Visualization System for the Interior of Objects with Internal Structures for Surgery Simulation

Dennis G. Balreira, Anderson Maciel, Marcelo Walter
Institute of Informatics
Federal University of Rio Grande do Sul (UFRGS)
Porto Alegre, Brazil
Email: {dgbalreira,amaciel,marcelo.walter}@inf.ufrgs.br

***Abstract*—Simulations present many advantages over other training methodologies since they reduce time and cost spent by professionals, offering less risk to the patients. Besides, studies show that the amount of realism seen in the simulators is positively correlated to the engaging of students in learning. Current simulations in virtual surgery use three-dimensional representations of organs without any internal structure. For some applications, however, there is a need to represent also the organs internal anatomical structures, such as blood vessels. We present, in this paper, a technique that allows arbitrarily oriented cuts through objects, particularly anatomical structures, reconstructing the mesh surface in the cutting zone. In the process, all internal structures participate in the final rendering of the generated surface. As a case study, we selected a human liver model with vessels and present the internal visualization of the liver in real time for arbitrary cutting planes. Our work has applications, for instance, in improving current state-of-the-art surgery simulators for training of students and medical doctors.**

***Keywords*-surgical simulators, solid textures, visualization system, computer graphics**

I. Introduction

Simulation is a safe way to educate and improve the skills of undergraduate students in Medicine. It allows students to learn and repeat procedures several times, which is impossible to do with real patients. Replacing real people's or animal's organs by virtual models addresses ethical, religious and legal issues, improving the skills of future Medical Doctors. Besides, researchers confirm that virtual surgery training systems provide several benefits when incorporated in medical education [1]. For many applications in Computer Graphics geometric representations using only the surface of objects is enough. However, there are some applications that require the visualization and possibly interaction with the interior of objects. Surgical operations such as hepatectomy [2] for instance, require cutting the liver in distinct regions depending on each patient. Furthermore, simulators involving these kind of organs [3] [4] [5] have focused their work on other aspects of the simulation.

Current solid texturing techniques generate seamless textures inside objects. However, they cannot deal with cases when there are other objects inside the surface geometry, such as blood vessels inside the liver. In this sense, we improve the solid texturing approaches in order to deal with those objects.

This work relates to a M.Sc. dissertation.

Contributions: This paper proposes a new technique that allows plane-oriented cuts in anatomical structures, not only reconstructing the surface's texture in the cutting zone but also taking into account its internal structures. Our approach considers the geometric objects as a whole, composed by surface and inside, and reconstructs the mesh in the cutting zone. In Fig. 1 we show an example of result from our solution. Next generation surgical simulators can benefit from this approach in order to improve realism, and, therefore, skills development.

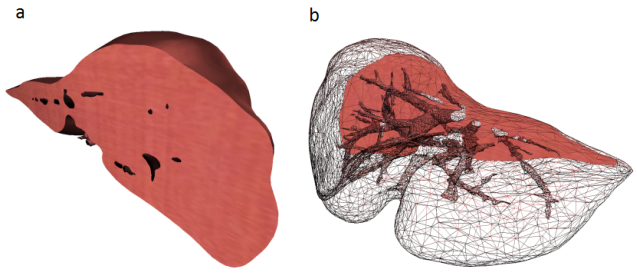

Fig. 1. Result from our technique. A liver with a cut showing the arteries as empty shapes (a) and the same result from a diferent perspective in wireframe (b). Information about this result as cut 1 in Table II.

II. Related work

The idea of solid texturing or 3D texturing was introduced in 1985 [6] [7] and has evolved much since. Solid textures are defined as a process in which a texture generating function is evaluated directly on $\mathbb{R}^3$. This function defines a volume, and the object looks as if it was *carved* from this volume. A survey published by Pietroni and colleagues [8] presents a detailed review of several techniques for solid texturing and texture synthesis. From the original papers, Ghazanfarpour and colleagues [9] [10] extended the idea using spectral analysis and texture synthesis applied to 2D orthogonal views [11]. Another method adapted a texture synthesis process called optimization-based with histogram matching, globally minimizing its energy function [12]. In the following year, a procedure introduced a new technique to synthesize solid textures restraining them to a voxel subset, forcing spatial determinism [13]. Recently, a new approach tried to deal particularly with regular and semi-regular patterns [14].

