Cuts in Organs with Internal Structures

D. G. Balreira[1], A. Maciel[1], L.T. Cavazzola[2], M. Walter[1]

[1]Institute of Informatics - Federal University of Rio Grande do Sul (UFRGS) - Brazil

[2]Institute of Health Sciences - Federal University of Rio Grande do Sul (UFRGS) - Brazil

dgbalreira, amaciel, marcelo.walter@inf.ufrgs.br, cavazzola@gmail.com

ABSTRACT

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. Simulations present many advantages over other training 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.

Keywords
Computer graphics, visualization system, solid textures, cuts in objects

1 INTRODUCTION

For many applications in Computer Graphics, geometric representations using only the surface of objects (boundary representation) is enough. However, there are some applications that require visualization and possibly interaction with the interior of the objects. Surgical simulators are an example. Surgical operations such as hepatectomy [Clavien et al., 2007] for instance, require cutting the liver in distinct regions depending on the condition of each patient. Furthermore, this possibility is missing from current research on virtual simulators [Delingette and Ayache, 2005] [Echegaray et al., 2014] [Endo et al., 2014] which 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. This
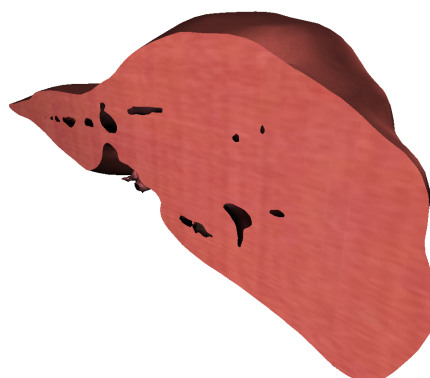
Figure 1: Result from our technique. A liver with a cut showing the blood-vessels as empty shapes. Information about this result as cut 1 in Table 2.

paper proposes a new technique that allows arbitrary cuts in objects, particularly anatomical structures. The technique allows not only the reconstruction of the surface's texture in the cutting zone, but also takes into account the object's internal structures. Our approach considers the geometric object as a whole, composed by surface and inside, and reconstructs the mesh in the cutting zone. Finally, it renders an internal texture on the cutting surface. In Fig. 1 we show an example of a result from our approach. Next generation surgical simulators, for instance, can benefit from this approach in

order to improve realism, and, therefore, skills development.

2 RELATED WORK

In this section, we discuss selected related work in the field of solid texturing.

The idea of solid texturing or 3D texturing was introduced in 1985 [Perlin, 1985] [Peachey, 1985] 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 [Pietroni et al., 2010] presents a detailed review of several techniques for solid texturing and texture synthesis. From the original papers, other researchers [Ghazanfarpour and Dischler, 1995] [Ghazanfarpour and Dischler, 1996] extended the idea using spectral analysis and texture synthesis applied to 2D orthogonal views [Dischler et al., 1998]. Another method adapted a texture synthesis process called optimization-based with histogram matching, globally minimizing its energy function [Kopf et al., 2007]. In the following year, a procedure introduced a new technique to synthesize solid textures restraining them to a voxel subset, forcing spatial determinism [Dong et al., 2008]. Recently, a new approach tried to deal particularly with regular and semiregular patterns [Du et al., 2013].

Cutler et al. [2002] presented a scripting language in order to define internal layers for objects. Two years later, [Owada et al., 2004] 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. [2007] used few images representing cross sections of an object in order to render any point inside it. Takayama's research [Takayama et al., 2008] extended the concept of lapped textures [Praun et al., 2000] to solid textures, covering the whole object's volume instead of only its surface. In the same year, a new system called volume painter [Owada et al., 2008] projected volumes from sketches defined by the user. In 2010, a new concept called diffusion surfaces [Takayama et al., 2010] was able to render structures that have a smooth color variation in its internal structures, like fruits and vegetables.

From the above review, it can be understood that, in the current state-of-the-art on solid texturing, no technique deals 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.

3 OUR TECHNIQUE

This section presents our technique for dealing with arbitrary cuts in objects with internal structures. 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.

