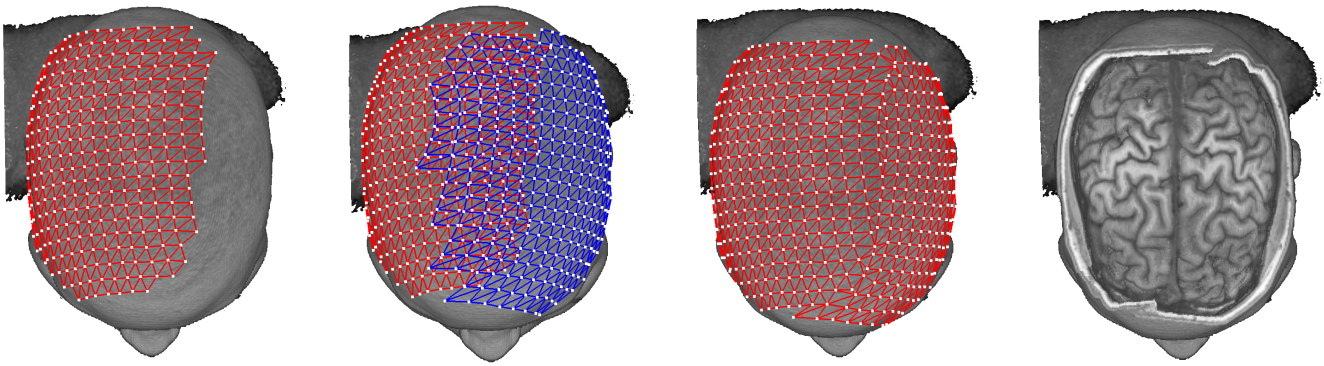
Curvilinear Reformatting in Native Space based on Mesh Zippering and Offsetting

Wallace Souza Loos, Wu, Shin-Ting
Department of Computer Engineering and Automation
School of Electrical and Computer Engineering, University of Campinas,
Campinas, Brazil
http://www.dca.fee.unicamp.br/{wloos,ting}



(a) (b) (c) (d)

Fig. 1. Curvilinear reformatting in native space: (a) the first region is selected and the mesh is built, (b) then the second region is selected and another mesh is built, (c) the meshes are zippered and (d) the curvilinear reformatting is performed.

***Abstract*—Most curvilinear reformatting algorithms are not processed on native space, which makes them inappropriate to be used for neuronavigation purposes. In this work, we present an interactive curvilinear reformatting algorithm that is performed in native space. The user selects the desired regions. Then, they are sampled in order to build their corresponding meshes, which are representations of the selected regions. A single complete mesh is formed by zippering the meshes that overlap each other. After that, we use an offset algorithm to crop the region corresponding to the complete mesh. In this work we present two algorithms: the zippering and the offset algorithms, which allow us to do larger crops in the brain in interactive time.**

***Keywords*-zippering; depth-map; curvilinear reformatting; focal cortical dysplasia (FCD)**

I. Introduction

Curvilinear reformatting is a noninvasive computational exploration technique, that computes a series of equidistant curvilinear slices on 3D neuroimaging data without physically entering the brain. Once it improves the anatomical display of the gyral structure of the hemispheric convexities [1], it is useful for finding lesions of focal cortical dysplasia, a common cause of refractory epilepsy. This is because that this kind of dysplasia is often associated with a blurring of the gray-white matter junction, spread along curvilinear layers of the brain parallel to the brain surface in the areas directly below the skull, and small dysplastic lesions are often missed by conventional MRI methods, such as multiplanar reformatting [1], [2].

Bastos *et al.* [1] have performed curvilinear reformatting on 3D neuroimaging data using an in-house developed software (Brainsight). In order to do it, manual delineation on 2D slices of the brain is needed to obtain the interpolation among them for extracting the curvilinear surface of the brain. This procedure demands much time for routine clinical use. Bergo and Falcão [3] and Huppertz *et al.* [2] overcame this limitation by proposing an automatic curvilinear reformatting method. Bergo and Falcão [3] have applied the Image Foresting Transform (IFT) to segment the envelope of the brain. Then, the euclidean distance transform is used to extract its isosurfaces. Huppertz *et al.* [2] have applied predefined masks to segment the brain in order to remove the skull and outer regions in different depths. Thus the outer brain regions are peeled by layers parallel to the brain surface. Differently from previous works, Wu *et al.* [4] presented a way to make the curvilinear reformatting in the native space, i.e. in the patient-centered reference system. Its implementation is based on the algorithm of visibility and mesh offsetting. The user selects a region on the visible part on which the curvilinear reformatting is carried out. Because the native space is preserved, the procedure

M.Sc. dissertation

can be used directly for neuronavigation purposes. However, only one visible region can be selected. For this reason a comparative diagnosis is not possible when the region to be analyzed is not in a visible region.

