Annals of Biomedical Engineering (© 2007)
DOI: 10.1007/s10439-007-9278-x

A Novel Method for Visualization of Entire Coronary Arterial Tree

THOMAS WISCHGOLL,[1] JOERG MEYER,[2] BENJAMIN KAIMOVITZ,[3] YORAM LANIR,[3] and GHASSAN S. KASSAB[4,5,6]

[1]Department of Computer Science and Engineering, Wright State University, Dayton, OH, USA; [2]Department of Biomedical Engineering, University of California, Irvine, CA, USA; [3]Department of Biomedical Engineering, Israel Institute of Technology, Haifa, Israel; [4]Department of Biomedical Engineering, Indiana-Purdue University, Indianapolis, IN 46202, USA; [5]Department of Surgery, Indiana-Purdue University, Indianapolis, IN 46202, USA; and [6]Department of Cellular and Integrative Physiology, Indiana-Purdue University, Indianapolis, IN 46202, USA

(Received 6 April 2006; accepted 1 February 2007)

Abstract—The complexity of the coronary circulation especially in the deep layers largely evades experimental investigations. Hence, virtual/computational models depicting structure-function relation of the entire coronary vasculature including the deep layer are imperative. In order to interpret such anatomically based models, fast and efficient visualization algorithms are essential. The complexity of such models, which include vessels from the large proximal coronary arteries and veins down to the capillary level (3 orders of magnitude difference in diameter), is a challenging visualization problem since the resulting geometrical representation consists of millions of vessel segments. In this study, a novel method for rendering the entire porcine coronary arterial tree down to the first segments of capillaries interactively is described which employs geometry reduction and occlusion culling techniques. Due to the tree-shaped nature of the vasculature, these techniques exploit the geometrical topology of the object to achieve a faster rendering speed while still handling the full complexity of the data. We found a significant increase in performance combined with a more accurate, gap-less representation of the vessel segments resulting in a more interactive visualization and analysis tool for the entire coronary arterial tree. The proposed techniques can also be applied to similar data structures, such as neuronal trees, airway structures, bile ducts, and other tree-like structures. The utility and future applications of the proposed algorithms are explored.

Keywords—Coronary vasculature, Geometry reduction, Large-scale visualization, Occlusion culling, Tree-shaped data set.

INTRODUCTION

To understand such a complex system as the coronary circulation, it is essential to employ anatomically based mathematical models that integrate the physical and biological interactions. It is important for these virtual models to include high detail at the microvasculature (including capillary vessels) as well as on a macroscopic scale (epicardial vessels) in order to integrate the entire coronary vasculature. A visual representation of the anatomical model should include the various parameters of the model. For example, diameters and lengths and their relative changes throughout the vasculature should be visualized for every vessel segment. The visual representation should enable a user to better analyze the parameters of the data set compared to tabular data. In addition, further information should be accessible to the user by selecting a vessel segment and displaying information, such as vessel volume and surface area. The system should also allow the user to edit the individual vessel segments and change their radii or location. Obviously, representing the entire geometry of the vasculature results in a huge set of geometrical data. Ideally, the visualization should be interactive; i.e., the rendering algorithm has to output at least several frames per second (fps).

Rendering such a large-scale model is quite challenging for currently available computing hardware since commodity graphics cards are presently not able to display this amount of information interactively. For the complete coronary arterial model, a total of 6 giga-byte (GB) of geometric information is needed to be transferred from main memory to the graphics card, which presents a limit for interactive rendering. Furthermore, most desktop computers are not capable of handling this amount of data due to insufficient main memory. Hence, the size of such a large-scale anatomical model is prohibitive for rendering on desktop computers without employing out-of-core techniques.

The objective of this study is to develop a visualization method for a view-dependent, interactive decimation of massive tree-shaped data sets. The proposed

Address correspondence to Ghassan S. Kassab, Department of Biomedical Engineering, Indiana-Purdue University, Indianapolis, IN, 46202, USA. Electronic mail: gkassab@iupui.edu

approach will combine a spatial data structure and occlusion queries to reduce the number of triangles necessary to render tree-shaped data sets that exceed the memory present in the computer system. The topology of tree-shaped data sets is exploited in order to reduce the complexity of the triangle mesh coherently. The proposed software system makes use of recent improvements in graphics hardware and employs hardware occlusion queries that allow a faster and more precise occlusion test as compared to software-based approaches. The techniques described in this article can be easily applied to data extracted from any tree-like structures.

METHODS

Anatomically Based Model

Recently, Kaimovitz et al.[15] developed a three-dimensional (3-D) geometric model of the entire coronary arterial tree (right coronary artery, RCA; left anterior descending artery, LAD; and left circumflex, LCx arterial tree) based on Kassab et al.'s coronary morphometric data base.[16] The model spans the entire coronary arterial tree down to the capillary vessels in a prolate spheroid model of the heart and encompasses about 10 million segments. The 3-D tree structure was reconstructed initially in rectangular slab geometry by means of global geometrical optimization using a parallel Simulated Annealing (SA) algorithm. The SA optimization was subject to a global boundary avoidance constraint and local constraints at bifurcations prescribed by previously measured data on branching asymmetry in the coronary arterial tree.[38] Subsequently, the reconstructed tree was mapped onto the prolate spheroidal geometry of the heart. The transformation was made through least squares minimization of the deformation in segment lengths as well as their angular characteristics.

Rendering of Massive Tree-Like Structures

In the previous publication.[15] vessel segments were visualized using standard cylinders. Since consecutive vessel segments do not necessarily form 180 degree angles, these result in visible gaps at the point of transition. To avoid these gaps, the proposed system represents vessel segments as conic cylinders with rotated ends, which are not necessarily orthogonal to the cylindrical axis. In this way, a smooth transition from one segment to the daughter segment(s) can be achieved, thus avoiding any gaps. The individual conic cylinders are pieced together using triangles that are fitted in such a way that an optimal, gap-less approximation is achieved. This results in an accurate visual representation of the entire vascular structure as defined by the data set.

Since several triangles are needed to represent a single conic cylinder, rendering a vascular structure which consists of 10 million vessel segments requires about 220 million triangles to achieve a sufficiently accurate approximation. This in return results in geometry data that amounts to several GB in size which exceeds the main memory of common desktop computers. In addition, transferring this amount of data to the graphics hardware and processing this information overwhelms both the bus system (usually advanced graphics port, AGP, or PCI Express) as well as the graphics hardware. Consequently, techniques are needed that allow the system to handle data sets that exceed the amount of main memory present in the computer as well as reduce the number of triangles to generate the visualization.

Hence, the proposed software system deploys out-of-core techniques which store the entire geometry data on the hard drive only. During the rendering process, only parts of the data are transferred to the main memory. Once these parts are processed, the system automatically removes these parts and loads the next ones for further processing. In this way, the geometry data is loaded in a streaming fashion from the hard drive and then transferred to the graphics hardware for visualization.

In addition, the proposed system reduces the number of triangles using view-dependent geometry reduction, backface-culling, and occlusion-based reduction. View-dependent geometry reduction automatically reduces the amount of detail that is used for representing the vessel segments based on the distance to the viewer. Hence, the vessel segments that appear far away are drawn with less detail (using a lower number of triangles per vessel segment) while the ones in the front are shown in full detail. Since usually only half of a conic cylinder is visible at a time, only the visible half needs to be processed in order to generate the visualization. Accordingly, the number of triangles can be reduced significantly by removing those triangles of a conic cylinder that face away from the viewer.

Similarly, occlusion-based reduction removes those conic cylinders that represent vessel segments which are obstructed by several other vessel segments and therefore invisible from the current location of the viewer. Since these vessel segments are not visible, they can be eliminated without changing the visualization. This reduces the number of triangles that need to be transferred to the graphics hardware. Note that all these techniques for reducing the number of triangles are view-dependent; i.e., whenever the location of the viewer changes these need to be recomputed to ensure that only those triangles are removed that minimally contribute to the current visualization. As a consequence, all computations required need to be implanted very efficiently. For example, better efficiency can be achieved by grouping vessel segments that are close together and then applying the geometry reduction techniques to the entire group. This reduces the computation effort for visibility tests significantly. Grouping of vessel segments can be achieved by, for example, sub-dividing the bounding box of the entire vascular structure into equal sub-areas. A detailed description, including implementation details, can be found in the appendix.

RESULTS

Tree Rendering

Figure 1 shows the complete vascular model including all three major branches. The left image depicts an overview, while the right displays a close-up view of the marked region. Even after zooming into the model, there is still an enormous amount of detail which underscores the complexity of the generated model. The geometric model exceeds the main memory of most desktop computers. In addition, the geometry data results in slow performance due to the enormity of information that needs to be processed to compute a single projected image. Without applying out-of-core methods, only one branch of the vasculature, for example the LCx as depicted in Fig. 2 (a), can be visualized since it is significantly smaller in size (1.8 million vessel segments).

Geometry Reduction

The geometry reduction techniques applied to the data significantly reduced the number of triangles. For example, Fig. 2(b) shows the results for the view-dependent geometry reduction where 20% less triangles were required for generating the image. Figure 3 shows an example of hardware occlusion queries applied to the LCx data set. A reduction of 56% was achieved. The vessel segments in those areas that were identified as occluded are colored in red and drawn at the lowest level of detail. As can be seen in the figure, only those parts that are far away from the observer and obstructed to a large extent by other vessel segments are displayed using a lower level of detail (red).

Performance

For performance testing, two different systems were used. The first one was a Pentium4 2.6 giga-Hertz (GHz) central processing unit (CPU) equipped with 2 GB of main memory and an AGP version of an Nvidia GeForce fx5200 graphics card. The second one was equipped with two AMD64 Opteron 246 2.0 GHz processors and 1 GB of main memory. This system used the PCI Express version of an Nvidia Quadro FX 4400.

