Extracting Intersections of Coplanar Surfaces (Boolean-operation on touching meshes)

Release 1.00

Roman Grothausmann[1]

December 14, 2014

[1]grothausmann.roman@mh-hannover.de,
Institute of Functional and Applied Anatomy, Hannover Medical School and
REBIRTH Cluster of Excellence, Hannover, Germany

Abstract

The contribution to VTK presented in this article is specialized on the extraction of contact surfaces (CS). This extraction can be regarded as the intersection Boolean-operation of only touching meshes. The `vtkCoplanarSurfaceExtractor` filter produces either polygonal or triangulated CSs by reconstructing the contact faces of co-planar triangles. Specified tolerances account for discrepancies in coplanarity of faces which might occur due to rounding effects. This article is accompanied with the source code, input data, parameters and output data that were used for validating the VTK-filter.

Latest version available at the Insight Journal [http://hdl.handle.net/10380/3504]

Contents

1 Introduction

Analyzing geometric properties of segmented regions contained in label-images is a common task in digital image analysis.[1] These measures always concern each label individually. However, depending on the segmentation, for example the results of borderless watershed transforms[2], labeled regions are adjacent to other labels. Although the surface of each label can be determined[3], a geometric property that cannot be analyzed for each label individually is the contact surface between adjacent labels (CS). This can be of special interest for example in biological/clinical studies to measure the contact surface of adjacent organs or if blobs (cells, particles) were separated at constrictions using a watershed transform on a distance map as in Fig. 1, see also Beare and Lehmann[2]. A simple method to measure CSs is the creation of a "lego" surface for each label (see e.g. http://www.vtk.org/Wiki/VTK/Examples/Cxx/Medical/GenerateCubesFromLabels or Mueller[4]) followed by extraction of coincident faces of adjacent labels with e.g. `vtkDistancePolyDataFilter`[5]. However, the resulting surface area is only a very rough estimate due to the not-smoothed discretization. Subsequent smoothing with e.g. `vtkWindowedSincPolyDataFilter` to improve the result has to preserve the boundaries and is not constrained by the initial voxel values any more. This can falsify the results because the resulting surface will tend to form the minimal spanning surface ot the given boundary (depending on the amount of smoothing iterations chosen). Preserving the boundaries also means that the boundary itself will not be smoothed yielding jagged borders. Smoothing before the extraction of coincident faces is not possible because the smoothing process for each label will remove the face coincidence essential for the extraction. Instead, marching-cubes surfaces created by `vtkDiscreteMarchingCubes` (http://www.vtk.org/Wiki/VTK/Examples/Cxx/Medical/GenerateModelsFromLabels) can be used because it is guaranteed to yield CS for adjacent labels, Fig. 1. Marching cubes takes the local voxel arrangement into account and results in 26 ($3^3 - 1$) possible face orientations (instead of just 6 for "lego" surfaces) yielding a "smoother" re-