Contributions: This work proposes an approach to overcome the limitation of the curvilinear reformatting presented in [4] using a zippering algorithm based on depth-maps and also introducing a new way to avoid local self-intersections. Fig. 1 shows the reformatting curvilinear result after zippering the meshes.

A. Related work

In order to make the curvilinear reformatting, we need first to zipper the meshes and then compute the offset. The basic ideia for zippering meshes is: 1) to remove the overlap area and 2) to remesh. Turk and Levoy [5] have presented a method for combining a collection of range images into a single polygonal mesh. The overlapping portions of the meshes are removed, only triangles with all vertices on the overlap area are removed. Then, for an edge of a triangle on a mesh that intersects the boundary of another mesh, a new vertex is created and the triangulation is performed. Shostko *et al.* [6] have presented a method based on an algorithm that automatically generates a surface mesh from intersecting surfaces by means of Boolean intersection/union operations. Figueiredo *et al.* [7] have presented an algorithm for intersecting finite-element meshes defined on parametric surface patches. The algorithm proposed has three main steps: 1) determination of the intersection points, 2) determination of the trimming curves and 3) remeshing. The mesh intersections are computed using numerical methods and the trimming curves are used to join the meshes. The previous works suit well to our problem; however our meshes geometries are simple and almost coplanar, therefore to take advantage of these features, we decided to propose another approach.

The main problem we face in triangular mesh offsetting is local self-intersection. Some local self-intersections can appear during the mesh offset due to the degenerated triangles. The global self-intersections are not treated in this work, because our geometries are open and convex. The basic idea for sel-intersection free mesh offsetting in most work is: 1) to check at each iteration whether it occurs, and 2) to discard it colliding triangles. Jung *et al.* [8] presented an algorithm to remove self-intersections from the triangular offset mesh. It can be summarized in 4 steps: 1) using topological operators, edge collapse and flip edge, remove the degenerated triangles 2) compute the self-intersections (local and global), 3) find a valid seed triangle, and 4) the valid region grows from the seed triangle to neighboring triangles and invalid triangles are removed. A valid seed triangle is a triangle which belongs to a set of triangles that defines the outer boundary of the offset volume from the raw triangular mesh. However, their method works only if the mesh geometry is closed. They use a numerical method to compute the intersections.

Yi *et al.* [9] have presented a mitered offsetting method of a triangular mesh. A modified version of quadric error metrics (QEM) is used to compute the offset vertex position. The offset vertex position computation can be formulated as a least square problem. If the vertex position which minimizes the sum of squared distance error from the triangles around the original mesh vertex is within tolerance, the offset vertex is accepted. Otherwise, the offset vertex is split repeatedly until the error is acceptable. Vertex split occurs at the sharp features. As a post-processing step, self-intersections can be removed by the method in [8]. Liu *et al.* [10] presented an offset surface generation to construct intersection-free offset surfaces. The idea of the algorithm is to sample the offset surface into a uniform grid and then employ a contouring algorithm to build the resultant offset mesh surface from the signed distance-field. The cost to compute the signed distance-field is not feasible for interactive time. Wu *et al.* [4] developed a way to avoid the local self-intersections by removing the degenerate triangles in a preventive way. For each displacement, each triangle of the mesh is checked whether some triangles have their area smaller than a predefined threshold. All triangles with area smaller than a predefined threshold are removed. However, these local operations can create problems in the geometry of the mesh, since they are dependent of the sequence they are performed.

II. Algorithm Overview

In order to understand why the method proposed in [4] does not allow comparative diagnosis, we provide a brief description of the algorithm in this section. Initially, a user selects a ROI by brushing with the cursor on the head's surface. When the interaction is finished, the region is sampled and a mesh is built. A mesh is a collection of vertices, edges and faces that approximate the scalp's geometry. It plays an important role in the curvilinear reformatting, because it allows us to apply known mesh oriented geometric procedures for computing intersection-free laminar layers. The mesh is displaced in the direction of the vertices' normal vector until a previously specified depth is reached. For each displacement, the mesh is voxelized in the same resolution of the raw brain volume.

