Published in final edited form as:
Conf Proc IEEE Eng Med Biol Soc. 2014 ; 2014: 3308–3312. doi:10.1109/EMBC.2014.6944330.

GPU Based Multi-histogram Volume Navigation for Virtual Bronchoscopy

Ruida Cheng,
Image Science Laboratory, Center of Information Technology, National Institutes of Health, Bethesda, MD

Sheng Xu,
Interventional Radiology Section, NIH Center for Interventional Oncology, Clinical Center, Bethesda, MD

Alexandra Bokinsky,
Geometric Tools, Inc, Chapel Hill, NC

Evan McCreedy,
Image Science Laboratory, Center of Information Technology, National Institutes of Health, Bethesda, MD

William Gandler,
Image Science Laboratory, Center of Information Technology, National Institutes of Health, Bethesda, MD

Bradford J. Wood, and
Interventional Radiology Section, NIH Center for Interventional Oncology, Clinical Center, Bethesda, MD

Matthew J. McAuliffe
Image Science Laboratory, Center of Information Technology, National Institutes of Health, Bethesda, MD

Abstract

An interactive navigation system for virtual bronchoscopy is presented, which is based solely on GPU based high performance multi-histogram volume rendering.

I. Introduction

Virtual endoscopy has become an increasingly important tool for training, diagnosis, pre-operative planning and intra-procedural image guidance. In virtual bronchoscopy navigation, real-time video images of the bronchoscope are compared to the virtual images generated from a diagnostic CT volume to locate the bronchoscope relative to the patient's airways and pre-procedural CT. Current clinical methods for registration used in bronchoscopy procedures include video or image based as well as electromagnetic tracking based solutions

Ruida Cheng, ruida@mail.nih.gov, tel: (301) 496-5363.

[1]. Various registration algorithms have been developed to automatically register real and CT-derived virtual bronchoscopy images [2,3]. These algorithms require rendering virtual bronchoscopy images at different locations and viewing angles to find the best match of the real image. Since the registration is carried out during the clinical procedure, fast virtual bronchoscopy rendering is highly desirable for less latency and more real time procedural adjustment. Current endoscopy navigation systems either strive for interactive rendering of an iso-surface from the polygonal representations, or high-quality volume rendering that has to be generated from a 3D image. While surface-based virtual bronchoscopy is very fast, it relies on an accurate segmentation of the airways. However, semi or automatic airway segmentations are not reliable when the airways are small or the resolution of the diagnostic CT image is low [4, 5]. As an alternative, volume rendering can be used when airway segmentation is not accurate. Even if volume rendering is ambiguous in small airways due to volume averaging, the clinician may use his or her knowledge to effectively locate the position of the bronchoscope. The registration algorithms may still find the best match between the real and virtual bronchoscopy images.

Intensive research has been conducted in developing fast volume rendering based endoscopy navigation. Wan et al. [6] proposed a fast direct volume rendering technique that exploits distance information stored in a potential field of a camera control model, and is parallelized on a multiprocessor. The proposed method is CPU based, and the navigation system requires a high-end workstation to run, such as the SGI power challenge with 16 processors. Kruger et al. [7] proposed a GPU raycasting method that combines volume rendering with iso-surface rendering. On the pixel shader, a first pass performs single pass ray traversal and early ray termination on the surface to gather the depth and density info for each pixel. Then a second pass performs composition and shading to get the final image. Yuan [8] proposed a fast GPU raycasting algorithm to improve image quality. A linear interpolation is used to estimate the intersection between a ray and iso-surfaces in order to reduce resampling artifacts. Diepenbrock et al. [9] proposed a GPU based volume raycaster that generates two proxy geometries, a cube one for rendering, and a sphere one to map the user’s input to camera movement. An image analyzer overlays the two for final rendering. Limited research has been dedicated to support pure volume rendering based navigation without 3D surface rendering. Wan’s work is pure volume rendering, however, it is CPU based. All the above GPU based navigation systems constrain with 3D surface rendering to achieve high rendering quality.