Then, 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 3 presents the pipeline of our technique with the two primary processes labeled: A-Remeshing the Objects, detailed in Section 3.1; and B-Texturing the Object, detailed in Section 3.2.

3.1 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.
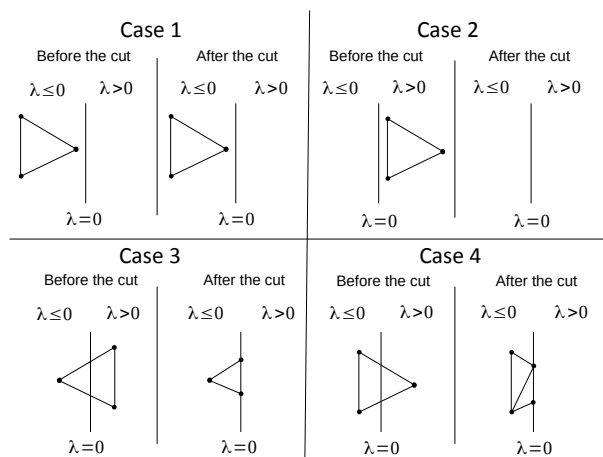

Figure 2: Possible cases of cuts applied to each triangle of the mesh in 2D view.

3.1.1 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 2.

For all cases, the three vertices of each triangle are tested. In case 1, if λ of all vertices satisfy $\lambda \leq 0$, they
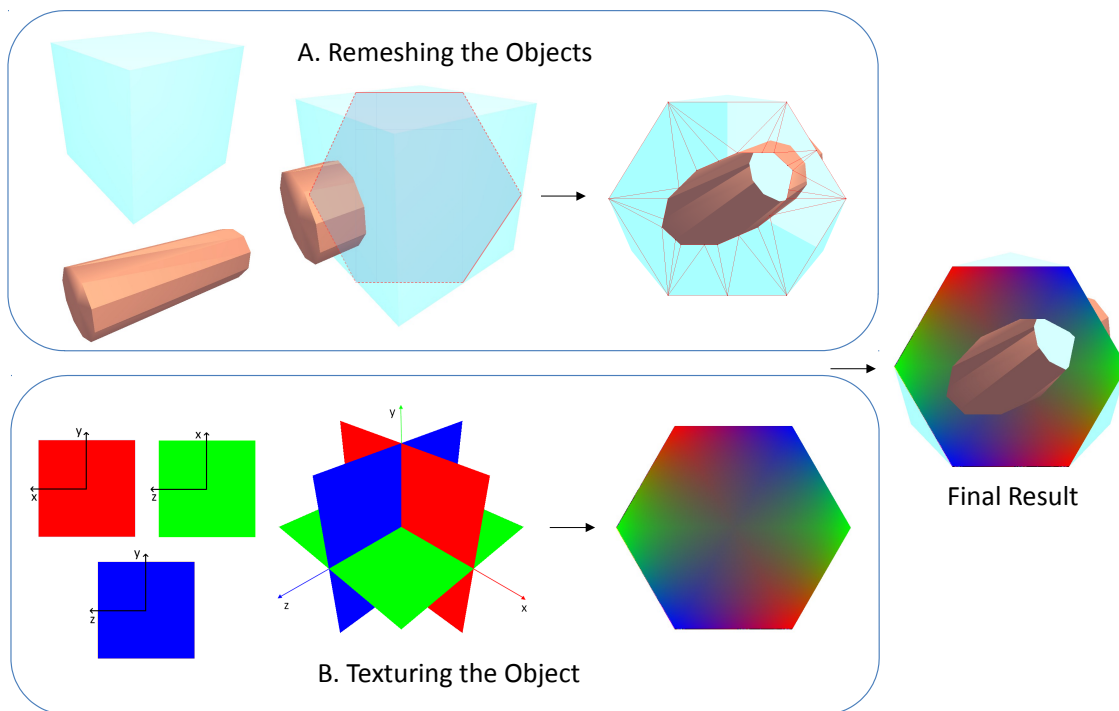
Figure 3: Overview of our technique. Given two 3D meshes, a set of three images and a cutting plane as input, our technique calculates the resulting geometry (A) and a texture to be mapped on the cutting zone (B).

