can be obtained from the Fiedler eigenvector of the graph Laplacian. If $\mathbf{G}$ is a simple connected graph with n vertices and if $\mathbf{L}$ is the Laplacian matrix for $\mathbf{G}$, then $\mathbf{L}$ has

n real eigenvalues that satisfy $0 = \lambda_1 < \lambda_2 \leq \lambda_3 \leq ... \leq \lambda_n$. The Fiedler Value ($\lambda_2$) is the second smallest eigenvalue of its Laplacian matrix $\mathbf{L}$ and the Fiedler vector is the corresponding eigenvector [Fiedler 1973]. The Fiedler vector is of interest since it defines a 1-dimensional embedding of the graph on a line that respects the edge lengths of the graph. To enhance geometry awareness, Levy et al. replace the graph Laplacian with the Laplace-Beltrami operator, allowing the generation of a function basis over a mesh of arbitrary topology by solving the generalized eigenvalue problem $\mathbf{L}x = \lambda \mathbf{M}x$ where $\mathbf{L}$ is the Laplacian of the mesh and $\mathbf{M}$ is the per vertex mass matrix. The Fiedler eigenvector of this equation provides a natural vertex ordering that satisfies all of our specified requirements.

In our implementation, we adopt the Laplacian formulation of [Goes et al. 2020], which allows us to compute a discrete Laplacian matrix for arbitrary mesh topologies. Applying the Fiedler vector of the Laplacian back to the mesh as a per-vertex attribute gives us a parameterization that is quick to compute and does not require tweaking per mesh (section 4).

2.2 Generating contour lines

Once we have computed the Fiedler eigenvector, we trace out isolines at evenly spaced isovalues.

When the input mesh lacks sufficient resolution or is not closed, the isolines near the ends of the mesh may be open curves (Figure 2(b)) which can lead to artifacts in the hair generation process. To address this, we utilize a half-edge data structure to identify the open edge loops of the input mesh. Subsequently, we interpolate between the last closed isoline and the open ends of the tube (Figure 4).

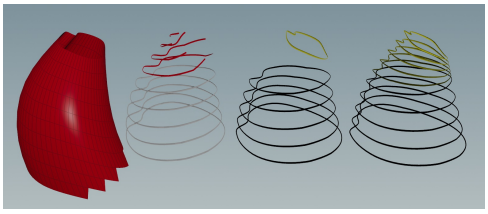

Figure 4: Open Isolines cleanup.

2.3 Generating the outer hair

It is not guaranteed that the winding order of isolines is consistent across the mesh. To achieve a smooth line of hair that follows the shape of the input mesh, we need to determine how to join the points in the isolines, denoted $\mathbf{I}_{i=0..N}$.

To accomplish this, we first re-sample all isolines to have the same number of points, n. Then we perform two closest point lookups between the isolines $\mathbf{I}_i$ and $\mathbf{I}_{i+1}$ to find the corresponding point indices. The first pair of closest points defines a starting line that we use to determine the relative orientation of the second pair of points. We use this information to compute a consistent point ordering and resolve inconsistent winding.

(see Algorithm [1]). Alternatively, it is possible to use a polygon vector area to resolve the winding inconsistencies. To prevent zigzagging of hair along the mesh, we provide the option to smooth

Algorithm 1 Generating Outer Lines.

1: sort($\mathbf{I}_{0..N}$) // sort by iso value
2: resample($\mathbf{I}_{0..N}$, n) // n := user input
3: compute_point_ordering($\mathbf{I}_{0..N}$) // as described above
4: Finally, sort and join corresponding points in isolines

the lines. We then do closest point lookups to the input mesh to make sure the overall shape remains the same.

2.4 Inner Hairs and attributes

Now that the isocircles have a consistent number of points and are sorted, we can generate inner hairs by interpolating the outer hairs. The scheme is similar to a uniform scattering of points within a circle, where each point is represented in polar coordinates as $(\sqrt{r}, \theta)$. In the process, we compute the centroid of each of the isolines, which allows us to get the medial axis of the input mesh for free.

