Published in final edited form as:

J Vis Lang Comput. 2015 August ; 29: 54–62. doi:10.1016/j.jvlc.2015.02.006.

3D Modeling of Branching Structures for Anatomical Instruction

William A. Mattingly, Ph.D.¹, Julia H. Chariker, Ph.D.², Richard Paris, M.S.¹, Dar-jen Chang, Ph.D.¹, and John R. Pani, Ph.D.²

¹Department of Computer Engineering and Computer Science, University of Louisville, Louisville, Kentucky USA

²Department of Psychological and Brain Sciences, University of Louisville, Louisville, Kentucky USA

Abstract

Branching tubular structures are prevalent in many different organic and synthetic settings. From trees and vegetation in nature, to vascular structures throughout human and animal biology, these structures are always candidates for new methods of graphical and visual expression. We present a modeling tool for the creation and interactive modification of these structures. Parameters such as thickness and position of branching structures can be modified, while geometric constraints ensure that the resulting mesh will have an accurate anatomical structure by not having inconsistent geometry. We apply this method to the creation of accurate representations of the different types of retinal cells in the human eye. This method allows a user to quickly produce anatomically accurate structures with low polygon counts that are suitable for rendering at interactive rates on commodity computers and mobile devices.

1 INTRODUCTION

Branching structures can be found throughout human anatomy and can be counted among some of the most challenging structures to learn in medical education. Some examples of branching structures include blood vessels, neural pathways, and air passages in the lungs. Typically, a long branching structure winds in and around multiple structures within an organ or tissue. Understanding the relationship between individual branches and adjacent structures is important because it has a direct bearing on normal and abnormal function in a particular tissue.

Branching structures present challenges for learning that computer-based instructional models can help overcome. Their three-dimensional structure is generally complex with individual branches that are visually indistinct from one another. Computer models embedded in highly interactive educational programs, allow students to explore complex

Corresponding Author: William A. Mattingly, Ph.D., Postdoctoral Associate, University of Louisville, Louisville, Kentucky 40202, Phone: 270.776.5660, wamatt02@louisville.edu.

Publisher's Disclaimer: This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final citable form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

structures from multiple perspectives promoting a better understanding of their shapes and relationships to nearby tissues [1,2].

Students must also learn to identify branching structures in sectional anatomy. Here, structures are identified in two-dimensional sections of tissue that have been imaged or removed from a three-dimensional structure (for example, magnetic resonance images (MRI) and histological sections viewed under a microscope). The two- and three-dimensional views appear very different, and multiple mappings are possible, adding to the challenges in learning. In this case, computer models can be sectioned at different depths and orientations, providing students with an opportunity to explore the relationship between the two- and three-dimensional structures [1,2].

Several factors are important in considering the different methods available for modeling branching structures. Ideally, a method would allow for the generation of anatomically correct structures within a reasonably short period of time. It would also provide ease of use for individuals with little experience in modeling 3D structures. In order to embed models in interactive learning programs, the method should also allow for the creation of individual structures in complex scenes with low polygon counts. This is necessary to provide interactive capabilities such as highlighting, naming, rotating, and removing/adding structures during learning.

Currently available methods for modeling 3D structures vary in terms of these factors with each having particular costs and benefits. Although volume reconstruction of branching structures might provide an accurate representation of anatomical structure, production is slow and expensive. In addition, volume rendered models are not useful because no individually modeled structures are produced that can be treated as separate objects in an instructional program. There are tools available in 3D modeling software such as Maya [3] and 3DS Max [4] from Autodesk or Blender [5] from Blender Foundation that allow for rapid development of individual structures. However, the modeling tools in these programs are often based on the simplistic extrusion or loft tool, and produce a self-intersecting mesh at branch points, making them unsuitable for anatomical instruction. Positioning each vertex and polygon of a branching structure using an interactive modeler can produce an accurate representation, but the process is too time consuming even for an experienced user.

To overcome these challenges, we developed software for creating polygonal mesh models of branching structures with the following requirements. First, structures needed to be modeled as individual objects that can be distinguished from a larger scene. Second, modeled structures needed to accurately represent the shape of anatomical structures in the body, meaning at each branch point the mesh needed to possess a smooth curvature. Third, the model must have a low polygon count to facilitate real time interaction. And finally, the models need to be constructed quickly and inexpensively.

