Polygon Mesh Generation of Branching Structures

Jo Skjermo and Ole Christian Eidheim

Norwegian University of Science and Technology,
Department of Computer and Information Science,
Jo.Skjermo@idi.ntnu.no
Ole.Christian.Eidheim@idi.ntnu.no

Abstract. We present a new method for producing locally non-intersecting polygon meshes of naturally branching structures. The generated polygon mesh follows the objects underlying structure as close as possible, while still producing polygon meshes that can be visualized efficiently on commonly available graphic acceleration hardware. A priori knowledge of vascular branching systems is used to derive the polygon mesh generation method. Visualization of the internal liver vessel structures and naturally looking tree stems generated by Lindenmayer-systems is used as examples. The method produce visually convincing polygon meshes that might be used in clinical applications in the future.

1 Introduction

Medical imaging through CT, MR, Ultrasound, PET, and other modalities has revolutionized the diagnosis and treatment of numerous diseases. The radiologists and surgeons are presented with images or 3D volumes giving them detailed view of the internal organs of a patient. However, the task of analyzing the data can be time-consuming and error-prone. One such case is liver surgery, where a patient is typically examined using MR or CT scans prior to surgery. In particular, the position of large hepatic vessels must be determined in addition to the relative positions of possible tumors to these vessels.

Surgeons and radiologists will typically base their evaluation on a visual inspection of the 2D slices produced by CT or MR scans. It is difficult, however, to deduce a detailed liver vessel structure from such images. Surgeons at the Intervention Centre at Rikshsopitalet in Norway have found 3D renderings of the liver and its internal vessel structure to be a valuable aid in this complex evaluation phase. Currently, these renderings are based on a largely manual segmentation of the liver vessels, so we have explored a way to extract and visualize the liver vessel structure automatically from MR and CT scans.

The developed procedure is graph based. Each node and connection corresponds to a vessel center and a vessel interconnection respectively. This was done in order to apply knowledge based cost functions to improve the vessel tree structure according to anatomical knowledge. The graph is used to produce a polygonal mesh that can be visualized using commonly available graphic acceleration hardware.

A problem when generating meshes of branching structures in general, is to get a completely closed mesh that does not intersect itself at the branching points. We build on several previous methods for mesh generation of branching structures, including methods from the field of visualization for generation of meshes of tree trunks. The main function of a tree's trunk can be explained as a liquid transportation system. The selected methods for the mesh generation can therefore be extended by using knowledge of the branching angles in natural systems for fluid transportation. This enables us to generate closed and locally non-intersecting polygon meshes of the vascular branching structures in question.

2 Previous Work

In the field of medical computer imagery, visualization of internal branching structures have been handled by volume visualization, as the data often was provided by imaging systems that produced volume data. However, visualization of polygon meshes is highly accelerated on modern commonly available hardware, so we seek methods that can utilize this for our visualization of branching vascular transportation structures.

Several previous works have proposed methods for surface mesh generation of trees that handles branching. We can mention the parametric surfaces used in [1], the key-point interpolation in Oppenheimer [14], the "branching ramiforms" [2] (that was further developed by Hart and Baker in [9] to account for "reaction wood"), and the "refinement by intervals" method [11].

In [12], [17], *rule based mesh growing* from L-systems was introduced. The algorithm used mesh connection templates for adding new parts of a tree to the mesh model, as L-system productions was selected during the generation phase. The mesh connection templates were produced to give a final mesh of a tree, that could serve as a basis mesh for subdivision. This method could only grow the mesh by rules, and could not make a mesh from a finished data set.

The work most similar to our was the *SMART* algorithm presented in [6], [7]. This algorithm was developed for visualization of vascular branching segments in the liver body, for use in a augmented reality aided surgery system. The algorithm produced meshes that could be used with Catmull-Clark subdivision [3] to increase surface smoothness and vertex resolution.

The *SMART* algorithm defined local coordinate axis in a branching point. The average direction of the incoming and outgoing segments was one axis, and an *up* vector generated at the root segment was projected along the cross sections to define another axis (to avoid twist). The child closest to the average direction was connected with quads, at a defined distance. The *up* vector defined a square cross section, and four directions, at a branching point. All remaining outgoing segments were classified into one of these directions according to their angle compared with the average direction. The child closest by angle in each direction was connected with the present tile, and this was recursively repeated for any remaining children.