Debate exists regarding the merits of surface rendering versus volume rendering [10, 11]. One advantage of iso-surface based volume rendering is the realistic lighting and shading effects, which enhance the perception of spatial relations of anatomic structure. One major limitation of this approach is segmentation. For example, in the context of virtual bronchoscopy, small airways are extremely easy to miss with semi-manual based human segmentation. Those small branches might be the diagnosed as highly suspicious lesion. In the pure volume rendering based navigation environment, an appropriate adjusted transfer function is desired for viewing the fine details of the small airways. Though interactive direct volume representations would be highly desirable, no system has yet been presented that is capable of delivering sufficient quality with a truly interactive frame rate [12].

To overcome the inherent problems of the above approaches in a pure volumetric rendering context, we developed a bronchoscopy navigation system that satisfies the following requirements: 1) Pure volumetric view without any 3D iso-surface rendering to minimize the human based segmentation errors; 2) Intuitive mouse centric navigation control that is capable of steering the surrounding anatomic structure from the viewpoint; 3) Path planning with automatic fly through; 4) Achieve high rendering quality with interactive frame rate. With those simple requirements in mind, we developed a prototypical GPU based multi-histogram volume navigation system.

The proposed GPU navigation system is built on top of the MIPAV [13] visualization framework. The framework utilizes the WildMagic [14] game engine to handle the rendering. In general, 3D camera control in computer games has only limited use in the area of volume rendering [9] because the game engine based rendering pipeline leads to a substantial problem with volume navigation on the GPU. We developed a work around method to solve the problem, which is presented as the following: 1) overview of the GPU based rendering architecture; 2) explanation of the bottleneck GPU rendering constraint in the context of volume navigation; 3)the work around method;4) basic functionalities of a prototype bronchoscopy navigation system.

Methods

A. Old GPU-based Multi-histogram Rendering Framework

The previous published paper [14] introduced MIPAV visualization framework on top of the WildMagic game engine. WildMagic is a C++ game engine [15] that provides the low-level geometry routines and complete high-level scene graph management for game applications. We ported the WildMagic library to Java specifically for the shader-based rendering pipeline and shader-effects library. WildMagic is optimized for fast and efficient use of GPU memory by sharing texture and shader data.

Figure 1 depicts the layered MIPAV visualization framework. The WildMagic library is layered between MIPAV and the Jogl library to implement ray-cast based volume rendering. GPU based GLSL shaders produce very efficient rendering results through the use of extensive graphics hardware optimization.

In later work [16], we integrated the GPU-based multi-histogram rendering into the MIPAV visualization framework. The multi-histogram rendering framework proposed by Kniss et al. [17] gives the volumetric data the 3D impression of overlaid multiple shapes (iso-surfaces). Multi-histogram rendering gives greater emphasis to boundary visualization than does the traditional one dimensional transfer function based volume rendering. Figure 2 depicts the rendering pipeline.

In the preprocessing stage, the gradient magnitude, Laplacian volume, and the volume normal are calculated by OpenCL routines on the GPU. The WildMagic library renders the 3D volume proxy geometry to sample the volume data. Each time the proxy geometry is rendered, the pixel shader calculates the current location in the volume to sample, renders the volume data with gradient magnitude and Laplacian values, and calculates the output color and opacity. If more than one 2D multi-histogram widget is applied, the values are combined before being written to the framebuffer. Each rendering pass is blended with alpha blending in the framebuffer.

The proxy geometry is rendered as a color cube, which is sized to fit the volume data. At the start of rendering, the back-facing sides of the cube are rendered, where back face color is determined by the x, y, z location on the rendered face. This produces a 2D texture as seen in Figure 2. Next the proxy geometry is rendered again, this time rendering the front-facing sides of the cube. When the front faces are rendered, the 2D back-facing texture is passed to the pixel shader program. The pixel shader program uses the positions on the front-face of the cube and the positions of the back-side of the cube, which it reads from the 2D texture, to calculate a view-direction along which to sample the volume data. During each pass the pixel shader reads the volume data, gradient magnitude, Laplacian, and normal map from the 3D textures, and computes the color and opacity from the multi-histogram transfer function. Blinn-Phong illuminations model is used to compute the shading at each voxel. The pixel shader program calculates a single location along the ray to render, and multiple rendering passes are used to generate the final image in the framebuffer. Thus, ray-casting is realized in sense of multi-pass, which utilizes the parallel GPU processing power of the modern graphics hardware.

B. Multi-histogram framework rendering constraint in the context of navigation

Due to historic reasons, the MIPAV visualization framework relies heavily on the WildMagic game engine architecture to handle the GPU based volume rendering. The proxy geometry, the color cube, is the key to rendering the volume.