In this paper, we propose an algorithm for the creation of 3D branching models meeting our necessary criteria. We describe the necessary parameters for defining any type of branching structure and develop an algorithm for the construction of its polygonal mesh representation.

We implement the algorithm in an interactive application and in a batch application for procedurally generated structures.

The main benefit of interactive modeling tools is the real time feedback they provide to the user. This allows users to visualize the exact effect of certain parameters on the shape of a geometric object. Most 3D scene modeling software packages have many geometric primitives that can be created and modified in real time. This provides increased control over the final shape. The object is stored internally by its parameters and only at the final stage of rendering does it become simply a collection of polygons. The low polygon count of the branching meshes created by our method allow for the real time modification of the position of a branch and the thickness of its connections even after smoothing the structure using surface subdivision methods.

Our algorithm can also be used in batch or scripted applications as a means of producing a large complex structure defined by a list of connected branching structures and their parameters. This list can be constructed one branch at a time by a 3D modeler or created procedurally using various methods such as L-systems [6] and particle systems [7].

We show the polygonal meshes of several anatomical structures in the eye created using our algorithm. These models provide interactive and accurate representations of several key anatomical structures in the retina. The method is applicable to many other types of branching structures but most notably to organic structures, at both microscopic and macroscopic levels.

The remainder of the paper is organized as follows. In section 2 we review the previous methods for creating 3D representations of branching structures. We describe our branching structure 3D mesh construction algorithm along with the parameters required for the construction in section 3. In section 4 we show rendered images of meshes created using our algorithm. We also describe the application of our algorithm to the creation of anatomical cell structures in the human retina. Finally, we conclude in section 5 with our summary and intended directions for future work.

2 PREVIOUS WORK

3D Modeling of branching structures is necessary for the visualization of human vasculature for diagnosis of medical conditions, anatomy education, and vascular therapy planning. Preim and Oeltze [8] provide an overview of the different ways these visualizations are approached in practice, as model-free or model-based.

In model-free approaches, a voxel, or volumetric pixel, is an element of volume representing some value in a regularly spaced 3D grid. Once a data set has been stored in this format from CT, ultrasound, or MRI scans, it can then be visualized using volume rendering. [9] Each voxel of the data set must also have associated information that will determine its appearance, such as color and opacity. Generation of the visualization can be performed by rendering a polygon mesh extracted from voxel data, or by direct volume rendering. The well known marching cubes algorithm creates the polygonal mesh from volume data [10].

Model-based approaches refer to those that will generate a surface according to some input parameters, such as a skeleton centerline and accompanying radius. This type of visualization is applicable to modeling problems in many disciplines and is commonly used to generate geometry representing natural plant life, such as trees. In the model-based approaches, described below, generation of accurate mesh geometry is the primary goal.

The geometry of branching structures was first explored by Bloomenthal [11] and involved implicit formulas for computing intersections. Convolution surfaces have also been applied specifically to vascular structures [12]. Felkel et al. [13] explored modeling tree structures as collections of radii and a skeletal structure. Tobler et al. [14] first demonstrated the modeling of branching structures using surface subdivision by refining a low polygon coarse mesh. Methods for modeling higher order branching using subdivision have also been explored [15].

Some works focus both on producing a skeletal structure and generating a branching mesh to accompany it. In the context of botanical trees, Neubert et al. [16] give an overview of the different methods that can be used to generate the skeletal structure representing a branching surface. The two most prevalent processes for this creation involve either rule-based systems in which specific rules map to certain characteristics of structure [6], [17], [18] and image based systems which reproduce the structure of actual trees upon analyzing some photographic representation [19], [20], [21]. Since these systems go immediately from the generated rule to the mesh representation, there is little need to provide formal constraints for a skeletal structure. But in order to utilize results from a rule-based structure and apply it to different types of mesh representation, an intermediate structure is needed. Certain constraints for this structure would be obvious, such as not allowing distinct nodes to overlap. Many of these principles involving skeletal structures are also used in applications for generating 3D geometries from 2D sketches [22] [23].

