PERGAMON

Computers & Geosciences 28 (2002) 841–850

COMPUTERS & GEOSCIENCES

www.elsevier.com/locate/cageo

TRICUT: a program to clip triangle meshes using the rapid and triangle libraries and the visualization toolkit☆

C.H. Lindenbeck[a], H.D. Ebert[b], H. Ulmer[a,*], L. Pallozzi Lavorante[b], R. Pflug[a]

[a] *Geologisches Institut der Albert-Ludwigs-Universitaet, Albertstr. 23-B, D-79104 Freiburg i. Br., Germany*
[b] *Instituto de Geociências, Universidade Estadual Paulista, Campus de Rio Claro, BR 13.506-900 Rio Claro-SP, Brazil*

Received 11 September 2000; received in revised form 20 August 2001; accepted 28 August 2001

Abstract

An efficient technique to cut polygonal meshes as a step in the geometric modeling of topographic and geological data has been developed. In boundary represented models of outcropping strata and faulted horizons polygonal meshes often intersect each other. TRICUT determines the line of intersection and re-triangulates the area of contact. Along this line the mesh is split in two or more parts which can be selected for removal. The user interaction takes place in the 3D-model space. The intersection, selection and removal are under graphic control. The visualization of outcropping geological structures in digital terrain models is improved by determining intersections against a slightly shifted terrain model. Thus, the outcrop line becomes a surface which overlaps the terrain in its initial position. The area of this overlapping surface changes with respect to the strike and dip of the structure, the morphology and the offset. Some applications of TRICUT on different real datasets are shown. TRICUT is implemented in C++ using the Visualization Toolkit in conjunction with the RAPID and TRIANGLE libraries. The program runs under LINUX and UNIX using the MESA OpenGL library. This work gives an example of solving a complex 3D geometric problem by integrating available robust public domain software. © 2002 Elsevier Science Ltd. All rights reserved.

Keywords: Visualization; Surface modeling; Intersection computation; Surface cutting; Delaunay triangulation

1. Introduction

This paper describes the program TRICUT, which is developed to cut intersecting triangle meshes. Applications are shown where geological horizons and faults intersect digital terrain models (DTMs). The data is represented by irregularly spaced triangle meshes, which are used for shaded, perspective visualization. As a result of different data sampling techniques underground geological structures are mostly less detailed than the terrain surface. The program determines the outcrop of the structures and re-triangulates the area of contact. Overlapping parts of the structures can be removed from the refined meshes. A new visualization technique is developed by clipping these meshes against a slightly shifted terrain.

The program is most efficient using the RAPID library for collision detection and the TRIANGLE library for a robust constrained Delaunay triangulation in 2D. The algorithms are implemented using the Visualization Toolkit to gain interactive control over the removal process. Examples of landscapes from the Swiss Jura Mountains and southeastern Brazil, as well as tectonic structures of a sandbox experiment are shown.

☆Code available from server at http://www.iamg.org/CG-Editor/index.htm

*Corresponding author.

E-mail addresses: hdebert@rc.unesp.br (H.D. Ebert), ulmer@uni-freiburg.de (H. Ulmer).