The voxelization is used to find the voxels which the mesh intersects, and these voxels are tagged with their depth relative to the scalp. The voxelization algorithm is an adaptation of the depth map based algorithm proposed in [11]. As the user can select only what he can see, for selecting another region it is necessary to rotate the head. Nevertheless every time that the user makes a selection, a mesh is built, then if s/he turns the head to select another region a new mesh is built. Now to ensure that the transitions between the meshes are smooth we need to zipper these meshes.

III. Problems

Based on the scalp geometry, we assume that: the meshes used to approximate the surfaces of the scalp are convex and open. Then, we can formulate two questions: 1) How to join two almost coplanar meshes? and 2) How can we avoid the local self-intersection in the mesh offsetting?

We propose to apply the zippering algorithm to join the meshes. It is important to mention that the union is made pairwise and only on overlapped meshes. After a user selects an area to be explored, we join it immediately with the existing mesh. Then, to explore the fact that our surfaces are almost coplanar, we decided to use depth-maps to replace the costly computations needed to find the mesh intersection. The depth-maps are used to identify which triangles are in the regions that overlap. If a triangle belongs to both depth-maps, the triangle is in an overlap region. After the overlapping triangles are identified, the overlapping triangles of one of the meshes are removed. After that the meshes are zippered.

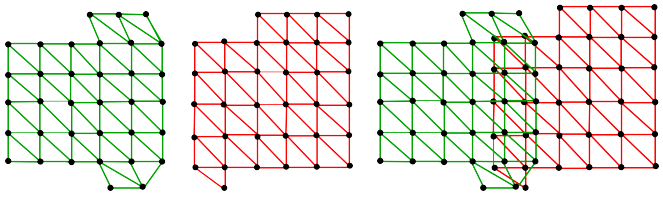
For avoiding the local self-intersection, we propose to use a simplification algorithm to remove the degenerated triangles and, consequently, the local self-intersections. This is because that using a simplification algorithm we increase the area of the triangles.

IV. Zippering based in Depth-map

We divided the zippering algorithm into 4 steps: 1) identifying the overlap area, 2) finding transition vertices, 3) getting new fronts, and 4) joining the meshes.

A. Identifying the Overlap Area

To identify the regions that overlap, the depth-maps are acquired for both meshes. The procedure to acquire the depth-maps are: the meshes are rendered individually, and their depth-maps are saved in an offscreen buffer. As the meshes are almost coplanar, their depth-maps projected on the screen should overlap. To discover which triangles are in the overlap area, it is necessary to find which triangles are in the area where the depth-maps are overlapping. It is the common area between the depth-maps, $(D_{M_1} \cap D_{M_2})$. Without loss of generality, choose one mesh, M_1, and for each edge, e_i, of the triangle, t_i, it is sampled and applied to a map that takes it from the world coordinates to the screen coordinates, $T{:}R^3 \rightarrow R^2$. Then for each sampled point x, we apply the coordinate transformation T to find its screen coordinate. Thus if $T(x) \in (D_{M_1} \cap D_{M_2})$, then the triangle belongs to the overlap area. This process is made for both meshes. Fig. 2 shows the



(a) M_1. (b) M_2. (c) M_1 and M_2 overlapped.

Fig. 2. Meshes: M_1 and M_2.

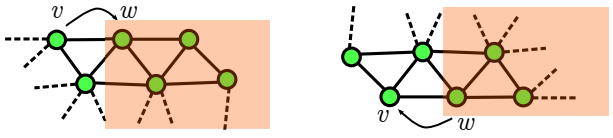
meshes individually (Fig. 2(a) and Fig. 2(b)) and when they are overlapped (Fig. 2(c)). Their respective depth-maps are shown in Fig. 3. The overlapped area $(D_{M_1} \cap D_{M_2})$ is colored in black in Fig. 3(c).

(a) Depth-map D_{M_1}. (b) Depth-map D_{M_2}. (c) D_{M_1} and D_{M_2} overlapped.

Fig. 3. Depth-maps: D_{M_1} and D_{M_2}.