Furthermore, the *SMART* algorithm did not include any method for automatic adjustment of the mesh with respect to the areas near forking points, and could produce meshes that intersected locally if not manually tuned. Our method automatically generates meshes without local intersection as long as the underlying structures loosely follows natural branching rules.

3 Main Algorithm

The proposed algorithm is loosely based on the *SMART* algorithm. It also uses knowledge of the branching angles in natural systems for fluid transportation as described in Sect. 3.1.

3.1 Natural Branching Rules

Leonardo Da Vinci presented a rule for estimating the diameter of the segments after a furcation in blood vessels, as stated in [15]. The *Da Vinci* rule states that the cross-section area of a segment is equal to the combined cross section area of the child segments, as seen in the following section.

$$\pi r_0^2 = \pi r_1^2 + \pi r_2^2 + ... + \pi r_n^2 \tag{1}$$

A generalization, as seen in 2, was presented by Murray in [13]. Here, the *Da Vinci* rule has been reduced so that the sum of the diameters of the child segments just after a furcation is equal to the diameter of the parent just before the furcating, where d_0,d_1, and d_2 are the diameters of the parent segment and the child segments, respectively. α was used to produce different branching. α values between 2 and 3 are generally suggested for different branching types.

$$d_0^\alpha = d_1^\alpha + d_2^\alpha \tag{2}$$

From this Murray could find several equations for the angle between 2 child branches after a furcation. One is shown in 3, where x and y are the angles between the direction of the parent and each of two child segments. As seen from the equation, the angles depend on the diameter of each of the child segments. Murray also showed that the segments with the smallest diameter have the largest angle.

$$\cos(x + y) = \frac{d_0^4 - d_1^4 - d_2^4}{2d_1^2 d_2^2} \tag{3}$$

Thus, we assume that the child segment with the largest diameter after a furcation, will have the smallest angle difference from the direction of the parent segment. This forms the basis for our algorithm, and it will therefore produce meshes that do not locally intersect as long as the underlying branching structure mostly follows these rules that are based on observations from nature. We now show how this is used to produce a polygon mesh of a branching structure.

3.2 New Algorithm

The algorithm is based on the extended *SMART* algorithm, but uses the *Da Vinci* rule to ensure mesh consistency. It uses data ordered in a DAG (Directed Acyclic Graph) where the nodes contains the data for a segment (direction vector, length and diameter). The segments at the same level in the DAG are sorted by their diameters. An *up* vector is used to define the vertices at each segments start and end position. The vertex pointed to by the *up* vector, is set to be a corner of a square cross section of a segment. The sides of this square defines the directions used in sorting any child segments. The sorting results in four sets of child segments (one for each direction of the square), where the child segments in each set are sorted by the largest diameter.

To connect segments, we basically sweep a moving coordinate frame (defined by a projected *up* vector) along a path defined by the segments data. However, at the branching points we must build another type of structure with vertices, so we can add the child segments on to the polygon mesh. This is done by considering the number of child segments, and their diameters and angles compared to the parent segment.

Starting at the root node in the DAG we process more and more child segments onto the polygon mesh recursively. There are four possible methods for building the polygon mesh for any given segment. If there are one or more child segments in the DAG, we must select one of the methods described in Sect. 3.3 (for one child segment), Sect. 3.5 (for more then one child segment), or in Sect. 3.4 (for cases where the child segment with largest diameter has an angle larger then 90 degrees with respect of the parent segment). If there are no child segments, the branch is at its end. The segment is closed with a quad polygon over four vertices generated on a plane defined by the projected *up* vector and the segments diameter at the segments end.

3.3 Normal Connection

If there is one child segment (with angle between the present and the child segment less then 90 degrees), we connect the child segment to the present segment, and calculates the vertices, edges and polygons as described in this section. Each segment starts with four vertices on a plane at the segments start position. As the algorithm computes a segment, it finds four vertices at the segment's end. It then closes the sides of the box defined by these eight vertices (not the top and bottom).

The first step is to calculate the average direction between the present segment, and the child segment. This direction is the *half direction*. Next, the up vector is projected onto the plane defined by the *half direction* and the segments end point. A square cross section is then defined on the plane at the segment's end position, oriented to the projected *up* vector to avoid twist. The length of the *up* vector is also changed to compensate for the tilting of this plane compared to the original vector. The corners of the cross section are the four end corner vertices for the present segment. These vertices, along with the four original vertices, defines the box that we close the sides of with quad polygon faces.