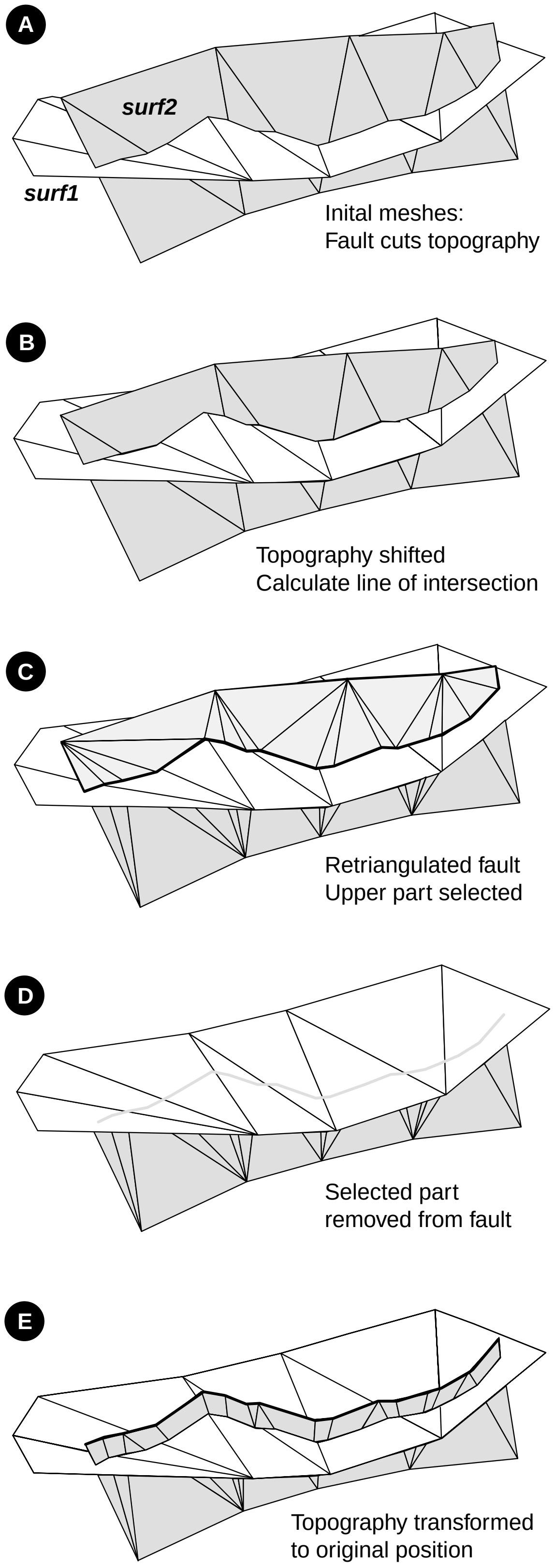
2. Previous work

Cutting of triangle meshes was not an unsolved problem when we started to design TRICUT. Extensive experience in the development and application of cutting functionality was made in the gOcad group[1] (Euler, 1999; Huang, 1990). The program GO2VV (gOcad to VOXELVIEW) uses a function of the gOcad libraries called TSURF_Cut_by_Scissors to cut subvolumes from extended sets of faults (Lindenbeck and Ulmer, 1995). A joint project between the gOcad research group, the Chevron Company and the T-Surf Company introduced the computation of intersections by using geological constraints. An online document[2] provides details and further references. Another powerful program which gives reliable results with a distinct approach is the ray-tracing program PoVRay.[3] Others algorithms for intersecting and trimming parametric meshes have been published by Coelho et al. (1999) and Shostko et al. (1999).

The decision to implement TRICUT as a standalone tool was made while investigating the possibilities of the RAPID library, the TRIANGLE library and the Visualization Toolkit. The project was based on the functionality and reliability of these packages, which are in the public domain. The implementation of TRICUT was started on a high level using the selected libraries and C++ classes. Using VTK e.g. we could start graphically debugging the intersection problem early on. Merging different toolkits also has disadvantages, such as the lack of a common interface and a bundled distribution. This project demonstrates some of the advantages of the bazaar method[4] in software development.

3. Geology and landscape

Our geometric models are boundary representations of geological structures. Using 3D graphics these surfaces are visualized together with the topographic data. Different modeling techniques are used to approximate the surfaces as triangle meshes. The horizons and faults are triangulated between serial sections, known as geological cross sections. Mostly, the observation points of the subsurface structure are rare and the cross sections are representing one possible geological interpretation only, which has to be tested. The elevation models of the land surface (DTMs) are derived by digitizing contour lines. The data are gridded followed by an interpolation of the sparse grid and its triangula-



[1] GOCAD Hompage. http://www.ensg.inpl-nancy.fr/GOCAD/

[2] Euler, N., Sword, Ch.H., Dulac, J.C., A new tool seal a 3D Earth Model: a cut with constraints. http://www.ensg.inpl-nancy.fr/GOCAD/research/papers/1998.09.seg.euler.pdf

[3] PoVRay-Persistence of Vision Raytracer. http://www.pov-ray.org/

[4] Raymond, E.S., The Cathedral and the Bazaar. http://www.tuxedo.org/~esr/writings/cathedral-bazaar/

tion using different tools. Another approach was the direct 2D Delaunay triangulation of the digitized contour lines, which can yield more accurate results, specifically for geomorphometric applications (Weibel and Brändli 1995). Meshes of regular and irregular grids can be handled and processed with TRICUT. In some of the models the meshes are decimated with respect to the slope gradient. The resulting meshes are dense in areas of irregular relief and relatively coarse in regions of low relief variation. The geological elements are also irregularly spaced polygonal surfaces, but with a free orientation in a 3D volume. Overturned parts are widespread in regions with folded strata. The meshes often cross the polygonal surface of the terrain model.

4. Basic concepts of TRICUT

The program TRICUT determines lines of intersection between a given mesh of the lithologic or tectonic surface and the mesh of the terrain. Along these lines—the outcrop—all points of intersection are used to re-triangulate the outcropping mesh. The new triangulation will refine the area of contact between the meshes. The divided triangles will not cross the land surface so that all triangles of the upper, overlapping part can be removed recursively. The crudely shaped geological element achieves a detailed border along the outcrop and the mesh will fit perfectly to the terrain along this re-triangulated border.

Fig. 1 shows a visualization technique developed using this program: In the scheme a fault (gray) intersects the topography (white). Before determining the line of intersection, the terrain is shifted a certain amount along the z-axis (Fig. 1B). A pseudo-outcrop will be determined with the uplifted terrain. All intersected triangles are re-triangulated constrained by the intersection line segments (Fig. 1C). Triangles of the overlapping part will be recursively removed (Fig. 1D). After translating the terrain back to its former position a part of the fault projects through the terrain model (Fig. 1E). The small protruding feature follows the landform along the outcrop. In complex geometric models this method of outcrop extension provides a better visual interpretation of the layering.

Fig. 1. Schematic application of TRICUT on two intersecting polygonal surfaces. (A) Initially *surf2* (gray) cuts *surf1* (white). (B) *Surf1* is displaced before intersecting line is determined. (C) All triangles of *surf2* are re-triangulated, constrained by intersection line segments. (D) Selected part of *surf2* is (light gray in (C)) is removed. Line of intersection is visible on *surf1*. (E) *Surf1* reset to its initial position. Clipped boundary of *surf2* becomes visible above *surf1*.

5. The Visualization Toolkit

The program is developed using the Visualization Toolkit (VTK).[5] VTK is an object oriented visualization system accessible through an extended C++ class library and bindings to the script languages Tcl and Python (Schroeder et al., 1998). VTK is in the public domain and can be used on a wide range of hardware platforms. We tested the program TRICUT on a SGI/CRAY supercomputer, different SGI- and SUN workstations and on the INTEL platform using LINUX with the MESA OpenGl package. The basic programming concept in VTK is the setup of a visualization pipeline. The triangle meshes are loaded as *vtk Polydata* and are mapped as actors. The actors are displayed with their properties using a VTK renderer to coordinate the scene including the lights and the camera. The object-oriented architecture provides high-level data structures such as the spatial search structure *vtkCellLocator*. There are useful methods to access the datasets, e.g. to use the adjacency information of the triangles. The Visualization Toolkit provides a picker to select cells from an actor in the scene. It is used in TRICUT for interactive selection of the parts that shall be removed.