One typical step with the traditional WildMagic game engine rendering pipeline is described in Figure 3.

From the model space, the image scene graph tree contains all the 3D geometry objects in the game scene. View frustum culling goes through each 3D geometry surface, eliminating portions of the object, possibly the entire geometry object that is outside the view frustum. Then the remaining geometry figures in the image scene graph are projected into the homogeneous canonical view volume (CVV) for rasterizing. Each geometry object is attached with its own shader for GPU surface rendering. To navigate into the game scene, the camera simply moves around inside the game environment, viewing the closest 3D geometry figures that are visible inside the view frustum. Frustum culling is performed at each frame of the rendering loop. One point needs to mention is the clipping. Clipping against the view frustum tests the intersection of the front-facing object’s triangle meshes with the view frustum clipping planes. The portions outside will be clipped away. The remaining parts will regenerate the triangle meshes for the new 3D objects. In the context of game based navigation, clipping against the 3D geometry objects turns out to be unnecessary since the game player never needs to navigate inside the 3D figures. Occasionally, a hollow 3D surface shows up when the player moves the camera into the 3D object. Clipping against the 3D figure to regenerate the triangle meshes turns out to be the performance overhead in the gaming environment. Therefore, culling without clipping is a general operation in the game engine architecture.

We build the multi-histogram rendering framework based on top of the WildMagic game engine. The culling without clipping scenario introduces a substantial design pitfall in the context of volume navigation. Figure 4 illustrates the problem.

In the context of volume navigation, the proxy geometry is the only geometry object under the image scene graph tree. From the old multi-histogram rendering framework, rendering works smoothly when the camera is far from the volume proxy geometry since the geometry resides entirely inside the view frustum. To navigate into the 3D volume, we move the camera close to the volume, as shown in Figure 4 b,c. The volume data is rendered with a pixel shader on the GPU. The shader is activated by rendering the volume proxy geometry. One problem with this technique is that the WildMagic library culls off the bounding-volume proxy geometry cube with the view-frustum near plane before rasterizing. The front-face and back-face of the proxy geometry are no longer rendered on the GPU, and therefore the fragment shader is no longer activated. This causes the whole volume rendering to disappear as the user tries to navigate inside or zoom into the volume. This pitfall from the game engine architecture design becomes the bottleneck in the context of volume navigation with the GPU based rendering framework.

C. Work around method in the context of navigation

One work around method we developed is to automatically detect clipping of the volume proxy geometry by the view frustum near plane as the camera moves into the volume. When clipping is detected, the algorithm generates a new triangle mesh by intersecting the near plane with the original proxy bounding cube. The newly generated triangle mesh serves as the new volume proxy geometry to pass down to the shader for rasterizing. The resulting effect is an appearance of navigating into the volume. Before rasterizing, the plane-cube intersection and re-meshing are computed for each frame inside the CPU based rendering loop. We follow the references [18, 19] to implement the algorithm. Figure 5 demonstrates the possible clipping configurations of the near-plane and the proxy geometry cube.

The proxy cube and the view frustum intersection test is the key to solving the clipping problem. As illustrated in Figure 5, the intersection between a cube and the near plane generates an irregular polyhedron, which contains at most 6 vertices. The intersection test algorithm is outlined as follows. When the near plane intersects an edge of the cube, the plane is given in Hessian normal form,

$$n \cdot x = d \qquad (1)$$

where n is the unit normal vector of the plane and d the length of the normal vector. An edge $E_{i \to j}$ between two vertices V_i and V_j of the cube can be described by the straight line,

$$E_{i \to j} = V_i + \lambda(V_j - V_i) \quad \text{with } 0 \le \lambda \le 1 \qquad (2)$$

The intersection point between the near plane and the straight line spanned by edge $E_{i \to j}$ is calculated as,

$$\lambda = \frac{d - n \cdot V_i}{n \cdot (V_j - V_i)} \quad (3)$$

The near plane intersects the edge of cube when $\lambda \in [0, 1]$, otherwise there is no intersection. A key point in performing the plane-edge intersection calculation is to maintain a clockwise or counter-clockwise ordering of the intersection points such that the final result forms a valid polygon. Figure 6 shows the remapping of the points on the cube so that the nearest point is V_0, and is on the cube top-face. This corner is clipped by the view frustum and will serve as the starting corner in creating the new mesh surface. Vertex V_7 is the furthest vertex lying on the opposite corner of the cube back-face. There are three independent paths from V_0 to V_7 as marked in Figure 6a by the solid lines with different shaded gray colors. Each path consists of a sequence of three edges $\{E_1, E_2, E_3\}$, such as, $E_1 = E_{0 \to 1}$, $E_2 = E_{1 \to 4}$ and $E_3 = E_{4 \to 7}$ for the light gray path. Any viewport-aligned clipping plane that intersects the cube will have exactly one unique intersection point along each of the three paths.