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.
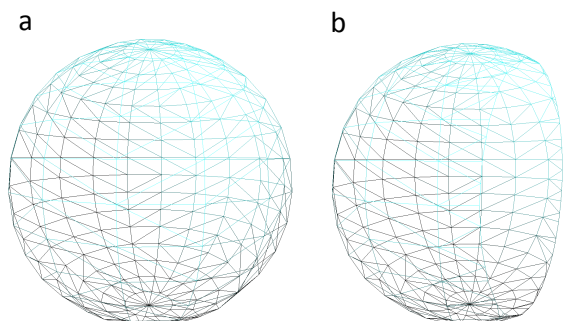
3.1.2 Segmentation in Topologically Connected Sets

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 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 their respective order. Figure 5 shows an example where two connected sets are detected.
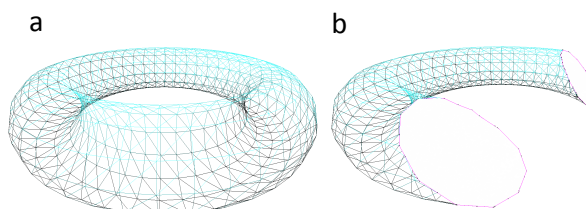

Figure 4: Example of a sphere model before (a) and after (b) the cut.

Figure 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.

3.1.3 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.

3.1.4 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. 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).
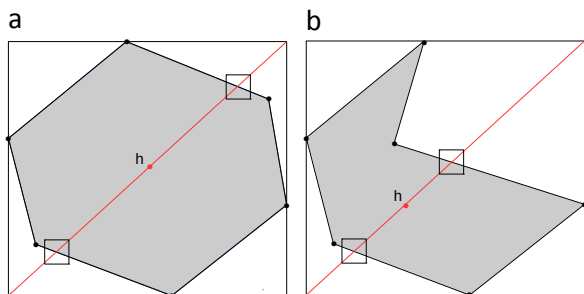

Figure 6: Examples of holes. h marks the point inside the hole. It is computed as the average of the two intersection points with one of the diagonals.

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.
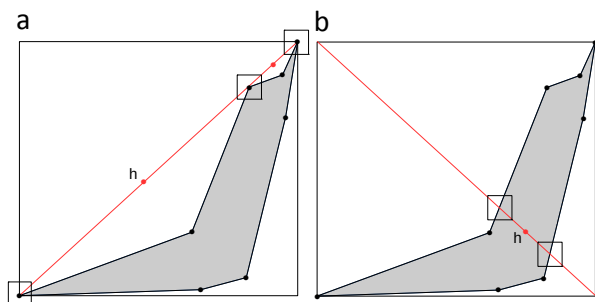

Figure 7: A case when the average of the intersection points is outside the hole (a). In this case we use the other diagonal to compute h (b).

3.1.5 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 [1996] to triangulate the vertices that define 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.

3.1.6 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.
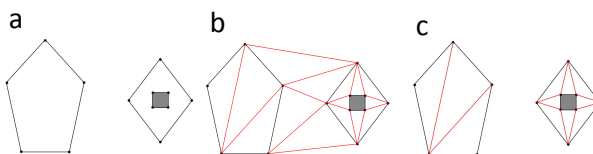

Figure 8: A set of vertices grouped in two segments with a hole in the second one (hatched) (a). Respective triangulation of the set in red lines before (b) and after (c) the removal of external triangles.

3.1.7 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.

3.2 Texturing the Object

This section describes the steps comprising the process B of Figure 3, *Texturing the Object*. At the end of this process, we will obtain a rendering for the cutting plane taking into account the holes. We use a simple solid texturing technique based on the interpolation of images. 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.

3.2.1 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.
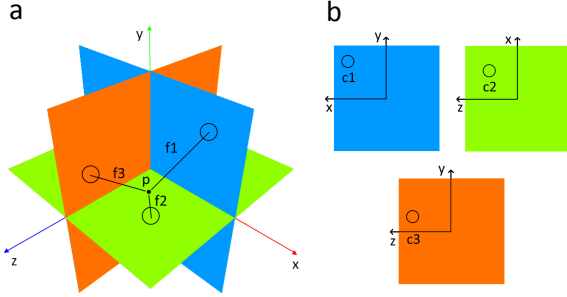

Figure 9: Color sampling for a point p from three images arranged in the 3D space (a) and their respective color contributions c_k (b).