6. The RAPID library

Determining the contact between polygonal surfaces in a conventional way can require a great deal of computational resources. An average terrain model usually contains more than 100k triangles, whereas a modeled fault or horizon only a few hundreds. Performing the intersection test for every possible triangle pair is time consuming. The RAPID library (Robust and Accurate Polygon Interference Detection,[6]) provides an efficient way to detect the set of intersecting triangles. It has been designed for collision detection between moving polygonal objects. RAPID implements the OBBTree data structure, which is a hierarchy of oriented bounding boxes (see footnote 6).

7. Implementation of TRICUT

TRICUT is divided into 7 main program steps as shown in Fig. 2. The visual results of some program steps are synthesized in Fig. 1 and illustrated in Figs. 6–8.

[5] Schroeder, W., Martin, K., Lorensen, B., Visualization Toolkit (VTK). http://www.kitware.com/vtk.html

[6] Gottschalk, S., Lin, M.C., Manocha, D., RAPID—Robust and Accurate Polygon Interference Detection. http://www.cs.unc.edu/~geom/OBB/OBBT.html

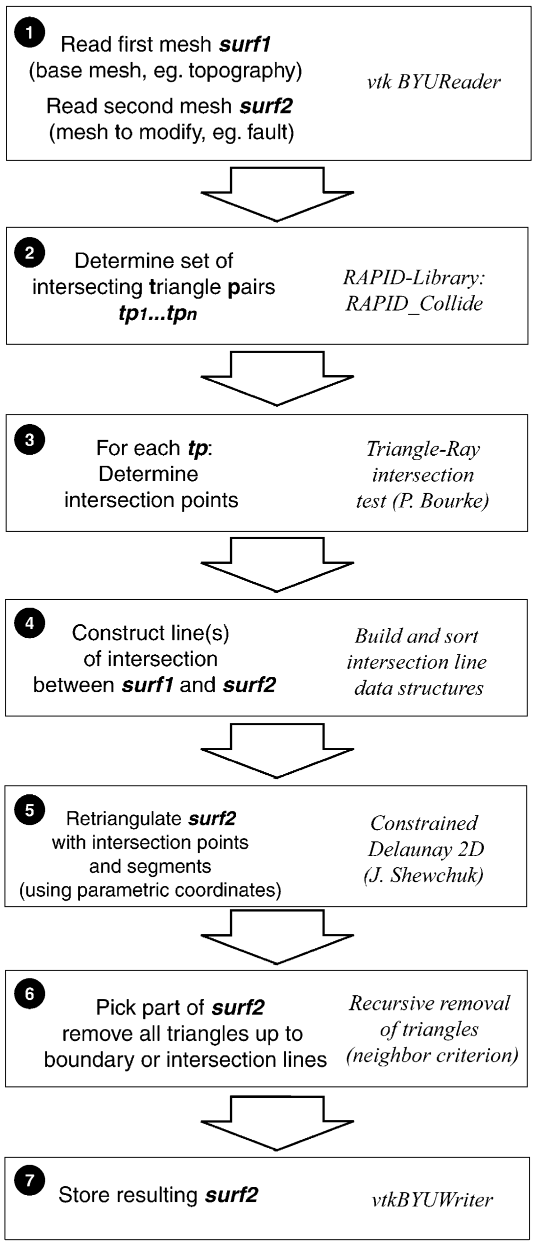

1 Read first mesh surf1
(base mesh, eg. topography)
Read second mesh surf2
(mesh to modify, eg. fault)
vtk BYUReader
2 Determine set of intersecting triangle pairs tp1...tpn
RAPID-Library: RAPID_Collide
3 For each tp: Determine intersection points
Triangle-Ray intersection test (P. Bourke)
4 Construct line(s) of intersection between surf1 and surf2
Build and sort intersection line data structures
5 Retriangulate surf2 with intersection points and segments (using parametric coordinates)
Constrained Delaunay 2D (J. Shewchuk)
6 Pick part of surf2 remove all triangles up to boundary or intersection lines
Recursive removal of triangles (neighbor criterion)
7 Store resulting surf2
vtkBYUWriter

Fig. 2. Program flow of TRICUT: 7 steps explained schematically (left) with notes on implementation (right).

7.1. Loading the surfaces and optional shifting of one surface

Two objects of the *vtkBYUReader* class were used to read the polygonal meshes. These are stored as *vtkPolyData,* a structure for polygonal data (Fig. 3). The first polygonal surface is referred to as *surf1,* the second one as *surf2*. The geometry of polydata is a list of *xyz*-coordinates (*vtkFloatPoints*) defining the vertices of

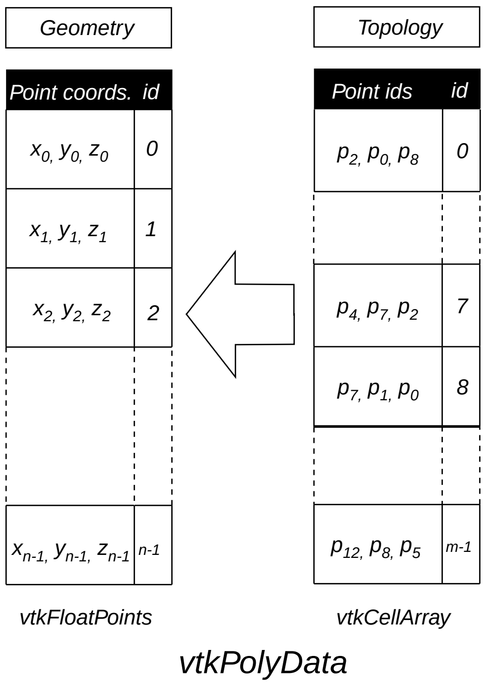


Fig. 3. Structure of a *vtkPolyData* class. Topology (right) references geometry data (left).

the surface. Each point has its own identification number (*Id*). The topology of polydata is the connectivity list (*vtkCellArray*) of the vertices describing the triangles. The *vtkPolyData* class provides useful methods to access geometry and topology of the surfaces. Of special interest is the method to query the neighborhood of triangles. The digital terrain model is loaded as the first surface (*surf1*) and can be shifted along the *z*-axis using the object *vtkTransformPolyDataFilter* (Fig. 1B).