As one of the reference data sets, the LCx coronary artery data was visualized. For displaying the data set in full detail (considering all vessel segments at full resolution), the geometry consisted of 25 million triangles. On the first test system without any enhancements, the data set could be rendered at an average frame-rate of 0.5 fps. Figure 4(a) provides a comparison between the frame rates obtained for the full data set and for the instantly reduced data set using occlusion culling. The frame rates over time are depicted for three different rendering methods using the same data set, and are shown while navigating through the model. In full level-of-detail mode, the

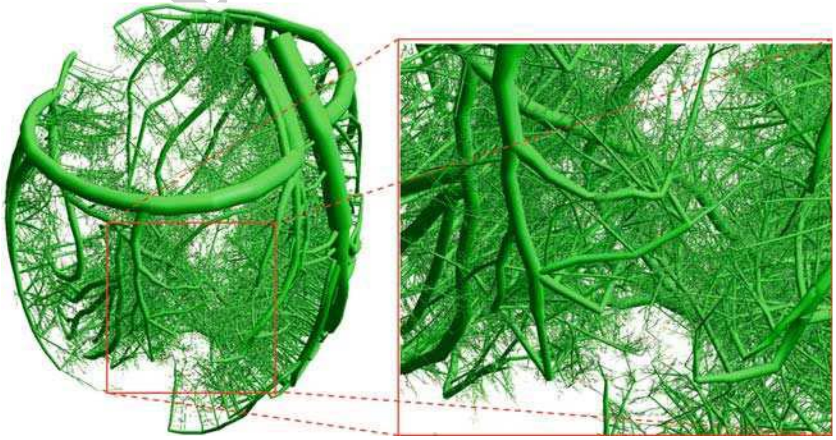


FIGURE 1. Complete representation of the vasculature of a heart and close-up view depicting the large amount of detail in the model.

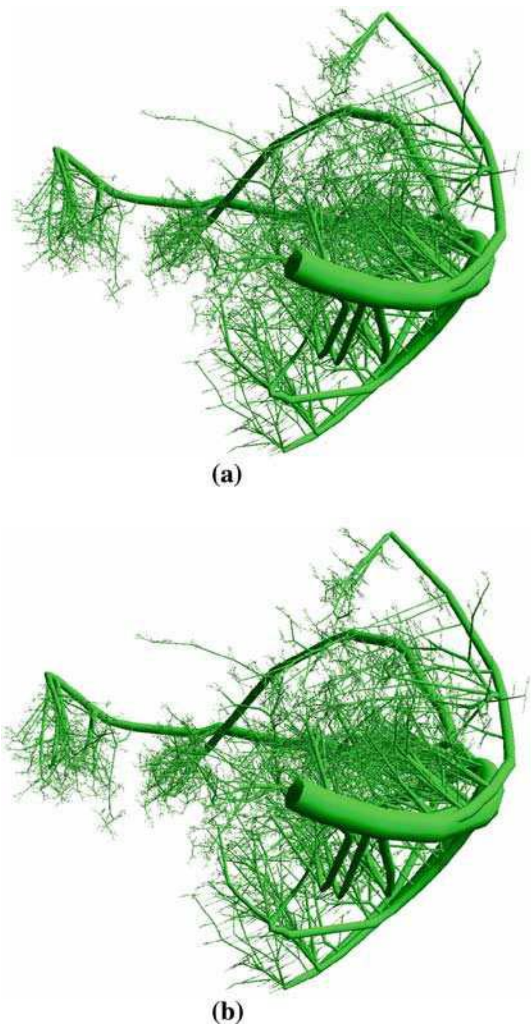


FIGURE 2. Rendering of the geometry of the left circumflex coronary artery (LCx) data set at full resolution (a) and LCx branch rendered with view-dependent geometry reduction enabled (20% reduction) (b).

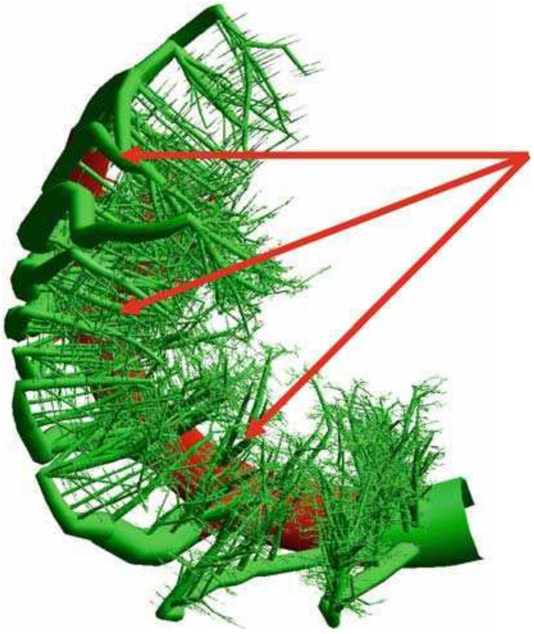


FIGURE 3. Close up of the LCx branch rendered with hardware occlusion culling enabled (56% reduction). Areas rendered with reduced resolution are shown in red as marked by arrows.

graphics hardware could not be used to its full extent when the first computer system was used. By enabling occlusion culling combined with view dependent geometry reduction, the number of triangles that need to be displayed for each frame was trimmed down to an average of 11 million triangles according to Fig. 4(b). Consequently, the frame rate increased to an average of 2.1 fps due to the reduced number of triangles. Since the performance increased by a factor of four while cutting the number of triangles in half only, the saturation of the AGP bus and CPU of the test system is improved resulting in more efficient usage of the graphics hardware.

For the occlusion test, the number of sub-areas used should not be too large since the more time is spent on occlusion culling the less time is available for the actual rendering of the image. In this case, a heuristically determined equidistant scheme of $10 \times 10 \times 10$ subvolumes was used for the performance tests. Due to the fact that the number of triangles that need to be rendered for each frame could be reduced to 11 million triangles, the utilization of the AGP bus is improved. As a result, the rendering system was able to render about 23.1 million triangles per second. Therefore, to achieve a significantly better rendering performance using the graphics hardware, we incur only a minor performance loss for conducting the occlusion tests.

The performance of the backface culling implemented in the system was tested on the RCA data set (consists of 4.3 million vessel segments) which was represented by 77 million triangles. The data set was rendered on the system equipped with an Nvidia GeForce fx5200 graphics card as previously described and utilized the implemented out-of-core technique. Figure 5(a) shows the number of triangles used during rendering. The backface culling was done in a conservative way where only about one third of the triangles were removed. In this way, only invisible triangles are removed. This is especially necessary when rendering vessel segments that are represented by a very low number of triangles. For example, for a vessel segment represented by eight triangles, six of these can be seen in the worst case. According to Fig. 5(a), about 50 million triangles were required for rendering after removing those triangles that face away from the view point. This then increased the rendering rate accordingly as can be seen in Fig 5(b). Originally,