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 $c_k = I_{k_{i,j}}$.

3.2.2 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:

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

3.2.3 Rendering by Texture Mapping

This step has two main goals: the first shows how to obtain the final color of the point p and the second explains how to render the triangles on the cut plane using texture mapping.

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

$$c = \sum_{k=1,2,3} c_k f_k \qquad (1)$$

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.
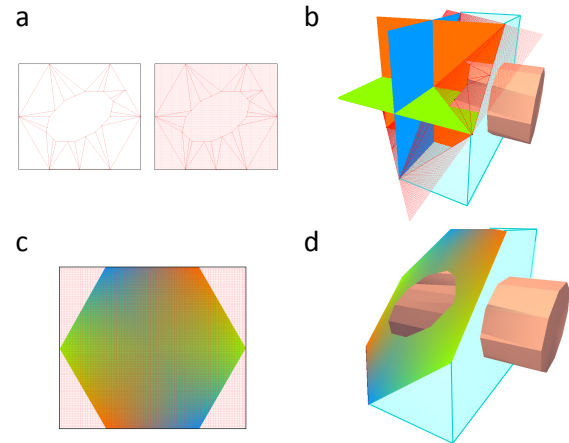

Figure 10: Rendering and texture mapping process. Given a triangulation and the points on the grid in the plane space (a), the points are represented in the world space (b). Then, the colors are calculated, producing a new texture image (c) that is mapped on the triangles to be rendered (d).

4 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 [Heimann et al., 2009]. We used the Training data - part 1 pack containing about 300 images with resolution 512x512 from the abdominal region already with binary masks to segment the liver. Then we used MeVisLab to obtain the geometric models (Figures 11-a and 11-b) as detailed in Table 1.

Model	Vertices	Triangles
Liver	3350	6673
Vessels	11216	22662

Table 1: Details of the input models.

We used the textures for the liver from Xue-mei et al. [2009] where they addressed texture synthesis for surgery simulators and the texture for the vessels from ElHelw et al. [2004]. 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 separately and superimposed are illustrated in Figures 11-d, 11-e and 11-f, respectively.

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.

The cuts are organized into different groups to illustrate our technique for various goals. The first cut is shown with full texture (Figure 1) and in wireframe (Figure 12) to illustrate a plane containing several intersections with vessels.

Cuts 2 to 7 (Figure 13) 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.

Cuts 8 to 13, shown in Figure 14, 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.

5 CONCLUSIONS

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.

Although we presented our technique focusing on organs, it can also deal with other types of three-dimensional models that have an internal structure. For the sake of generalization, we tested our approach using a watermelon model. Results are presented in Figure 15. The seeds are geometrically modeled as internal objects. The watermelon texture at the cutting plane looks plausible, and the seeds are correctly cut.

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.

6 ACKNOWLEDGMENTS

REFERENCES

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*, 356(15):1545–1559, 2007.

B. Cutler, J. Dorsey, L. McMillan, M. Müller, and R. Jagnow. A procedural approach to authoring solid models. *ACM Trans. Graph.*, 21(3), July 2002.

H. Delingette and N. Ayache. Hepatic surgery simulation. *Communications of the ACM*, 48(2):31–36, 2005.

J.-M. Dischler, D. Ghazanfarpour, and R. Freydier. Anisotropic solid texture synthesis using orthogonal 2d views. 17(3):87–95, 1998.

Y. Dong, S. Lefebvre, X. Tong, and G. Drettakis. Lazy solid texture synthesis. In *Proceedings of the Nineteenth Eurographics Conference on Rendering*, EGSR '08, pages 1165–1174, 2008.