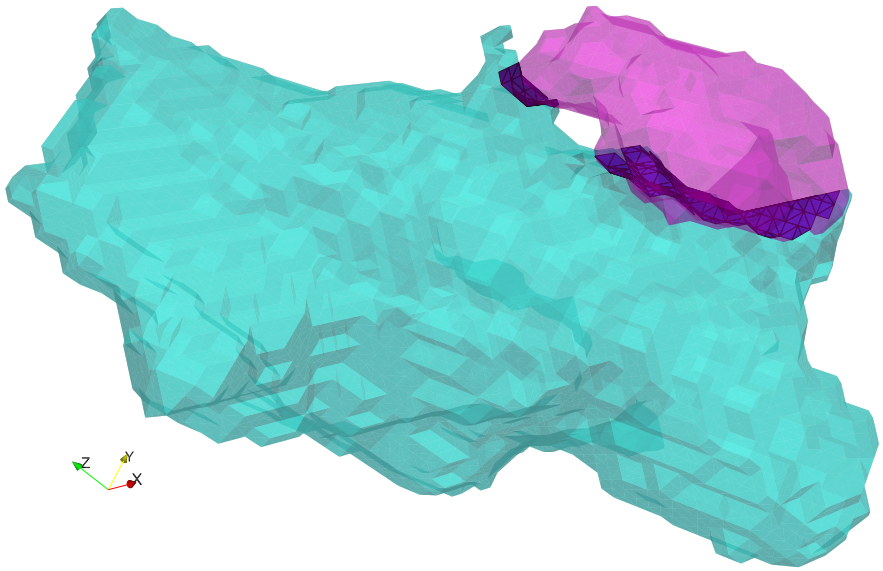

Figure 1: Examplary touching marching cubes meshes

Two test meshes (colored magenta and cyan) originating from differently labeled but touching regions in a voxel data set. The meshes were created by `vtkDiscreteMarchingCubes`. The marching cubes meshes touch each other but do not intersect because the two labels touched each other in the voxel data set. The result of `vtkCoplanarSurfaceExtractor`, which extracted the parts of the surfaces where both magenta and cyan meshes touch, is rendered in blue (faces and edges).

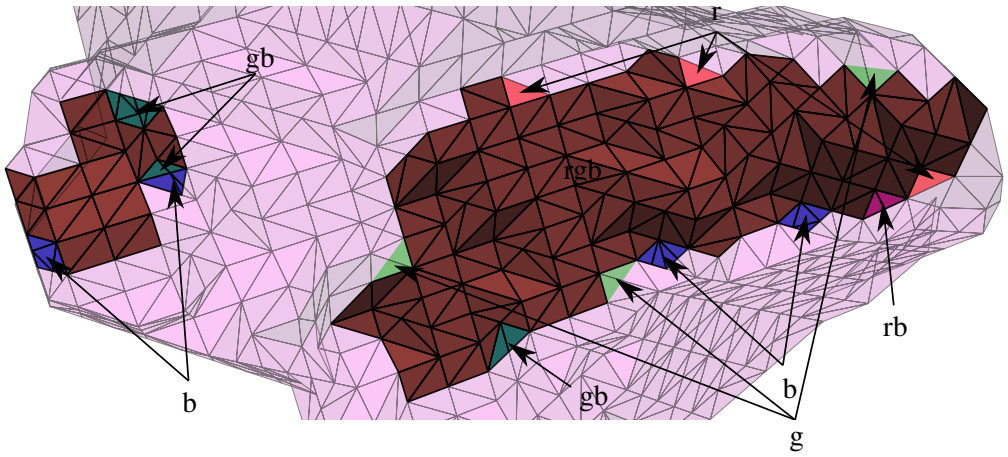

Figure 2: Faces and edges of the magenta mesh rendered with transparency (and front-face culling). Additionally the result of `vtkCoplanarSurfaceExtractor` is rendered in blue (faces and edges). The part of the magenta mesh whose point positions are also found in the cyan mesh (not shown) is colored red and vice versa green.
r: faces only in red mesh; g: faces only in green mesh; b: faces only in blue mesh; rb: faces only in red and blue mesh; gb: faces only in green and blue mesh; rgb: faces in red, green and blue mesh (only one face marked)