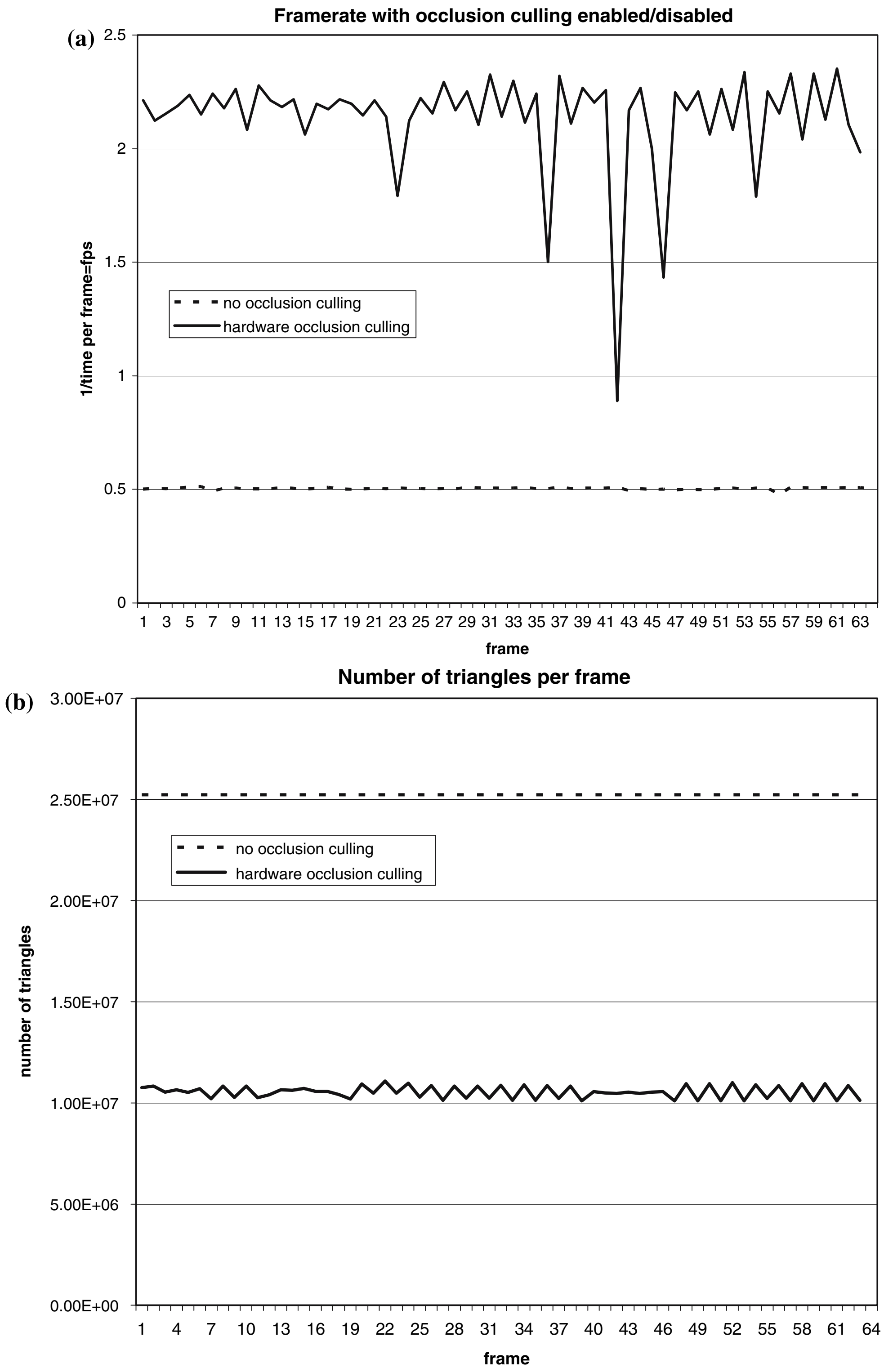
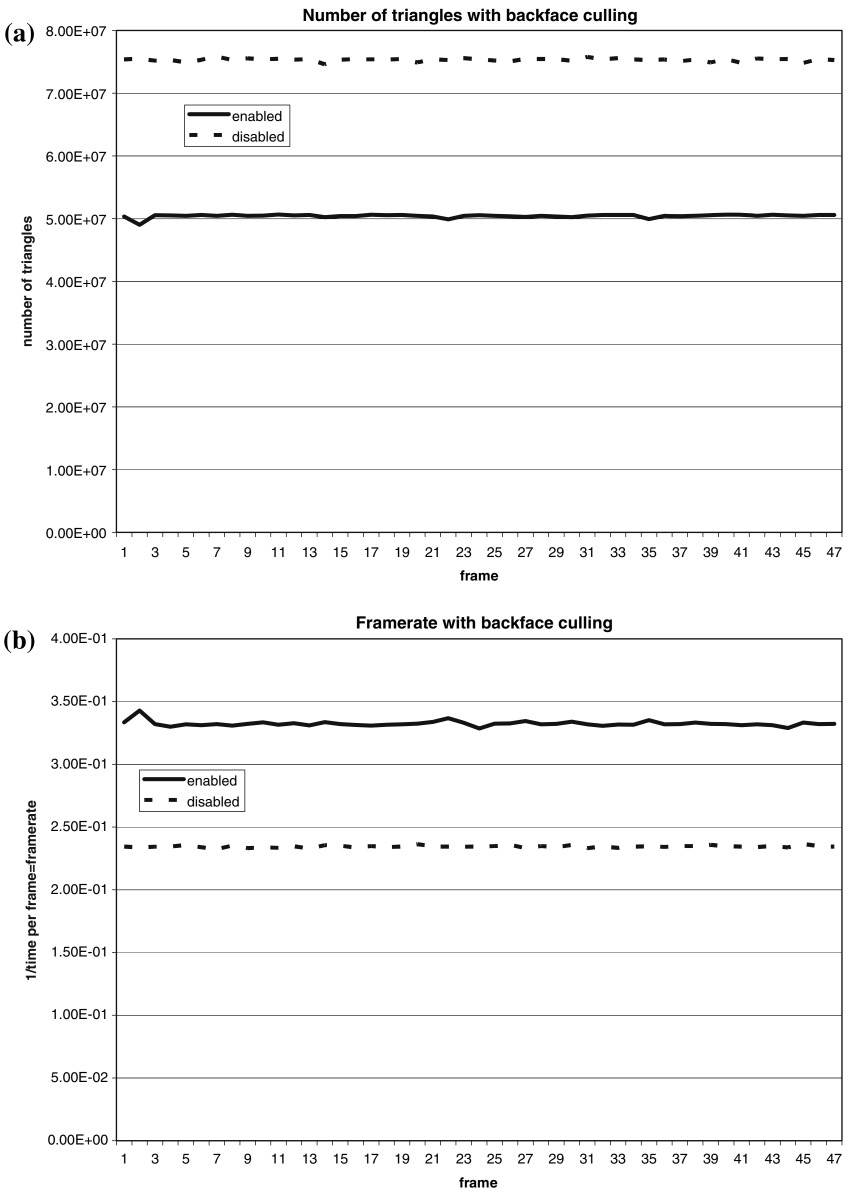


FIGURE 4. Resulting frames per second (a) and number of triangles used (b) when displaying an overview rendering of the left circumflex coronary artery (LCx) branch on a desktop PC equipped with an Nvidia GeForce fx5200 graphics card.

a rendering speed of an average of 0.23 fps was achieved. After removing backfacing triangles, the data set is rendered at 0.33 fps, a 39% improvement in performance. During rendering, the software system had a memory footprint of 1.6 GB mainly used for caching most of the data set.


(a) Number of triangles with backface culling

(b) Framerate with backface culling

FIGURE 5. Number of triangles used (a) and frames per second (b) when rendering an overview of the right coronary artery (RCA) branch on a desktop PC equipped with an Nvidia GeForce fx5200 graphics card with and without backface culling.

The out-of-core technique was tested using all three branches of the coronary arterial tree model. The geometry of this model is represented by 220 million triangles to represent all 10 million vessel segments resulting in 6 GB of geometry information. Rendering was performed both on the system equipped with an Nvidia GeForce fx5200 graphics card as well as an Nvidia Quadro FX 4400. According to Fig. 6(a), rendering a single frame of this data set on the first system took about 3 min and 52 s. Using the second system, rendering the full data set took only about 62 s as can be seen in Fig. 6(b). Obviously, the implemented visualization system benefits from the faster graphics hardware and the 64bit architecture available in the test system. Due to the out-of-core visualization, the full model could be rendered using less than 64 MB of main memory as observed via the Windows task manager. As pointed out previously, the out-of-core approach exploiting standard memory mapping techniques benefits from the caching capabilities of the operating system especially well when the user zooms to a certain area so that the geometry required for rendering this part fits into main memory. Figure 7 shows the performance of the rendering for such a case. An average of 1.9 fps was achieved. The system was able to render the data at more than two fps. Only in those cases where it is required to load the geometry from a sub-area that was not displayed before, the system's

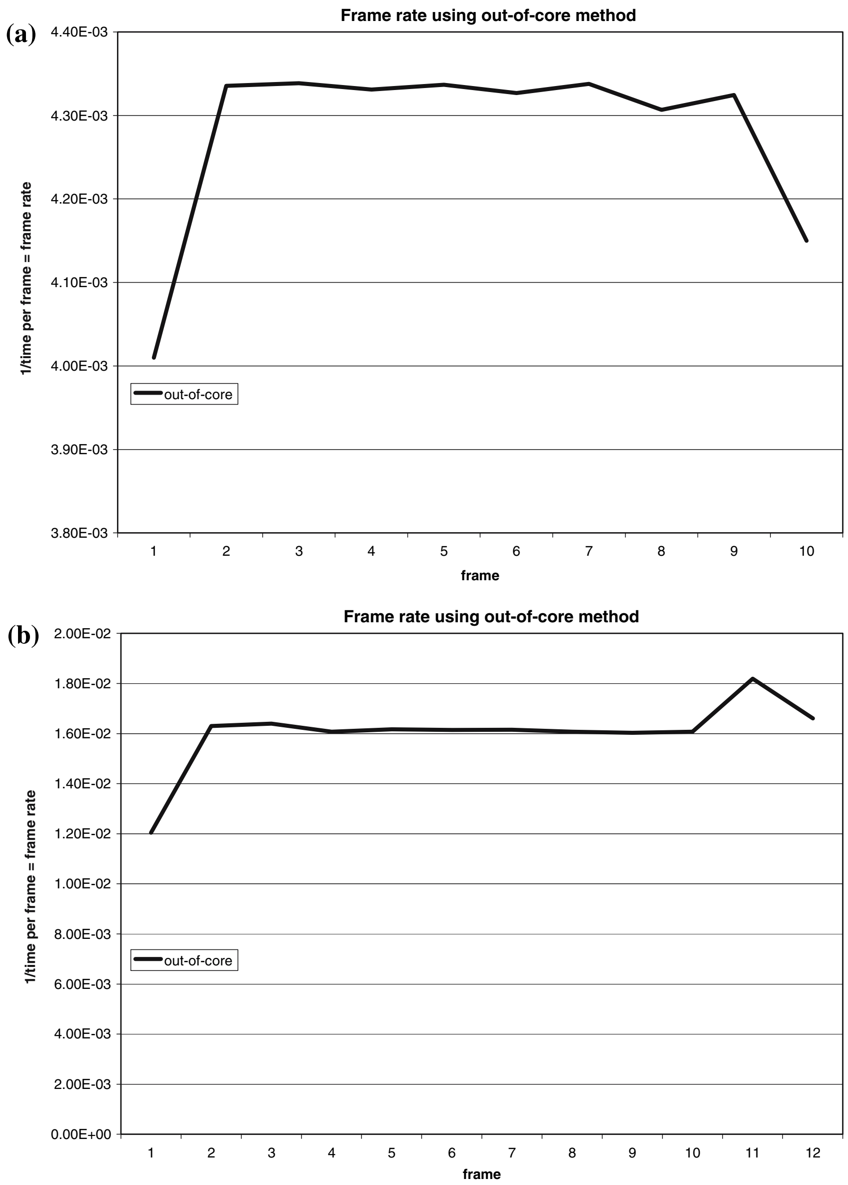


FIGURE 6. Resulting frames per second for rendering an overview of the complete geometric model of the coronary artery tree on two different desktop PCs, one PC equipped with an Nvidia GeForce fx5200 graphics card (a), the other one equipped with an Nvidia Quadro FX 4400 graphics card (b).

performance dropped. Overall, the system was capable of rendering even huge data sets as can be seen from these examples and allows visualization at respectable frame rates even with small main memory computers.

DISCUSSIONS

This study provides a visualization method for a quantitative anatomical model of tree structures (such as coronary arterial trees) that can be used, for instance, to model the temporal and spatial distribution of blood flow in the heart. In this study, the described simulated data set was visualized. It is possible, however, to use the system for other types of data sets, such as microCT scanned specimens[20,26] or data retrieved by an imaging cryomicrotome.[32] The visualization features of the software will serve as an educational tool as well as for data interpretation. It is expected that the software will become a valuable tool for cardiologists, physiologists and students. The details are discussed below.