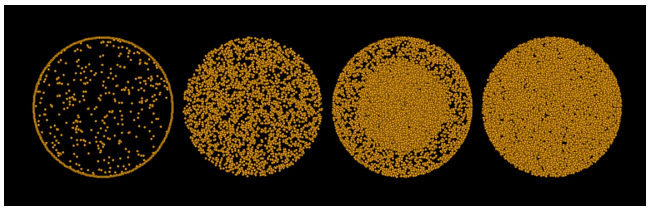

Figure 5: Tube cross sections with varied hair distributions.

In practice, pairing this quick but naive scattering scheme with a ramp to control the density has been flexible enough for artists to obtain their desired hair distribution (Figure 5). This model makes it easy to generate attributes such as distance from the medial axis, which, when combined with the implicit point ordering obtained from the isolines, allows standard post-processing like scraggles and the addition of strays (section 3). In addition, we can also sample the input mesh at the corresponding points for any shading attributes like texture coordinates or colors, allowing artists to paint maps for the tube input mesh and have the tool automatically transfer them to the generated hairs.

3 STYLING AND ADDITIONAL TOOLING

We can use the process from section 2 to generate clumped looks from a larger coarsely modeled hair mesh or style individual bundles differently. Our mesh parameterization and contour line generation scheme allows us to split larger hair meshes into smaller tubes, as shown in Figure 6. The splitting of tubes can be defined through two parameters, a cross section and a fracture pattern. In Figure 6, the first isoline and a voronoi fracture pattern are shown in the red box. The fracture pattern is parameterized using barycentric coordinates corresponding to its associated cross section. These coordinates are then used to project the pattern to the remaining isolines. Each isoline is split into separate polygons on the basis of the projected pattern, and the corresponding polygons are connected to form separate smaller tubes. These smaller tubes can optionally be modified to add separation and tapering. Once the new tubes have the desired look, the hair tube workflow from section 2 can be used to generate hair.

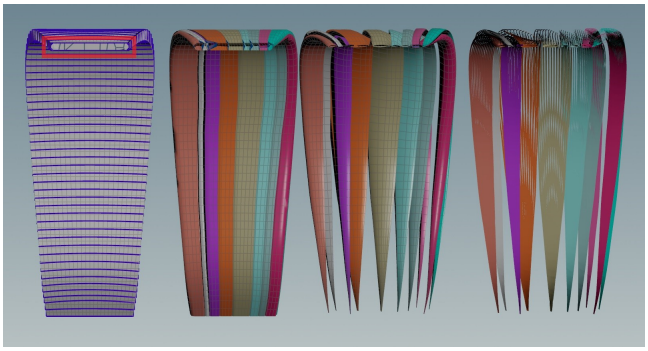

Figure 6: Workflow for large mesh to smaller clumped tubes.

To add realism, each individual tube can be styled to add strays as shown in Figure 7. Strays are chosen as a percentage of total hair, and their distribution can be controlled as a function of distance from the medial axis. Similarly, the points on the hair have an implicit ordering as a result of sorting the isolines and therefore, it's trivial to define a periodic noise function on the strand to add some variety to the look and perform other standard hair workflows like clumping.

Figure 7: Strays and Scraggles.

4 RESULTS

Our algorithm is implemented in Houdini, using Eigen [Guennebaud et al. 2010] for linear algebra routines and Spectra [Qiu 2015] to solve the eigenvalue problem. We implement the discrete Laplacian operator for polygonal meshes according to the formulation by [Goes et al. 2020]. In practice, we found that a hyper parameter value of $\lambda = 1.5$ worked best for the eigenvalue problem solve for quad meshes. We also observed that Spectra generates more consistent results when we compute the first 4 or 5 eigenvectors instead of just the first.

For a typical hair set up with 50-100 tubes and 4000 hairs per tube like in Figure 1, hair styling from loading the mesh to a final result takes between 3-10 seconds on an Intel Xeon Gold 6226R @ 2.90GHz. The most expensive operation in this setup is computing the Laplacian eigenvector in (Section 2.1) but it only needs to be computed once per mesh and can be cached. The rest of the process is relatively cheap to compute, allowing artists to tweak the look interactively at 2-3 fps for a full character groom.