Recently, methods for joining separate branches together using a convex hull of their geometric cross sections demonstrate the most robust and consistent means for modeling branches. Creating a 3D convex hull involves generating a polygon mesh from a set of 3D points that will necessarily envelop all of the points in the set. A method for constructing a mesh around an input skeleton using a convex hull algorithm to create joints at branching skeleton vertices is first described in [24]. Between the convex hulls at branching locations there are solid segments created that surround the segments of the input skeleton. This method was further improved upon in [25], where the points used for finding a convex hull, the arguments to the convex hull procedure, are first projected onto a sphere to ensure each point is included as a vertex in the resulting triangulation. Later [26] used this method to create a system which would take a skeleton representing a vascular structure and create a polygon mesh, and [27] improved on this by creating a mesh that is always quad dominant.

In the following section we build upon these methods by developing a new construction method that uses an additional parameter that affects the central shape of a branching structure. By defining geometric limits to this parameter our algorithm provides a mesh with consistent geometry, and supports user modification of the central parameter.

3 ALGORITHM

3.1 Input and Output

The input to generate a branching structure using our algorithm is a skeleton composed of line segments connecting each tube that composes the branch. Each endpoint of the skeleton is a node and its 3D position is used as input to the algorithm for generating the final structure. The thickness of a branch is also needed to generate a correct shape, and so each endpoint has an associated thickness value. Figure 1 shows the user interface for our algorithm implementation, which allows selecting and moving the parameters that define the branch structure. Table 1 shows the input values for each node position and thickness used to generate the structure in Figure 1.

3.2 Input Constraints

For a mesh representing a branching structure we first examine what is involved in constructing a coarse mesh intended for subdivision. Many 3D modeling packages provide an extrude operation, in which a 2D shape is swept along some curve to create a surface. For example, extruding an open square along a line segment and then applying some subdivision scheme to the resulting surface could make a cylinder. The simplicity of this operation leads us to use it as the basis of our mesh construction.

We first consider the intersection of two of these structures at a common point as shown in Figure 2. This input would consist of three connected points each with an associated radius representing the thickness of the structure. A plane bisecting the angle between two segments would partition the structure into two regions. To prevent intersecting faces in the mesh, it is necessary for node positions to always be a sufficient distance from this partition plane, namely, at least the distance of its associated radius.

These criteria also extend to branching segments that have more than three nodes. In those cases a partition plane must be computed for every pair of nodes connected to a branch center, as shown in Figure 3. For a node n_i with radius r_i , n_i must be at least r_i distance from every plane which partitions n_i from other nodes connected to b. Input that does not satisfy this constraint will have a structure with intersecting faces. A partition plane must be computed for every distinct pair of nodes connected to the branch center, and so a total of $\binom{n}{2}$ or $n(n-1)$ planes are needed for a branching structure with n connected nodes. The plane partitioning nodes n_i and n_j and connected to central node b would have the normal:

$$p(n_i, n_j) = \frac{\vec{bn_i}}{|\vec{bn_i}|} - \frac{\vec{bn_j}}{|\vec{bn_j}|}$$

The partition planes are also used to determine the vertex positions for the mesh of the branching structure as discussed in the next section.

3.3 MESH CONSTRUCTION

Our geometric construction process is similar to others that make use of a 3D convex hull algorithm. In this approach vertex groups that represent the 2D shape of the final structure are taken as input to a 3D convex hull operation. This provides the robustness of the 3D convex hull algorithm while still allowing flexibility in the final shape of the structure.

[For clarity we separate the construction process into two parts: boundary vertex generation and central hull generation. Algorithm 1 shows the process for generating boundary vertices and Figure 4 shows the output for a sample input.] The first step of mesh construction involves placement of the vertices that serve as the vertices of the boundary edges of the final structure. Point b represents the center of the branch or the location where other nodes are joining. Connected to b is a list of nodes $n_1, n_2, \dots, n_i$ each with an associated radius value, which determines the position of its associated vertex group. Point b also has a scalar parameter that is used to control the shape of the mesh at the center. Our system is similar to previous solutions in that the vertices for the boundaries of the branching structure are formed from a two-dimensional convex shape centered at the origin. For the input of Figure 4 this shape is a vertex group of size 4 forming a square. Copies of this shape are rotated and translated into position surrounding each of the connecting nodes.

Algorithm 2 details the process for generating the central hull vertices and Figure 5 depicts the output using the same sample used in Figure 4. Once the boundary vertices for the branching structure have been created, the next step is to use the partition planes to position the vertices of the central hull. This is the portion of the mesh that covers the central point b of the input in such a way as to prevent any intersecting faces in the completed mesh. Each boundary vertex serves as the starting point for a ray going in the direction of the connected node to the central node. This ray is intersected with each partition plane associated with the connected node of the boundary vertex. The intersection that is the closest to the boundary vertex must have a position that cannot cross over into the space of any other nodes. These intersections form new vertices of the mesh that we refer to as the central hull.