If the intersection polygon has only three vertices, there are exactly three intersections points with the three paths. We can compute three intersection points P_i by checking intersections with a sequence of edges, respectively, as shown in Figure 6b.

$P_0 =$ Intersecting with $E_{0 \to 1}$ or $E_{1 \to 4}$ or $E_{4 \to 7}$

$P_2 =$ Intersecting with $E_{0 \to 2}$ or $E_{2 \to 5}$ or $E_{5 \to 7}$

$P_4 =$ Intersecting with $E_{0 \to 3}$ or $E_{3 \to 6}$ or $E_{6 \to 7}$

In case that the intersection polygon has more than three vertices, we need to insert new points at the dotted lines. For example, insert a new point P_1 at the light gray dotted line $E_{1 \to 5}$. As a consequence, there must be two consecutive edges linking to P_1, and the two points reside either on the solid light gray path or the solid mid gray path as shown in Figure 6 c,d. If the intersection with the dotted line doesn't exist, we just set the point to P_0, which is on the light gray path. The intersection points with the dotted lines are expressed as:

P_1 = Intersection with $E_{1 \to 5}$, otherwise P_0

P_3 = Intersection with $E_{2 \to 6}$, otherwise P_2

P_5 = Intersection with $E_{3 \to 4}$, otherwise P_4

Following the above idea, we find all the six intersection points of the near plane with the proxy bounding cube in a sequence that forms a valid polygon while simultaneously re-meshing the clipping plane. The newly generated proxy geometry is passed down to the shader for GPU based rendering. Clipping the proxy cube with the near plane solves the problem of the volume navigation with the general game engine based rendering pipeline. After applying the clipping algorithm, users are able to navigate into the volume.

D. Prototype Virtual Bronchoscopy Navigation

By solving the clipping problem with the general game engine rendering architecture, we build a prototype volume navigation system for the virtual bronchoscopy based on the GPU enhanced multi-histogram rendering framework. Multi-histogram is a very effective way to visualize material boundaries from the complicated anatomy structure without rendering the 3D surfaces. It avoids the tedious semi-manual human segmentation with the large dataset, i.e. 512×512×256. Our motivation for the multi-histogram is to view the tiny branch structures along the bronchial airway during the navigation. Those small airways can easily be neglected with the human based segmentation

To unleash the full potential of the volume navigation, we implement two control modes: one with mouse centric control to steer and fly into the volume; the other is the path planning based automatic fly-thru. Users can easily switch between the two modes with a single button click.

In the mode of mouse centric control, press and hold down the left mouse button with the control key to pick a target point as the flying direction. Then rotate the mouse wheel forward once to fly toward the target direction or rotate it backward once to fly in the opposite direction. Fly through is implemented with Java thread control. The flying motion continues until the user presses the mouse again to stop at the current location. This is necessary when the user needs to switch between branches during navigation. Flying is realized by moving the camera at a constant speed. Features with the camera centric navigation metaphor are also implemented. The following key and mouse button combinations affect the view orientation and direction: 1) Ctrl + right mouse button – Counterclockwise roll rotation; 2) Alt + right mouse button – Clockwise roll rotation; 3) Shift + right mouse drag – Move the camera location horizontally or vertically, independent of the volume; 4) Shift + left mouse drag – Yaw, pitch rotation around the volume with camera centric location. The camera centric steering features are convenient for observing the small airways in bronchoscopy. Figure 7 shows a mouse centric control view. The upper panel shows the virtual bronchoscopy volumetric view. The lower three panel cross-hairs show the current virtual camera location.