Cutler *et al.* [15] presented a scripting language in order to define internal layers for objects. Two years later, Owada and colleagues [16] proposed a new method that consists in specifying the interior of an object by using a browsing and modeling interface, controlled by the user. Pietroni *et al.* [17] used a few images representing cross sections of an object in order to render any point inside it. Takayama's research [18] extended the concept of lapped textures [19] to solid textures, covering the whole object's volume instead of only its surface. In the same year, a new system called volume painter [20] projected volumes from sketches defined by the user. In 2010, a new concept called diffusion surfaces [21] was able to render structures that have a smooth color variation in its internal structures, like fruits and vegetables.

From the review above, we can see that the current state-of-the-art on solid texturing does not deal with objects with internal structures. Our technique considers internal structures and allows visually and geometrically consistent arbitrary cuts on the objects, extending, in this way, the state-of-the-art in solid texturing.

III. Our technique

This section presents our technique for dealing with arbitrary cuts in objects with internal structures, with possible application in surgery simulation. Overall, our algorithm receives two 3D meshes and a set of three images as input. The two meshes represent respectively an object's surface and the content in its interior. The three images are orthogonal samples of the 3D internal texture of the object.

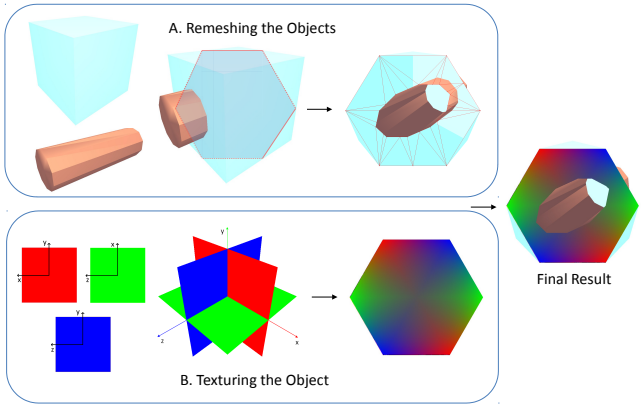

Fig. 2. Overview of our technique.

For any given cutting plane, our technique returns a consistent geometry – object's surface plus interior – and a texture to be mapped on the region defined by the cut. We generate the final solid texture by an interpolation function applied at each point on the cutting plane. If the plane intersects with the internal structure, the triangles in the overlap area are defined as holes and thus are not rendered. Figure 2 presents the pipeline of our technique with the two primary processes labeled: A-Remeshing the Objects, detailed in Section III-A; and B-Texturing the Object, detailed in Section III-B.

A. Remeshing the Objects

This section describes the sequence of steps for remeshing the input objects, surface and interior, according to the cutting plane. Our algorithm works for any triangular mesh, even concave and with disconnected parts. At the end of process A, each object is sectioned at the intersection of the model with the cutting plane, the surface model is retriangulated, and the internal parts are flagged as holes.

1) Cutting the Models: The first step consists of sectioning the object's triangles according to the cutting plane. Depending on the spatial location of the vertices with respect to the plane, defined by the λ value, different solutions are applied. We calculate λ by applying the coordinates of each triangle vertex on the equation of the cutting plane, producing the possible cases seen in Figure 3.

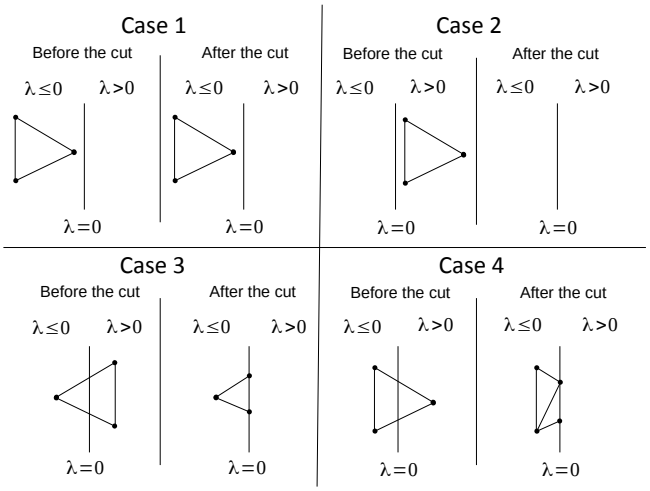

Fig. 3. Possible cases of cuts applied to each triangle of the mesh in 2D view.