B. Finding the Transition Vertices

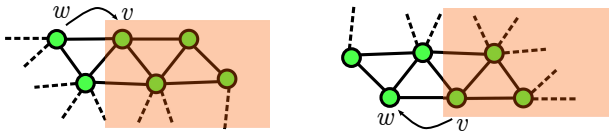
There are two kinds of transition vertices: external and internal. These vertices help to define the new fronts of the meshes with the overlap area removed, as explained in IV-C. Fronts are the boundaries made for the vertices between the transition vertices. Let B be the set of vertices that belong to the boundary of the mesh M_1. Let A be the set of vertices that belong to the overlap area of the meshes M_1 and M_2. The external vertex is a vertex $v \in (B \setminus A)$ and has one vertex adjacent $w \in (B \cap A)$. Fig. 6 shows the external vertices of the mesh M_1. The internal vertex is a vertex $v \in A$ and has



(a) First external vertex. (b) Second external vertex.

Fig. 4. External transition vertices.

one vertex adjacent $w \in (B \setminus A)$. Fig. 5 shows the internal vertices of the mesh M_2. Fig. 6 shows the external and internal vertices on the meshes M_1 and M_2. After getting the transition



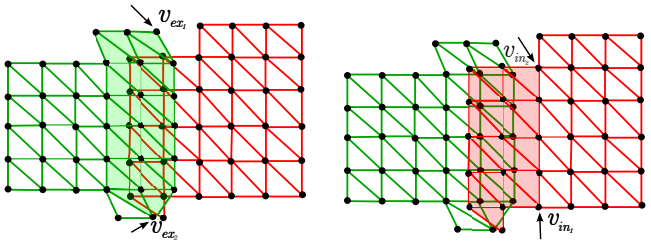
(a) First internal vertex. (b) Second internal vertex.

Fig. 5. Internal transition vertices.

vertices we remove all triangles of one mesh which are in the overlap region.

C. Getting the New Fronts

A front is a part of the boundary used to join the meshes. The new fronts are created by the sequence of vertices that connects the transition vertices, after removing triangles in the overlap area. For one mesh, we obtain the external transition vertices and for the other the internal transition vertices. The external front is the front made of the vertices connecting the external transition vertices, and the internal front is the front made of the vertices that connect internal transition vertices.



(a) External transition vertices. (b) Internal transition vertices.

Fig. 6. Transition vertices are shown on the meshes.

D. Joining the meshes

Due to the overlap area of the meshes being almost coplanar, the fronts that are at a distance less than or equal to one voxel, are used to join the meshes. The first and the last vertices of the internal front are merged with the closest vertices of the external front. The other vertices of the internal front are merged with other vertices of external front, only if the minimal distance is smaller than a pre-specified threshold. Fig. 7 shows this process. The first and the last vertices of the

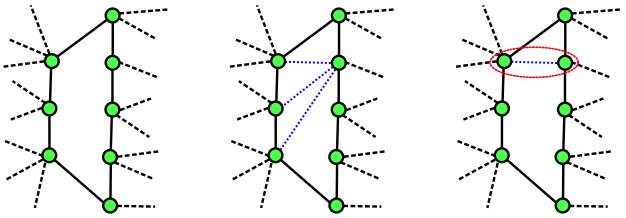
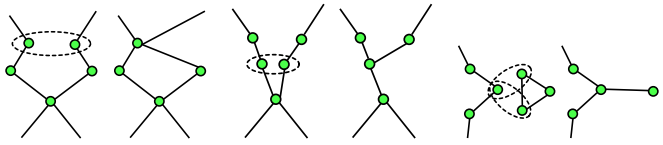


Fig. 7. Merging the closest vertices: (a) the first and the last are contracted, (b) the minimal distance is computed and (c) if the minimal distance is smaller than threshold fixed the vertex is contracted.

internal front are merged with the closest vertex of the external front (Fig. 7(a)), then for the other vertices the distances are computed (Fig. 7(b)), and the closest distance is chosen (Fig. 7(c)). If this distance was smaller than the threshold fixed, then the vertices are merged. There are three configurations after merging: 1) holes (Fig. 8(a)), 2) edge fusion (Fig. 8(b)), and 3) triangle collapse (Fig. 8(c)). Only in the case of holes