7.2. Determination of intersecting pairs

The expression intersection pair (*tp1*, ..., *tpn*) will be used for each pair of overlapping triangles (*tr1*, *tr2*), where *tr1* belongs to *surf1* and *tr2* to *surf2*. Several methods can be used to determine the pairs of intersecting triangles. Testing each triangle of *surf1* against each triangle of *surf2* is expensive with large terrain models. Other techniques are developed to perform intersection tests only on subsets of adjacent triangle pairs. Spatial structures and efficient algorithms are discussed in the "collision detection" literature. As stated in Gottschalk et al. (1996), many efficient algorithms have been proposed to perform collision detection between two polygonal models. Some have not yet been implemented, whereas others cannot be used for interactive applications. To determine the intersecting triangles, TRICUT uses the oriented bounding boxes data structure (OBB-Tree), by including the RAPID library (Gottschalk et al., 1996). It is implemented in C++. It has a simple user interface: one function call to RAPID_Collide returns the set of all intersecting triangle pairs (*tp1*, ..., *tpn*).

7.3. Determination of intersection points

For each intersecting triangle pair (*tp1*, ..., *tpn*) the points of intersection are computed. They define the polygonal intersection lines between *surf1* and *surf2* discussed in Section 4. On each pair *tr1*, *tr2* six line plane intersection tests are performed. TRICUT uses an algorithm to compute the intersection point between a line segment and a planar 3-vertex facet, encountered on Paul Bourke's computational geometry pages.[7] First the edges of *tr2* are considered as three line segments which are tested against the plane on which *tr1* lies. Next the plane is defined by *tr2* and the line segments are the edges of *tr1*. If the detected intersection point lies within the triangle facet, the point will be an element of the polygonal intersection line running along the contact between between *surf1* and *surf2*. The locator object *vtkMergePoint* is used to prevent the insertion of points that are already present in the geometry of *surf2*. This class divides the 3D space into efficient traversable nodes (buckets). The *vtkMergePoint m*ethod for fast point location operates on this spatial data structure. A structure of linked lists for each *tr2* in *surf2* is built to reference the intersection points for later re-triangulation.

7.4. Constructing the line(s) of intersection

The intersection line will be constructed from the unsorted intersection points. This line—or this set of lines—will be important as the boundary criteria for the removal process later on. Up to now there is no information about the number and the topology of the polylines. Along the contact between the meshes two types of lines can evolve: closed lines (loops) and open lines depending on the geometry of *surf1* and *surf2*. Fig. 5 gives an example of an open intersection line traversing a topography. The chaining of all computed intersection points is made under the assumption that each intersected triangle in *surf2* has two intersection points. Each of these point pairs builds one segment of the intersection line. The neighborhood relationships of line segments are computed on the assumption that two adjacent segments share one common intersection point. This chaining process results either in detection of a closed line or an open line (start and endpoint, each with only one neighbor). The process is repeated until all intersection points belong to a line. The last process in this points-sorting stage is the reorganization of each linked list corresponding to *tr2* triangles. The points contained in each linked list are sorted in accordance with the intersection polylines' traversal order. After this step, a linked-list traversal will mean traversing the corresponding triangle from side to side, following the set of boundary segments defined by the traversed points. This step is necessary to perform correctly the next step.

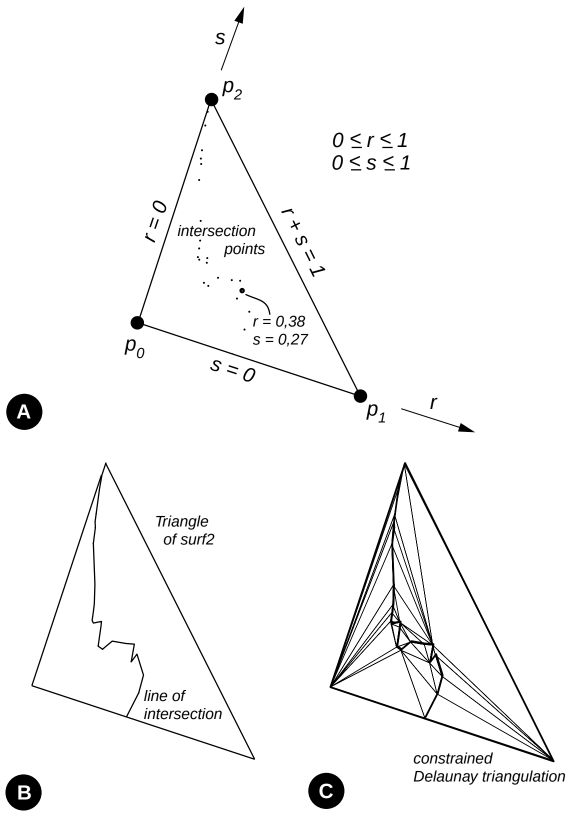


Fig. 4. Constrained Delaunay triangulation of intersected triangles in *surf2*. (A) Parametric coordinate system for each intersecting triangle and its intersection points. (B) Line of intersection. (C) Resulting triangulation constrained by all intersection line segments and border segments of triangle.

