PROCEEDINGS OF SPIE

Real-time interactive visualization and manipulation of the volumetric data using GPU-based methods

Dietrich, Carlos, Nedel, Luciana, Olabarriaga, Silvia, Comba, Joao, Zanchet, Dinamar, et al.

Real-time interactive visualization and manipulation of the volumetric data using GPU-based methods

Carlos Augusto Dietrich[a], Luciana Porcher Nedel[a], Silvia Delgado Olabarriaga[b],
João Luiz Dihl Comba[a], Dinamar José Zanchet[d],
Ana Maria Marques da Silva[c], Edna Frasson de Souza Montero[d]

[a]Informatics Institute, Federal University of Rio Grande do Sul, Porto Alegre, Brazil
[b]Image Sciences Institute, University Medical Center, Utrecht, The Netherlands
[c]Physics Faculty, Catholic University of Rio Grande do Sul, Porto Alegre, Brazil
[d]Federal University of São Paulo, São Paulo, Brazil

ABSTRACT

This work presents a set of tools developed to provide 3D visualization and interaction with large volumetric data that relies on recent programmable capabilities of consumer-level graphics cards. We are exploiting the programmable control of calculations performed by the graphics hardware for generating the appearance of each pixel on the screen to develop real-time, interactive volume manipulation tools. These tools allow real-time modification of visualization parameters, such as color and opacity classification or the selection of a volume of interest, extending the benefit of hardware acceleration beyond display, namely for computation of voxel visibility. Three interactive tools are proposed: a cutting tool that allows the selection of a convex volume of interest, an eraser-like tool to eliminate non-relevant parts of the image and a digger-like tool that allows the user to eliminate layers of a 3D image. To interactively apply the proposed tools on a volume, we are making use of some so known user interaction techniques, as the ones used in 2D painting systems. Our strategy is to minimize the user entrainment efforts involved in the tools learning. Finally, we illustrate the potential application of the conceived tools for preoperative planning of liver surgery and for liver vascular anatomy study. Preliminary results concerning the system performance and the images quality and resolution are presented and discussed.

Keywords: Texture-based volume visualization, interactive volume clipping, real-time volume rendering, medical imaging

1. INTRODUCTION

The recent advances in medical imaging technologies turned image-based diagnosis and therapy planning into important clinical aids. In practice, however, the interactive inspection of the medical image three-dimensional (3D) data is constrained to very simple procedures, largely performed in 2D (slice-by-slice). Direct volume rendering (VR), a technique introduced by Levoy[1], is an alternative for 3D inspection of non-segmented volumetric data. In VR, the image on the screen is generated by accumulating appearance properties (color, opacity) of voxels traversed by rays casted from the camera into the volume. The selection of proper appearance attributes to the voxels (called "classification") and the removal of uninteresting volume portions (called "clipping"), allows for the visualization of selective objects on the screen. Two difficulties are faced when using this technique in real applications: performance and usability.

Firstly, VR is a costly process that normally requires specialized hardware for attaining real-time image generation. Still today, real-time visualization methods for large datasets challenge the computer graphics community[2]. Several efforts have been made for the development of dedicated VR architectures[3] and programmable graphics hardware techniques[4,5]. Among others, one research line has leaded to VR that exploits hardware-assisted texture mapping capabilities[2]. Cabral *et al.*[6] show that it is possible to provide interactive reconstruction and visualization using 3D texture mapping in hardware. Today, texture-based approaches provide the basis for efficient techniques for VR in consumer-level graphics cards with graphics processing units (GPUs)[2,4,5]. One of the reasons – maybe the most important of them – behind the recent research efforts in texture-based VR is the incredible advance of graphics hardware[7]. This advance is driven by forces like computer games and semiconductor industries, which demand cheaper and faster

graphics hardware at consumer level. Recent GPUs offer capabilities like 3D texture mapping and programmable instruction sets, enabling personal computers (PCs) to approach the power of graphics workstations.

Secondly, the configuration of classification and clipping parameters for the visualization of complex structures typically require much user intervention. For example, operations like 3D clipping are restricted to simple clipping volumes or planes, and, in the case of some human anatomical structures (e.g. liver vasculature), defining the volume of interest (VOI) only in terms of such simple geometric elements would require an enormous amount of interaction – if possible at all. Flexible and efficient interactive tools are required for the delineation of complex VOI. Moreover, considering the advance in GPUs and the efforts in texture-based volume rendering, it is surprising that the interaction with volume data using GPU has been largely overlooked as a research topic by the scientific visualization community[5]. The work of Weiskopf et al.[5] represent an exception that address hardware assisted interactive techniques to manipulate volumetric data. In their work, they present a tool called "depth-based clipping", where the VOI is given by a polygon mesh, as well as a clipping tool, in which a second texture keeps the visibility information relating the original image.

In this work we present the implementation of three different tools for VOI selection based on interactive 3D clipping techniques. The first tool, called "clipper" allows the interactive selection of a convex volume of interest with an arbitrary number of faces. By managing a cutting plane, the user interactively defines the convex polyhedron that will be intersected with the 3D image to generate the VOI. A new fast clipping algorithm (called "weighted sweep graph", or WSG) efficiently resamples this polyhedron according to any given viewing direction to generate the 3D texture supplied to the GPU. The two other tools allow interactive, real-time, modification of visualization parameters for classification and clipping. Their implementation relies on the programming capabilities of modern consumer-level GPUs, namely the control available over the calculations used to generate the color of each pixel on the screen. The cutting tools, called "eraser" and "digger", are based on a simple interactive technique largely adopted in painting systems, by means of which the user progressively deletes from the visualization the voxels pointed out with the mouse. These three tools can be used for preoperative surgery planning or for anatomic studies, allowing the user (a surgeon or a medical student) to carefully inspect the organ's anatomy, as well as to specify resection regions in a user-friendly way.

This paper is structured as follows. In Section 2 some background related to our work is provided, with a short introduction to texture-based techniques for volume rendering and classification. Section 3 presents the three clipping tools. In Section 4 we present results obtained with our techniques and performance measurements. Conclusions and a discussion of future work end the text.