The vertices of the central hull need to be triangulated into a polyhedron in order to complete the mesh. A simple and robust way to do this is to translate the points outward from the central node until they each lie on a common sphere and then compute their 3D convex hull. The sphere only needs to be large enough to ensure an intersection with the central hull vertices. The largest distance from the central node to a connecting node is sufficient and simple to calculate. Figure 6 shows the sphere and convex hull created for the sample input. Since the 3D convex hull creates a closed polyhedron, some faces must be removed to connect the central hull to the boundary vertices. Removing any faces on the convex hull for which each vertex of the face belongs to the same vertex group surrounding a connecting node does this. After the central hull is triangulated, the vertices can be translated back to their original positions and the faces to complete the mesh are added by connecting the central hull to the boundary vertices.

3.4 Central Hull Shape

Our method allows modifying the shape of the central hull by using a scalar parameter associated with the central node. This scalar serves as a distance that each central hull vertex is translated along a line towards its complimentary boundary vertex. This parameter allows the user to interactively modify the shape of the branch. Figure 7 shows meshes with a different central hull shape achieved by varying the parameter associated with the central node. This parameter has constraints that must be checked to ensure the faces of the mesh do not intersect. The minimum value is 0, which means the vertices of the central hull are not moved at all from their original positions. The maximum value is the shortest distance from the central node to a connected node. Values higher would translate some central hull vertices past their boundary vertex counterparts.

Our algorithm has the advantage of being general enough that it can support any type of closed convex loop as its vertex group pattern. Branching structures are most commonly tubular and so a vertex group that approximates a circle gives the most accurate results. A balance is needed, as a higher number of vertices in the base vertex group will result in more faces in the final subdivided structure. In addition to square vertex groups we have also tested our algorithm with triangular and hexagonal groupings. Examples of each with Catmull-Clark subdivision are shown in Figure 8.

Another advantage of our system is the way it handles input nodes arranged in acute angles, or when all of the connected nodes of a branching structure are confined to one side of the central node as shown in Figure 9. Using partition planes ensure that some vertices of the central hull will have their planar intersections on the opposite side of the central node, which allows the structure to maintain a more volumetrically accurate shape.

4 Application to Modeling Retinal Cell Structures

In this section we discuss the application of our tool to the creation of structures intended for use in anatomical instruction. Specifically, several cell structures in the human retina were modeled based on the user-defined input of a series of branching nodes. Since the models were to be used as instructional aids in a learning environment, low polygon count was important to provide interactive speeds on typical desktop and laptop machines. The models also needed to be constructed without access to large amounts of image data from scans and volume rendering software. In these cases our method is ideal as a modeling tool as it allows for fast prototyping of complex branching structures.

The retina is a structure located at the back of the eye and is one of several structures involved in early visual information processing. It is composed of a variety of cell types, all of which are branching structures. In neural cells, information is transmitted through electrochemical signals. These signals travel from branches on one end of the cell, called dendrites, through a cell body and down a long tube known as an axon, to terminal branches at the other end of the cell. Figure 10 shows an illustration of the different types of cell structures found in the retina and eye.

In Figure 11 a retinal bipolar cell has been described by user input in the form of connected nodes representing the general skeleton shape of the structure, along with approximate thickness measurements for each node. Any node with three or more connections is a branching node created using our algorithm. Separate branches can be connected by performing a loft or sweep extrusion from one branch connection along a path to another. On the right side of Figure 11 the polygonal mesh has been smoothed using a single iteration of Catmull-clark subdivision in order to provide the final mesh for study. Like many neural structures the bipolar cell has a thick cell body that appears as a bulge at the end of the axon. This bulge can be represented in the input by a larger thickness value associated with a node at positions near the bulge.