7.5. Re-triangulation of intersecting triangles

TRICUT is developed to remove parts of *surf2* delimited with respect to *surf1* as a boundary. To do that, *surf2* has to be separated into different parts along the intersection line. Each intersecting triangle of *surf2* will be split by a new triangulation using the intersection points, which lie within these triangles (see Fig. 1C and Fig. 4). This set of points is read from the linked list

[7]Paul Bourke's computational geometry pages. http://www.swin.edu.au/astronomy/pbourke/geometry/planeline/

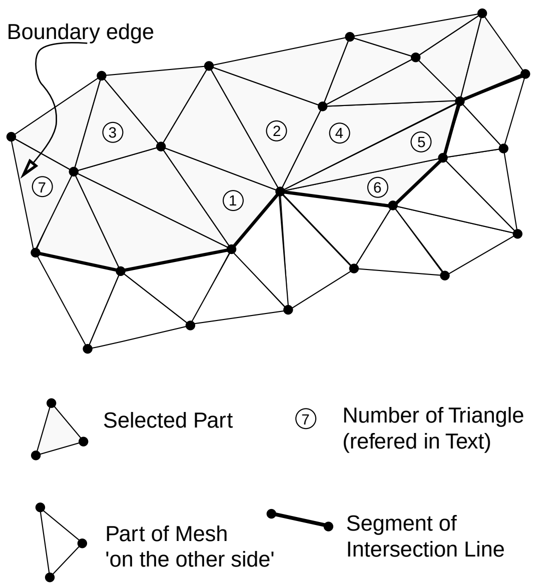

Boundary edge
3
2
4
5
7
1
6
Selected Part
7 Number of Triangle (refered in Text)
Part of Mesh 'on the other side'
Segment of Intersection Line

Fig. 5. Recursive part selection algorithm applied to re-triangulated fault is initialized by picking triangle.

built in step (3) and updated in step (4). The triangulation library used (TRIANGLE, developed by Jonathan Shewchuk[8]) works with 2D pointsets, but the original triangles in *surf2* can have any orientation in 3D space. Changing the coordinate system of each pointset before applying the triangulation is not efficient. To avoid this, parametric coordinates (Fig. 4) are used for the triangulation. Each point is described using two parametric coordinates (r, s), where $0< = r + s< = 1$. The vertices *P0*, *P1*and *P2* are represented by the pairs (0,0), (0,1) and (1,0). The *vtkCell* class provides the parametric coordinates. The triangles are inserted into *surf2* using methods of the *vtkCellArray* class.

Triangles crossing the intersection line would lead to problems during the recursive selection process in step (6). To avoid this, a constrained Delaunay triangulation on each intersecting triangle in *surf2* is performed using the TRIANGLE library. The triangulation of the intersection points and the vertices of the current triangle are constrained by the segments of the intersection line and the set of boundary segments of the triangle (Fig. 4C). This set contains intersection and triangle vertices sorted in a boundary traversal order by using parametric coordinates for each point. A constrained triangulation is necessary to ensure that the triangulation will include all intersection line segments. The use of an unconstrained Delaunay triangulation could lead to new triangles crossing some intersection segments.

7.6. Visualization, part selection and removal of triangles

Each of the new triangles in *surf2* shares an edge with a segment of an intersection line. This is the pre-condition for the removal process. At this point of the program execution the meshes are displayed in the renderer window. The result of the intersection computation can be investigated. The intersection lines are visualized using the *vtkTubeFilter* which generates a polygonal mesh around the intersection lines. Interaction with the renderer window includes the possibility of picking an arbitrary triangle. Fig. 5 shows a fault mesh, divided into two parts by an intersection line. The part selection algorithm is initialized by picking a triangle in the upper part. All triangles on this side of the intersection line are marked for deletion by traversing the mesh structure. Therefore the neighborhood of each triangle's edge is inspected. If an edge is part of the intersection line the neighbor triangle lies on the other side of the intersection line. The criteria for visiting neighbor triangles are derived from the number of contacts of triangle vertices with the intersection line: all neighbors are marked for deletion if there is none or one contact (triangle 3 or 2). In case of two contacts it must be decided, whether the edge is a segment of the intersection line (triangle 1) or not (triangle 4). Three contacts result in two different scenarios: triangle 6 has only one neighbor to mark for deletion, whereas triangle 5 has two (triangles 4 and 6). The boundary edge of *surf2* is the second criterion to stop searching (triangle 7). To retrieve the neighbor triangles a method of *vtkPolyData* is used. For visual control the color of the selected part will change and the removal has to be confirmed. Other parts can be deleted if more than one intersection line is present.

7.7. Saving the clipped surface

After removing one or more parts from *surf2* (Fig. 1D), the remaining triangles can be saved in the Movie.BYU data file format. To write the polygonal mesh VTK provides the class *vtkBYUWriter* as a companion to the class *vtkBYUReader* used in step (1). As previously stated, the terrain model can be translated back to its original position (Fig. 1E), in order to show the upper part of the cut mesh protruding through the topography.