2. BASIS OF HARDWARE-ASSISTED VOLUME RENDERING

2.1. Visualization

Hardware-based VR techniques reduce the problem of tracing a 3D ray in a 3D volume to the 2D problem of rendering a set of texturized planes. The 3D image, stored as one or more 3D textures, or stacks of 2D textures, is resampled by geometric primitives such as planes aligned with the volume axes, the viewing direction, or other configurations (such as spherical shells)[2]. These planes are rendered directly by the graphics hardware in an efficient manner. The rasterization step in the rendering pipeline breaks down each geometric primitive (normally a square) into pixel-sized "fragments", one for each pixel that it covers in the image plane. The visibility of each fragment is controlled with programmable operations and tests performed by the GPU, so that slow CPU operations are avoided. To compute the color of a pixel on the screen, the GPU composes the visibility attributes of each associated fragment in all planes. Fig. 1 illustrates the main concepts explained above.

In traditional texture-based volume rendering, all texture-space (entire 3D image) is resampled and taken into account at the fragments composition phase, including volume portions outside the VOI or where the voxels are completely transparent. In this scenario, the number of fragments that do not contribute to the final image can be large and slow down computation significantly.

Some authors[2,8] suggest to constrain resampling (and fragment generation) to texture areas (sub-volumes) where there is something to be seen. As in traditional texture-based VR, the sub-volumes always need to be rendered in visibility order (to preserve transparency), and resorted whenever the viewpoint (for perspective projection) changes. Such operations, however, can be heavy, so acceleration techniques are required. Li et al.[8] suggest an approach where the sub-volumes (in this case, axes-aligned boxes) are reorganized into an orthogonal BSP tree to simplify the sorting, and their intersections with the slicing planes are computed incrementally. The incremental computation of slicing planes coordinates uses the characteristics of axes-aligned boxes, which, however, often cannot properly describe the shape of volume portions to be eliminated from the computation.
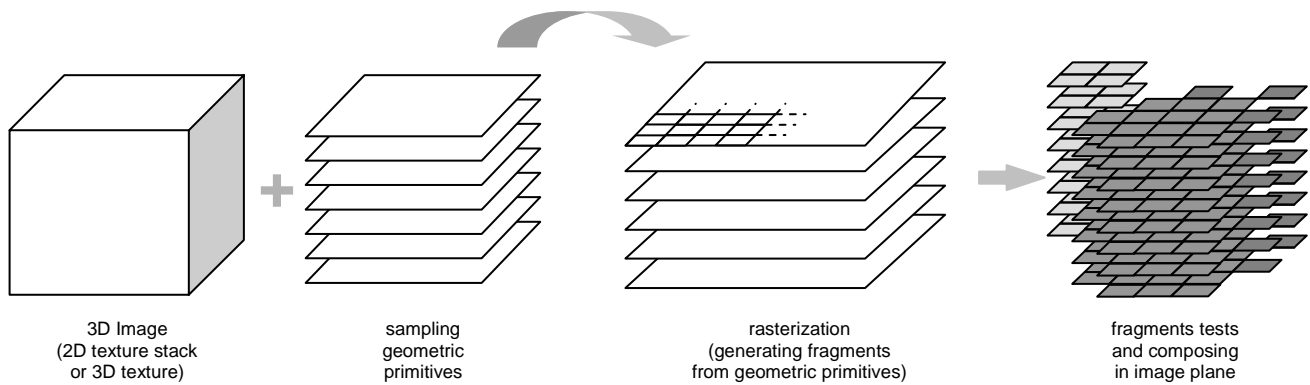
Fig. 1. 3D image visualization using texture mapping. The 3D image, stored as a 2D texture stack (or 3D texture), is first resampled by a set of geometric primitives. Next, in the graphics hardware, the color of each fragment of these primitives is computed (fragment shading) and them they are composed to generate the final pixel color on the screen.

Another important feature implemented in GPUs is multitexturing, which allows additional information to be stored into textures and used to compute the final color of each fragment. For example, this is quite useful in the VOI selection process to define cutting masks[5] (Fig. 2). In this approach, two 3D textures are used to store the input volumetric dataset and a binary mask that defines the geometry of the VOI. The VOI is obtained by multiplying (for each fragment) the result of the first texture by its corresponding in the second one.
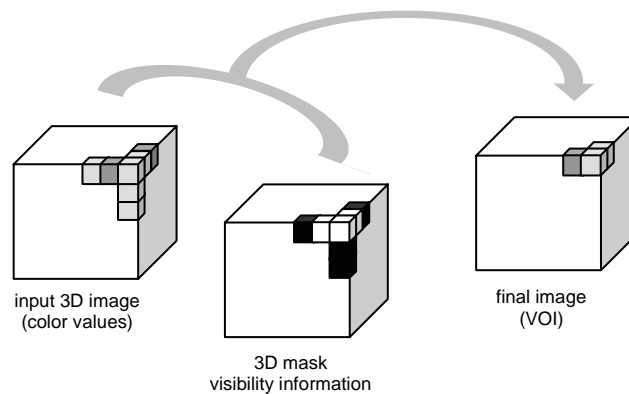

Fig. 2. Illustration of VOI selection tool based on multitexturing capabilities. The first 3D texture stores the volumetric data to be visualized. The second defines a binary mask defining the geometry of the VOI (white=visible, black=not visible). The last texture is generated by combining the two others.

2.2. Image classification

A transfer function assigns optical properties, such as color components (red, green and blue) and opacity (alpha) to scalar values of the dataset. Such values are used by an optical model that estimates the volume rendering integral inside each cell during rendering. The task of automatically finding the best transfer function for a given dataset is very difficult, requiring in most cases human interaction. This approach requires a system with a real-time or interactive response, such that modifications in the transfer function can be immediately visualized by the user. Typical speedups used in this calculation are the pre-computation of some components of the volume rendering integral and subsequent storage into 2D texture. During the visualization step, each processed fragment recovers the data from the 2D textures (a post-classification approach) and combines with other calculations to produce the final color of the fragment. This approach was used to store the transfer functions used in this work.