Visualization

When rendering such a large model, there are three factors that can limit performance. First, the amount of information can saturate the bus system so that the amount of data cannot be transferred fast enough to

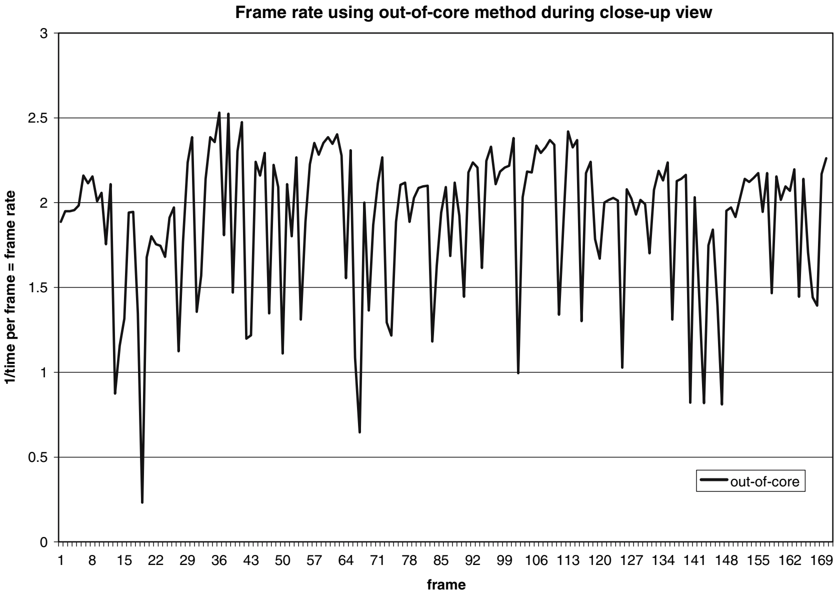


FIGURE 7. Resulting frames per second for rendering the complete geometric model of the coronary artery tree on a desktop PC equipped with an Nvidia GeForce fx5200 graphics card during close-up view.

the graphic card. Even though the PCI Express (PCIe) bus at 2.5 Gbit/s is faster than the advanced graphics port (AGP) with transfer rates of up to 2 Gbit/s, both bus systems can be saturated due to the large size of the model. Second, the capabilities of the graphic card itself may not be sufficient to render the data fast enough. Third, there may not be enough main memory available to store the entire data set. Common 32-bit desktop computers can only be equipped with up to 4 GB of main memory which is not enough for storing the entire geometric representation. If additional information is to be displayed, such as colors for the vessel segments to superimpose pressure or flow, this increases even further. However, the memory requirements for storing color information are significantly less than those for the geometry (about 1%) so that it would not reduce the performance significantly.

Sixty-four bit computers are becoming available, which can be equipped with up to 16 GB of main memory. BIOS and driver limitations, however, often still limit the amount of memory to 3 GB. Whenever the available memory is not sufficient for storing the entire geometry, the hard disk needs to be used as secondary storage which results in significantly slower data access.

In order to alleviate these limiting factors, the number of triangles employed to display the model needs to be reduced resulting in less data that needs to be processed. Suitable techniques include geometry reduction techniques which implies that the geometric model is displayed with a reduced number of triangles in those areas that are far from the view point. In addition to a view-dependent level-of-detail representation based on distance, occlusion-based methods can be used. These techniques allow the identification of areas in the model that contribute little to the final image. This is usually due to the fact that the segments located in those areas are obscured by other segments which are positioned more closely to the view point when projected onto the viewing plane. These occluded areas of the model can then be either eliminated or rendered using a lower resolution and consequently reduce the overall number of triangles.

Tree-shaped geometric structures have certain unique properties that render most traditional occlusion culling algorithms inefficient. For instance, when rendering architectural models or iso-surface representations of objects, occlusion frequently occurs. As an example, if the camera is located inside a room with no windows of an architectural model, the entire outside world is not visible, thus occluded. This is not likely to happen in tree-shaped data sets because the scene consists predominantly of relatively skinny elements which make partial occlusion much more likely than complete occlusion. Consequently, occlusion query techniques can only be used as a measure for visibility indicating the extent to which the precision of the model can be reduced without changing the visual appearance very much. Due to the limited occlusion within such a vascular tree structure, occlusion culling should not be used as a method for completely eliminating parts of the scene because certain areas may still be partially visible in most cases. Therefore, the present system uses OpenGL occlusion queries over the

GL_HP-occlusion-test which allows the system to determine the amount of occlusion.

Comparison with Other Studies

The resulting geometrical representation of the simulated vascular tree consists of millions of vessels. Due to the tremendous complexity of this model, the images shown in the original article[15] were generated using the POVray ray tracer. Those only include vessels down to order five to prevent POVray from being overwhelmed by the complexity of the model. It took about two minutes to generate an image of the reduced vasculature using POVray. In this article, however, all vessels down to the capillaries are shown and less time is needed for creating an image.

The proposed method is based on a real-time, view-dependent simplification of complex models. Several publications exist that employ similar methods. Progressive meshes as introduced by Hoppe[13] were designed to obtain increasingly coarser representations of a mesh by applying edge collapse operations. Applying this method, a level-of-detail description of the model is derived. In one of his later publications, Hoppe[14] describes efficient data structures and algorithms for implementing progressive meshes. Xia et al.[34] defined the notion of a *merge tree* that stores the edge collapse operations in a hierarchical manner to create a continuous-resolution representation of an object. A similar approach was proposed by El-Sana et al.[5] where a binary view-dependence tree is created containing general vertex-pair collapses. This tree can then be used to generate the required triangles for display at run time. Andujar et al.[1] used classical occlusion culling algorithms and computed potentially visible sets (PVS) which consist of those polygons that are likely to be visible. These sets are supersets of the sets of all visible polygons for which the degree of visibility is determined to create view-dependent occlusion culling. Shaffer et al.[31] developed a progressive mesh simplification algorithm which clusters the vertices using a BSP-tree resulting in an adaptive simplification of the polygonal mesh. Pajarola[23] introduces *FastMesh* which defines a hierarchy on half-edges that reduces the storage cost in comparison to vertex hierarchies. El-Sana et al.[6] combine a view-dependence tree with spatial sub-division techniques to avoid scanning of active nodes that do not contribute to the incremental update of the selected level of detail.

Several algorithms for reducing the complexity of a scene using occlusion culling are available both implemented in software and in hardware.[3] Greene[10] developed an algorithm based on hierarchical tiling that is able to determine whether a convex polygon is inside, outside, or intersecting an image hierarchy. Bartz et al.[2] render bounding volumes into a virtual occlusion buffer using OpenGL and read back the results from the graphics hardware to determine occlusion. Since reading back from the OpenGL buffer is slow, an interleaving scheme is applied to speed up read-back. Zhang et al.[39] describe hierarchical image-space occlusion maps for visibility culling. The culling algorithm uses an object-space bounding volume hierarchy and can be implemented using graphics hardware. Klosowski et al.[17] propose a visibility culling algorithm based on *Prioritized Layered Projection (PLP)* that can be implemented using graphics hardware. El-Sana et al.[6] combine the PLP approach with view-dependency resulting in a view-dependent occlusion culling. Yoon et al.[36] use a clustering hierarchy for refining the underlying grid to obtain a level-of-detail representation for arbitrary triangle meshes in addition to hardware occlusion culling. Recent efforts show that current hardware improvements and the usage of a *clustered hierarchy of progressive meshes* can improve rendering speed even further.[37] However, most of the described methods are not suitable for directly reducing the complexity of a model of tree-like anatomical structures, such as the coronary vascular tree.

Different techniques for visualizing vascular structures can be found in the literature. Gerig et al.[9] describe how to derive a skeletal structure from a volumetric image based on hysteriosis thresholding and binary thinning. Hahn et al.[12] employ geometrical primitives, such as truncated cones, to visualize vessels inside the human liver. A similar approach has been taken for the rendering method described in this article. The model is represented by conic cylinders as previously described. Masutani et al.[18] used cylinders aligned to the vessel skeleton to visualize the vasculature. Different radii at branchings resulted in discontinuities when using this method. Felkel et al.[8] reconstructed liver vessels from center line and radius information to supply an augmented reality tool for surgery. Puig et al.[24] developed a system for exploring cerebral blood vessels using a symbolic model with a focus on geometric continuity and on realistic shading. Oeltze et al.[20,22] use convolution surfaces to obtain a smoother representation of blood vessels extracted from CT or MR data.

Deussen et al.[4] use points and lines to represent complex systems of plants as approximation reducing the overall number of triangles compared to their original representation. Gumhold et al.[11] use a splatting approach based on ellipsoids for rendering scientific data sets. The advantage of such a glyph-based approach is the potential of deploying the hardware for rendering. Reina et al.[25] showed this when rendering molecular visualizations of 500,000 particles at 10 fps.

In a comparative study, the present system was compared to an implementation on a high-end visualization server: the Sun Fire V880z visualization server. This server is equipped with 16 GB of main memory allowing the system to store the entire geometric model in main memory. Despite the fact that this server is geared towards optimal rendering performance for large data sets, the overall performance was slower compared to the second test system using an Nvidia Quadro FX 4400. Generating a single projected image took 90 s on the Sun server while the second test system using the present system completed the task after 62 s.

Significance

Virtual models of normal hearts are needed as a physiologic reference. Pathological states can then be studied in relation to changes in model parameters that alter coronary perfusion. With such computational models, researchers can analyze the effects of different treatment options (medical and surgical), and ultimately find rational ways to prevent and treat coronary heart disease. Based on detailed anatomically based models, computational fluid dynamics simulations can yield accurate simulation of blood flow in health and disease. In order to visualize the present anatomically based models that may include future hemodynamic and physiological data, it is essential to have efficient and fast visualization techniques. The present study is the first step in that direction.