For all cases, the three vertices of each triangle are tested. In case 1, if λ of all vertices satisfy $\lambda \leq 0$, they remain the same because they are in the side of the object that remains. In Case 2, if λ of all vertices satisfies $\lambda > 0$, the triangle is removed from the object, since it is on the side which will be eliminated. In Case 3, if only one of the three λ values satisfies $\lambda \leq 0$, then the remaining two vertices are repositioned to the intersection point of the plane with the triangle. In the last case, if two λ satisfy $\lambda \leq 0$, the remaining vertex is moved to one of the intersection points and another triangle is created connecting the two intersection points with the generated triangle. At the end of this step, the results are consistent new meshes taking into account the cutting plane, as shown in Figure 4.

2) Segmentation in Topologically Connected Sets: The result of the previous step is the two original objects modified by the cut. Now we need to group sets of vertices into topologically connected sets, called *segments*. This step classifies in separate groups all the connected vertices on the cutting plane and allows our solution to work with geometries where a cut will split the original object into two disjoint parts. The key idea is to make a depth search along the vertices which

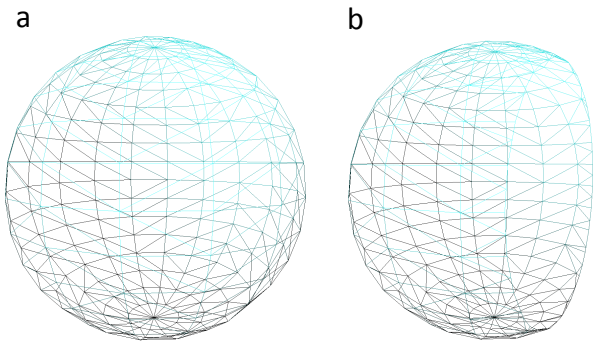

Fig. 4. Example of a sphere model before (a) and after (b) the cut.

were cut in the previous step, searching for the neighboring ones under the same conditions until the first vertex is reached again, forming a topologically connected set and its respective order. Figure 5 shows an example where two connected sets are detected.

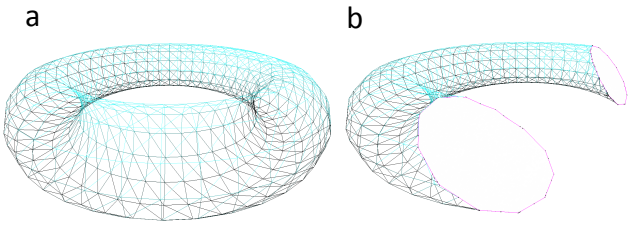

Fig. 5. Example of a torus mesh before (a) and after (b) being cut by a plane. In (b) two distinct connected sets can be observed.

3) Transforming the vertices to 2D: All the vertices in the cut area already segmented in connected sets are defined in world coordinates. In order to apply two-dimensional algorithms in the next steps, these vertices are transformed to a new basis determined by u, v and n, with n being the cutting plane's normal vector.

4) Computing the Holes for each Segment: The triangulation algorithm, presented in the next step, needs information about which parts are of the surface object and which parts are of the interior of the object, to triangulate only the parts which are part of the object's surface. We need therefore to identify these parts by finding a point inside each segment. We call h a 2D point located inside a two-dimensional segment which will not be triangulated (a hole). We compute h by calculating the average of the first two ordered intersection points found between a line connecting one of the diagonals from the hole's bounding box and the intersected edges (Figure 6).

After, we apply a point-in-polygon algorithm to check if h is inside the hole. Otherwise, we repeat the process with the remaining diagonal, as seen in Figure 7.

5) Triangulation: Once we have all the segments and holes identified, we can compute a triangulation of the segments; holes are not triangulated. We use the *Triangle* library presented by Shewchuk [22] to triangulate the vertices that define

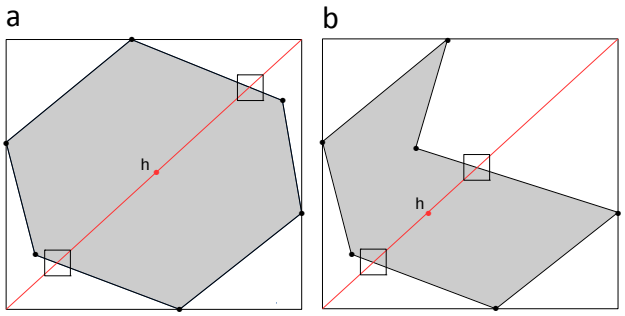

Fig. 6. Examples of holes. h marks the point inside the hole. It is computed as the average of the two intersection points with one of the diagonals.