In the mode of planning automatic based fly-thru, the user selects the annotation points to create the tracking path. A point of interest is captured by clicking the left mouse button on one of the bottom three planar panels while holding down the shift key. After a new point has been captured, it is added along with the current location to a list of other captured points, and then a small green sphere is rendered as the selected point of interest. When user clicks the generate path button, a B-Spline smoothed path is created. Figure 8 shows a tracking path in red color with the tri-planar view. Basically, the user creates a tracking path starting from the lesion region as the initial anchor point, and traces back to the entry point of the bronchus structure. To fly-thru, the user simply rotates the mouse wheel forward once to fly along the B-Spline path, or backward once to fly back. Figure 9 shows the volumetric view of path planning fly-thru, and the corresponding path hidden behind the volume scene.

III. Result

To conduct the performance measure, we evaluate the frame per second (fps) with the path planning automatic fly-thru mode. Frames rates were captured during the fly-thru with fine tuning the sampling rate. Measurements were conducted on a general desktop PC equipped with 8 GB RAM and an Nvidia GeForce GTX 550 Ti card with 1.0 GB texture memory. Table 1 shows the figures with different CT datasets. The GPU base multi-histogram volume navigation can achieve an acceptable interactive frame rate with smoother visual rendering quality.

IV. Conclusion

In virtual bronchoscopy simulation, it is important to develop a navigation method that enables users to control the position and orientation of virtual camera easily and intuitively. We propose a prototype navigation system that utilizes the GPU based multi-histogram to render a clear bronchial airway in a pure volumetric context, without any 3D surface rendering involved. The proposed system overcomes the GPU volume navigation constraint [9] imposed by the traditional game engine based medical simulation architecture. It can generate a trajectory path with a simple path planning tool, allowing users to fly-thru the airway easily and efficiently. The system also provides a mouse centric control to steer the camera around the volume. In addition, with the multi-histogram widget control it is intuitively easy to extract a bronchial airway view. It avoids a tedious step of semi-manual bronchial airway segmentation and minimizes the human based segmentation errors in the small airways structure. The proposed navigation system runs on general PC with a low end 3D graphics card, but can still achieve a relatively high rendering quality with interactive frame rate. The developed tool can be adapted for different non-invasive virtual endoscopy simulations, such as colon, bronchi, and blood vessels. We develop the system as a proof of concept tool for future simulation of minimally invasive image guided procedures.

Acknowledgments

This study supported in part by the NIH Intramural Research Program.

References

1. Schwarz Y, Greif J, Becker HD, Ernst A, Mehta A. Real-time electromagnetic navigation bronchoscopy to peripheral lung lesions using overlaid CT images: the first human study. Chest. Apr; 2006 129(4):988–94. [PubMed: 16608948]
2. Mori K, Deguchi D, Sugiyama J, Suenaga Y, Toriwaki J, Maurer CR, Takabatake H, Natori H. Tracking of a bronchoscope using epipolar geometry analysis and intensity-based image registration of real and virtual endoscopic images. Medical Image Analysis. 2002; 6:321–336. [PubMed: 12270236]
3. Higgins WE, Helferty JP, Lu K, Merritt SA, Rai L, Yu KC. 3D CT-video fusion for image-guided bronchoscopy. Comput Med Imaging Graph. Apr; 2008 32(3):159–173. [PubMed: 18096365]

4. Kiraly AP, Higgins WE, McLennan G, Hoffman EA, Reinhardt JM. 3D human airway segmentation methods for clinical virtual bronchoscopy. Academic Radiology. Oct; 2002 9(10):1153–1168. [PubMed: 12385510]

5. Graham, M. thesis. Jun, 2008 Robust Methods for Human Airway-Tree Segmentation and Anatomical-Tree Matching.

6. Wan M, Tang Q, Kaufman AE, Liang Z, Wax M. Volume rendering based interactive navigation within the human colon. IEEE visualization, '99. Oct.1999 :397–400.

7. Kruger A, Kubisch C, Straub G, Perim B. Advanced GPU volume rendering for virtual endoscopy. Eurographics. 2009

8. Yuan F. An iterative programmable graphics process unit based on ray casting approach for virtual endoscopy system. Optica Applicata. 2008; XXXVIII(3)

9. Diepenbrock S, Ropinski T, Hinrichs K. Context-aware volume navigation. IEEE Pacific visualization symposium (Pacific Vis). Mar.2011 :11–18.

