Survey of Errors in Surface Representation and their Detection and Correction

Veleba, D., Felkel, P.
Department of Computer Science
Czech Technical University, Faculty of Electrical Engineering
Karlovo náměstí 13
121 35 Praha 2, Czech Republic
{velebd1 | felkel} @ fel.cvut.cz

ABSTRACT

In this paper, a survey on the most typical mesh errors is given. Each error is described in detail, it is illustrated on an example and surface based techniques for its detection and correction are presented. Covered errors include cracks, holes, T-joints, overlaps, zero volume parts, duplicated geometry, self intersections, inconsistent normal orientation, invisible polygons, degenerate faces and concavities.

We consider the separation of the detection and the correction phases advantageous as it gives the user a better control over the mesh correction process, allowing better corrected meshes without introducing new errors, simplifications, or deformations.

Keywords

Mesh errors, Crack, T-joint, Inconsistent normal orientation, Swapped normals, Hole, Concavities, Invisible polygons, Detection, Correction, Degeneracies, Mesh repair

1. INTRODUCTION

At present, a growing number of models contain errors [Ken98, Ju04, Bis05] that either originate due to human mistakes or are produced by incorrectly implemented modeling software. These errors cause problems during every subsequent reuse of the model. Search and repair of these errors, which are often hidden, are highly time-consuming.

There are two different approaches to repairing polygonal models: a classic *mesh repair* [Mur97, Bar98, Bor02] and a newer *voxel based repair* [Noo03, Ju04, Bis05]. The latter is based on conversion to voxel representation and back. The most recent research in this field made Bischoff [Bis05]. He overcomes the main disadvantage of the voxelization, i.e. giving away the original model, by keeping the vertices' coordinates the same in the corrected model as in the input model. The former approach is more straightforward. Errors are first detected and each group of errors is corrected uniquely. However, the future seems to lie in combination of these two techniques. This paper is focused on the classic approach.

In Section 2, a survey of the typical mesh errors is provided. Each error is described and illustrated on an example. Error origin is discussed and methods for detection and correction of the error are presented. Section 3concludes.

2. ERRORS AND THEIR HANDLING

In this section, the following mesh errors are described: cracks, holes, T-joints, overlaps, dangling walls, duplicated geometry, self intersections, inconsistent normal orientation, invisible polygons, degenerated faces and concavities. First three of them, i.e. cracks, holes and T-joints, are well-known and hence they will be described only briefly. The concern will be devoted to the remaining errors.

2.1. Cracks

Cracks [Nie99] are small, elongated gaps in the model surface (see Fig.1 for an example). Cracks usually come along with T-joints, which are described in Section 2.3.

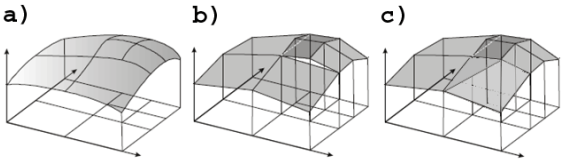

Fig.1 An example of a crack. a) A smooth surface [Nie99] is in b) represented by planar faces. c) shows the surface from panel b) after the correction.

2.1.1. Origin

Cracks originate mainly due to wrong triangulation of smooth sinuous surfaces and due to round-off errors.

2.1.2. Detection

Cracks, as well as holes, are demarked by boundary edges. Problems may arise if we need to distinguish cracks from holes – see [Vel06].

2.1.3. Correction

There are more approaches to correcting cracks. However, the correction technique should handle the cracks with respect to their origin (triangulation × shifted vertices).

Cracks arising from triangulation can be eliminated by *retriangulation* of the larger (rougher) face with respect to the boundary curve of the crack.

Cracks arisen due to multiplicity of vertices are corrected by *vertex contraction* [Gar97, Pop97].

More recently Borodin et. al. [Bor02] introduced further generalization of *edge contraction* operator, a *vertex-edge contraction*. Disadvantage of these operators is that they can produce non-manifold meshes.

2.2. Holes

By a hole [Lie03] [Var05], we understand a closed cycle of boundary edges. Problems with holes are that they either should be triangulated or that they are triangulated and should not be. See Fig.2 below.

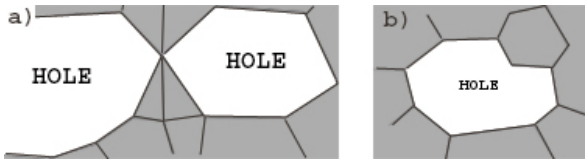

Fig.2 Two cases of holes. a) shows two connected holes and b) shows a single polygonal hole.

2.2.1. Origin

Holes arise mainly during imprecise surface reconstruction but they can often be intentional in the model.