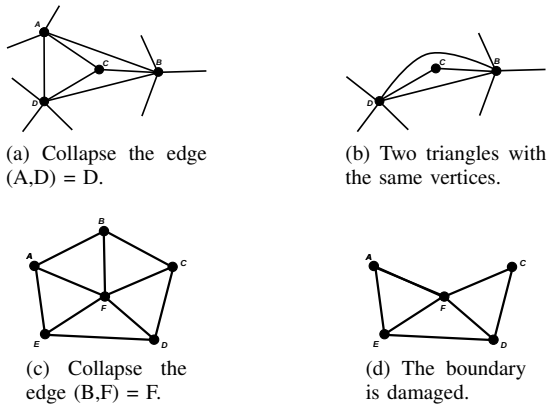
(a) Holes. (b) Edge fusion. (c) Triangle collapse.

Fig. 8. As result of merging two vertices we have the following cases: holes, edge fusion and triangle collapse.

further processing is necessary to fill them. We project the vertices of the hole on a plane and then we triangulate these vertices based on their internal angles.

V. Intersection-Free Mesh Offsetting

In this work we used the simplification algorithm proposed in [12]. The algorithm computes the cost to contract a pair of vertices using quadric error metrics (QEM). After the costs have been computed, the contractions are made from the lowest to the highest cost. In addition, we introduce penalties to avoid illegal contractions. In our work, illegal contractions are: 1) contractions that lead to two faces with the same set of vertices, such as the contraction of the edge (A,D) to the vertex D in Fig. 9(a) and results in the situation presented in Fig. 9(b), and 2) contractions of edges with at least one vertex on the boundary that may alter the boundary shape, as illustrated in Fig. 9(c) Fig. 9(d). The contraction of the edge (B,F) to the vertex F in Fig. 9(c) deforms the mesh's contour as depicted in Fig. 9(d).



(a) Collapse the edge (A,D) = D. (b) Two triangles with the same vertices.

(c) Collapse the edge (B,F) = F. (d) The boundary is damaged.

Fig. 9. Illegal contractions.

To prevent degenerate triangles, triangles whose vertices are tending to be colinear, we use the same heuristic that was used in [14], called measure of triangle compactness

$$\lambda = \frac{4\sqrt{3}w}{l_1^2 + l_2^2 + l_3^2}, \tag{1}$$

where the l_i are the lengths of the edges and w is the area of the triangle. It is assigned 1 to an equilateral triangle and 0 to a triangle whose vertices are colinear. For each displacement, it is checked if a triangle is a candidate to turn degenerate, in case of any candidate is found, the simplification algorithm is used. To avoid that the simplification algorithm remove too many triangles, we set a maximum number of triangle that can be removed.

VI. Results

The experiment images were acquired by Philips Achieva 3T Hospital of University of Campinas. The dimension of the volumes were 180 × 240 × 240. We performed the experiments on a *desktop* Intel ®Core i7 2.8 GHz with 8GB of RAM and graphic card NVIDIA GeForce GTX 650 Ti with 2GB of VRAM. The operating systems used were: *Windows* 7 and *Ubuntu* 14.02. The language used was C++ and the

TABLE I
TIME PERFORMANCES OF ZIPPERING SHOWN IN FIG. 11.

Mesh #1	Mesh #2	Mesh #3	Time(s)
372	361	665 (Fig. 11(a))	0.472
665	352	940 (Fig. 11(b))	0.384
940	167	1054 (Fig. 11(c))	0.170
1054	244	1204 (Fig. 11(d))	0.211
1204	126	1302 (Fig. 11(e))	0.092
1302	125	1361 (Fig. 11(f))	0.116

libraries were $wxWidgets$ and $OpenGL$. The data structures used to represent the mesh was half-edge.

The first experiment validates the zippering algorithm. We built a mesh as result of a sequence of zippering. Fig. 10 depicts the final mesh created on the head of a patient and Fig. 11, the meshes that were zippered. Table I shows the total time in seconds needed to make the zippering of the meshes showed in Fig. 11.

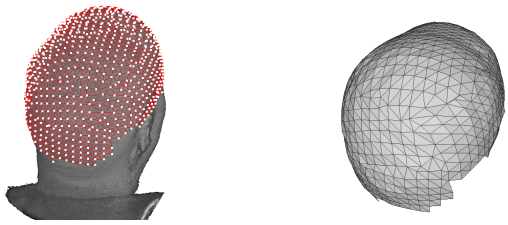


Fig. 10. Final mesh as the result of the zippering algorithm:(a) mesh on the head of the patient and (b) mesh being seen individually.