The interaction steps used to define transfer functions are very simple. The user can define multiple transfer functions corresponding to non-overlapping intervals (windows) of scalar values of the input dataset. The specification of each window is done by a central value and its width, color components (red, green, blue), opacity (alpha) and opacity function (linear, constant or Gaussian). The informed opacity value is used to position the function by assigning it to the

opacity of the rightmost scalar value in this window. For linear and Gaussian curves the opacity of the leftmost scalar is forced to be zero. In Fig. 3 we illustrate the definition of the three possible transfer functions used in this work.
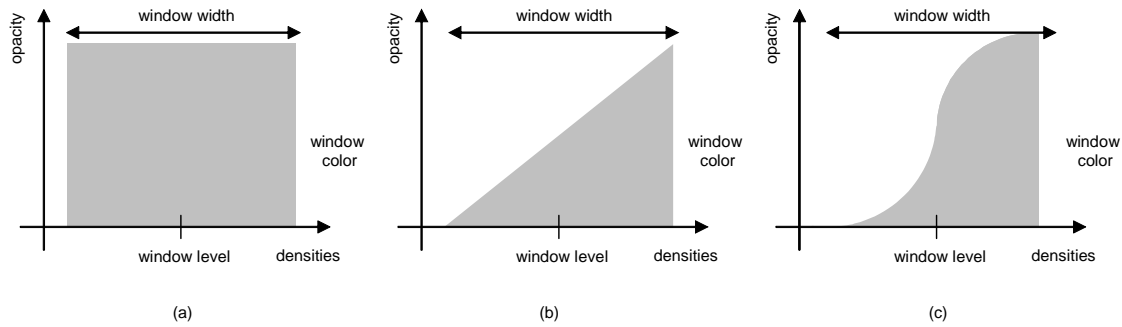

Fig. 3. Types of transfer functions used: constant (a), linear (b), and Gaussian (c)

3. INTERACTIVE CLIPPING TOOLS

In this work we adopt an interaction paradigm in the line of "what you see is what you get" (WSIWYG) in which the user progressively sculpts the VOI with three different tools: clipper, eraser and digger. Their primary goal is to provide an efficient inspection of inner parts of the VOI by removing parts of the original image from the visualization. At each interaction step, the tools affect only the voxels that are visible to the user on the screen, taking into account the current classification, clipping and viewing parameters. The clipper runs on the CPU, and its primary goal is efficiency, achieved by reducing the computation to a given VOI. The eraser and digger tools run on the GPU and only perform computations for regions that have passed through the previous clipping step.

3.1. Clipper

One aspect of 3D texture sampling using several parallel planes is that all fragments for each given plane are processed by the GPU. Although current graphics boards have enough performance that allows a reasonable large number of such planes, there is no efficient way to interrupt processing in a given fragment. In some datasets, empty regions can represent a substantial part of the volume, and not performing computation in such areas could free the GPU to perform other tasks (more sampling planes, more elaborate classification functions, etc). Since the GPU is not able to do this task, we use a clipping algorithm running on the CPU that intersects the sampling planes against a VOI, and only the resulting clipped planes are send to be processed by the GPU.

The VOI used in this work is defined by a convex region that lies inside the 3D volume. The specification of this convex volume by the user follows a *carving* approach. Starting with a basic convex shape (a cube for example), the user can define a cutting plane that slices the current convex volume, leading to a convex shape with one additional face. The process is repeated until necessary. Each cutting plane is informed by the user by its normal vector and a point on the plane. The user controls the normal direction by using the mouse to drag the second endpoint of a line that has as starting point the center of the VOI. The reference point in the plane is controlled by a slider that specifies the distance between the VOI center and the cutting plane. The set of cutting planes can be easily modified at any time, and arbitrary complex shapes can be defined without significantly affecting performance (Fig. 4).

Once the VOI convex region is defined, an algorithm is applied to compute its intersection against a sequence of parallel planes. Many algorithms in the literature[9,10] compute the intersection between a convex volume and a single plane. The problem here involves the intersection against the VOI of not *one* but *several* planes (parallel to each other), allowing the coherence of intersection calculations to be exploited.

The intersection of a single plane against the convex VOI results in a convex polygon, defined by an ordered (clockwise or counter-clockwise) list of vertices. These vertices lie over the edges of the VOI, and these edges play an important role in subsequent calculations. For instance, the calculation of a parallel plane near the previous plane is very likely to generate a convex polygon with vertices that lie over the same list of edges. Saving this list of edges from the previous calculation allow the vertices of the new polygon to be computed by intersecting these edges against the parallel plane. The necessary ordering of vertices to form the polygon is obtained from the list of edges, which is kept sorted along the vertex ordering used to create it. However, in some cases this polygon can not be computed from the previous

list of edges. This happens when a vertex of the VOI lies in the space between the two planes, requiring the list of edges needs to be updated by adding/removing edges incident to this vertex.
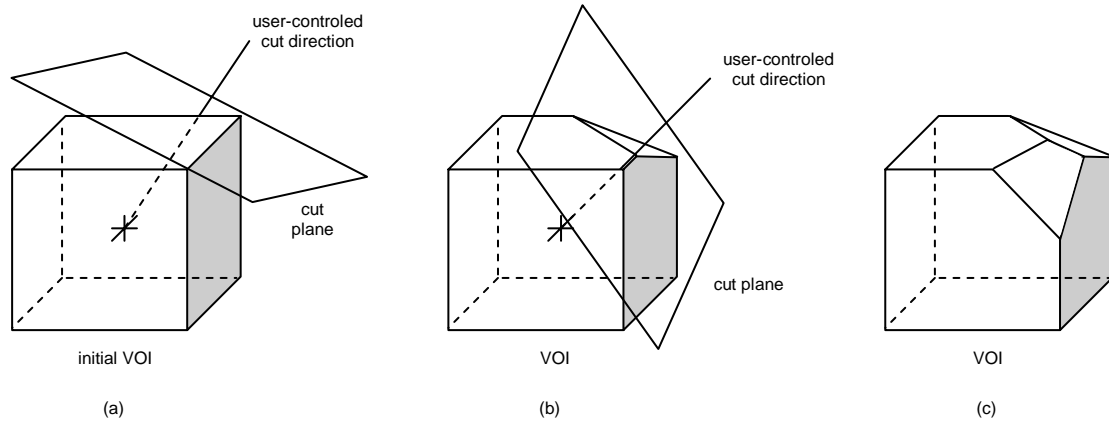

Fig. 4. Clipping using cutting planes. The cubic initial shape (a) and two subsequent steps in the creation of the VOI