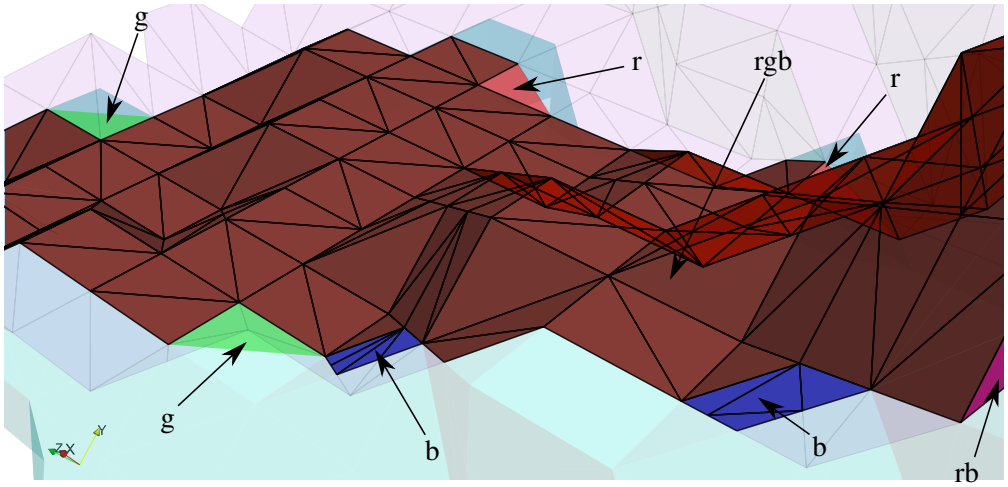

Figure 3: Different, close-up view of Fig. 2 also showing cyan mesh rendered with high transparency and back-face culling. Since the edges of magenta mesh are rendered while those of the cyan mesh are not, the important difference concerning the CS of these two meshes can be seen.
r: faces belonging to the red/magenta mesh but not touched by any face of the green/cyan mesh; g: faces belonging to the green/cyan mesh but not touched by any face of the red/magenta mesh; b: faces only in the blue mesh, i.e. regions where faces of the magenta and the cyan mesh touch (intersecting coplanar faces) but where these faces are not coincident (therefore not contained in the red or green mesh); rb: faces only in red and blue mesh; rgb: faces in red, green and blue mesh (only one face marked)

sult still suitable for CS extraction. However, the resulting CS between two adjacent labels is not guaranteed to consist of coincident triangles. Therefore, extraction with `vtkDistancePolyDataFilter` will not give an exact result of the CS, Fig. 2 and 3. The problem of extracting CSs in this case can be regarded as an intersection Boolean-operation on meshes.[6] However, Boolean-operations performed by e.g. `vtkBooleanOperationPolyDataFilter` are intended for intersections of meshes with volume overlaps not just touching surfaces.[5,7,8] Even implementations outside VTK, like the GNU Triangulated Surface Library (GTS), Carve or Blender (`www.blender.org`), have problems with these special cases.

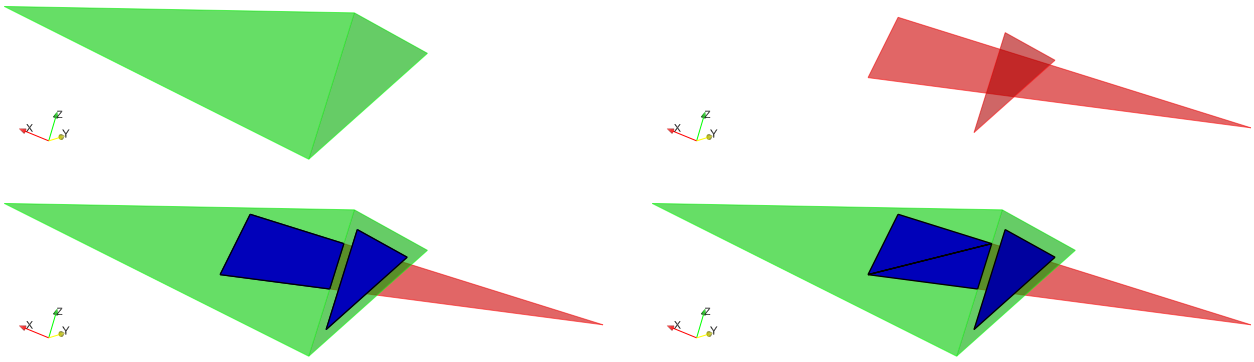

Figure 4: Test triangles in 3D

First input data set rendered in transparent green, second input data set in transparent red. Note that the triangles do not lie within the same plane in 3D, the red ones do not even touch. The output of `vtkCoplanarSurfaceExtractor` is colored blue with black edges using `vtkConvexHull2D` on the lower left, on the right `vtkDelaunay2D` consisting only of triangles (note the black line in the quad). The input data was created with Blender.

The contribution to VTK presented in this article is specialized for the extraction of CSs. The `vtkCoplanarSurfaceExtractor` produces either polygonal or triangulated CSs by reconstructing the contact faces of co-planar triangles. Specified tolerances account for discrepancies in coplanarity of faces which might occur due to rounding effects of point coordinates. Fig. 4 shows two input meshes consisting only of very few triangles in 3D (i.e. not all contained within the same plane) to ease visualization. One red triangle is completely contained in a green one and the other red triangle only partially overlaps with the other green one. The results of `vtkCoplanarSurfaceExtractor` are rendered in blue with black lines to reveal triangulations, see also Sec. 2. Fig. 5 shows some other cases that can occur.[6]

2 Installation and Usage

To make use of `vtkCoplanarSurfaceExtractor` just copy the `vtkCoplanarSurfaceExtractor.cxx` and `vtkCoplanarSurfaceExtractor.h` into your project directory and include it in `CMakeLists.txt`, take the demo program `CoplanarSurfaceExtractor` and `CMakeLists.txt` from this contribution as an example.