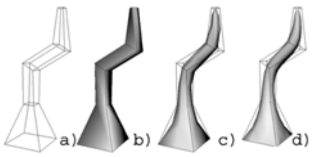

Fig. 1. Simple mesh production. a) the produced mesh, b) shaded mesh, c) one subdivision, d) two subdivisions

In Fig. 1, the mesh of a stem made of four segments connected in sequence can be seen. After processing the segment, the child segment is set as the present segment.

3.4 Connect Backward

When the first child segment direction is larger than 90 degrees compared to the present segments direction. special care has to be taken when producing the mesh (the main part of the structure bends backward). We build the segment out of two parts, where the last part is used to connect the child segments onto the polygon mesh.

The first step is to define two new planes. The *end plane* is defined along the direction of the segment at the distance that equal to the segments' length, plus the first child's radius (from the segments start position). The *middle plane* is defined at a distance equal to the diameter of the first child, along the negative of the present segments direction (from the segments end position). Two square cross sections are defined by projecting the *up* vector into the two newly defined planes. The cross section at the segments top can be closed with a quad surface, and the sides between the segments start and the middle cross section can also be closed with polygons. The sides between the middle and the top cross sections that has no child segments in its direction, can also be closed with polygons.

All child segment (even the first one) should be sorted into sets, defined by their direction compared to four direction. The directions are defined by the *middle* cross section, and each set should be handled as if they were sets of normal child segments, as described in Sect. 3.5. Vertices from the newly defined *middle* and *end* cross sections are used to define the start cross sections (the four start vertices) for each of the new directions. An example can be seen in Fig. 2.

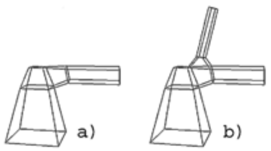

Fig. 2. Mesh production for direction above 90 degrees. a) first child added (direction of 91 degrees), b) next child

3.5 Connect Branches

If there is more than one child segment, we start with the first child segment. The first segment has the largest diameter, and should normally have the smallest angle compared to its parent segment.

To connect the first branching child segment onto the mesh, we first use the same method as in section 3.3. We make a square cross section at the end of the present segment, and its sides are closed by polygons. The distance from the segments start to the new cross section can be decreased to get a more accurate polygon mesh (for instance by decreasing the length by half of the calculated $l + x$ value found later in this section). In the example where vessels in a liver was visualized (Sect. 4.2), the length was decreased as we just described.

A new square cross section is also defined along the *half direction* of the first child, starting at the center of the newly defined cross section. These two new cross sections defines a box, where the sides gives four cross sections (not the top or bottom side). The first child segment (and its children) are recursively added to the top of this box (on the cross section along the *half direction*), while the rest are added to the four sides. Note that the end position of a segment is calculated by vector algebra based on the parent segment's end position, and not on the cross section along the *half direction.* This means that the segment's length must be larger than the structure made at the branching point, to add the child.

When the recursion returns from adding the main child segment (and its child segments), the remaining child segments are sorted into four sets. The sorting is again done by the segments angle compared to the sides of the cross section around the present segment's end point. One must remember to maintain ordering by diameter while sorting.

The vertices at the corners of the two new cross sections defines a box where the sides can be used as new cross sections for each of the four directions (not the top and bottom sides). For each of the four directions, a new *up* vector is defined as the vector between the center of the directions cross section, and a corresponding vertex on the present segment's end cross section. Figure 3 a) shows the *up* and *half direction* when adding a child segment in one of the four directions.

The main problem one must solve is to find the distance along the *half vector* to move before defining the start cross section for the main child segment. This to ensure that there is enough space for any remaining child segments. If the distance moved is too small, the diameter of any remaining child segments will seem to increase significantly after the branching. A too large distance will result in a very large branching structure compared to the segments it connects.

Our main contribution is the automatic estimation of the distance to move, to allow space for any remaining child segments. In Fig. 3 b), we can see the situation for calculating the displacement length m for a given half angle.