Conclusions and Future Work

A rendering system has been described which exploits the tree-shaped topology to increase rendering performance. Due to the nature of tree-shaped structures, hierarchical meshes to obtain different levels of detail can be generated based on the topological structure of the data; i.e., individual segments can be clustered as entities. Geometry reduction techniques as well as occlusion culling enables the system to render each frame four times faster than the standard method that displays the full model directly without simplification methods. For the LCx data set, the number of triangles can be reduced in such a way that the amount of geometric information is small enough to be transferred to the graphics hardware and fast enough to utilize the full performance potential of the hardware.

Using out-of-core techniques, the full model can be displayed even on computer systems equipped with relatively small amount of memory since only 64 MB are sufficient for the algorithm. With a high-end PC system, rendering can be even faster using out-of-core techniques compared to workstations equipped with much more main memory, such as the Sun Fire V880z visualization server.

In the future, ray tracing or ray casting algorithms will be applied to the data set to explore if there are any performance benefits from this approach. Additionally, GPU based methods that render the conic cylinders completely on the GPU might increase performance by reducing the amount of information that needs to be stored in memory and avoids data transfer on the bus systems. Using other types of geometry approximations for the tree segments in combination with hardware based approaches might yield even better performance, such as line primitives.[32]

ACKNOWLEDGMENTS

The authors would like to thank Falko Kuester, UC Irvine, for providing the high-end PC system equipped with an Nvidia Quadro FX 4400. This work was supported in part by the National Institute of Health—National Heart, Lung, and Blood Institute Grant 2 R01 HL055554-06 (GSK), by the National Institute of Mental Health (NIMH) through a subcontract with the Center for Neuroscience at the University of California, Davis (5 P20 MH60975), by the National Partnership for Advanced Computational Infrastructure (NPACI), Interaction Environments (IE) Thrust (10195430 00120410).

APPENDIX

Rendering of Massive Tree-Like Structures

In order to analyze large-scale tree-like structures, appropriate visualization methods are necessary. Whenever the geometry data of such a structure exceed the amount of main memory of the computer, the application of several techniques to both be able to handle the data set as well as improve performance are required. This Appendix provides details about the data format and explains the different techniques that were applied to visualize the data on common desktop computers.

Visualizing Tree-Like Structures

The structure is given as a sequence of consecutive segments where one segment can have multiple daughter vessels (mostly two as bifurcations) as successors, forming a tree-shaped structure with a highly asymmetric branching pattern. Each segment in the tree is characterized by the coordinates and radii of its proximal and distal nodes. This data format is similar to the one provided by commercial software packages, such as *Analyze*.[26,28,29] Since the radii of two consecutive nodes are not necessarily equal, a conic cylinder is defined based on this data resembling each segment. All conic cylinders together then define a representation of the vasculature as prescribed by the model. For a coronary arterial tree, there exist three major branches in the data set representing the RCA, LAD and LCx arterial tree, respectively. Every branch includes a complete set of linked vessel segments from the macrovasculature down to the first capillary bifurcation.

In order to generate a visual representation of the vascular tree, OpenGL and computer graphics techniques are used. In computer graphics, a camera analogy is followed similar to taking a photograph. A virtual camera is placed next to the objects, in this case the arterial tree. The orientation of the camera identifies the view direction; while the view direction combined with the location of the camera define the view. Using this definition of a view, all objects are then projected onto a virtual image plane. One can think of this image plane as the film inside the camera that was just defined. This projected image is then displayed on the computer screen. Consequently, the view as defined here identifies exactly what parts of the objects are displayed on the computer screen.

Figure 8 shows an illustration of this configuration. Since current graphics hardware does not support the display of complex objects; simple, more universal primitives have to be used. Due to its universal nature, triangles are the most common primitives in computer graphics since every complex object can be approximated using a set of triangles. As can be seen in Fig. 8, the conic cylinders that are used to represent the arterial tree are broken down into a series of triangles as well. The triangles are arranged in such a way that they approximate a conic cylinder; i.e., two of the edges of each triangle run along the main direction of the conic cylinder while one edge is parallel to one of the end caps of the cylinders. Hence, circles are computed at the transition between vessel segments in a first step. Since cylindrical arteries are considered, these circles are perfectly round. The circles are connected along each vessel segment using triangles to approximate the conic cylinder. For this, the circles are approximated by a series of points by computing a fixed number of points at equidistant locations along the circle. A set of two points (one on the circle at the one end of the cylinder, the second on the opposite side) are then selected in such a way that they form the closest distance between. This forms the first edge of the first triangle. By connecting the next points following the discretization of the circles in an alternating fashion, triangles are formed that approximate the conic cylinders. Figure 8 illustrates this by showing a sample triangle imposed on one of the conic cylinders.

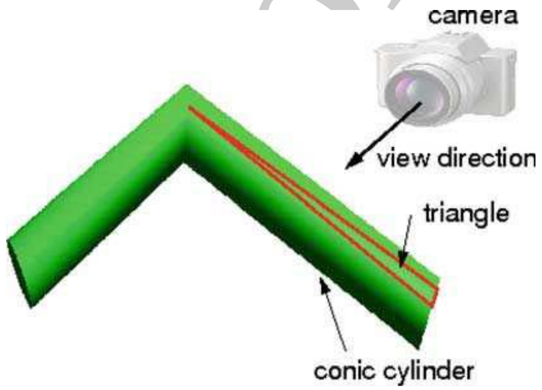


FIGURE 8. Interface between two successive vessel segments shown as conic cylinders which consist of several triangles. Camera and view direction are also shown.

To increase performance, OpenGL provides so-called triangle strips that require less data to be transferred to the graphics hardware for image display. In this case, instead of specifying all three vertices for all triangles, only the vertices for the first triangle need to be specified completely. For the subsquent triangles, only one vertex is specified and the last two vertices of the previous triangle are re-used, forming a new triangle that is directly connected to its predecessor. Since the triangles approximating a conic cylinder (and therefore a vessel segment) are all attached to each other, a triangle strip can be used to reduce the number of vertices that need to be specified. As an additional performance increase, OpenGL vertex arrays are used. Vertex arrays require less function calls and hence can be processed by the hardware more efficiently. To use OpenGL vertex arrays, all vertices for a single triangle strip are stored in a consecutive memory area. This memory area can be passed onto OpenGL in a single function call which results in drawing the entire triangle strip. Hence, an entire vessel segment is represented by one vertex array.

To achieve a smooth transition between consecutive vessel segments, the circle at the end of each cylinder is not necessarily orthogonal to the cylinder itself. Instead, these circles are created in such a way that the plane in which the circle resides divides the angle between the center-lines of the two consecutive conic cylinders into two equal halves (Fig. 8). Using this approach, the center lines of two subsequent segments can form an angle of up to 180 degrees (reverse direction). However, the most common angles in the coronary arterial tree are much smaller. Rotation of the cylinder ends as previously described does not change the way it is approximated by a triangle strip.

For the smallest of the three branches (LCx arterial tree), the number of triangles that is required for visualization of all 1.8-million vessel segments is 25 million triangles. The entire coronary tree consisting of 10 million vessel segments is represented by 220 million triangles based on a discretization of each conic cylinder using sixteen triangles. The conic cylinder representing each segment is approximated by a single triangle strip which is constructed as follows: the two ends of the conic cylinder are discretized using a maximum of eight points. These points are then connected with triangles. Using a point on each end alternately, this results in a triangle strip. At full resolution, this triangle strip consists of 16 triangles rendered by 18 vertices. Due to the size and detail of the data set, the geometry of the vascular structure requires representation of the coordinates of the vertices using 32-bit floating-point numbers. Lower-precision representations result in truncated positions of the vertices and therefore can change the overall geometry significantly. For correct illumination of the geometry, normal vectors are included using another set of three 32-bit floating point values. These normal vectors are required for computing the correct reflection of light on top of the conic cylinders used as geometric representation of the vascular structure. The normal vectors computed for every vertex based on the original geometry (the conic cylinders). This yields significantly better results and achieves an additional depth cue and therefore a more realistic image. In this way, a user is much better able to recognize the 3-D geometry in less time even from a projected 2-D image. Overall, for representing 220 million triangles using vertex coordinates and normal vectors, about 6GB of memory is required in order for the entire geometric representation of the vascular structure to be stored. This entire information needs to be processed for every projected image that is used as visualization of the vascular structure.

Geometry Reduction

In order to increase performance, a common approach is to reduce the amount of geometry information that needs to be processed for a single image. Usually, this is achieved by using a simpler representation and/or removing parts of the data set that is either invisible or only visible to a small extent. Compared to arbitrary triangle meshes, tree-shaped data sets have special topological features that can be taken advantage of to speed up the visualization. First, the connectivity between different segments can be used to simplify the structure by skipping segments. This results in a simpler representation of the data. Secondly, the cylindrical shape of the segments can be used to identify backfacing triangles on a per-segment basis instead of determining this information for each triangle which removes data invisible due to the projection. Since the cylinders are rendered as single triangle strips, the connectivity information can also be exploited when rendering the model; e.g., for backface culling as described later. Different levels of detail can be defined based on the precision at which a conic cylinder is drawn by reducing the number of points for each delimiting circle of the cylinders.

In the current implementation, three levels of detail are used: a full resolution level where each conic cylinder is represented by 16 triangles, a reduced level with 8 triangles per cylinder, and a low level of detail that skips every other segment and renders each remaining cylinder with 8 triangles. Obviously, the low level-of-detail mode should only be used in areas far away from the view point and mostly occluded; i.e., covered by a multitude of other vessel segments and therefore almost invisible. However, since it is almost completely occluded, a user would need to rotate or zoom in order to inspect this part of the vasculature. Once such an area is rotated and therefore more visible, the system would automatically increase the level of detail. Similarly, cracks that occur at the transition between different levels of detail are not noticeable because these transitions occur sufficiently far from the view point and only in at least partly occluded areas.

In order to decide the resolution for a particular segment, one could determine the distance between the current camera position and the segment itself or determine the number of pixels that would be projected onto the screen to represent this segment. However, due to the enormous amount of segments, the computational effort is too costly which would slow down the rendering speed to several seconds per frame even for the LCx data set. In fact, computing the distances between all vessel segments and the camera would take longer than computing the projected image for the entire data set without any reduction techniques.