2.2.2. Detection

Holes are detected as closed cycles of boundary edges, as well as cracks.

Unfortunately, this technique is not able to tell apart gaps from the natural boundary of the object.

In case there are more holes connected together, problems may arise with choosing a correct boundary edge belonging to the hole we are just detecting. This is done by taking the edge with the smallest angle to its previous edge.

Detection of holes on the natural boundary of objects is described in [Vel06].

2.2.3. Correction

For filling the holes Liepa [Lie03] introduced a 3-step method that firstly creates a patch that minimizes a weight function, secondly it shortens long edges and thus doesn't introduce skinny triangles, and at last, it uniformly spreads the vertices of the patch. The best results are achieved with weight function that considers a dihedral angle between existing neighboring faces and the face area.

Removing the extra triangles from the triangulated holes is quite problematic. We need to detect these triangles and differentiate them from the correct ones. To do this, we utilize the fact that redundant triangles have usually much longer edges. Unfortunately, this method fails on the boundary of the hole where extra triangles may be left or correct triangles may be discarded. Recent patching algorithms also do not produce triangles with long edges.

2.3. T-joints

T-joint [Bar98, Mur97] is a place, where two parallel edges connect (e_2 and e_3 in Fig.3b), while there is no appropriate vertex on the neighboring edge (e_1). The situation is illustrated in Fig.3b.

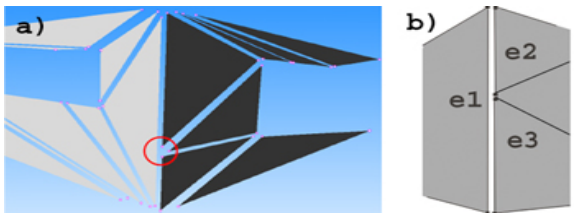

Fig.3 a) A T-joint. b) illustration to the definition c) A meeting of two surfaces with different levels of detail in a clipmap

2.3.1. Origin

T-joints are introduced by adding new vertices on existing edges and by wrong modeling in the design systems. T-joints also occur while handling a model in different levels of details (LOD).

2.3.2. Detection

All edges are checked to neighbor with exactly one another edge, i.e., they have the same end-vertices. Edges not matching this criterion are either boundary edges (have no neighbors) or edges participating in

T-joints (have more than one neighbor). Edges with more than one neighbor can also be non-manifold edges.

2.3.3. Correction

The best correction of T-joints is joining the multiple edges (e.g. in Fig.3b these would be e_2 and e_3). The triangles corresponding to these edges are replaced with one polygon (which may be subsequently triangulated) so the needless geometry is discarded.

T-joints on touch of different LODs during rendering are eliminated either by subdivision of neighboring triangles as in the ROAM algorithm [Duc97] or by introducing zero area triangles to "fill" the cracks on the touch of two levels and by interpolation of geometry and texture in the transition region [Los04].

2.4. Overlapping triangles

Overlapping triangles [Var05] have one or more of their vertices placed improperly into vertices, whose neighborhood is already fully triangulated (see Fig.3). Such triangles overlap the mesh (or are overlapped by other triangles) instead of filling the empty space on the surface. As a side effect, gaps are introduced into the model. In fact, the overlapping triangles are special case of duplicated geometry (2.6).

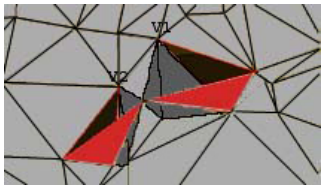

Fig.4 Overlaps: The red triangles denote overlapping triangles, the black triangles mark holes, and the dark grey triangles denote surface overlapped by the red triangles [Var05].

2.4.1. Origin

Overlapping triangles originate during triangulation of nonuniformly sampled models as a result of local undersampling. In this case, the tessellation algorithm positions one of the face vertices into an incorrect vertex. This is usual in triangulations of point clouds and in models taken by 3D scanners.

2.4.2. Detection

The overlaps are best detected by using the triangle (polygon) fans. We iterate through the list of vertices. For each vertex we construct its triangle fan and then, we compute the overall angle as a sum of angles between pairs of edges of each face connected to the fan's central vertex. Overall angle not equal to 2π implies a problem. If the angle is less than 2π, we have found a crack or a hole; if the angle is larger than 2π, we have found an overlap. Precision of computation has to be considered.

The detection can be made more efficient if we construct the fans only for vertices referenced by boundary edges.

2.4.3. Correction

To correct overlaps we delete the redundant triangles from the fans with angle larger than 2π and triangulate the holes. However, choosing the correct triangle to remove may be problematic.