The length of m must at least be as large as the root of the $d_1^2 + d_2^2 ... + d_n^2$, where $d_1, d_2..$ are the diameter of the child segments. This because we know from the *Da Vinci rule* and murray's findings that every child segment at this point

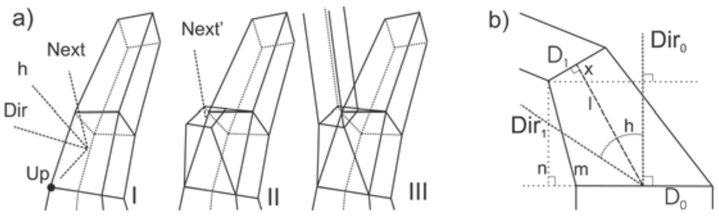

Fig. 3. a) Adding a second child segment. I) new *up*, *direction* and *half direction* vectors for this direction, II) first part of child segment, III) resulting mesh. b) Finding minimum distance m to move along the half vector to ensure space for any remaining branches (as seen from a right angle)

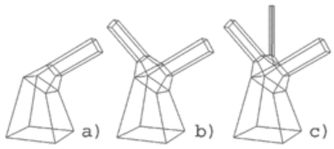

Fig. 4. The mesh production in a branching point. a) First child added, b) next child added, c) third child added

will have equal or smaller diameter then the parent segment (hence the sorting by diameter of child segments). Note that the *half angle (h)* will be less or equal to 45 degrees, as any larger angle will lead to the segment being handled as in section 3.4 (as the main angle then will be larger then 90 degrees).

We could find the exact length of m, but observe that as long as the length of n is equal to d_1, we will have enough space along m. Setting $n = d_1^2 + d_2^2 ... + d_n^2$ gives 4 for calculating the length to move along the *half vector.*

$$l + x = \sqrt{d_1^2 + d_2^2 + ... + d_n^2}/cos(h) + tan(h) * d_1/2 \qquad (4)$$

The error added by using $x + l$ instead of m, will introduce a small error in the mesh production. However, we observe that setting $n = d_1$ seems to give adequate results for most cases. An example result from using the algorithm can be seen in Fig. 4.

4 The Examples

A preliminary application has been produced in OpenGL to test the algorithm. This section shows this application at work. We have used it to produce polygon meshes for both naturally looking tree stems from a L-system generator, as well as meshes of the derived portal vein from a CT scan of a liver. Normal Catmull-Clark subdivision was used for the subdivision step.

4.1 Lindenmayer Generated Tree Stems

An extension to the application accepted an L-system string, representing a tree stem after a given number of Lindenmayer generation steps, as input. The extension interpreted the L-system string into a DAG that the application used to produce a base polygon mesh from. The application then subdivided this mesh to the level set by the user. An example with a shaded surface can be seen in Fig. 5.

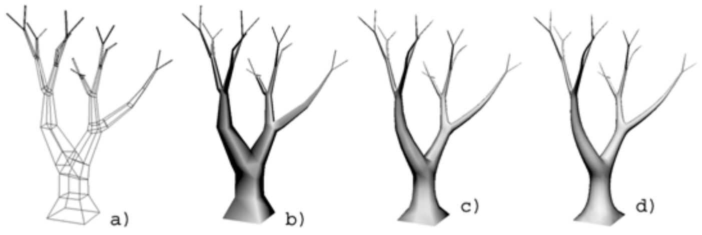

Fig. 5. A tree defined by a simple L-system. a) the produced mesh, b) shaded mesh, c) after one subdivision, d) after two subdivisions

4.2 Delineation of Hepatic Vessels from CT Scans

Several processing steps has to be completed in order to visualize the hepatic vessels from a CT scan. In the preprocessing phase, histogram equalization [8] is first conducted to receive equal contrast on each image in the CT scan. Next, the blood vessels are emphasized using matched filtering [4]. After the preprocessing phase, the blood vessels are segmented using entropy based thresholding [10] and thresholding based on local variance with modifications using mathematical morphology [16]. A prerequisite to our method is that the liver is segmented manually beforehand.

After the vessel segments are found, the vessels' centers and widths must be calculated. These attributes are further used in a graph search to find the most likely vessel structure based on anatomical knowledge. First, the vessel centers are derived from the segmentation result using the segments' skeletons [16]. The vessels' widths are next computed from a modified distance map [8] of the segmented images.

The last step before the vessel graph can be presented is to make connections between the located vessel centers. Centers within segments are interconnected directly. On the other hand, interconnections between adjacent CT slices are not as trivial. Here, as previously mentioned, we use cost functions representing anatomical knowledge in a graph search for the most likely interconnections [5]. The resulting graph is finally visualized using the outlined algorithm in this paper.