To remedy the situation, a spatial data structure is used. It is essential to the overall performance of the system that a simple data structure is used which requires only minimal computation. Hence, a simple subdivision scheme of the space covered by the data set is used. This space is equally divided in each dimension into sub-areas of the same size. Then, only the distance between the center of this sub-area and the camera needs to be calculated during the rendering process to determine the level of detail for the whole sub-area. Based on a set of thresholds provided by the user, all segments contained in each sub-area are rendered in full, reduced, or low level of detail, respectively. These thresholds describe the distance between the center of the sub-area and the camera at which the algorithm will automatically switch to a lower geometric resolution. Consequently, these thresholds determine the distance to the camera at which the system switches to a lower resolution. Using this type of geometry reduction, the number of triangles (e.g., the LCx coronary artery) can be reduced from 25 to about 20 million triangles without introducing noticeable artifacts.

Backface Culling

To reduce the number of triangles even further, all triangles that are located at the backside of the conic cylinders facing away from the camera can be removed since they are not visible. OpenGL is able to remove the backfacing triangles but then they still have to be transmitted to and processed by the graphics card. In order to avoid transmitting this amount of information, these triangles can be identified in software on the CPU. Of course, a particular vessel segment can be aligned at virtually any angle to the viewing direction with respect to the first triangle within the triangle strip. Therefore, the set of triangles facing backward can be different for each individual vessel segment. This implies that the computation has to be done for each vessel segment individually. Consequently, these computations have to be carried out in a very efficient way in order to avoid slowing down the rendering process. Again, the fact that the topology of a vessel segment is known can be exploited. Each vessel segment is represented by a conic cylinder. Consequently, usually one half of the cylinder is visible, while the other half is not. The triangles representing the invisible half can be identified using the normal vectors since these are computed in such a way that they are always pointing outwards with respect to the conic cylinder. One approach for identifying those triangles with normals facing away from the view point is to check the normals of every triangle individually. This would represent a computational burden, however, that would slow down the rendering process. Since the vessel segments are rendered as triangle strips, both the visible and the invisible halves are represented by two sets of consecutive triangles. Thus, in order to identify the set of back-facing triangles only, the transition from triangles with normals facing towards the camera and those pointing away from the camera has to be found which is significantly less expensive computationally. Consequently, only the triangles facing the camera which are visible triangles need to be drawn resulting in a significant reduction in the number of triangles that need to be sent to the graphics hardware.

Occlusion-Based Reduction

Another way of reducing the number of triangles required for a geometric representation of the vascular structure is to remove triangles that represent conic cylinders which are hidden behind a multitude of other vessel segments with respect to the current projection (referred to as occlusion). In a tree-shaped data set, complete occlusion is not likely to occur since a single segment does not significantly obstruct the geometry located behind it. Many of the segments need to be co-located and packed very densely in a particular area to occlude other parts of the vascular tree. However, complete occlusion is not likely to occur. Thus, those parts of the tree which are detected as (partly) occluded are still displayed using the lowest level of detail.

The present software system employs occlusion queries implemented in OpenGL 1.5.[30] During such a query, the OpenGL library keeps track of whether the specified graphical primitives result in pixels actually drawn to the projected image. In contrast to the *GL_HP-occlusion_test*, which only returns a binary true or false result depending on whether pixels were drawn or not, the occlusion queries defined in OpenGL 1.5 allow the retrieval of the number of fragments (pixels) that contribute to the current projected image during the query. Assuming frame coherence (two consecutive projected images being similar), we can use these queries to check for occlusion. By drawing a bounding box of a sub-area, these queries allow the software to determine how much of a specific sub-area is visible based on the previous projected image. These sub-areas are identical with the ones defined by the spatial data structure used for the previously described geometry reduction; i.e., an equidistant sub-division into cubical areas. Based on a user-defined threshold describing the number of pixels that need to be visible, the present system can determine the level-of-detail to be used for the vessel segments contained in this sub-area. The smaller the threshold the less vessels are required to occlude a sub-area. Since the vessel segments are spread over the entire volume relatively evenly, a more sophisticated sub-division technique such as binary-space-portioning (BSP) trees or k-d trees, which sub-divide space recursively at arbitrary planes instead of using a fixed scheme, would not result in a significant improvement. Also, a simpler sub-division scheme allows for faster processing of the individual sub-elements during testing for occlusion.

For each element of the sub-division, a hardware occlusion query is issued as described above to check how many fragments pass the depth test; i.e., contribute to the current projected image. This results in an estimate of how much of the specific sub-area contributes to the current projected image. To avoid actual drawing of the bounding boxes, the color mask is set to zero in OpenGL. Similarly, the OpenGL depth buffer is marked as read-only to prevent the bounding boxes from changing the depth values and therefore occlude each other.

Occlusion queries supported on GeForce 3 and subsequent NVidia GPUs allow many queries to be performed simultaneously. Therefore, all bounding boxes that are needed for the occlusion queries are drawn first. The result of each occlusion query is stored in the memory of the graphics hardware. To avoid stalling of the graphics pipeline, the result is read back only once for all sub-areas after all occlusion queries are finished. Each occlusion query returns the number of fragments of the bounding box that would actually pass the depth test and would have been drawn if the color mask would not have been set to zero. Consequently, an occlusion query provides a precise measure of how much of a certain sub-area is occluded. Based on a user-specified threshold, the rendering system can then decide whether to draw the vessel segments contained in that specific area at the full level or at a lower level of detail.

Out-of-Core Rendering

Since the whole data set representing the complete model of the arterial vascular tree does not fit into the main memory of regular desktop computers, an out-of-core method was implemented to support larger data sets. This technique uses hard drives as main storage medium, while main memory is only used for caching data. In this way, only the portion of the data set which the algorithm is currently using needs to be present in main memory. The system automatically updates this portion; i.e., the system loads another portion of the data set into main memory and removes another whenever necessary. This enables rendering of the entire model of the vasculature on current commodity hardware. In this approach, the geometry is determined in a pre-computational step. For each of the spatial sub-areas that are used during the rendering, the triangles needed for displaying all of the vessel segments contained in this specific sub-area are computed and then stored in a file. After that, the geometry data can be removed from main memory. Using such a streaming technique reduces the memory footprint significantly. In our experiments the memory consumption of the software implementation was less than 64 MB (as observed via the Windows task manager).

The geometric representation is pre-computed and written to a file in form of binary arrays in the same way it is used by OpenGL when rendered using vertex arrays. Note that only the full resolution needs to be stored in the out-of-core file since the lower levels-of-detail can be derived by masking elements within the vertex array. Offsets are stored at the beginning of the binary file as depicted by Fig. 9. This section of the file allows the system to determine where to find all vertex arrays within the file for each of the sub-areas used by the spatial data structure. With this information, the exact location within the file can be determined and mapped to memory resulting in a pointer to the base address of all vertex arrays representing the geometry of all vessel segments within a specific sub-area. From this starting point, all vertex arrays can be processed as if they are stored in memory.

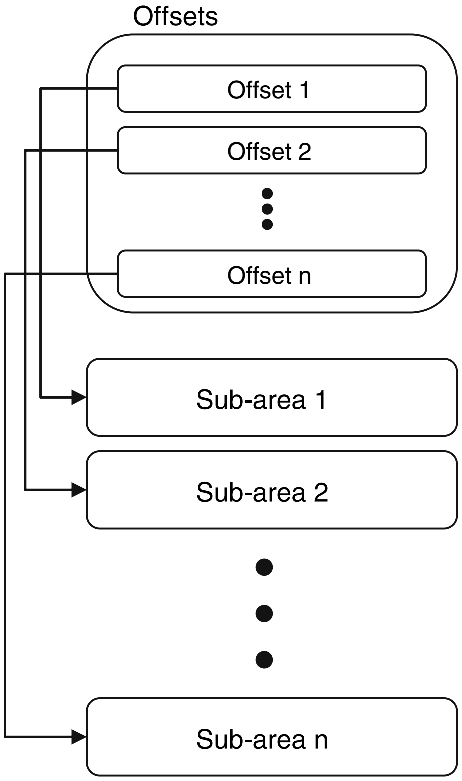
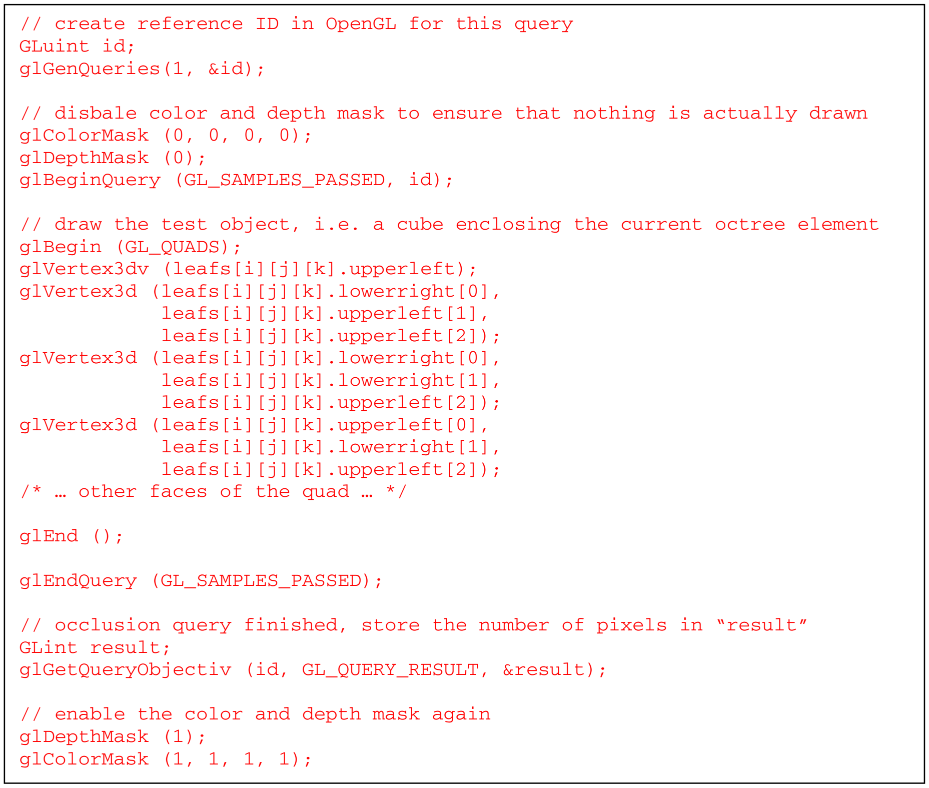


FIGURE 9. Out-of-core data file structure reflecting each sub-area within the spatial data structure and their offsets.