The proposed algorithm computes all intersections by sweeping a plane through the volume using the current set of edges to compute the resulting intersection. We propose a new structure called *WSG* (weighted sweep graph) to conveniently encode the topological structure of the convex volume in such way that: (1) the list of current edges is maintained in polygon order, (2) vertex events are easily detected, and (3) the list of edges is processed by deleting/inserting edges incident to the vertex. Fig. 5 presents an example of WSG.
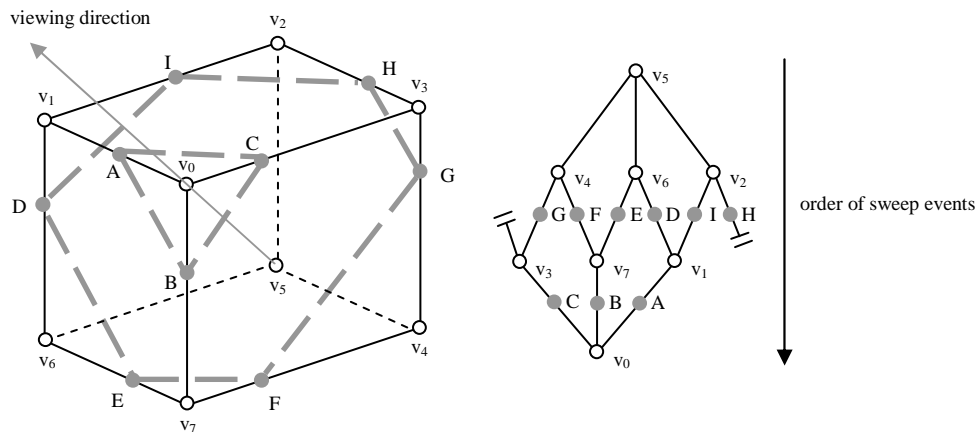

Fig. 5. A cubic VOI (left), the resulting WSG (right), and its use to compute two intersecting polygons. The WSG encodes the topology of the cubic region. The ordered list of edges of one intersected polygon is defined by a horizontal cut in the graph (G-F-E-D-I-H, or A-C-B, for example). Edge lengths are proportional to the distance between its vertices. Vertex events are defined when the sweeping of a cut through the WSG reaches a vertex. Edge list updates simply removes edges incident to the vertex that generated the event. In our planar representation of WSG (right), we consider that vertex v_3 and v_2 are connected.

3.2. Eraser and digger

The eraser and digger are special tools that allow the selection of voxels following a "painting" paradigm. This selection is achieved by only considering voxels that are inside an imaginary volume along a line-of-sight controlled by the user. The direction of this line is specified by placing (using the mouse) a 2D point on the screen (called sight-reference-point or SRP). For each 2D point, an inverse viewing transformation is applied to obtain the corresponding line of projective equivalent points. The extents of the imaginary volume are defined over the projection plane by a circular region. This is controlled by a slider that defines the radius of this circle-of-interest. The resulting imaginary volume will be either a cone (perspective projection) or a cylinder (parallel projection).

Clipping occurs by first projecting the center of every voxel into the projection plane. For each projected center, a distance to the SRP is computed (called projected distance of a voxel). Note that this computation happens over the projection plane, and corresponds to a 2D Euclidean distance. In the eraser tool, all voxels with projected distance smaller than the radius of interest are removed. In the digger tool, the same voxels considered for removal by the eraser tool are evaluated, but only voxels that are within a given distance from the first point of intersection with the volume (along the viewing direction) are removed. Another slider is used to specify this distance (also called depth). Fig. 6 illustrates the cutting tools behavior.
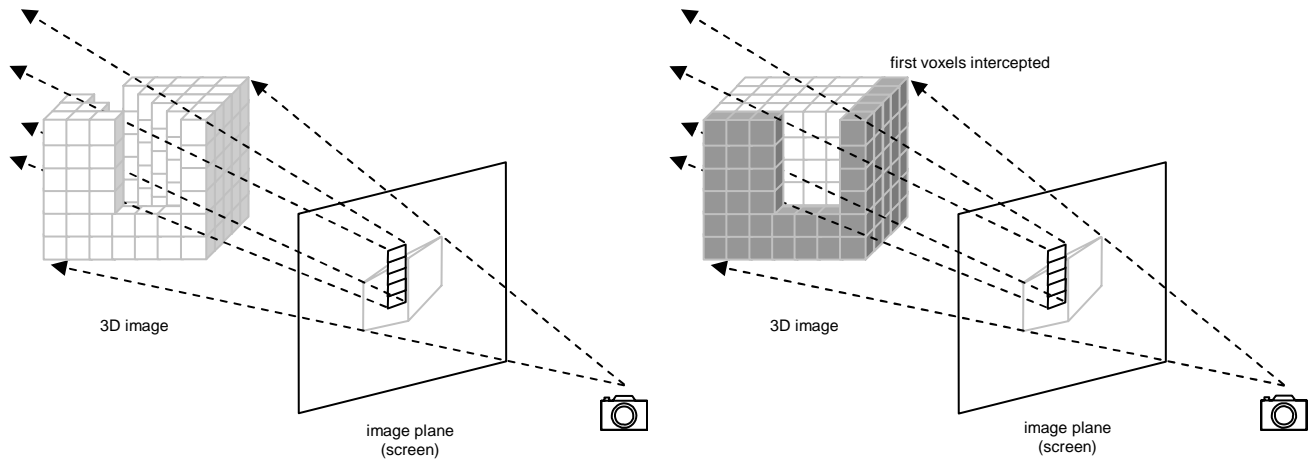

Fig. 6. Illustration of 3D interaction based on the projection image on the screen. The user points pixels on the screen (SRPs) and the voxels spanned by the "in depth" projection of the circle-of-interest are removed. This case illustrates a vertical trajectory with the mouse, using five SRPs and circle-of-interest = 0.5 pixels. (a) Eraser: all voxels along the ray are removed. (b) Digger: voxels within a given depth are removed.

Both tools are efficiently implemented using multitexturing facilities provided by modern GPUs, similar to the idea proposed by Weikopf *et al*[5]. In their work, one of the textures buffers stores grey values (corresponding to the volume to be visualized), while the other stores visibility information (a mask indicating, for each voxel, whether it should contribute to the image or not, as explained in Section 2 and illustrated in Fig. 2). In our work, instead of directly storing a binary visibility mask, one of the texture buffers contains distance information that is used to control visibility and to efficiently implement the eraser and digger tools.

Fig. 7 illustrates the main data structures and operations involved in the tools implementation. The polyhedron defined by the user with the clipper (see Section 3.1) resample the original 3D image, defining the 3D image (*ClippedImage*) to be visualized. The VOI is implicitly produced by combining the *ClippedImage* with a texture containing visibility information (*SculptBuffer*), and rendered directly by the GPU to produce the resulting image on the screen.