5 CONCLUSIONS AND FUTURE WORK

By relaxing constraints in the hair creation process, we have developed a robust hair workflow that enables artists to design hairstyles with ease.

As with any discrete differential geometry setup, the quality of the meshing affects the accuracy of the computation. Our current implementation is catered to regular, production-quality meshes. While the method performs reasonably well for lower quality meshes with varying edge lengths, it can be improved to handle highly irregular geometries by employing intrinsic triangulation methods [Sharp et al. 2019] to construct the discrete Laplacian matrix. Another approach is to perform the eigenvector calculation on an explicitly remeshed surface and then transfer the computed parameterization to the original surface using UV coordinates. It is worth noting that these more complex computations can be performed as a pre-processing step without impeding the overall workflow.

A limitation of the current approach arises when dealing with certain complex meshes, such as star-shaped inputs, where manual intervention is required to divide the mesh. An extension to address this scenario is to treat the Fiedler vector as a scalar field and utilize it to compute the Reeb graph of the mesh [Doraiswamy and Natarajan 2013]. This Reeb graph can subsequently be employed to automatically partition the mesh into distinct regions, each of which can then be processed using the hair tube algorithm to generate hair. Another easy improvement would be to automatically determine the number of isolines needed for a given mesh based on heuristics related to the curvature and size of the mesh.

ACKNOWLEDGMENTS

We would like to thank Anna Molamphy for the tooling to break down a hair mesh into smaller clumps, Beau Parkes for their help in integrating the hair tubes setup in the Animal Logic pipeline, and Joshua Matthews for their feedback, testing, and setting up the teaser figures.

REFERENCES

Tony DeRose and Mark Meyer. 2006. Harmonic Coordinates. https://api.semanticscholar.org/CorpusID:6098355

Harish Doraiswamy and Vijay Natarajan. 2013. Computing Reeb Graphs as a Union of Contour Trees. *IEEE Transactions on Visualization and Computer Graphics* 19, 2 (2013), 249–262. https://doi.org/10.1109/TVCG.2012.115

Miroslav Fiedler. 1973. Algebraic connectivity of graphs. *Czechoslovak Mathematical Journal* 23, 2 (1973), 298–305. http://eudml.org/doc/12723

Ashraf Ghoniem and Ken Museth. 2013. Hair Growth by Means of Sparse Volumetric Modeling and Advection. In *ACM SIGGRAPH 2013 Talks*. Association for Computing Machinery, New York, NY, USA, Article 34.

Fernando De Goes, Andrew Butts, and Mathieu Desbrun. 2020. Discrete differential operators on polygonal meshes. *ACM Transactions on Graphics* 39, 4 (2020).

Gaël Guennebaud, Benoît Jacob, et al. 2010. Eigen v3. http://eigen.tuxfamily.org.

M. Isenburg and P. Lindstrom. 2005. Streaming meshes. In *Visualization Conference, IEEE*. IEEE Computer Society.

B. Levy. 2006. Laplace-Beltrami Eigenfunctions Towards an Algorithm That "Understands" Geometry. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*.

Yixuan Qiu. 2015. Spectra: C++ Library For Large Scale Eigenvalue Problems. https://spectralib.org.

Nicholas Sharp, Yousuf Soliman, and Keenan Crane. 2019. Navigating Intrinsic Triangulations. *ACM Trans. Graph.* 38, 4 (2019).

Kelly Ward, Florence Bertails, Tae yong Kim, Stephen R. Marschner, Marie paule Cani, and Ming C. Lin. 2007. A Survey on Hair Modeling: Styling, Simulation, and Rendering. *IEEE Transactions on Visualization and Computer Graphics* 13, 2 (2007).

Cem Yuksel, Scott Schaefer, and John Keyser. 2009. Hair Meshes. *ACM Trans. Graph.* 28, 5 (2009).