8. The VTK visualization pipeline

In the VTK system, data visualization is the result of a pipeline execution: starting with the internal geometric representation of the object and resulting in the

[8]Shewchuk, J.R., A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator. http://www.cs.cmu.edu/~quake/triangle.html

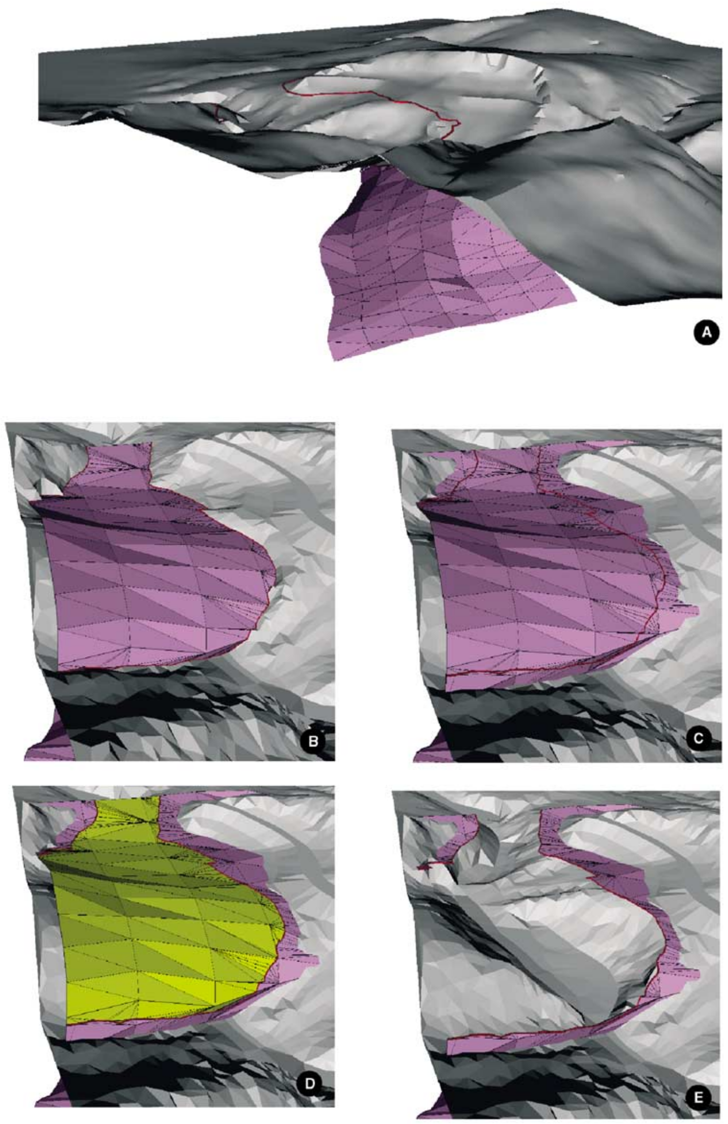


Fig. 6. Folded fault in Swiss Jura. Birs valley south of Delemont. (A) Viewing to NE. Fault is clipped without translation of terrain. (B) Fault cut against shifted terrain. (C) Line of intersection (red). Terrain is translated back. (D) Overlapping part selected for removal (yellow). (E) Resulting extended outcrop.

visualization in a 3D-rendering window. The following VTK-classes (Schroeder et al., 2000) are used in TRICUT:

- *vtkDataSet*: defines data sources. *vtkPolyData* objects have been used to provide the geometric information necessary for the representation of polygonal objects in 3D space.
- *vtkPolyDataMapper*: converts polygonal data information into graphic primitives used by the hardware/software rendering libraries.
- *vtkActor*: to represent data in the rendering scene. It controls object properties (color, shading, etc.) as well as matrix transformations associated with them.
- *vtkRenderer*: performs all transformations for the generation of 3D images, involving the properties of the actor, matrix transformations and camera options.
- *vtkRenderWindow*: creates the window where renderers draw their images.

- *vtkRenderWindowInteractor:* enables the user to interact with the visualized scene: 'w' key forces wireframe representation; 's' key switches to shaded view of the objects. By placing the mouse pointer over an object in the window and pressing 'p', this object can be 'picked' and its properties can be manipulated.

After the visualization step is completed, the *Start()* method from the *vtkRenderWindowInteractor* object is invoked and the user can interact with the rendered surfaces. After removal of the selected part, the user can save the remaining triangles of *surf2* or continue the interaction with the data and select other parts from the same surface. If a translation of *surf2* was applied at the beginning of the program, the surface might be shifted back to its initial position (Fig. 1E) after picking of *surf2*, using again (see Section 7.1) instances of the *vtkTransform* and *vtkTransformPolyData* filter.

9. Applications

The TRICUT program was successfully used to process surface cutting on several models with topographic and geological data. Three models rendered by