The eraser and digger tools operate only in the *SculptBuffer*. Each voxel of this buffer keeps a scalar value corresponding to the distance, in screen coordinates, between a SRP defined by the user and the voxel projection on the screen. This distance indicates whether the projected voxel is affected by the tool (distance < circle-of-interest) and, consequently, should be removed from further visualization. The *SculptBuffer* is initialized with the largest representable number, indicating that all voxels are visible. When a new SRP is set, the system recalculates all the distances and updates the *SculptBuffer* in the following manner: if the new distance is smaller than the old one, the value is updated; otherwise, the value is unchanged. In this manner, the buffer registers the smallest distance from the voxels in the *ClippedImage* to the SRP defined. All voxels "touched" by the tools will have distance smaller or equal to the circle-of-interest, being ignored for further image generation.

The mechanism described above is sufficient to implement the eraser tool. For the digger tool, however, depth information is also required to determine if a voxel will be removed from further visualization. In this case, the update of the *SculptBuffer* considers the depth measured along the viewing direction, and only those voxels that have depth greater than the digging depth are discarded. For voxels that have depths smaller than the digger depth, a value is written into the *SculptBuffer* that forces it to be invisible in the final rendering step.

Unfortunately, a current limitation of graphics hardware prevents 3D textures to be written as the output of fragment processing. Since we intend to use the *SculptBuffer* as a 3D structure, it requires an intermediate step that

generates a 2D representation of the 3D buffer (using several slices stored side-by-side). This structure is called the *2DSculptBuffer*. Once the GPU generates this buffer, it is read back into the CPU. Fortunately, sending this information back to the GPU as a 3D texture does not require any further processing of this data. Therefore, only the pointer to the memory location can be passed to the GPU to create the *SculptBuffer* as a 3D texture in the GPU. This limitation should be removed in future generations of graphics boards and improve even further this processing.
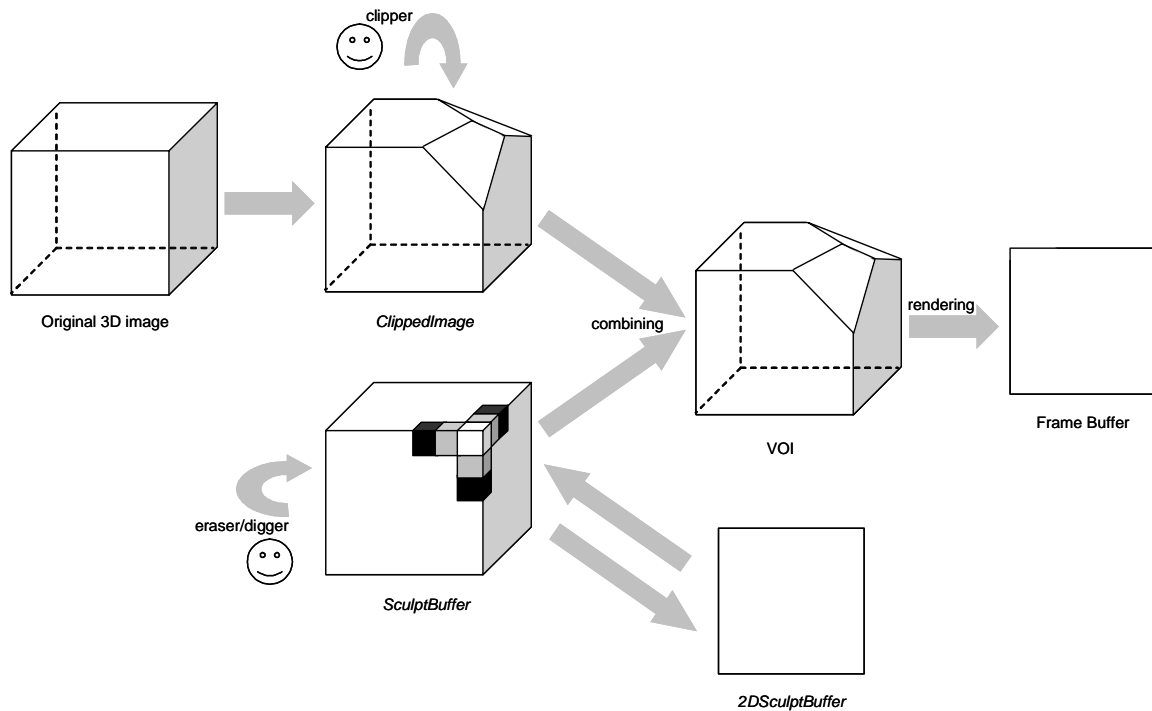

Fig. 7. Interactive visualization pipeline, considering the application of the three developed tools: clipper, eraser and digger

Once the *SculptBuffer* is generated, the visualization algorithm starts by rendering the slices that are contained in the *ClippedImage*, which were generated by the clipping algorithm described in the previous section. Each slice corresponds to a convex polygon in 3D that is rendered into several fragments that correspond to a 2D matricial approximation of this polygon. Associated with each fragment we maintain the position of the 3D point that generated it, (for fragments internal to the polygon we use interpolated positions of polygon vertices). This position is used as index into the *SculptBuffer*. If the position happens to lie exactly on the center of a voxel of the volume, then the *SculptBuffer* returns the exact distance to the SRP. Otherwise, a linear interpolation of the distances of the closest voxels is used, allowing a smooth transition between distance values that reduces aliasing artifacts during rendering.

After the distance associated with each fragment is computed, a fragment program running on the GPU discard fragments that have distances smaller than the radius-of-interest. For visible fragments, the shading is computed by accessing the look-up table generated by the transfer functions (see Section 2.2) and rendered into the frame buffer.

4. CASE STUDY

The complex anatomy of the liver and associated vascular systems causes difficulties not only for medical students, but also to experienced surgeons when preparing and performing a liver surgery (e.g. transplantation). Preoperative imaging plays an important role in patient selection and surgical planning[11], and contrast-enhanced spiral CT is the gold standard imaging technique for visualizing in-vivo liver vascular anatomy. However, the conventional 2D approach to inspect the volumetric image fails to reveal the complex liver structure, and only experimented surgeons and radiologists can correctly interpret them.

The cutting tools presented in Section 3 were developed as an attempt to facilitate the interactive 3D inspection of volumetric images, in particular for preoperative planning of liver surgery and for study of liver anatomy based on CT scans. The case study presented below illustrates the potential of the developed tools in the analysis of a complex 3D structure that would be very difficult to inspect satisfactorily with conventional cutting tools.