S.-P. Du, S.-M. Hu, and R. R. Martin. Semiregular solid texturing from 2d image exemplars. *IEEE Transactions on Visualization and Computer Graphics*, 19 (3):460–469, 2013. ISSN 1077-2626. doi: http://doi. ieeecomputersociety.org/10.1109/TVCG.2012.129.
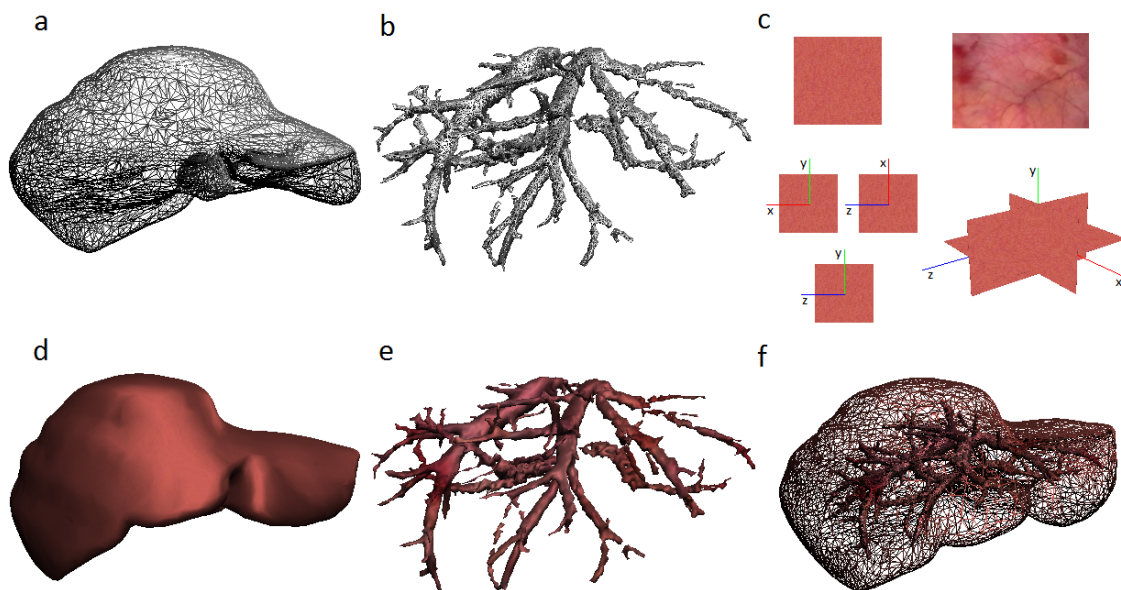
Figure 11: Overview of the models and textures for the case study. Liver (a) and vessels models (b). Textures for the liver and for the vessels and internal images (c). Textured liver (d) and vessels (e), and both superimposed (f).

Cut	1	2	3	4	5	6	7	8	9	10	11	12	13
IP	667	182	586	569	618	476	269	479	479	479	479	479	479
PT	657	178	600	597	647	498	273	492	492	492	492	492	492
GR	256	256	256	256	256	256	256	32	64	128	256	512	1024
ET	136.21	41.32	120.8	118.02	143.25	93.86	55.58	37.73	40.42	51.06	92.49	255.65	905.47

Table 2: 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.
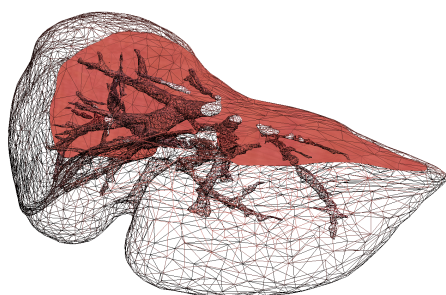

Figure 12: Cut illustrating a plane with several intersections with the vessels in wireframe. Same result presented in Fig. 1
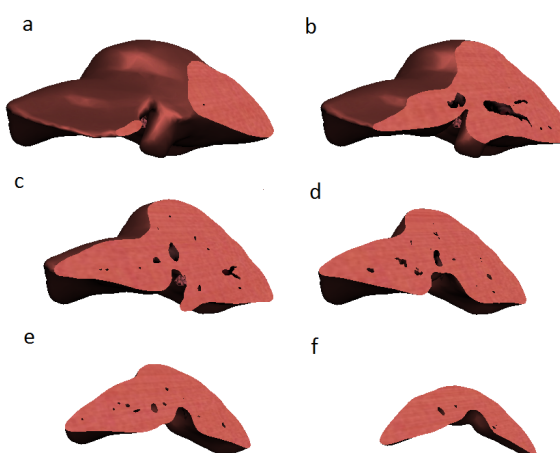

Figure 13: Cuts 2 (a), 3 (b), 4 (c), 5 (d), 6 (e) and 7 (f) presenting six different cuts maintaining the same direction and grid resolution.