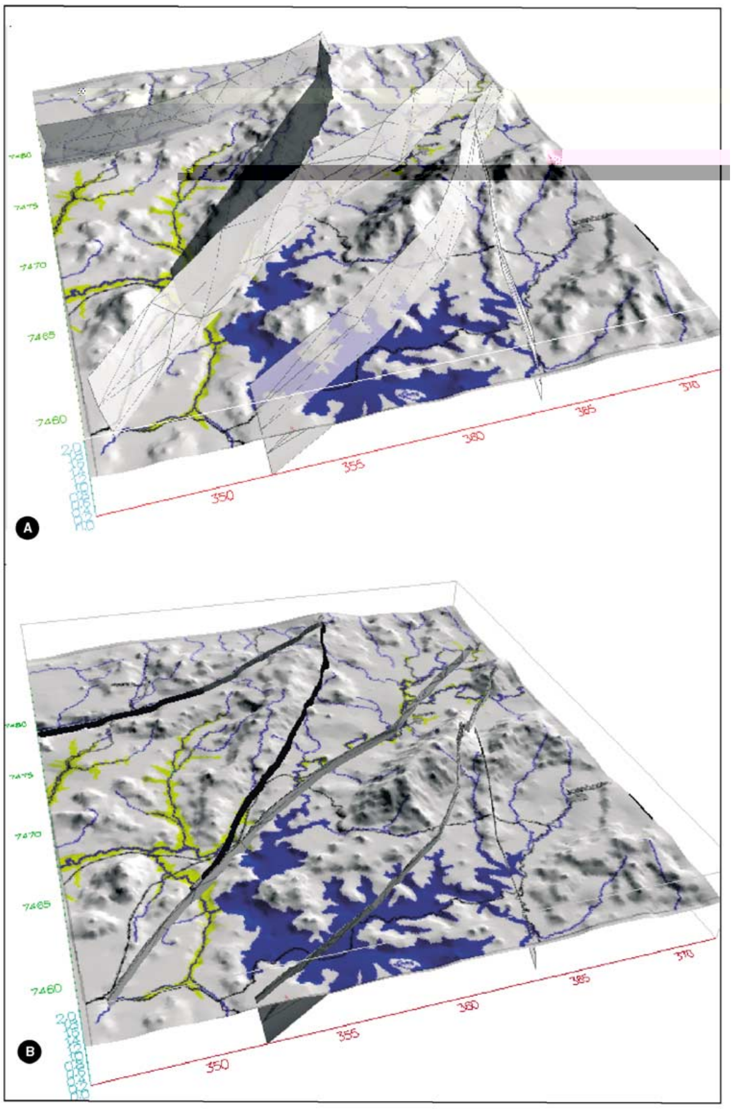
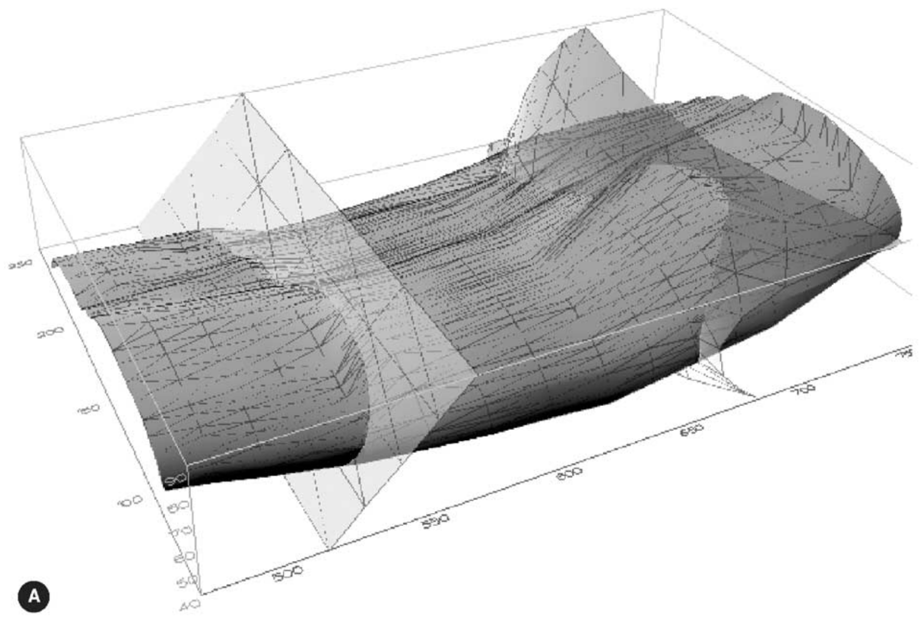


Fig. 7. Gouraud-rendered Extrema model: topography and four reactivation faults (A) before and (B) after cutting process (faults projecting 150 m over surface).



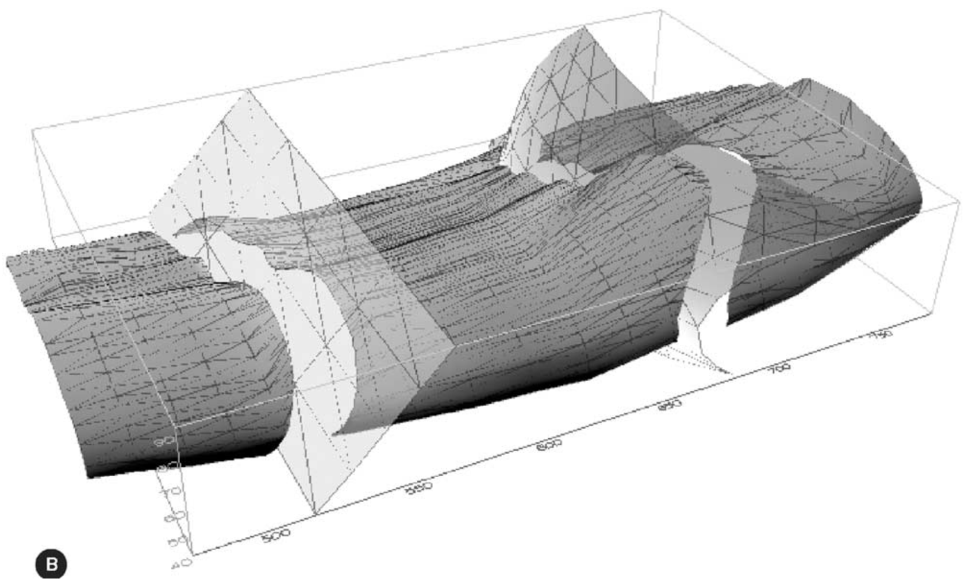


Fig. 8. Sandbox model with folded surface dissected by two fault planes. (A) Before cutting process and (B) after splitting folded surface in three parts, in accordance with fault kinematics.

GEO3VIEW and listed bellow are represented in this paper (Figs. 6–8).