4.1. Image acquisition

The experimental results presented here correspond to in vitro livers of pigs, where the portal and hepatic veins were catheterized and ionic contrast medium was injected. The CT datasets were obtained using a spiral acquisition scanner CT Toshiba Asteion®), consisting of transversal slices obtained at 120 kV peak kilovoltage, 150 mA tube current, pitch 1.5, slice thickness 3mm, 512 x 512 pixels (0.625 mm in plane spacing) and 16-bits/pixel grey-scale information. Timing of the image acquisition relative to the contrast agent injection is not important at in vitro studies.

4.2. Image classification

As mentioned in Section 2.2, our system allows the specification of multiple transfer functions. For the experimental results with the liver datasets, we used two transfer functions defined according to the image acquisition protocol. Basically, our interest lies on: visualizing the liver parenchyma and vessels; suppressing background artifacts. The protocol includes the image windows and a pre-defined transfer function, both defined by the radiologist with Merge eFilm® workstation software. The acquired images, as well as the pre-defined transfer function are received in DICOM medical image file format. Using the pre-defined transfer function and the Hounsfield density values (the Hounsfield values of liver tissue and contrast agent are known *a priori*) we recover the transfer functions shown in Fig. 8, including color and opacity attributes. These transfer functions emphasize the liver parenchyma and the contrast agent, i.e., the intra-hepatic vessels.
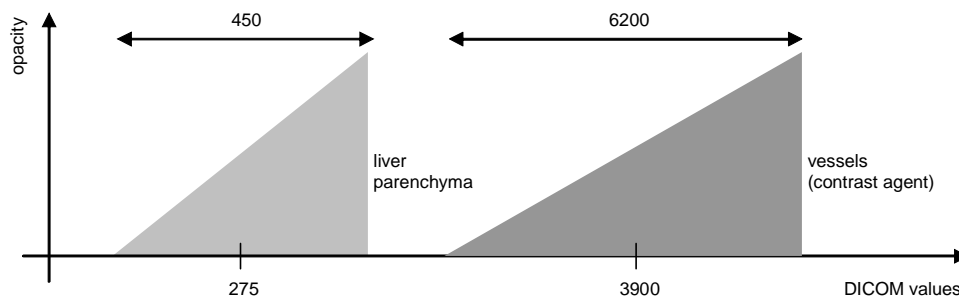

Fig. 8. Two linear transfer functions used in the liver dataset

Fig. 9 shows the visualization of a CT scan of the liver with the application of the two look-up tables presented in Fig. 8. The parenchyma opacity was reduced to allow for vessel visualization. The application of the look-up function on each voxel of the dataset is completely done by using the graphics hardware and achieving real-time frame rates.
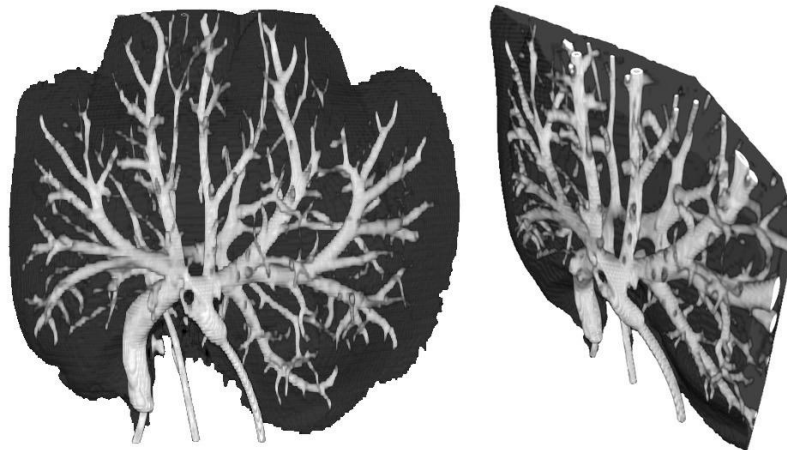

Fig. 9. Visualization of a CT scan of the liver with different look-up tables, for vessels and parenchyma

4.3. VOI selection

Fig. 10 shows a VOI selection exclusively with the application of the clipper tool (see Section 3.1). In this example, the VOI is determined by means of a polyhedron, which allows the determination of the section planes in hepatectomy planning. Similar results can also be achieved with the eraser tool, but the clipper effectively reduces the number of

voxels that should be processed, improving the rendering performance. In our experiments, we observed that the rendering performance is directly proportional to the number of voxels in the VOI.

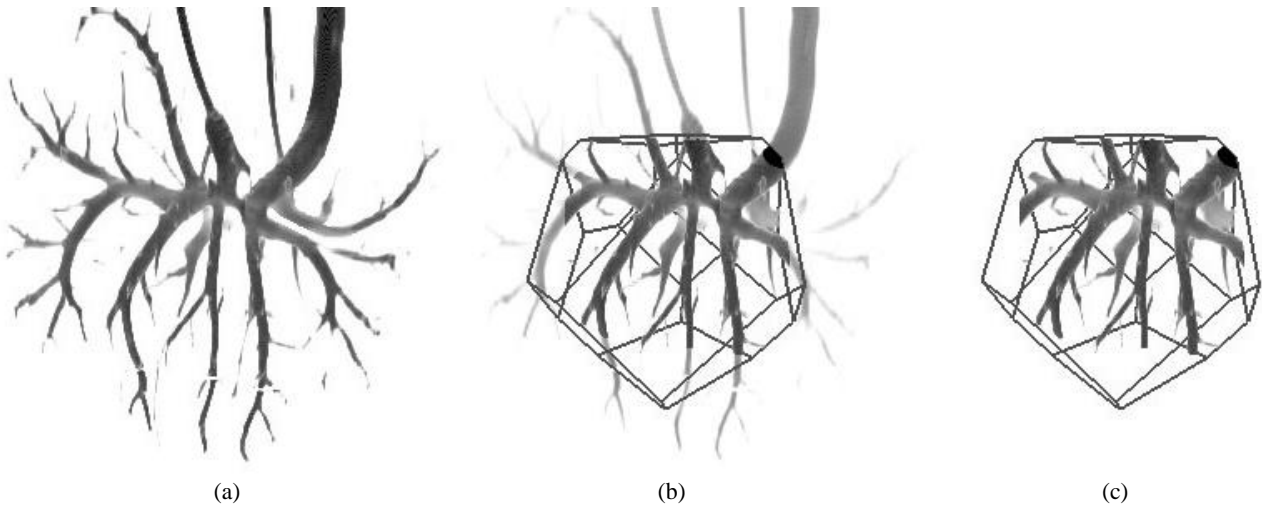(a)   (b)   (c)

Fig. 10. Using the clipper to select the primary portal vein branches in the volumetric vascular liver image. The branches show the anatomy of hepatic segments (liver anatomy segmentation). The definition of a VOI by means of a polyhedron allows the determination of the section planes in hepatectomy planning: the original dataset (a); the definition of a new convex clipped VOI (b,c).