10. Rubin GD, Beaulieu CF, Argiro V, Ringl H, Norbash AM, Feller JF, Dake MD, Jeffrey RB, Napel S. Perspective volume rendering of CT and MR images: applications for endoscopic imaging. Radiology. May; 1996 199(2):321–330. [PubMed: 8668772]

11. Higins WE, Ramaswamy K, Swift R, McLennan G, Hoffman EA. Virtual bronchoscopy for 3D pulmonary image assessment: state of the art and future needs. Radiographics. May-Jun;1998 18(3):761–778. [PubMed: 9599397]

12. Bartz D. Virtual endoscopy in research and clinical practice. Computer graphics forum. 2005; 24(1)

13. McAuliffe MJ, Lalonde FM, McGarry D, Gandler W, Csaky K, Trus BL. Medical image processing, analysis, and visualization in clinical research. 14th IEEE CBMS, Proceedings. 2001:381–386.

14. Cheng R, Bokinsky A, Hemler P, McCreedy E, McAullife M. Java based volume rendering frameworks. SPIE Medical Imaging 2008: Visualization, Image-Guided Procedures, and Modeling. Mar.2008

15. Eberly, DH. 3D Game Engine Design: A Practical Approach to Real Time Computer Graphics. second. New York: The Morgan Kaufmann Series in Interactive 3D Technology; 2006.

16. Senseney J, Bokinsky A, Cheng R, McCreedy E, McAuliffe M. SPIE medical imaging. Feb.2013

17. Kniss J, Kindlmann G, Hansen C. Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets. Proc IEEE Visualization Conf. 2001; 01:255–562.

18. Rezk-Salama C, Kolb A. A vertex program for efficient box-plane intersection. In proceeding of: Proc Vision, Modeling, and Visualization (VMV). Jan.2005