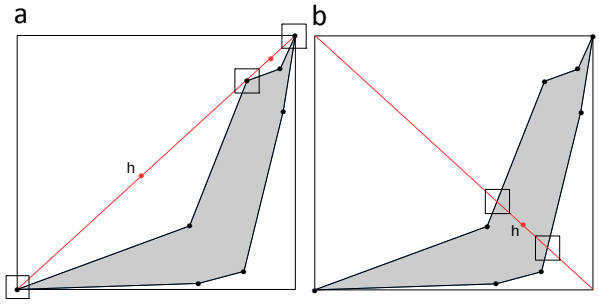

Fig. 7. A case when the average of the intersection points is outside the hole (a). In this case we use the other diagonal to compute h (b).

a segment. This library correctly creates a set of triangles leaving the holes without triangles. Figure 8 shows an example for two disconnected segments, one of them containing a hole, marked in gray in the figure.

6) Removal of External Triangles: In this step, we remove all the triangles that are outside the original segments by calculating the barycenter of each one followed by a point-in-polygon technique, as shown in Figure 8. Since we decided to compute the triangulation algorithm only once due to its cost instead of for each segment separately, this simple procedure correctly eliminates all triangles which are not part of the mesh.

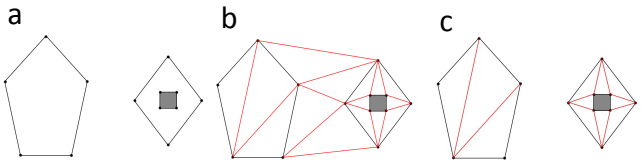

Fig. 8. A set of vertices grouped in two segments with a hole in the second one (hatched) (a). Respective triangulation of the set in red lines before (b) and after (c) the removal of external triangles.

7) Mapping back to 3D: Since the next step, texturing, requires the points in world coordinates, we transform the computed triangles back to world coordinates. This is the last step in remeshing the original objects according to the cutting plane, resulting in a new cut object, triangulated at the defined plane.

B. Texturing the Object

This section describes the steps comprising the process B of Figure 2, *Texturing the Object*. At the end of this process, we will render the cutting plane taking into account the holes and proper texturing. We use a simple solid texturing technique based on the interpolation of images. We interpolate three images representing each of the object's orthogonal planes in order to obtain internal information from different perspectives. Although this solution can be used on its own as a texturing technique, we will explain it as a continuation of the previous section. Given the triangles on the plane already computed and three images defining the object's internal texture, we apply a function that interpolates the images, returning a color for each point according to its spatial location.

1) Color Sampling: Let I be an image formed by a matrix $I_w \cdot I_h$ of pixels, each pixel accessed by $I_{i,j}$ with i as rows and j as columns. The three input images denoted by I_k, $k \in \{1, 2, 3\}$ are set in the orthogonal planes $z = 0$, $y = 0$ and $x = 0$, respectively. Also, they are centered at the origin of the surface object and limited according to the object's bounding box defined by $max = (max_x, max_y, max_z)$ and $min = (min_x, min_y, min_z)$.

Our color sampling technique can compute a color for any point inside this domain although we are only interested in the cutting plane's triangles. Figure 9 shows a graphical representation of the images on the three-dimensional space.

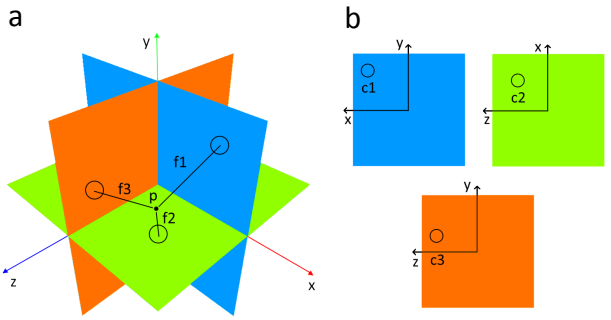

Fig. 9. Color sampling for a point p from three images arranged in the 3D space (a) and their respective color contributions c_k (b).