G. Echegaray, I. Herrera, I. Aguinaga, C. Buchart, and D. Borro. A brain surgery simulator. *Computer Graphics and Applications, IEEE*, 34(3):12–18, 2014.

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*, pages 346–352. Springer, 2004.

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*, 3(1):1–8, 2014.
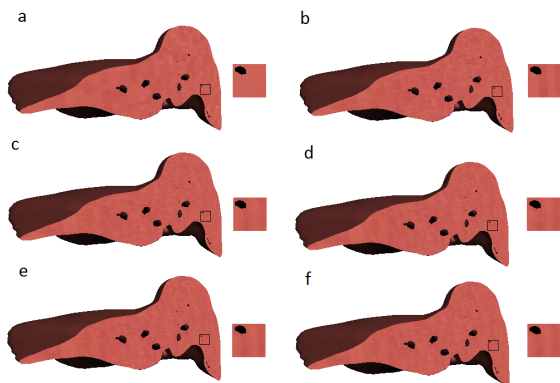
Figure 14: Cuts 8 (a), 9 (b), 10 (c), 11 (d), 12 (e) and 13 (f) showing the same cutting plane for different grid resolutions.
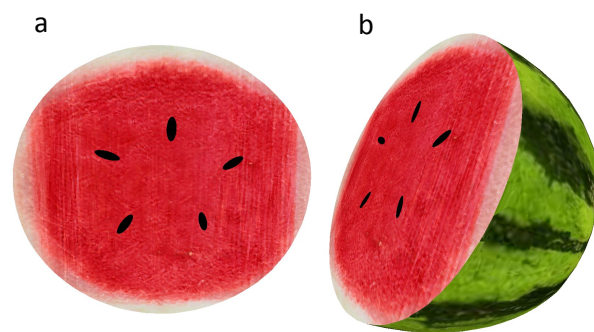

Figure 15: Watermelon cut using our technique.

D. Ghazanfarpour and J.-M. Dischler. Spectral analysis for automatic 3-d texture generation. *Computers & Graphics*, 19(3):413–422, 1995.

D. Ghazanfarpour and J.-M. M. Dischler. Generation of 3D texture using multiple 2D models analysis. 15 (3):C311–C323, C478–C479, Sept. 1996.

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*, 28(8):1251–1265, 2009.

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)*, 26(3):2, 2007.

S. Owada, F. Nielsen, M. Okabe, and T. Igarashi. Volumetric illustration: designing 3d models with internal textures. *ACM Transactions on Graphics (TOG)*, 23 (3):322–328, 2004.

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*, pages 9–16. Eurographics Association, 2008.

D. R. Peachey. Solid texturing of complex surfaces. *SIGGRAPH Comput. Graph.*, 19(3), July 1985.

K. Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, 1985.

N. Pietroni, M. A. Otaduy, B. Bickel, F. Ganovelli, and M. Gross. Texturing internal surfaces from a few cross sections. 26(3):637–644, 2007.

N. Pietroni, P. Cignoni, M. Otaduy, and R. Scopigno. Solid-texture synthesis: a survey. *Computer Graphics and Applications, IEEE*, 30(4):74–89, 2010.

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

J. R. Shewchuk. Triangle: Engineering a 2d quality mesh generator and delaunay triangulator. In *Applied computational geometry towards geometric engineering*, pages 203–222. Springer, 1996.

K. Takayama, M. Okabe, T. Ijiri, and T. Igarashi. Lapped solid textures: filling a model with anisotropic textures. 27(3):53, 2008.

K. Takayama, O. Sorkine, A. Nealen, and T. Igarashi. Volumetric modeling with diffusion surfaces. *ACM Transactions on Graphics (TOG)*, 29(6):180, 2010.

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*, volume 1, pages 238–241. IEEE, 2009.