19. http://cococubed.asu.edu/code_pages/raybox.shtml

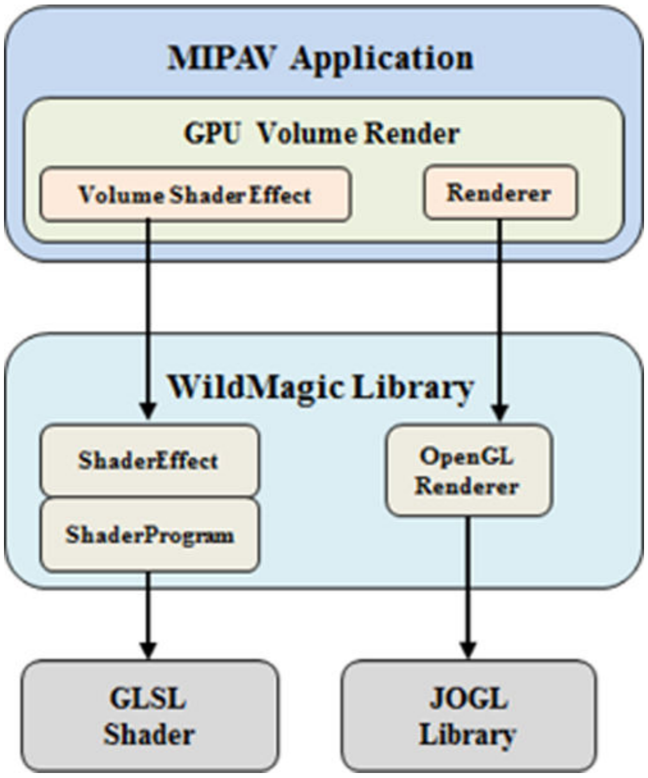


Figure 1.
MIPAV visualization framework layers

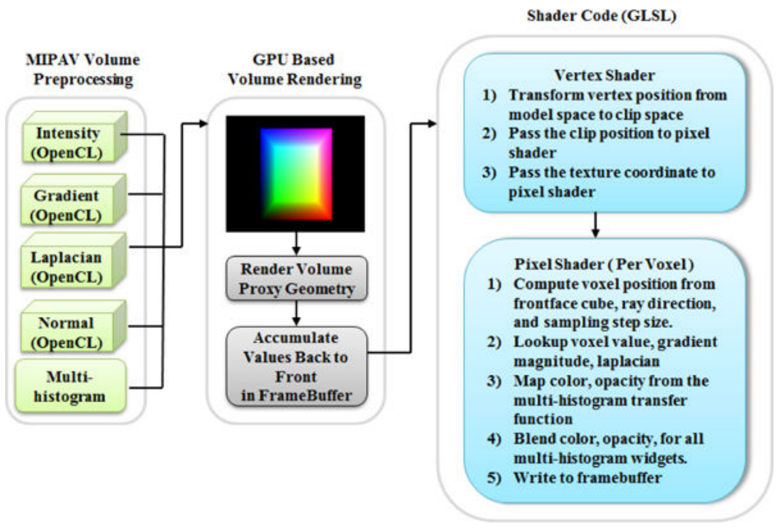

Shader Code (GLSL)
MIPAV Volume Preprocessing
GPU Based Volume Rendering
Intensity (OpenCL)
Gradient (OpenCL)
Laplacian (OpenCL)
Normal (OpenCL)
Multi-histogram
Render Volume Proxy Geometry
Accumulate Values Back to Front in FrameBuffer
Vertex Shader
1) Transform vertex position from model space to clip space
2) Pass the clip position to pixel shader
3) Pass the texture coordinate to pixel shader
Pixel Shader (Per Voxel)
1) Compute voxel position from frontface cube, ray direction, and sampling step size.
2) Lookup voxel value, gradient magnitude, laplacian
3) Map color, opacity from the multi-histogram transfer function
4) Blend color, opacity, for all multi-histogram widgets.
5) Write to framebuffer