Let c_k, $k \in \{1, 2, 3\}$ be the color related to a given pixel for each of the input images I_k and p a point defined inside the object's bounding box. The positions I_{k_i} and I_{k_j} of each pixel corresponding to each of the images associated with the point p can be calculated as:

$$I_{1_{i,j}} = \left\{ \left\lfloor \frac{(max_y - p_y)}{(max_y - min_y)} \cdot I_{1_h} \right\rfloor, \left\lfloor \frac{(max_x - p_x)}{(max_x - min_x)} \cdot I_{1_w} \right\rfloor \right\}$$

$$I_{2_{i,j}} = \left\{ \left\lfloor \frac{(max_x - p_x)}{(max_x - min_x)} \cdot I_{2_h} \right\rfloor, \left\lfloor \frac{(max_z - p_z)}{(max_z - min_z)} \cdot I_{2_w} \right\rfloor \right\}$$

$$I_{3_{i,j}} = \left\{ \left\lfloor \frac{(max_y - p_y)}{(max_y - min_y)} \cdot I_{3_h} \right\rfloor, \left\lfloor \frac{(max_z - p_z)}{(max_z - min_z)} \cdot I_{3_w} \right\rfloor \right\}$$

Therefore, each one of the colors associated with the point p can be computed as $c_k = I_{k_{i,j}}$.

2) Calculating the Weight Factors: In this step, we interpolate the three colors computed in the previous step in order to generate a unique color for the point p. This interpolation is based on the distance between p and the images, where each c_k has an associated weight factor when computing the final color.

Let f be a weight factor corresponding to a contribution percentage of an image I on a point p. We calculate the auxiliary factors f_q as:

$$f_q = 1 - \left| 1 - \frac{(max_q - p_q)}{\frac{max_q - min_q}{2}} \right|,$$

where q indicates the axes z, y and x, respectively.

Given the factors f_k related to the images I_k, as in Figure 9, then:

$$f_k = \frac{f_q}{f_x + f_y + f_z}$$

3) Rendering by Texture Mapping: This step has two parts: the first is how to obtain the final color of a point p and the second is how to render the triangles on the cut plane using texture mapping.

We generate the final color c for a given point p as:

$$c = \sum_{k=1,2,3} c_k f_k \tag{1}$$

In order to obtain the final texture for the triangles that define the cutting plane, we use a two-dimensional regular grid on the plane to sample the texture and create a texture map. We set this grid with n lines and m columns on the two-dimensional bounding box defined according to the triangles to be rendered in the plane space.

We compute the colors of these points with equation (1), assigning these colors directly onto a new image I. Further, we define this image as a texture and normalized (u, v) coordinates for the vertices on the plane. Finally, we map the created texture on the triangles and render the image. This process is illustrated in Figure 10.

IV. Results and Discussion

We used OpenGL for the graphics API and the tests ran on an Intel Core i7-4770 CPU 3.40 GHz with 16GB RAM, Windows 8 64 bits and NVIDIA GeForce GTX 770 for graphics. We choose a liver model with blood vessels as a case study to test our solution. We constructed the models from CT images of the SLIVER07 project [23]. We used the Training data - part 1 pack containing about 300 images from the abdominal region already with binary masks to segment the liver, and resolution 512^2 . Then we used MeVisLab [24] to obtain the geometric models (Figures 11-a and 11-b) as detailed in Table I.

We used the textures for the liver from Xue and colleagues [25] where they addressed texture synthesis for surgery simulators and the texture for the vessels from Elhelw et al. [26].

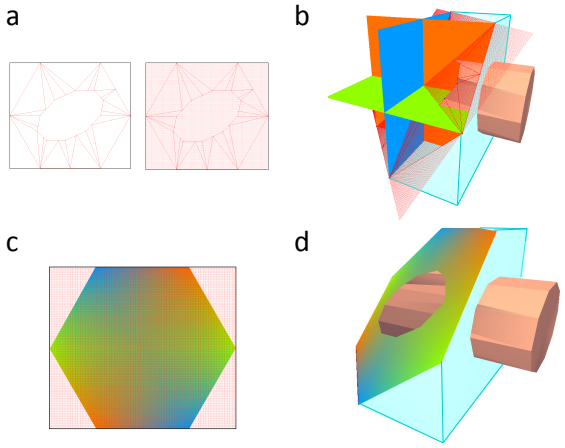

Fig. 10. Rendering and texture mapping process. Given a triangulation and the points on the grid in the plane space (a), the points are represented in the world space (b). Then, the colors are calculated, producing a new texture image (c) that is mapped on the triangles to be rendered (d).