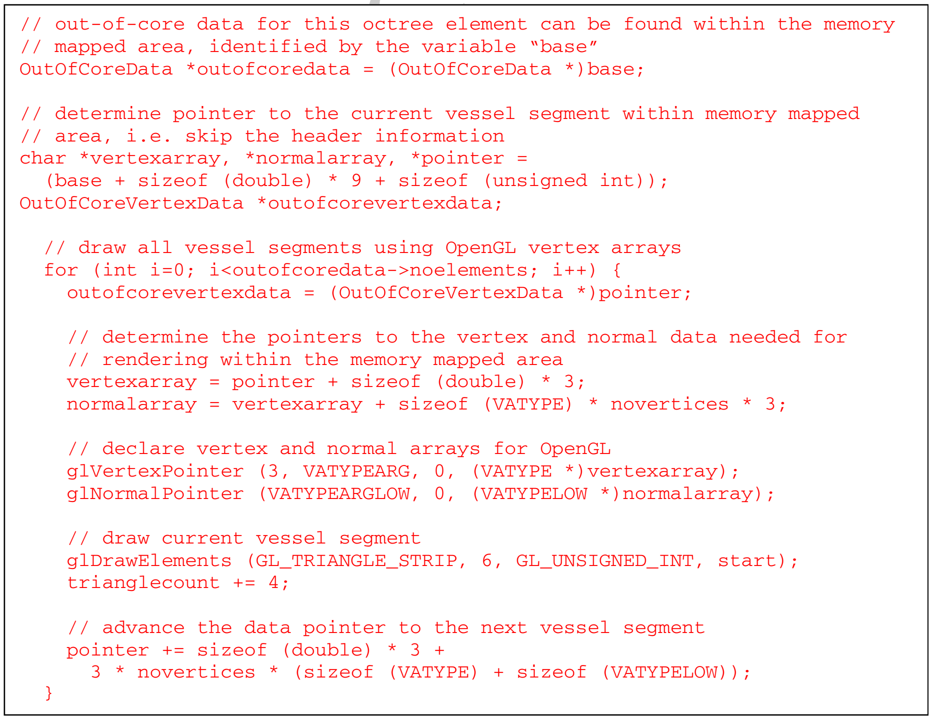
Implementation Details

The visualization system is based on OpenGL. Occlusion queries as defined in OpenGL 1.5 are used for the occlusion culling. Figure 10 shows sample code to use occlusion queries in this context. Similar to display lists, vertex arrays, which are significantly faster than immediate mode rendering,[34] are used for rendering the vessel segments. OpenGL allows selecting the vertices of a vertex array that are used during the rendering. Thus, a lower level of detail can be realized simply by providing OpenGL with a subset of indices that represents a lower level of detail. This index array can be pre-computed and then provided for every segment destined for lower resolution. Obviously, this index array is the same for each segment due to the fact that the conic cylinder representing a single segment is discretized in the same way for each segment. Figure 11 includes the source code used for rendering vessel segments using vertex arrays.



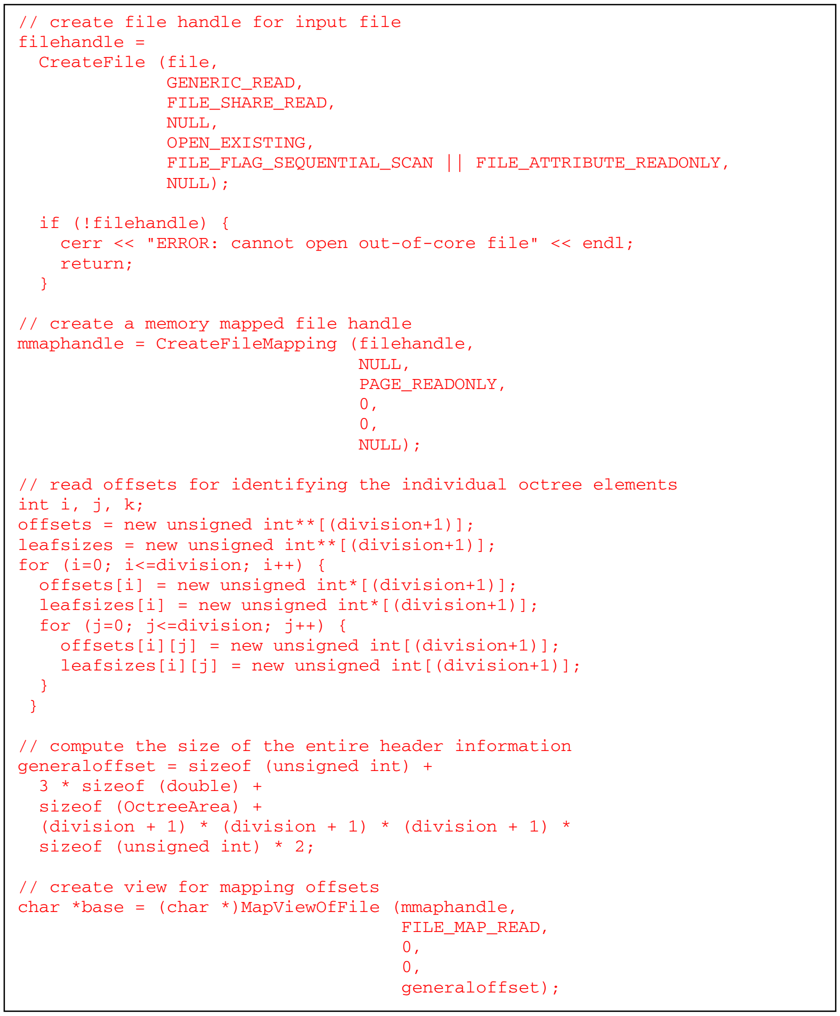
```
// create reference ID in OpenGL for this query
GLuint id;
glGenQueries(1, &id);

// disbale color and depth mask to ensure that nothing is actually drawn
glColorMask (0, 0, 0, 0);
glDepthMask (0);
glBeginQuery (GL_SAMPLES_PASSED, id);

// draw the test object, i.e. a cube enclosing the current octree element
glBegin (GL_QUADS);
glVertex3dv (leafs[i][j][k].upperleft);
glVertex3d (leafs[i][j][k].lowerright[0],
            leafs[i][j][k].upperleft[1],
            leafs[i][j][k].upperleft[2]);
glVertex3d (leafs[i][j][k].lowerright[0],
            leafs[i][j][k].lowerright[1],
            leafs[i][j][k].upperleft[2]);
glVertex3d (leafs[i][j][k].upperleft[0],
            leafs[i][j][k].lowerright[1],
            leafs[i][j][k].upperleft[2]);
/* … other faces of the quad … */

glEnd ();

glEndQuery (GL_SAMPLES_PASSED);

// occlusion query finished, store the number of pixels in “result”
GLint result;
glGetQueryObjectiv (id, GL_QUERY_RESULT, &result);

// enable the color and depth mask again
glDepthMask (1);
glColorMask (1, 1, 1, 1);
```

FIGURE 10. Sample code for OpenGL occlusion query to determine the number of fragments (pixels) that would contribute to the current image when drawing all vessel segments contained in a single octree element. The variable result will contain the exact number of pixels that would be drawn for the bounding box of this specific octree element (for the sake of simplicity only the drawing commands for the first face of the bounding box is shown).



```
// out-of-core data for this octree element can be found within the memory
// mapped area, identified by the variable “base”
OutOfCoreData *outofcoredata = (OutOfCoreData *)base;

// determine pointer to the current vessel segment within memory mapped
// area, i.e. skip the header information
char *vertexarray, *normalarray, *pointer =
  (base + sizeof (double) * 9 + sizeof (unsigned int));
OutOfCoreVertexData *outofcorevertexdata;

  // draw all vessel segments using OpenGL vertex arrays
  for (int i=0; i<outofcoredata->noelements; i++) {
    outofcorevertexdata = (OutOfCoreVertexData *)pointer;

    // determine the pointers to the vertex and normal data needed for
    // rendering within the memory mapped area
    vertexarray = pointer + sizeof (double) * 3;
    normalarray = vertexarray + sizeof (VATYPE) * novertices * 3;

    // declare vertex and normal arrays for OpenGL
    glVertexPointer (3, VATYPEARG, 0, (VATYPE *)vertexarray);
    glNormalPointer (VATYPEARGLOW, 0, (VATYPELOW *)normalarray);

    // draw current vessel segment
    glDrawElements (GL_TRIANGLE_STRIP, 6, GL_UNSIGNED_INT, start);
    trianglecount += 4;

    // advance the data pointer to the next vessel segment
    pointer += sizeof (double) * 3 +
      3 * novertices * (sizeof (VATYPE) + sizeof (VATYPELOW));
  }
```

FIGURE 11. Sample code for drawing all vessel segments within an octree element using OpenGL vertex arrays; the geometry data is retrieved from the memory mapped file. The starting location of the memory mapped area is indicated by the variable ‘‘base’’.



```
// create file handle for input file
filehandle =
  CreateFile (file,
              GENERIC_READ,
              FILE_SHARE_READ,
              NULL,
              OPEN_EXISTING,
              FILE_FLAG_SEQUENTIAL_SCAN || FILE_ATTRIBUTE_READONLY,
              NULL);

  if (!filehandle) {
    cerr << "ERROR: cannot open out-of-core file" << endl;
    return;
  }

// create a memory mapped file handle
mmaphandle = CreateFileMapping (filehandle,
                                NULL,
                                PAGE_READONLY,
                                0,
                                0,
                                NULL);

// read offsets for identifying the individual octree elements
int i, j, k;
offsets = new unsigned int**[(division+1)];
leafsizes = new unsigned int**[(division+1)];
for (i=0; i<=division; i++) {
  offsets[i] = new unsigned int*[(division+1)];
  leafsizes[i] = new unsigned int*[(division+1)];
  for (j=0; j<=division; j++) {
    offsets[i][j] = new unsigned int[(division+1)];
    leafsizes[i][j] = new unsigned int[(division+1)];
  }
 }

// compute the size of the entire header information
generaloffset = sizeof (unsigned int) +
  3 * sizeof (double) +
  sizeof (OctreeArea) +
  (division + 1) * (division + 1) * (division + 1) *
  sizeof (unsigned int) * 2;

// create view for mapping offsets
char *base = (char *)MapViewOfFile (mmaphandle,
                                    FILE_MAP_READ,
                                    0,
                                    0,
                                    generaloffset);
```

FIGURE 12. Memory mapped reading of the header information from the out-of-core file; similarly the geometric information is read from the out-of-core file for every octree element.