As of VTK-6.1.0 `vtkConvexHull2D` is not included within an installation[1] although contained in the folder `Infovis/Core/`. To include `vtkConvexHull2D` into the installation process add to `Infovis/Core/CMakeLists.txt` under `set(Module_SRCS` a line containing `vtkConvexHull2D.cxx` and in `Infovis/Core/module.cmake` under `DEPENDS` a line with `vtkRenderingCore`.

As intersections of co-planar triangles are always convex, the resulting surface can be created by either `vtkConvexHull2D` or `vtkDelaunay2D`. The difference between the two methods lies in the output: `vtkConvexHull2D` yields `vtkPolygon` whereas `vtkDelaunay2D` returns a triangulated version of this polygon. It is essential not to modify `SetMinHullSizeInWorld(0.0)` for `vtkConvexHull2D` in `vtkCoplanarSurfaceExtractor` or the results won't be correct, see Sec. 4 and Fig. 6. For the application of `vtkCoplanarSurfaceExtractor` on marching-cubes surfaces, the *guaranteed visibility* of hulls should be granted and therefore a `MinHullSizeInWorld` of `0.0` should lead to no problems. Setting this property to `0.0` also makes `vtkConvexHull2D` perform as fast as `vtkDelaunay2D` in the test cases.

[1] This is a bug: http://www.vtk.org/pipermail/vtkusers/2014-December/089682.html