Model	Vertices	Triangles
Liver	3350	6673
Vessels	11216	22662

TABLE I
DETAILS OF THE INPUT MODELS.

We used the same image for the three textures to represent the inner liver. Ideally, three different images should be used. These results can be seen in Figure 11-c. The textured models superimposed are illustrated in Figure 11-d.

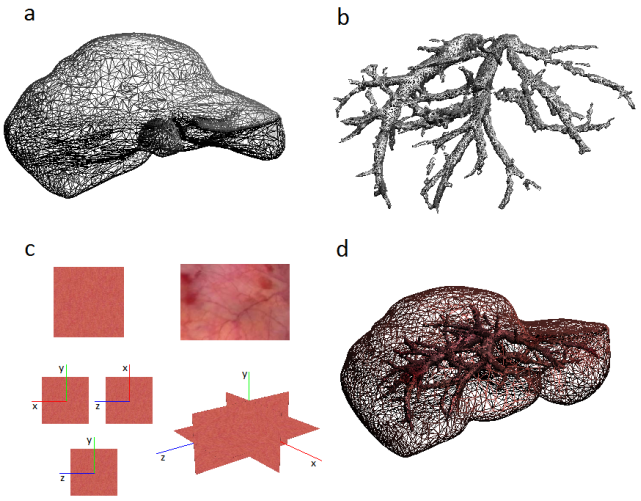

Fig. 11. Overview of the models and textures for the case study. Liver (a) and vessels models (b). Textures for the liver and for the vessels and internal images (c). Textured liver and vessels superimposed (d).

Next we show several results from the inputs, as a set of cuts generated by our system. Table 2 presents information related to these cuts, including the number of intersection surface points (IP), number of triangles on the plane (TP), plane texture grid resolution (GR) and execution time (ET) in milliseconds.

Cut	IP	TP	GR	ET
1	667	657	256	136.21
2	182	178	256	41.32
3	586	600	256	120.8
4	569	569	256	118.02
5	618	647	256	143.25
6	476	498	256	93.86
7	269	273	256	55.58
8	479	492	32	37.73
9	479	492	64	40.42
10	479	492	128	51.06
11	479	492	256	92.49
12	479	492	512	255.65
13	479	492	1024	905.47

TABLE II
INFORMATION ABOUT THE NUMBER OF INTERSECTION POINTS (IP), NUMBER OF TRIANGLES ON PLANE (TP), TEXTURE GRID RESOLUTION (GR) AND EXECUTION TIME (ET) IN MILLISECONDS FOR EACH CUT.

The cuts are organized into different groups to illustrate our technique for various goals. The first cut is shown with full texture (Figure 1-a) and in wireframe (Figure 1-b) to illustrate a plane containing several intersections with vessels.

Cuts 2 to 7 (Figure 12) present six cuts maintaining the same cutting direction. As the grid resolution remains constant, its related execution time increases according to the number of triangles intersecting with the vessels.

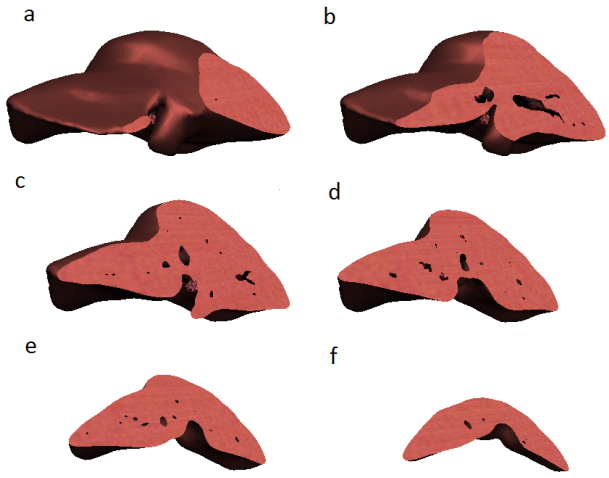

Fig. 12. Cuts 2 (a), 3 (b), 4 (c), 5 (d), 6 (e) and 7 (f) presenting six different cuts maintaining the same direction and grid resolution.