Fig. 11 shows the application of the eraser tool to simulate a resection of a liver region, revealing the intra-vascular structure. Fig. 12 illustrates the use of the digger to simulate incisions on the liver parenchyma, exposing, in another way, the vascular structure of interest for surgery planning.

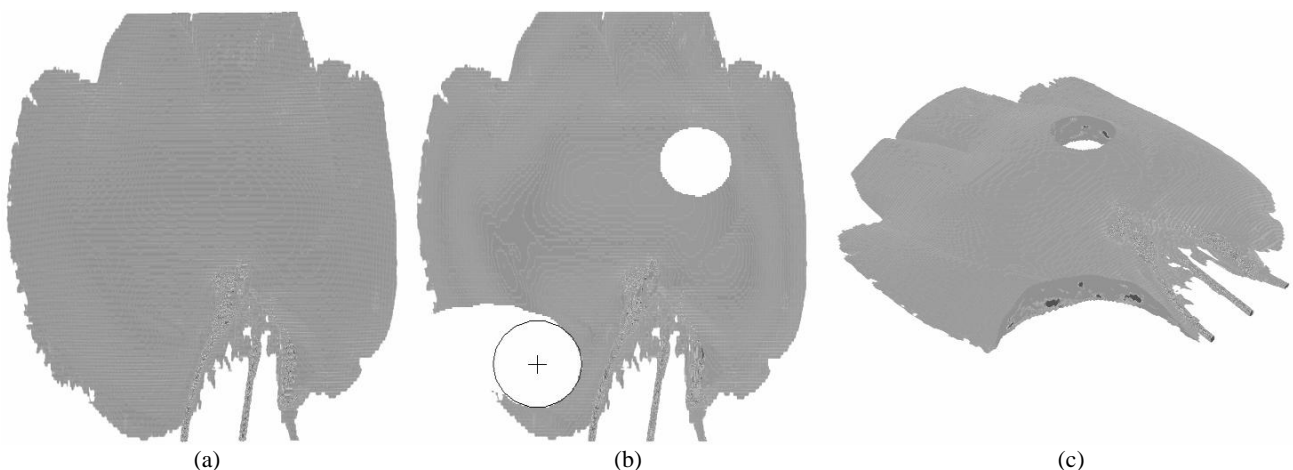(a)   (b)   (c)

Fig. 11. Eraser example: the eraser was used in two different places of the original liver (a,b); visualization of the same VOI from another point of view (c), from which it is possible to observe the inner structure at the resected places.

Fig. 13 illustrates the use of eraser and digger for hepatectomy planning. In this kind of procedure, the surgeon often needs to cut off a tumor, an entire liver segment, or a set of segments. A special case of liver segmentation is considered in transplantation surgeries and involves the definition of the Cantlie's line, the line that functionally divide the liver in two parts (left and right), guarantying that the two parts will keep its functionalities after the resection. The determination of the vessels inside the parenchyma is also an important step in the definition of a resection plane. This kind of exploration can be enabled by using our digger tool.

The images shown in Figs. 9 to 13 were produced from a dataset of size 512 x 512 x 128. The images were generated in a 512 x 512 pixels window in a PC Pentium 4, with 2 Ghz, 512 Mb, with a graphics board ATI® 9800. The performance achieved in this workstation was considered acceptable. While in a simple GPU based renderization of a 512 x 512 image obtained from a VOI of size 512 x 512 x 100 we achieved about 2.18 msec per image (i.e. 458 images

per second), with our method that considers the application of eraser and digger tools and the same conditions above, we achieved 4.38 msec per image (i.e. 228 images per second). Instead the fact our interactive visualization method seriously impacts in performance, we continue largely guaranteeing real-time rates.
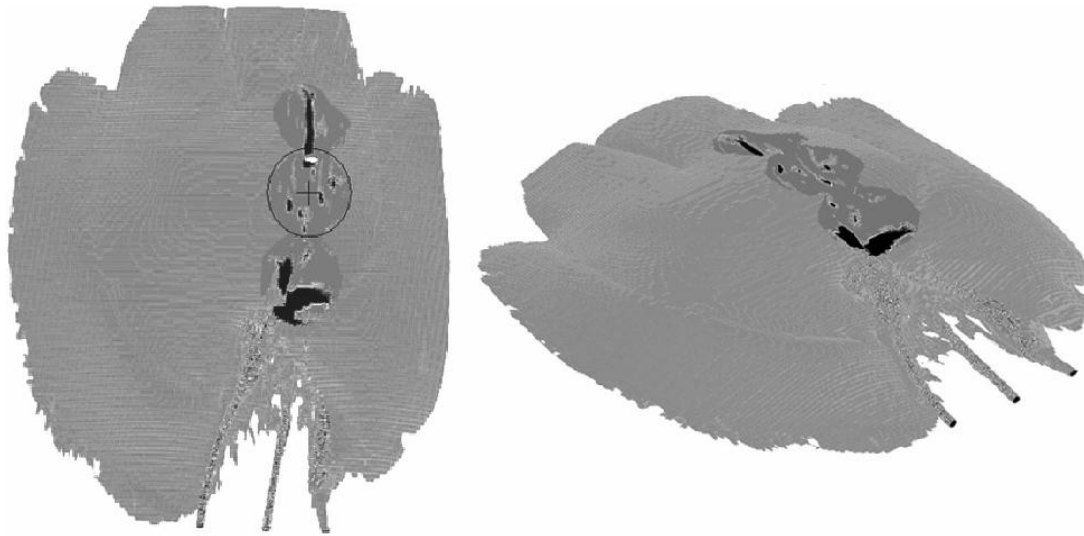