Each cell type needed to be modeled in 3D in order to give an accurate impression of the environment in the retina. In Figure 12 we compared the appearance of a model-free approach using volume data with that of semi-automated model-based software tools like lofting. The volume rendering produces a representation of neural cells that is very close to the structure of the source volume data. Good applications have been developed for the volume rendering of neural cells with particular focus on rendering the dendritic spines [28]. The left side of figure 12 shows a neural cell from the program P-view [29] that provides volume data 3D rendering. The example model in the figure has 800,000 polygons, which is far too many to render multiple structures in one scene on commodity hardware. This number is often reduced using algorithms for mesh simplification. However, simplification introduces other problems such as the loss of details that may be important. This result can be compared to the output of modeling tools like lofting which create meshes with a lower number of polygons but have unnatural geometry at branching sections. The right side of Figure 12 shows a mesh structure created with 3DS max using loft and boolean operations generated from a spline input. The mesh has 633 faces but branch sections are sharp and angled rather than smooth. These problems can be corrected by manipulating the vertices and faces of the mesh individually, which we call poly-modeling, but doing so requires more skill and time and defeats the purpose of using the automated tools.

Due to the drawbacks associated with volume rendered meshes and lofting tools, we compared the efficiency of structure creation using our interactive software implementation with that of poly-modeling only, which is able to produce satisfactory structures with respect to polygon count and geometric accuracy. We compared the time required to model a retinal structure using the open-source modeling tool Blender with the time required to generate the same structure with our software. The mean time required for structure completion has been displayed for each method in Table 2. On average there is a substantial time savings achieved using the method presented in this paper. Although the time required for manual modeling improved with each attempt as patterns emerged in the workflow that allowed for more rapid performance, the manual approach took over 30 additional minutes for structure completion. An additional drawback associated with this method is that the modeler must have sufficient knowledge of poly modeling techniques. Using our software tool, the only knowledge required is the ability to use an interface to move nodes in three dimensions and to use the interface to scale the thickness of the nodes. In contrast, a manual modeling method requires skill in manipulating the vertices, edges, and faces of polygonal mesh. A

user must also use complex modeling tools to extrude shapes, and add and remove vertices from a mesh and position them appropriately.

Figure 13 shows a group of 12 retinal structures representing four major cell types in the human retina created with our implementation. At the bottom of the scene in blue are ganglion cells. Adjacent to these cells are amacrine cells in red, and bipolar cells in yellow and green. Finally, horizontal cells are at the top shown in purple. Each mesh structure was refined with a single iteration of Catmull-clark surface subdivision. The entire scene has 20,869 faces. This is adequate for a scene in instructional software as it allows the entire group of structures to be displayed interactively on modern computers and mobile devices. Table 3 shows the input nodes, faces, and mesh creation times including subdivision for the various structures.

When comparing methods for creating anatomical models, we return to the four criteria discussed in the introduction for this paper. These are anatomical accuracy, production time, polygon count and the ability to easily separate models into individual structures. Our proposed method is able to satisfy each of these criteria by creating models with high structural accuracy in a short time and with a low polygon count. Additionally, our algorithm can be used to create independent structures providing the ability to interact with individual anatomical structures within a 3D model (e.g., highlight, name, and rotate structures). In Table 4 we show a comparison of the current methods for producing these structures and how each satisfies the four criteria outlined above.

5 Conclusion

We have presented an algorithm that allows for the fast creation of polygonal mesh representations of branching structures. The meshes created possess several useful properties such as, a low polygon count, smooth curvature, and a straightforward construction process. The partition planes used provide a means of validating input data as well as a general and accurate way to control the shape of branches with different shapes and appearance. We provide constraints for the parameters used by our algorithm to prevent the creation of branching structures with intersecting polygonal faces.

Models constructed by our algorithm are ideal for use in learning applications where branching structures need to be quickly prototyped and imported into an interactive environment. The algorithm can represent many types of biological structures such as neural cell networks, bronchial tubes, and vascular structures. Procedural methods for generating these structures can be used to create the input for our algorithm in the form of a connected graph in which each node possesses position and thickness information.

Applications focused on learning anatomical structures benefit greatly from using a consistent solid shading scheme for the color. However there could be situations in which mapping textures onto the surface would be beneficial, and thus a possible avenue for future work involves describing a formula for generating UV texture coordinates along the surface of a branching structure.

Acknowledgments

Funding was provided by grants R01LM008323 and R01LM008323-03S1 from the National Library of Medicine (NLM) at the National Institutes of Health (NIH) (John Pani, PI) and by grant P20GM103436 from the National Institute for General Medical Sciences (NIGMS) at the National Institutes of Health (NIH) (Nigel Cooper, PI).