Cuts 8 to 13, shown in Figure 13, use the same cutting plane but with increasing resolution of the grid, zoomed in on the right of each cut. In this sequence, we also see an increase in the rendering time, due to the increasing resolutions of the grid. Also, we can barely see differences in the final rendering with resolutions above 256.

For a video illustrating our technique in action, please visit https://vimeo.com/128415963. This video was captured directly from the screen.

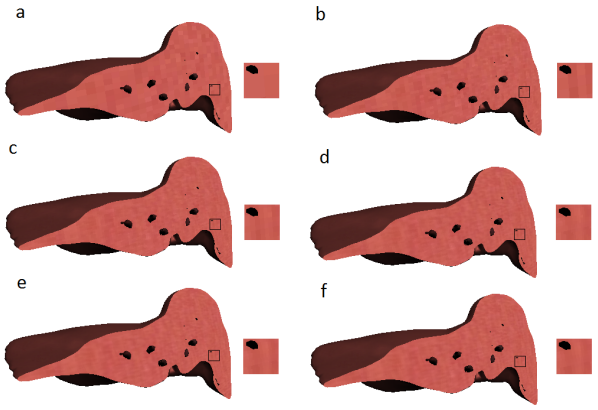

Fig. 13. Cuts 8 (a), 9 (b), 10 (c), 11 (d), 12 (e) and 13 (f) showing the same cutting plane for different grid resolutions.

V. Conclusion

We presented a technique for cuts in objects with internal structures with possible application on surgery simulators. In general, research related to surgical simulators does not deal with internal structures of organs, lacking information that could increase visual realism.

After this paper, a number of new avenues can be explored. First, other image combinations can be tested besides the trilinear approach we presented here. Then, validation studies should be planned to, using photographs of real objects, assess the outcome produced by the different texturing interpolations. For the cutting, we aim to extend our approach to support arbitrary cutting surfaces, such as those defined by implicit equations or 3D meshes, instead of a single planar surface. Further, as those improvements will require higher computation power, we intend to explore parallel implementations on the GPU before integrating our solution into an operational surgery simulator.

VI. Publications

A summary of this dissertation was accepted for publication as a full paper at WSCG 2015 conference.

Acknowledgments

We gratefully acknowledge the partial financial support from FAPERGS through grants 12/1944-2 and 2283-2551/14-8, and CNPq through grants 305071/2012-2 and 478730/2012-8.

References

[1] K. Kunkler, "The role of medical simulation: an overview," *The International Journal of Medical Robotics and Computer Assisted Surgery*, vol. 2, no. 3, pp. 203–210, 2006.

[2] P.-A. Clavien, H. Petrowsky, M. L. DeOliveira, and R. Graf, "Strategies for safer liver surgery and partial liver transplantation," *New England Journal of Medicine*, vol. 356, no. 15, pp. 1545–1559, 2007.

[3] H. Delingette and N. Ayache, "Hepatic surgery simulation," *Communications of the ACM*, vol. 48, no. 2, pp. 31–36, 2005.

[4] G. Echegaray, I. Herrera, I. Aguinaga, C. Buchart, and D. Borro, "A brain surgery simulator," *Computer Graphics and Applications, IEEE*, vol. 34, no. 3, pp. 12–18, 2014.

[5] K. Endo, N. Sata, Y. Ishiguro, A. Miki, H. Sasanuma, Y. Sakuma, A. Shimizu, M. Hyodo, A. Lefor, and Y. Yasuda, "A patient-specific surgical simulator using preoperative imaging data: an interactive simulator using a three-dimensional tactile mouse," *Journal of Computational Surgery*, vol. 3, no. 1, pp. 1–8, 2014.

[6] K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, 1985.

[7] D. R. Peachey, "Solid texturing of complex surfaces," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 279–286, Jul. 1985. [Online]. Available: http://doi.acm.org/10.1145/325165.325246

[8] N. Pietroni, P. Cignoni, M. Otaduy, and R. Scopigno, "Solid-texture synthesis: a survey," *Computer Graphics and Applications, IEEE*, vol. 30, no. 4, pp. 74–89, 2010.