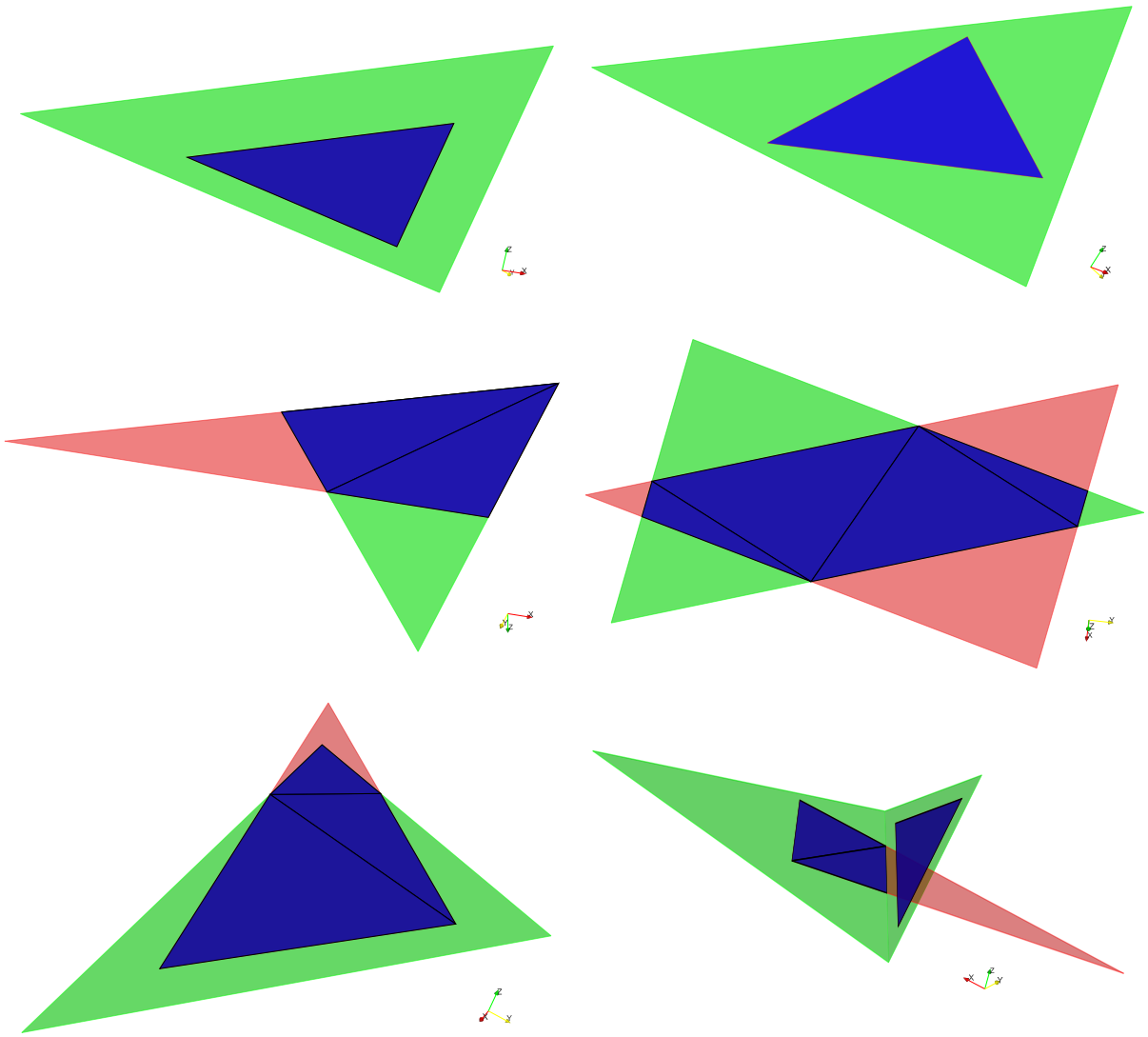

Figure 5: Coplanar test triangles, colored as in Fig. 4.

3 Usage notes

`vtkCoplanarSurfaceExtractor` depends on cell normals produced by `vtkPolyDataNormals`. The input meshes are passed to `vtkTriangleFilter` because the algorithm is only implemented for `VTK_TRIANGLE` so far. As of VTK-6.1.0 `vtkPolyDataNormals` passes its cell normal computation to `vtkPolygon::ComputeNormal` and that for `VTK_TRIANGLE` to `vtkTriangle::ComputeNormal` implemented in `vtkTriangle.h` which normalizes the normals with double precision. However, test showed that removing additional calls to `vtkMath::Normalize` (where appropriate to avoid unnecessary computations), caused failures for some test data sets. The problem seems to be caused by rounding errors occurring for values very close to but outside the range $[-1.0;+1.0]$. Therefore, a `SaveAcos()` function was introduced that handles these rounding errors. These errors are far smaller than the tolerances generally needed by `vtkCoplanarSurfaceExtractor` for correct execution. Removing `vtkMath::Normalize` yields a speed-up of around 4x for the test data sets.

Although VTK supports non-planar polygons, the result of e.g. a renderer in ParaView and that of `vtkTriangleFilter` might differ. `vtkCoplanarSurfaceExtractor` expects planar polygons and does not check this specifically, it just uses the result from `vtkTriangleFilter` as is.

4 Testing

The contribution comes with basic tests and some specific test data sets in `testing/triangle_tests` and `testing/mc-surf_tests`. This data was also used for the figures. The test program `TestCoplanarSurfaceExtractor` runs `vtkCoplanarSurfaceExtractor` on two input meshes and compares the filter output with an expected result mesh employing `vtkHausdorffDistancePointSetFilter`[9]. Generally, the output of `vtkCoplanarSurfaceExtractor` are open meshes with borders such that a given tolerance on the Hausdorff distance seems an appropriate measure for automated testing. For example, depending on the value of `MinHullSizeInWorld` of `vtkConvexHull2D` the result of `vtkCoplanarSurfaceExtractor` will differ, see Fig. 6. The automated tests catch this case even though both results lie in the same plane and have the same number of points and cells. Using `vtkHausdorffDistancePointSetFilter` for testing also allows to compare the output of `vtkCoplanarSurfaceExtractor` with either the result of `vtkConvexHull2D` or `vtkDelaunay2D` which (even though the surfaces are identical) generally differ in the number of cells.

5 Conclusions

In some cases it is important to extract the exact contact surfaces (CS) of two adjacent meshes, especially when the CSs are small and not well resolved due to limited resolution of e.g. a tomogram. This extraction can be regarded as the intersection Boolean-operation of only touching meshes. The presented contribution of a VTK-filter `vtkCoplanarSurfaceExtractor` can achieve this task, as shown in some exemplary data sets. The contribution also contains a test program, test/demo data, some test cases for the filter and an example application.