References

1. Chariker JH, Naaz F, Pani JR. Computer-based Learning of Neuroanatomy: A Longitudinal Study of Learning, Transfer, and Retention. *Journal of educational psychology*. 2011; 103(1):19–31. [PubMed: 23349552]
2. Pani JR, Chariker JH, Naaz F. Computer-based learning: Interleaving whole and sectional representation of neuroanatomy. *Anatomical Sciences Education*. 2013; 6(1):11–18. [PubMed: 22761001]
3. Autodesk. Maya 2015. 2014.
4. Autodesk. 3DS Max. 2014.
5. Foundation, B. Blender. 2014.
6. Prusinkiewicz, P.; Lindenmayer, A. *The algorithmic beauty of plants*. Springer-Verlag New York, Inc; 1990.
7. Reeves, W. Particle Systems: A Technique for Modeling a Class of Fuzzy Objects. *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*; Detroit, Michigan, USA: ACM; 1983.
8. Preim, B.; Oeltze, S. 3D Visualization of Vasculature: An Overview. In: Linsen, L.; Hagen, H.; Hamann, B., editors. *Visualization in Medicine and Life Sciences*. Springer; Berlin Heidelberg: 2008. p. 39-59.
9. Levoy M. Display of surfaces from volume data. *Computer Graphics and Applications, IEEE*. 1988; 8(3):29–37.
10. Lorensen WE, Cline HE. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput Graph*. 1987; 21(4):163–169.
11. Bloomenthal, J. Modeling the mighty maple. *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*; ACM; 1985.
12. Chiew-Lan XJ, et al. Convolution Surfaces for Line Skeletons with Polynomial Weight Distributions. *Journal of Graphics Tools*. 2001; 6:17–28.
13. Felkel, P.; Wegenkittl, R.; Buhler, K. Surface Models of Tube Trees. *Proceedings of the Computer Graphics International*; IEEE Computer Society; 2004.
14. Tobler, RF.; Maierhofer, S.; Wilkie, A. A multiresolution mesh generation approach for procedural definition of complex geometry. *Shape Modeling International*, 2002. *Proceedings*; 2002.
15. Ou S, Bin H. Subdivision method to create furcating object with multibranches. *Vis Comput*. 2005; 21(3):170–187.
16. Neubert, B.; Franken, T.; Deussen, O. Approximate image-based tree-modeling using particle flows. *ACM SIGGRAPH 2007 papers*; ACM; 2007.
17. Oppenheimer, PE. Real time design and animation of fractal plants and trees. *Proceedings of the 13th annual conference on Computer graphics and interactive techniques*; ACM; 1986.
18. de Reffye, P., et al. Plant models faithful to botanical structure and development. *Proceedings of the 15th annual conference on Computer graphics and interactive techniques*; ACM; 1988.
19. Reche-Martinez, A.; Martin, I.; Drettakis, G. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM SIGGRAPH 2004 Papers*; ACM; 2004.
20. Quan, L., et al. Image-based plant modeling. *ACM SIGGRAPH 2006 Papers*; ACM; 2006.
21. Tan, P., et al. Image-based tree modeling. *ACM SIGGRAPH 2007 papers*; ACM; 2007.
22. Farrugia P, Camilleri K, Borg J. A language for representing and extracting 3D geometry semantics from paper-based sketches. *Journal of Visual Languages & Computing*. 2014; 25(5):602–624.
23. Orbay G, Burak Kara L. Pencil-like sketch rendering of 3D scenes using trajectory planning and dynamic tracking. *Journal of Visual Languages & Computing*. 2014; 25(4):481–493.

24. Srinivasan, V.; Mandal, E.; Akleman, E. Solidifying wireframes. Proceedings of the 2004 bridges conference on mathematical connections in art, music, and science; Banf, Alberta, Canada. 2004.
25. Hart G. An Algorithm for Constructing 3D Struts. Journal of Computer Science and Technology. 2009; 24(1):56–64.
26. Younis, H. Fully-automatic Branching Reconstruction Algorithm: Application to Vascular Trees. 2010.
27. Bærentzen JA, Misztal MK, Welnicka K. Converting skeletal structures to quad dominant meshes. Computers & Graphics. 2012; 36(5):555–561.
28. Rodriguez A, et al. Automated three-dimensional detection and shape classification of dendritic spines from fluorescence microscopy images. PloS one. 2008; 3(4):e1997. [PubMed: 18431482]
29. Rodriguez, A. CNIC - Tools: P-View. 2014. Available from: <http://research.mssm.edu/cnic/tools-pview.html>

Highlights

- We develop an algorithm for easy construction of 3D branching surfaces.
- Constraints are defined which provide limits on useful input.
- Our algorithm is demonstrated constructing nerve cells of the human retina.