Figure 2.
Old GPU based multi-histogram rendering pipeline

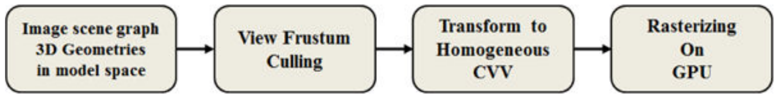


Figure 3.
General game engine rendering pipeline view frustum culling

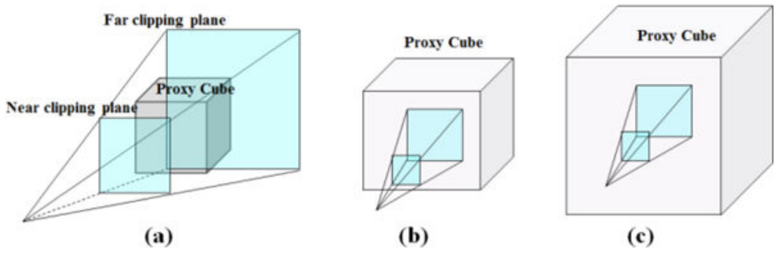


Figure 4.
View frustum culling in the context of navigation

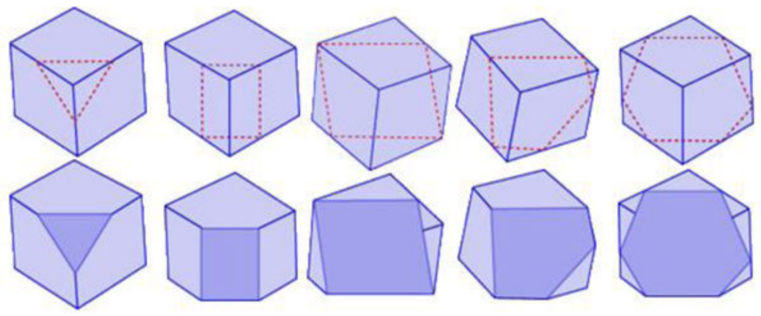


Figure 5.
Plane-cube clipping configurations

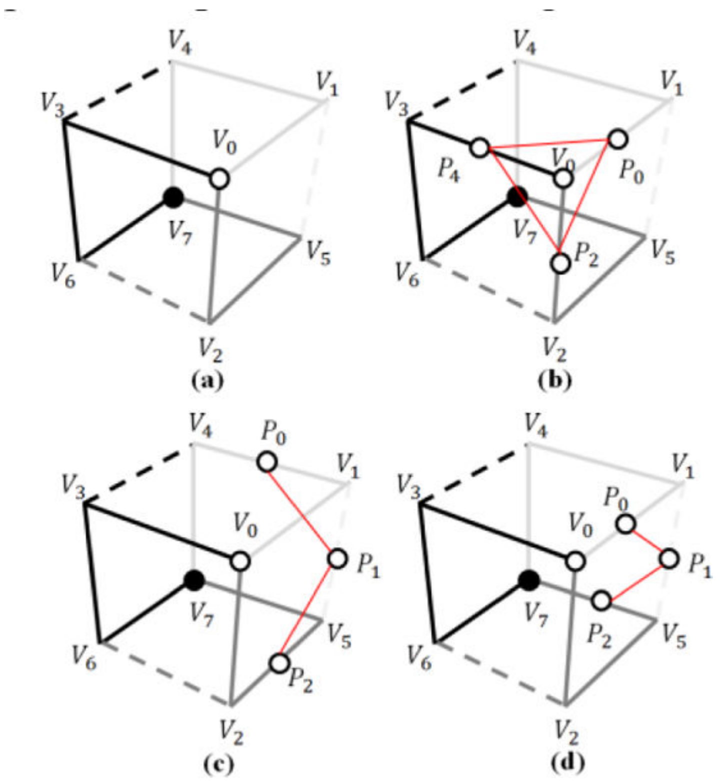
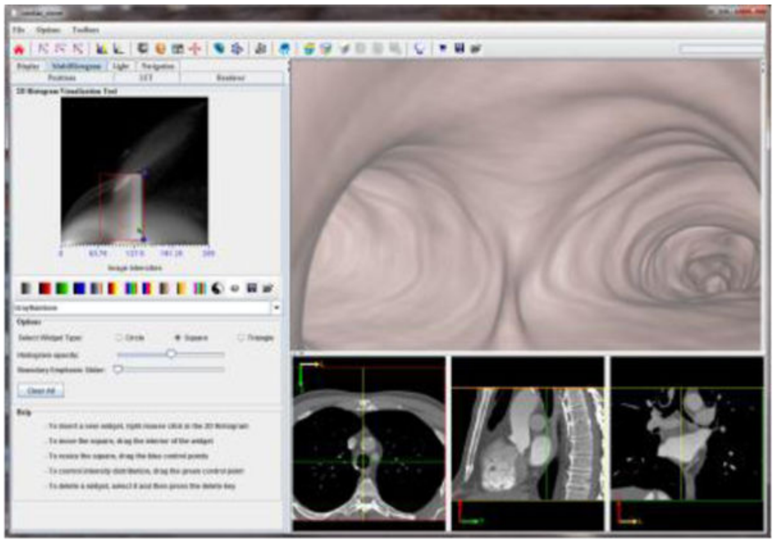


Figure 6.

Remapping of points on the proxy cube



- To insert a new widget, right mouse click in the 2D histogram
- To move the square, drag the interior of the widget
- To resize the square, drag the blue control points
- To control intensity distribution, drag the green control point
- To delete a widget, select it and then press the delete key

Figure 7.
Mouse centric control view

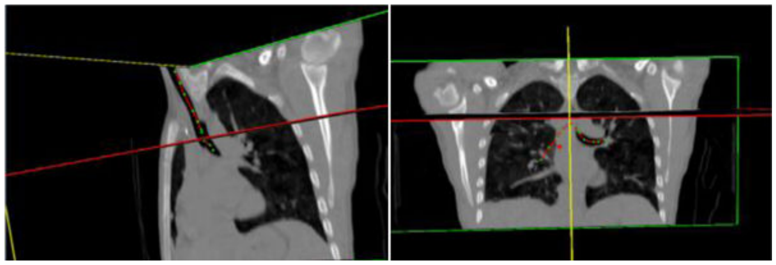


Figure 8.
Path planning

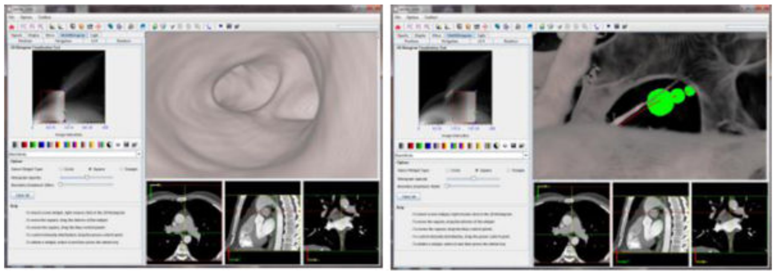


Figure 9.
Path planning mode fly-thru

Table 1

Performance (fps)

	Size	FPS
CTlung	512×512×73	28
Bronchus	512×512×256	20
Lung(female)	512×512×313	15