9.1. *Folded fault in the Swiss Jura Mountains*

The topography consists of an irregularly spaced triangle mesh. All points of the digitized contour lines were triangulated using an unconstrained Delaunay triangulation. The geological elements of three horizons and two fault planes were built by a constrained triangulation between adjacent cross sections (Tipper, 1977; Lindenbeck and Ulmer, 1995). Fig. 6A–E show a folded fault crossing a valley, the subsurface strata are not displayed. Initially, this fault hides a wide part of the valley. The fault is cut against the terrain (shifted 25 m, Fig. 6B). The former nearly equal-sized triangles of the fault are split into numerous small triangles along the mesh contact. Fig. 6C shows the line of intersection (red) between the topography (gray) and the fault (purple). The terrain is translated back and Fig. 6D shows the overlapping part of the fault already selected for removal (yellow). Fig. 6E shows the resulting extended outcrop, with the yellow part removed. This technique provides more visible information of the structural settings by expanding the line of outcrop without hiding too much of the terrain. The layering is visualized without using translucency which is often misleading in perspective views that are not displayed stereoscopically. The meshes in Fig. 6A–E are flat shaded to show the triangles. Processing this fault without translation will fit the fault exactly to the terrain (Fig. 6A).

9.2. Subvertical faults in a DTM of The Estrema region in southeastern Brazil

This model has been constructed to improve morphostructural and neotectonic investigations (Fig. 7). The data were obtained by digitizing 76,296 points along the topographic lines from the 1:50,000 scale Extrema sheet. After interpolation, the nodes of the regular grid (250×300) were triangulated. Finally, for faster rendering, the triangle meshes were decimated by 50–75% using *vtkDecimator* class. The faults are represented by gouraud-shaded meshes with 20% translucency and simultaneous display of the triangles. The drainage network and the Quaternary basins have been overlaid on the DTM with texture-mapping techniques. Their simultaneous visualization onto the rendered model enhances the recognition of morphotectonic patterns.

9.3. Simple geometric model of the Ribeira Belt

This model (Fig. 8) represents a sandbox experiment in which transpression between obliquely converging plates has been simulated (Ebert et al., 1995, Lindenbeck et al., 1997). A folded layer is dissected and translated along two transversal faults.

10. Conclusions

TRICUT has various applications for manipulating polygonal surfaces in different areas such as in structural geology, mining and petroleum geology, allowing the separation of individual meshes by dissecting faults; in tectonic geomorphology and neotectonic studies by enhancing the visual capabilities of outcropping layers and faults through the DTMs. Although introduced here in a geoscientific context, TRICUT, as an open source project based on the integration of seperate but reliable public-domain software packages, can also be adapted and improved for applications in other fields where intersecting meshes must be separated for the generation and analysis of complex geometries and shapes.

Acknowledgements

This work was part of a project funded by the Volkswagen Stiftung (I/73 762) and the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

References

Coelho, L.C.G., Gattass, M., Fiqueiredo, L.H., 1999. Intersecting and trimming parametric meshes on finite-elements shells. International Journal for Numerical Methods in Engineering 47 (4), 777–800.

Ebert, H.D., Neves, M.A., Hasui, Y., Lopes, J.A., Guerra, M., 1995. Compartimentação crustal e evolução cinemática da Província Mantiqueira através de modelagem física. In: 5° Simpósio Nacional de Estudos Tectônicos, Boletim Resumos Expandidos, Gramado, Brazil, pp. 26–28.

Euler, N., 1999. Modelisation volumique, contraintes et libertes. Ph.D. Dissertation, Institut National Polytechnique de Lorraine, Ecole Nationale Superieure de Geologie-gOcad research group, Nancy, 148pp.

Gottschalk, S., Lin, M.C., Manocha, D., 1996. OBBTree: a hierarchical structure for rapid interference detection. In: Proceedings. ACM Siggraph, New York, 27p.

Huang, Y.G., 1990. Modelisation et Manipulation des Surfaces Triangulees. Ph.D. Dissertation, Institut National Polytechnique de Lorraine, Ecole Nationale Superieure de Geologie-gOcad research group, Nancy, 156pp.

Lindenbeck, Ch., Ulmer, H., 1995. Entwicklung und anwendung von computerprogrammen zur visualisierung geologischer strukturen und prozesse (Development and application of computer programs for visualization of geological structures and processes). Freiburger Geowissenschaftliche Beitraege 9, 280pp.

Lindenbeck, Ch., Ulmer, H., Ebert, H.D., Pflug, R., 1997. Geometric modeling and visualization of a sandbox-experiment: transpressional tectonics in the Precambrian Ribeira-Belt (Se-Brazil). Zentralblatt für Geologie und Palaeontologie Teil I 3/4, 707–719.

Schroeder, W., Martin, K., Avila, L.S., Law, C.C., 2000. Vtk User's Guide. Prentice-Hall, Englewood Cliffs, NJ, 355pp.

Schroeder, W., Martin, K., Lorensen, B., 1998. The Visualization Toolkit. Prentice-Hall, Englewood Cliffs, NJ, 646pp.

Shostko, A.A., Löhner, R., Sandberg, W.C., 1999. Surface triangulation over intersecting geometries. International Journal for Numerical Methods in Engineering 44 (9), 1359–1376.

Tipper, J., 1977. A method and FORTRAN program for the computerized reconstruction of three-dimensional objects from serial sections. Computers & Geosciences 3 (4), 579–599.

Weibel, R., Brändli, M., 1995. Adaptive methods for the refinement of digital terrain models for geomorphometric applications. Zeitschrift für Geomorphologie Neue Folge Supplementband 101, 13–30.