[9] D. Ghazanfarpour and J.-M. Dischler, "Spectral analysis for automatic 3-d texture generation," *Computers & Graphics*, vol. 19, no. 3, pp. 413–422, 1995.

[10] D. Ghazanfarpour and J.-M. DISCHLER, "Generation of 3d texture using multiple 2d models analysis," in *Computer Graphics Forum*, vol. 15, no. 3. Wiley Online Library, 1996, pp. 311–323.

[11] J.-M. Dischler, D. Ghazanfarpour, and R. Freydier, "Anisotropic solid texture synthesis using orthogonal 2d views," *Computer Graphics Forum*, vol. 17, no. 3, pp. 87–95, 1998.

[12] J. Kopf, C.-W. Fu, D. Cohen-Or, O. Deussen, D. Lischinski, and T.-T. Wong, "Solid texture synthesis from 2d exemplars," *ACM Transactions on Graphics (TOG)*, vol. 26, no. 3, p. 2, 2007.

[13] Y. Dong, S. Lefebvre, X. Tong, and G. Drettakis, "Lazy solid texture synthesis," in *Proceedings of the Nineteenth Eurographics Conference on Rendering*, ser. EGSR '08. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, pp. 1165–1174. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2008.01254.x

[14] S.-P. Du, S.-M. Hu, and R. R. Martin, "Semiregular solid texturing from 2d image exemplars," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, no. 3, pp. 460–469, 2013.

[15] B. Cutler, J. Dorsey, L. McMillan, M. Müller, and R. Jagnow, "A procedural approach to authoring solid models," *ACM Trans. Graph.*, vol. 21, no. 3, pp. 302–311, Jul. 2002. [Online]. Available: http://doi.acm.org/10.1145/566654.566581

[16] S. Owada, F. Nielsen, M. Okabe, and T. Igarashi, "Volumetric illustration: designing 3d models with internal textures," *ACM Transactions on Graphics (TOG)*, vol. 23, no. 3, pp. 322–328, 2004.

[17] N. Pietroni, M. A. Otaduy, B. Bickel, F. Ganovelli, and M. Gross, "Texturing internal surfaces from a few cross sections," *Computer Graphics Forum*, vol. 26, no. 3, pp. 637–644, 2007.

[18] K. Takayama, M. Okabe, T. Ijiri, and T. Igarashi, "Lapped solid textures: filling a model with anisotropic textures," vol. 27, no. 3, p. 53, 2008.

[19] E. Praun, A. Finkelstein, and H. Hoppe, "Lapped textures," in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 465–470.

[20] S. Owada, T. Harada, P. Holzer, and T. Igarashi, "Volume painter: Geometry-guided volume modeling by sketching on the cross-section," in *Proceedings of the Fifth Eurographics conference on Sketch-Based Interfaces and Modeling*. Eurographics Association, 2008, pp. 9–16.

[21] K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi, "Volumetric modeling with diffusion surfaces," *ACM Transactions on Graphics (TOG)*, vol. 29, no. 6, p. 180, 2010.

[22] J. R. Shewchuk, "Triangle: Engineering a 2d quality mesh generator and delaunay triangulator," in *Applied computational geometry towards geometric engineering*. Springer, 1996, pp. 203–222.

[23] T. Heimann, B. Van Ginneken, M. A. Styner, Y. Arzhaeva, V. Aurich, C. Bauer, A. Beck, C. Becker, R. Beichel, G. Bekes *et al.*, "Comparison and evaluation of methods for liver segmentation from ct datasets," *Medical Imaging, IEEE Transactions on*, vol. 28, no. 8, pp. 1251–1265, 2009.

[24] MeVis Medical Solutions AG, "Mevislab - medical image processing and visualization," http://www.mevislab.de, MeVis Medical Solutions AG, MeVis Medical Solutions AG, Bremen, 2015.

[25] L. Xue-mei, H. Ai-min, and Z. Qin-ping, "Organ texture synthesis for virtual reality-based surgical simulators," in *Computer Science and Engineering, 2009. WCSE'09. Second International Workshop on*, vol. 1. IEEE, 2009, pp. 238–241.

[26] M. A. ElHelw, B. P. Lo, A. Darzi, and G.-Z. Yang, "Real-time photo-realistic rendering for surgical simulations with graphics hardware," in *Medical Imaging and Augmented Reality*. Springer, 2004, pp. 346–352.