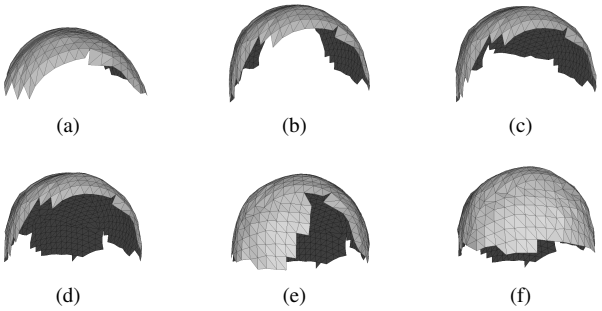


Fig. 11. Mesh zippering process.

The second experiment assesses the offset algorithm using the simplification algorithm to remove the degenerate meshes. Fig. 12 shows three meshes in different depths. The first mesh (depth = 0.0mm) begins to be displaced until some degenerate face is detected. When it occurs, the simplification algorithm is used and then the displacement continues. In our work we defined a maximum depth = 64.5mm. This depth is sufficient to our application. Fig. 13 shows the mesh on the patient's head and the local curvilinear reformatting.

The last experiment tests our curvilinear reformatting using the zippering algorithm and the algorithm of mesh offsetting. Fig. 14 shows the result of the curvilinear reformatting made over the top of the head. Fig. 15 shows another view of the curvilinear reformatting and Fig. 16 shows the curvilinear reformatting being performed only in the back region of the head. Table II shows the total time in seconds needed to make the curvilinear reformatting presented in Fig. 14, Fig. 15 and Fig. 16.

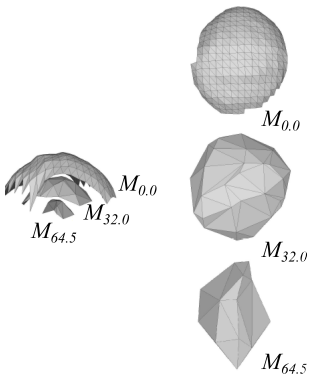


Fig. 12. Meshes in different depths: 0.0mm, 32.0mm and 64.5mm. The simplification algorithm was used to prevent degenerate triangles.

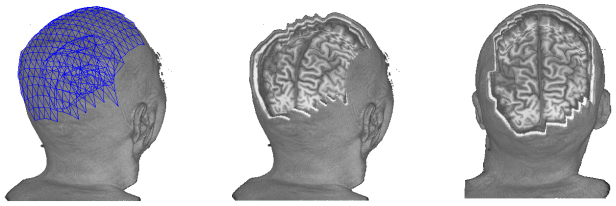


Fig. 13. The result of the curvilinear reformatting: (a) the mesh displacements (the same meshes shown in Fig. 12), (b) and (c) show the final result of the curvilinear reformatting.

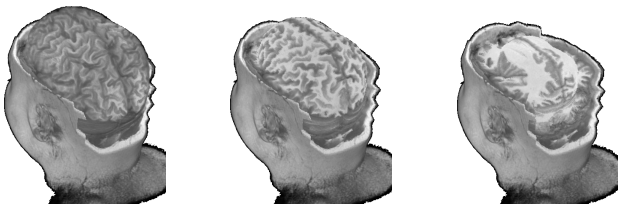
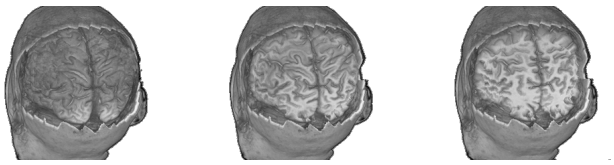


Fig. 14. Curvilinear reformatting in different depths: top view.

Fig. 15. Curvilinear reformatting in different depths: side view.



(a) 15mm (b) 23mm (c) 31mm

Fig. 16. Curvilinear reformatting in different depths: only in the back of the head.

TABLE II
TIME PERFORMANCE OF CURVILINEAR REFORMATTING.