The out-of-core rendering approach uses a single file that contains the pre-computed geometry describing the model. This file is accessed using memory mapping implemented in the Windows™ and Linux operating systems. Figure 12 outlines the necessary function calls for memory mapping the geometric representation of the vasculature for the Windows™ operating system. This has two major advantages. First, the file can be randomly accessed. File caching is handled entirely by the operating system. This approach utilizes the hardware capabilities of the memory management unit (MMU) within the CPU. Second, due to the file caching capabilities of the operating system, close-up views can be rendered at comparatively high frame rates. If the geometry that is needed for a close-up view fits into the file cache of the computer, no hard disk access is necessary making the rendering relatively fast. The out-of-core rendering mode has the advantage of handling data sets larger than the available main memory space.

REFERENCES

[1]Andujar, C., C. Saona-Vazquez, I. Navazo and P. Brunet. Integrating occlusion culling and levels of details through hardly-visible sets. *Computer Graphics Forum* 19(3), 2000.

[2]Bartz, D., M. Meißner, and T. Hüttner OpenGL-assisted occlusion culling for large polygonal models. *Computers Graphics* 23(5):667–679, 1999.

[3]Cohen-Or, D., Y. Chrysanthou, C. T. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Trans. Visualization Computer Graphics* 9(3):412–431, 2003.

[4]Deussen, O., C. Colditz, M. Stamminger, and G. Drettakis Interactive Visualization of Complex Plant Ecosystems. *IEEE Visualization* 2002:219–226, 2002.

[5]El-Sana, J and A. Varshney. Generalized view-dependent simplification. In P. Brunet & R. Scopigno (eds.) Computer

Graphics Forum (Eurographics ,99), Vol. 18(3). The Eurographics Association and Blackwell Publishers: 83–94, 1999.

[6] El-Sana J., Sokolovsky, C. T. Silva. 2001 Integrated occlusion culling with view-dependent rendering. IEEE Visualization 2001:371–378.

[7] El-Sana, J. and E. Bachmat Optimized view-dependent rendering for large polygonial datasets. *IEEE Visualization* 2002:77–84, 2002: IEEE Computer Society.

[8] Felkel, P, A. L. Fuhrmann, A. Kanitasar, and R. Wegenkittel. Surface reconstruction of the branching vessels for augmented reality aided surgery, BIOSIGNAL, Vol. 1, VUTIUM Press: 252–254, 2002.

[9] Gerig, G., T. Koller, G. Széhely, C. Brechbühler, O. Kübler. Symbolic description of 3-D structures applied to cerebral vessel tree obtained from MR angiography volume data, Information Processing in Medical Imaging, Springer, LNCS: 94–111, 1993.

[10] Greene, N. Hierarchical polygon tiling with coverage masks. *ACM SIGGRAPH* 1996:65–74, 1996.

[11] Gumhold, S. Splatting Illuminated Ellipsoids with Depth Correction. In Proceedings of 8th International Fall Workshop on Vision, Modelling and Visualization 2003: 245–252, 2003.

[12] Hahn, H. K., B. Preim, D. Selle, and H. O. Peitgen Visualization and interaction techniques for the exploration of vascular structures. *In IEEE Visualization* 2001:395–402, 2001: IEEE, IEEE Computer Society Press.

[13] Hoppe, H. Progressive meshes. *ACM SIGGRAPH* 1996:99–108, 1996.

[14] Hoppe, H. Efficient implementation of progressive meshes. *Computers Graphics* 22(1):27–36, 1998: ISSN 0097–8493.

[15] Kaimovitz, B., Y. Lanir, and G. S. Kassab Large-Scale 3-D Geometric Reconstruction of the Porcine Coronary Arterial Vasculature Based on Detailed Anatomical Data. *Ann. Biomed. Eng*. 33(11):1517–1535, 2005.

[16] Kassab, G. S., C. A. Rider, N. J. Tang, and Y. C. Fung Morphometry of pig coronary arterial trees. *Am. J. Physiol.* 265(Heart Circ. Physiol. 34):H350–H365, 1993.

[17] Klosowski, J. T. and C. T. Silva Efficient conservative visibility culling using the prioritized-layered projection algorithm. *IEEE Trans. Visualization Computer Graphics* 7(4):365–379, 2001.

[18] Linsen, L., B. J. Karis, E. G. McPherson, and B. Hamann Tree Growth Visualization. *J. WSCG* 13:81–88, 2005.

[19] Masutani, Y., K. Masamune and T. Dohi. Region-growing-based feature extraction algorithm for tree-like objects. Visualization in Biomedical Computing, Springer, LNCS: 161–171, 1996.

[20] Nordsletten, D. A., S. Blackett, M. D. Bentley, E. L. Ritman, and N. P. Smith Structural Morphology of Renal Vasculature. *Am. J. Physiol. Heart Circ. Physiol.* 291:H296–H309, 2006.

[21] Oeltze, S., B. Preim. Visualization of Anatomic Tree Structures with Convolution Surfaces, IEEE/Eurographics Symposium on Visualization: 311–320, 2004.

[22] Oeltze, S. and B. Preim Visualization of Vasculature with Convolution Surfaces: Method, Validation and Evaluation. *IEEE Trans. Medical Imaging* 24(4):540–548, 2005.

[23] Pajarola, R. FastMesh: efficient view-dependent meshing. In B. Werner (ed.) Proceedings of the nineth Pacific Conference on Computer Graphics and Applications (PACIFIC GRAPHICS-01). IEEE Computer Society, Los Alamitos, CA, Oct. 16–18: 22–30, 2001.

[24] Puig, A., D. Tost, and I. Navazo An interactive celebral blood vessel exploration system. *IEEE Visualization* 97:443–446, 1997.

[25] Reina, G. and T. Ertl. Hardware-Accelerated Glyphs for Mono- and Dipoles in Molecular Dynamics Visualization, Proceedings of EUROGRAPHICS – IEEE VGTC Symposium on Visualization 2005: 177–182, 2005.

[26] Ritman, E. L. Micro-computed tomography-current status and developments. *Annu. Rev. Biomed. Eng*. 6:185–2082004.

[27] Robb, R. A. The Biomedical Imaging Resource at Mayo Clinic. Guest Editorial. *IEEE Trans. Med. Imaging* 20(9):854–867, 2001.

[28] Robb, R. A. and C. Barillot Interactive display and analysis of 3-D medical images. *IEEE Trans Med Imaging* 8(3):217–226, 1989.

[29] Robb, R. A., D. Hanson, R. A. Karwoski, A. G. Larson, E. L. Workman, and M. C. Stacy ANALYZE: a comprehensive, operator-interactive software package for multidimensional medical image display and analysis. *Computerized Med Imaging Graphics* 13:433–454, 1989.

[30] Segal, M. andK. Akeley. The OpenGL® Graphics System: A Specification (Version 2.0–October 22, 2004), http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf.

[31] Shaffer, E. and M. Garland. Efficient Adaptive Simplification of Massive Meshes. In IEEE Visualization 2001: 127–134, 2001.

[32] Spaan, J. A., R. ter Wee, J. W. van Teeffelen, G. Streekstra, M. Siebes, C. Kolyva, H. Vink, D. S. Fokkema, and E. VanBavel. Visualization of intramural coronary vasculature by an imaging cryomicrotome suggests compartmentalization of myocardial perfusion areas. *Med. Biol. Eng. Comput*. 43(4):431–435, 2005.

[33] Stoll, C., S. Gumhold, and H. Seidel Visualization with stylized line primitives. *IEEE Visualization* 2005:695–702, 2005.

[34] Woo, M., J. Neider, and T. Davis. OpenGL Programming Guide. Addison Wesley, 3rd edn, 2003.

[35] Xia, J., J. El-Sana, and A. Varshney Adaptive real-time level-of-detail-based rendering for polygonial models. *IEEE Trans. Visualization Computer Graphics* 3(2):171–183, 1997.

[36] Yoon, S. E., B. Salomon, and D. Manocha. Interactive view-dependent rendering with conservative occlusion culling in complex environments. IEEE Visualization 2003 Proceedings: 163–170, 2003.

[37] Yoon, S. E., B. Salomon, R. Gayle, and D. Manocha. Quick-VDR: Interactive View-Dependent Rendering of Massive Models. In IEEE Visualization 2004 Proceedings: 131–138, 2004.

[38] Zamir, M. Nonsymmetrical bifurcations in arterial branching. J. General Physiol. 72(6):837–845, 1978.

[39] Zhang, H., D. Manocha, T. Hudson, and K. E. Hoff III Visibility culling using hierarchical occlusion maps. *Computer Graphics* 31(Annual Conference Series):77–88, 1997.