Fig. 12. Two different views of the same liver parenchyma sculpted with the digger tool
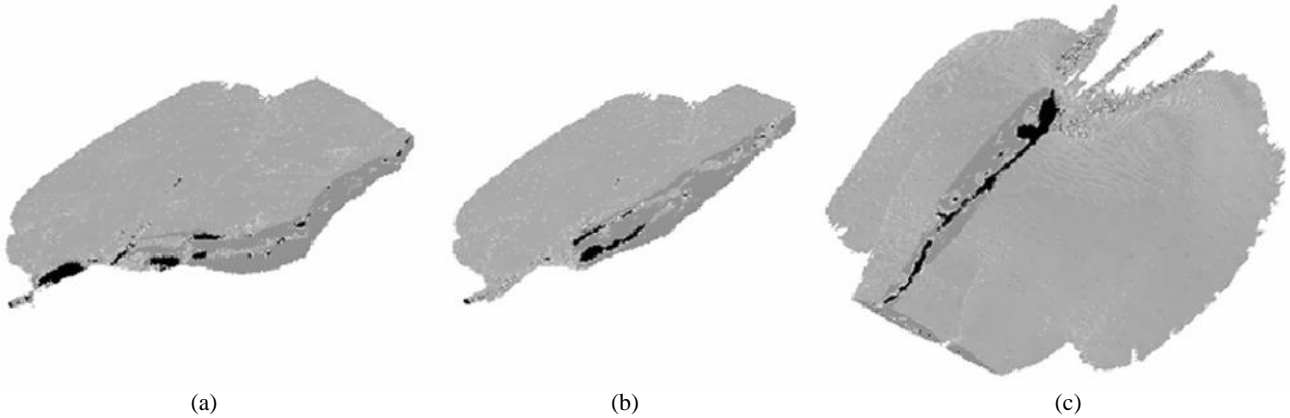

| (a) | (b) | (c) |

Fig 13. Hepatectomy planning examples: left lateral segmentectomy image (resection of liver segments 2 and 3) using the eraser tool (a); left hepatectomy image at the Cantlie´s line (resection of segments 2, 3 and 4) with the eraser (b); visualization of the liver parenchyma and middle hepatic vein trajectory and drainage of the middle and right middle hepatic veins in the cava vein obtained with the digger tool (c).

Color versions of the images presented in this section are available at LIVER3D project web site at http://www.inf.ufrgs.br/cg/liver3D.

5. DISCUSSION AND CONCLUSIONS

In the experimental results presented, the tools implemented were successfully employed in VOI selection. The use of 2D cutting tools following a "painting" paradigm shows itself as a simple and efficient approach to allow the interaction with 3D images. Additionally, the use of these tools – mainly the clipper – can improve the rendering performance, while eraser and digger do not drastically decrease efficiency with a reasonable number of sampling slices in the VOI. Concerning the efficiency, it is important to observe that the rendering pipeline presented here can be improved. The WSG structure capabilities could be used to split a 3D image in sub-volumes, where each sub-volume can be represented by a different WSG. Putting together, these WSGs can maintain the visibility order, while a visibility algorithm decides

who can be rendered or not. Even the concept of circle-of-interest, introduced in Section 3.2 and used to define the area affected by the eraser and digger tools can be easily extended, allowing the selection of arbitrary shapes (shape-of-interest) on the image plane. This interaction can also be assisted by the system, helping the user to select regions with some similarity criterion.

Finally, Fig. 11 shows the first results we obtained from the exploration of another improvement in our tools, which involves the use of eraser and digger affecting only one of the transfer functions in use. To produce this kind of result, the final rendering algorithm is being adapted.
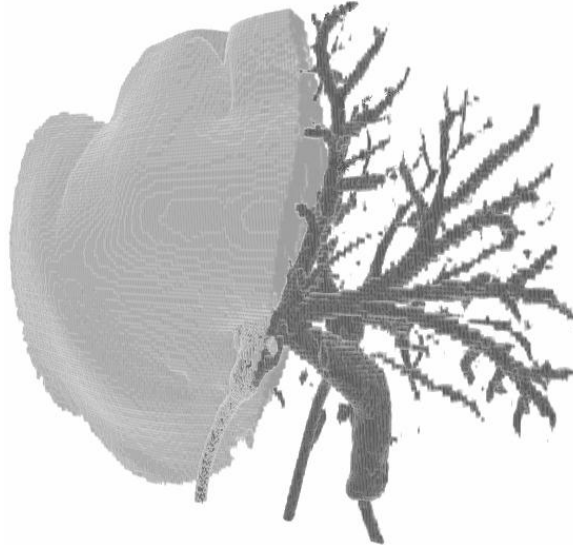

Fig. 14. Using the eraser tool to show the liver segmentation at the Cantlie's line, considering just the part of the image classified by the look-up table designed for the parenchyma.

ACKNOWLEDGMENTS

REFERENCES

1. M. Levoy. "Efficient Ray-Tracing of Volume Data". In: *ACM Transactions on Graphics*, v. 9, n. 3, p. 245-261, July 1990.
2. J. Krüger and Westermann. "Acceleration techniques for GPU-based volume rendering". In: *Proceedings of 14th IEEE Visualization Conference*, p. 287-292, October 2003.
3. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer and L. Seiler. "The VolumePro real-time ray-casting system". In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS*, p. 251-260, August 1999.
4. K. Engel, M. Kraus and T. Ertl. "High-quality pre-integrated volume rendering using hardware-accelerated pixel shading". In: *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS Workshop on Graphics Hardware*, p. 9-16, August 2001.
5. D. Weiskopf, K. Engel, and T. Ertl. "Volume Clipping via Per-Fragment Operations in Texture-Based Volume Visualization". In: *Proceedings of 13th IEEE Visualization Conference*, p. 93-100, October 2002.
6. B. Cabral, N. Cam, and J. Foran. "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware". Arie Kaufman and Wolfgang Krueger, editors, 1994.
7. R. Fernando and M. J. Kilgard. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison-Wesley, Boston, 2003.
8. W. Li, K. Mueller and A. Kaufman. "Empty space skipping and occlusion clipping for texture-based volume rendering". In: *Proceedings of 14th IEEE Visualization Conference*, p. 317-324, October 2003.

9. A. Glassner (series editor). Graphics Gems Series (Volume I-V). Academic Press, 1990-1995.
10. P. Schneider and D. Eberly. Geometric Tools for Computer Graphics. Morgan Kaufmann, 2003.
11. Kamel I.R., Kruskal J.B., and Raptopoulos V. "Imaging for Right Lobe Living Donor Liver Transplantation". In: Seminars in Liver Disease 2001, v. 21, n. 2, p. 271-282, May 2001.