Figure 1.
Interactive branch modeling application

Figure 2.
Branch input and partition plane

Figure 3.
Branch input and partition planes for three connected nodes

Figure 4.
Example boundary vertex generation with 4 nodes and central node

Figure 5.
Process of central hull construction

Figure 6.
Branches with varying central hull shape

Figure 7.
Branch structures using triangular and hexagonal vertex groups

Figure 8.
Branch shape for acute input angles

Figure 9.
Artist illustration of retinal cell structures

Figure 10.
Bipolar retinal cell model created using our software. Shown are input

Figure 11.

Two examples of 3D surface of a neuron. Left: Mesh generated from volume data using P-view software [29]. Right: Mesh created with 3D modeling loft and boolean operations using 3DS Max [4]. The insets illustrate the inorganic branch topology (A) and the poor structure of the adjoining mesh (B).

Figure 12. Scene with models created by our software. The structures represent four major types of retinal structures in the eye.

Table 1

Position and thickness of nodes in Figure 1

Node	Position: (x, y, z)	Thickness
b: Center Node	(1.92, 0.93, 2.07)	1.0
n1	(3.68, 6.59, 2.79)	1.0
n2	(1.29, 1.35, -3.91)	1.0
n3	(4.42, -2.82, 1.64)	1.0
n4	(5.78, -0.64, 7.09)	1.0
n5	(0.13, 0.41, 7.07)	1.0
n6	(-3.34, 0.44, 2.29)	1.0

Table 2

Time comparison between creating a retinal ganglion structure with 3D modeling tools versus our software

Trial	Modeling Time (min:sec) ganglion (blender)	Modeling Time (min:sec) ganglion (software)
1	47:19	9:34
2	42:25	9:36
3	41:15	9:40
4	35:17	9:29

Table 3Features of retinal models from **Figure 13**

Structure	Nodes	Faces	Subdivided	Time (s)
ganglion	129	784	2868	0.089
bipolar	60	334	1242	0.032
horizontal	104	573	2136	0.046
amacrine	83	455	1698	0.043

Table 4

Benefits comparison for 3D model creation methods. Volume reconstruction has high polygon count and difficulty producing independent structures. Lofting and extrusion provide low structural accuracy. Poly-modeling has high production time. Proposed method provides best balance.

Method	Structure Accuracy	Production Time	Polygon Count	Independent Structures
Volume Reconstruction	varies	long	high	no
Software Tools (e.g. lofting)	low	short	low	yes
Poly-modeling	high	long	low	yes
Proposed Algorithm	high	short	low	yes

Algorithm 1**Boundary vertex generation from node input**

Input: Central node b

List N of i nodes: $\langle n_1, n_2, \dots, n_i \rangle$

List S of i scalars: $\langle s_1, s_2, \dots, s_i \rangle$

List G of j vertices: $\langle v_1, v_2, \dots, v_j \rangle$ forming a convex 2D polygon around y -axis

Output: List V of $i \cdot j$ boundary vertices

foreach node n and scalar s in N and S

 Let M = rotation matrix from y -axis to $\overrightarrow{bn}$

 foreach vertex v in G

$V_{ij} = M \times (v \cdot s) + n$

 endfor

endfor

Algorithm 2**Central hull construction**

Input: Central node b

List N of i nodes: $\langle n_1, n_2, \dots, n_i \rangle$

List V of $i \cdot j$ boundary vertices from Algorithm 1

List P of p partition planes

Output: List C of $i \cdot j$ central vertices

foreach boundary vertex $v_{i,j}$ in V

 Let R = ray from $v_{i,j}$ in the direction of $\overrightarrow{bn_i}$

$C_{i,j} = \text{intersect}(R, p_1)$

 foreach plane p in P

 if (distance ($v_{i,j}, C_{i,j}$) > distance ($v_{i,j}, \text{intersect}(R, p)$))

$C_{i,j} = \text{intersect}(R, p)$

 endif

 endfor

endfor