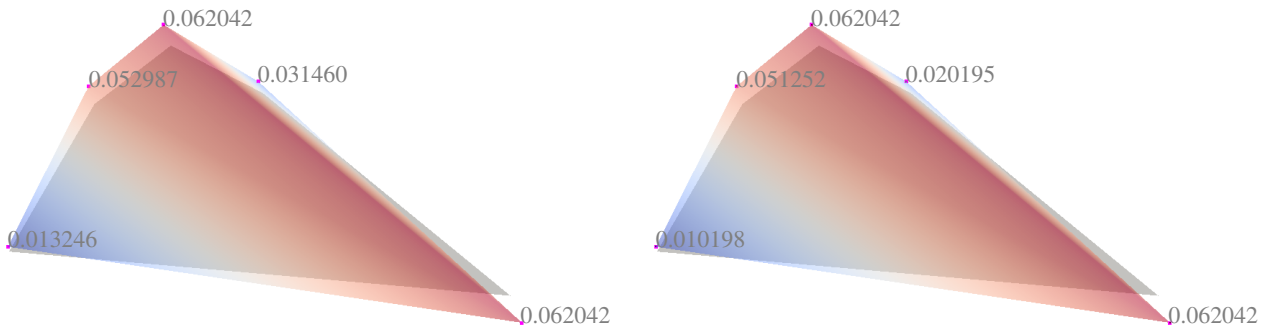

Figure 6: Visualization of output discrepancies caught by the automated tests

Correct output of `vtkCoplanarSurfaceExtractor` rendered in gray, overlaid with the output employing `vtkConvexHull2D` with a non-zero value (the default) for `MinHullSizeInWorld` colored according to the point-to-point distance (left) and point-to-cell (right). The Hausdorff distance between the meshes is 0.062042. The inputs are identical to Fig. 5 bottom left.

Intersection is only one of the typical Boolean-operations on meshes.[5–8] It is the only one implemented in `vtkCoplanarSurfaceExtractor` because union or difference operations seem to be hardly ever needed in the special case of purely adjacent meshes. If needed, the code of `vtkCoplanarSurfaceExtractor` can be adjusted to do these operations instead.

6 Acknowledgment

Thanks go to Tobias Sargeant concerning testing of Carve for CS extraction and to the members of the VTK-users mailing-list for their help and support.

References

[1] Gaëtan Lehmann. Label object representation and manipulation with ITK. *Insight Journal*, (176):1–34, Aug 2008. URL http://hdl.handle.net/1926/584. 1

[2] Richard Beare and Gaëtan Lehmann. The watershed transform in ITK - discussion and new developments. *Insight Journal*, (92):1–24, June 2006. URL http://hdl.handle.net/1926/202. 1

[3] Gaëtan Lehmann and David Legland. Efficient N-Dimensional surface estimation using Crofton formula and run-length encoding. *Insight Journal*, pages 1–11, 02 2012. URL http://www.insight-journal.org/browse/publication/852. 1

[4] Dan Mueller. Cuberille Implicit Surface Polygonization for ITK. *Insight Journal*, (740), Jul. 2010. URL http://hdl.handle.net/10380/3186. 1

[5] Cory Quammen, Chris Weigle, and Russell M. Taylor II. Boolean Operations on Surfaces in VTK Without External Libraries. *The VTK Journal*, (797), March 2011. URL http://hdl.handle.net/10380/3262. 1, 5

[6] Honggang Qu, Mao Pan, Bin Wang, Yong Wang, and Zhangang Wang. *Advances in Spatio-Temporal Analysis*, chapter Boolean operations on triangulated solids and their applications in 3D geological modelling, pages 21–28. ISPRS Book Series. CRC Press, 2007. ISBN 9780203937556. doi: http://www.crcpress.com/product/isbn/9780415406307. URL http://www.isprs.org/proceedings/XXXVI/2-W25/source/BOOLEAN_OPERATIONS_OF_TRIANGULATED_SOLIDS_AND_THEIR_APPLICATIONS_IN_THE_3D_GEOLOGICAL_MODELLING.pdf. 1, 1

[7] Bryn Lloyd. Boolean Operations on Surfaces for VTK. *The VTK Journal*, (726), 2010. URL http://hdl.handle.net/10380/3169. 1

[8] Tobias Sargeant. carve, 1.4.0. URL http://carve-csg.com. code repository: https://code.google.com/p/carve/. 1, 5

[9] Frédéric Commandeur, Jérôme Velut, and Oscar Acosta. A VTK Algorithm for the Computation of the Hausdorff Distance. *The VTK Journal*, 09 2011. doi: http://hdl.handle.net/10380/3322. URL http://www.insight-journal.org/browse/publication/839. 4