A few modification to the existing graph is made in order to make it more visually correct. First, nodes with two or more interconnected neighbors have their heights averaged since the resolution in the y-direction is normally lower

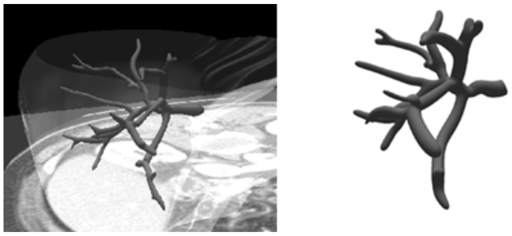

Fig. 6. Left: Portal vein visualized from a CT scan of a liver (the CT scan data can be shown at the same time for any part of the liver, for visual comparison). Right: The same structure withouth the scan data

than that in the image plane. Second, if two interconnected nodes are closer than a predefined limit, the two nodes are replaced by one node positioned between them. Fig. 6 shows the resulting visualization of the derived portal vein from a CT scan of a liver.

5 Findings

Our method for automatically calculating the distance for sufficient space for any remaining child segments after the first child segment has been added, seems to produce good results. The preliminary results from our method applied to visualization of hepatic vessels in the liver gives good results when compared with the CT data they are based on, but these results have only been visually verified (however the first feedbacks from the Intervention Centre at Rikshsopitalet in Norway has been promising). However, a more throughout comparison with existing methods, and verification against the data set values should be completed before using the method in clinical applications.

The algorithm is fast and simple, and can be used by most modern PC's with a graphic accelerator. The meshing algorithm mostly does its work in real-time, but the subdivision steps and any preprocessing slow things down a bit. Graphic hardware support for subdivision will hopefully be available in the relative near future. When this happens, the subdivision of the branching structures may become a viable approach even for large amounts of trees or blood vessels in real-time computer graphics.

References

1. Bloomenthal J.: Modeling the mighty maple. Computer Graphics 19, 3 (July 1985) 305–311
2. Bloomenthal J., Wyvill B.: Interactive techniques for implicit modeling. Computer Graphics 24, 2 (Mar. 1990) 109–116
3. Catmull E., Clark J.: Recursively generated b-spline surfaces on arbitrary topological meshes. Computer Aided Design 10, 6 (1978) 350-355

4. Chaudhuri S., Chatterjee S., Katz N., Nelson M., Goldbaum M.: Detection of blood vessels in retinal images using two-dimensional matched filters. IEEE Transactions on Medical Imageing 8, 3 (1989) 263–269
5. Eidheim O. C., Aurdal L., Omholt-Jensen T., Mala T., Edwin B.: Segmentation of liver vessels as seen in mr and ct images. Computer Assisted Radiology and Surgery (2004) 201–206
6. Felkel P., Fuhrmann A., Kanitsar A., Wegenkittl R.: Surface reconstruction of the branching vessels for augmented reality aided surgery. Analysis of Biomedical Signals and Images 16 (2002) 252-254 (Proc. BIOSIGNAL 2002)
7. Felkel P., Kanitsar A., Fuhrmann A. L., Wegenkittl R.: Surface Models of Tube Trees. Tech. Rep. TR VRVis 2002 008, VRVis, 2002.
8. Gonzalez R. C., Woods R. E.: Digital Image Processing, second ed. Prentice Hall, 2002.
9. Hart J., Baker B.: Implicit modeling of tree surfaces. Proc. of Implicit Surfaces 96 (Oct.1996) 143–152
10. Kapur J. N., Sahoo P. K., Wong A. K. C.: A new method for gray-level picture thresholding using the entropy of the histogram. Computer Vision, Graphics, and Image Processing 29 (1985) 273–285
11. Lluch J., Vicent M., Fernandez S., Monserrat C., Vivo R.: Modelling of branched structures using a single polygonal mesh. In Proc. IASTED International Conference on Visualization, Imaging, and Image Processing (2001).
12. Maierhofer S.: Rule-Based Mesh Growing and Generalized Subdivision Meshes. PhD thesis, Technische Universitaet Wien, Technisch-Naturwissenschaftliche Fakultaet, Institut fuer Computergraphik, 2002.
13. Murray C. D.: A relationship between circumference and weight in trees and its bearing in branching angles. Journal of General Phyiol. 9 (1927) 725–729
14. Oppenheimer P. E.: Real time design and animation of fractal plants and trees. Computer Graphics 20, 4 (Aug. 1986) 55–64
15. Richter J. P.: The notebooks of Leonardo da Vinc Vol. 1. Dover Pubns., 1970.
16. Soille P.: Morphological Image Analysis. Springer-Verlag, 2003.
17. Tobler R. F., Maierhofer S., Wilkie A.: A multiresolution mesh generation approach for procedural definition of complex geometry. In Proceedings of the 2002 International Conference on Shape Modelling and Applications (SMI 2002) (2002) 35–43