	No. of triangles	Time(s)
Fig. 14	1335	34
Fig. 15	1070	27
Fig. 16	806	22

VII. CONCLUSION

The average time to perform the curvilinear is less than 1 minute and the zippering algorithm less than 1 second. These results are appropriate for an interactive environment. From the videos available in the project site http://www.dca.fee.unicamp.br/~wloos/video.html, one can appreciate the performance of our algorithm. Thus we present some advantages of our algorithm: 1) few interactions are needed to make the curvilinear reformatting, 2) it works in interactive time, 3) space native is maintained and 4) the reformatting is made only in the region that is selected. However our algorithm still has some limitations. As our zippering algorithm is based on a visibility algorithm, the area to be zippered needs to be visible, otherwise it is not possible to use it. We also assume that only two transitions vertices for each mesh are found, otherwise the zippering is not performed. In other words, the mesh cannot have more than two intersecting points. When we are merging the vertices, some edges can cross each other as presented in Fig. 17. When the vertices w and v are merged (Fig. 17(b)), the edge e_v may cross e_w. A way to handle this problem is to split the edges in two edges.

Another limitation of our proposal is that we created new triangles based on their internal angles formed by their vertices to fill the holes. This methodology only works for convex holes. The small distance between the meshes guarantees the smoothness of the zippered mesh, however this can yield some degenerate triangles. One alternative is using the simplification algorithm to remove them immediately after zippering.

Fig. 17. The edges, e_w and e_v, cross each other after the vertices w and v are merged.

ACKNOWLEDGMENT

This work has been supported by CAPES (Coordination for the Improvement of Higher Level Personnel) and FAPESP (The State of São Paulo Research Foundation).

REFERENCES

[1] A. C. Bastos, R. M. Comeau, F. Andermann, D. Melanson, F. Cendes, F. Dubeau, S. Fontaine, D. Tampieri, and A. Olivier, "Diagnosis of subtle focal dysplastic lesions: Curvilinear reformatting from three-dimensional magnetic resonance imaging," *Annals of Neurology*, vol. 46, no. 1, pp. 88–94, 1999.

[2] H.-J. Huppertz, J. Kassubek, D.-M. Altenmller, T. Breyer, and S. Fauser, "Automatic curvilinear reformatting of three-dimensional {MRI} data of the cerebral cortex," *NeuroImage*, vol. 39, no. 1, pp. 80 – 86, 2008.

[3] F. Bergo and A. Falcao, "Fast and automatic curvilinear reformatting of mr images of the brain for diagnosis of dysplastic lesions," in *Biomedical Imaging: Nano to Macro, 2006. 3rd IEEE International Symposium on*, April 2006, pp. 486–489.

[4] S.-T. Wu, C. Yasuda, and F. Cendes, "Interactive curvilinear reformatting in native space," *Visualization and Computer Graphics, IEEE Transactions on*, vol. 18, no. 2, pp. 299–308, Feb 2012.

[5] G. Turk and M. Levoy, "Zippered polygon meshes from range images," in *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '94. New York, NY, USA: ACM, 1994, pp. 311–318.

[6] A. A. Shostko, R. L. O. Hner, and W. C. Sandberg, "Surface Triangulation Over," vol. 1376, no. June 1998, pp. 1359–1376, 1999.

[7] L. H. D. E. Figueiredo, "Intersecting and Trimming Parametric Meshes on Finite-Element Shells," vol. 0, no. 0, pp. 1–28, 1999.

[8] W. Jung, H. Shin, and B. K. Choi, "Self-intersection Removal in Triangular Mesh Offsetting," vol. 1, pp. 477–484, 2004.

[9] I. L. Yi, Y.-s. Lee, and H. Shin, "Mitered Offset of a Mesh Using QEM and Vertex Split," *Proceedings of the 2008 ACM symposium on Solid and physical modeling*, pp. 315–320, 2008.

[10] S. Liu and C. C. L. Wang, "Fast Intersection-free Offset Surface Generation from Freeform Models with Triangular Meshes," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 852, pp. 347–360, 2011.

[11] E.-A. Karabassi, G. Papaioannou, and T. Theoharis, "A fast depth-buffer-based voxelization algorithm," *J. Graph. Tools*, vol. 4, no. 4, pp. 5–10, Dec. 1999.

[12] M. Garland and P. S. Heckbert, "Surface simplification using quadric error metrics," *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*, pp. 209–216, 1997.

[13] M. Garland, "Quadric-based polygonal surface simplification," Ph.D. dissertation, Pittsburgh, PA, USA, 1999, aAI9950005.