2.5. Zero volume parts

Though rarely, zero volume parts (also called dangling walls) [Bøh95] sometimes occur in the model. Bøhn defines them as *sets of faces that that do not contribute to the definition of the volume occupied by one solid or more solids in the space.*

Very often, zero volume parts are used intentionally – for example to create paintings on the walls in models of interiors (called *decals)* or to connect two separate shells (artifact faces).

2.5.1. Origin

Dangling walls are usually mistakes of a model designer who might forgot to remove them from the model. They can be caused by imprecise floating point arithmetic as well.

2.5.2. Detection

In contrary to Bøhn, who detects only exactly matching pairs of faces with different orientation, we extend the detection to all patches that do not delimit any volume.

To distinguish cases of intentional use of zero volume parts from errors, we should only detect zero volume parts which are more distant from any face than a user-provided constant. By this, we ensure we won't detect the decals etc. On the other hand, this restriction prevents us from detecting artifact faces used for connecting two separate shells for example.

It is quite problematic to distinguish zero volume parts (ZVP) from cracks and holes. To do so, we may count number of vertices (or edges, faces etc) reachable from the boundary edges that delimit this error. Generally, number of vertices reachable from ZVP should be smaller than number of vertices reachable from holes. This is due to fact that from the hole the entire model could be possibly reached as opposed to ZVP from which only ZVP itself is reachable. Of course, this may also crash on holes in small objects and zero volume parts consisting of many faces, but both of these cases are extraordinary.

2.5.3. Correction

If required, dangling walls can be simply removed from the model.

2.6. Duplicated geometry

We differentiate the following cases of duplicated geometry:

- Concurrent vertices
- Concurrent edges
- Concurrent faces
    - Same normals
    - Opposite normals
    - Same triangulation
    - Different triangulation

Within all these cases we tell apart:

- Identical double geometry
- Mutually shifted double geometry

Concurrent vertices are sometimes used intentionally to model a sharp edge. This case is depicted in Fig.5b and described in [Vel06].

In Fig.5a you can see a cube with one side triangulated twice by mutually shifted faces (which include also shifted edges and shifted vertices) and even with opposite face normals.

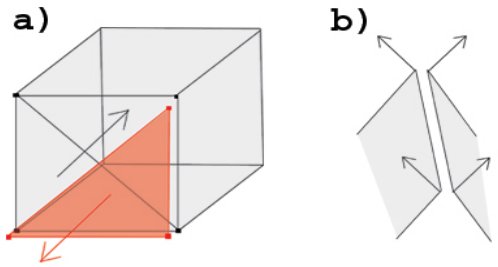

Fig.5 A concurrent geometry. a) shows a cube with one side triangulated by mutually shifted different triangles. They even have an opposite normal orientation. b) gives an example on doubled vertices and edges along the sharp edge

2.6.1. Origin

Beside the above mentioned intentional cases, concurrent geometry also originates during export of a model into another format. Owing to an incorrect export procedure, some objects in the model are duplicated. Moreover, the duplicated objects may also be triangulated in another way than its original copy, as shown in Fig.5a above.

Concurrent vertices also originate due to round-off mistakes. These are the most common reason in cases where vertices of neighboring triangles do not have identical coordinates. Instead, every triangle's vertex is located in a slightly different position in the space. As a side effect, a crack is introduced into the model.

2.6.2. And finally, duplicated objects might arise due to a human mistake. For example, duplicities appear as a result of copy & paste operation where the paste operation is unintentionally performed twice or even more.

2.6.3. Detection

Cases where concurrent geometry is identical are easy to detect: we find all vertices with the same coordinates. Then, we have to find out whether they were assigned to any edges or polygons; this information will be used in the correction process.

Detecting mutually shifted concurrent geometry is only slightly different. We are searching for duplicated geometry within a user provided ε-tolerance—inside the tolerance, the geometry is considered to be duplicated in contrary to the geometry outside the tolerance. A *k*d-tree can be efficiently used for such a search.

For finer search for duplicated vertices we can apply different ε value for each axis (x, y, and z).

2.6.4. Correction

Once the duplicated geometry has been found, either identical or mutually shifted, we might iterate through the duplicated vertices and leave only one of all the vertices with identical coordinates. The remaining vertices will be discarded—we choose the ones that do not form polygons. If there are more vertices forming identical polygons (edges), we discard also the redundant polygons (edges). Before such a deletion, we check the normal orientation of these polygons. In case of opposite normals, we must decide which one will be left and which one discarded. This can be done by counting the number of inside / outside oriented normals over the object. The majority decides and the user is involved in irresolute cases. Unfortunately, the majority can be also mistaken and thus, an incorrect orientation would be chosen. If the surfaces corresponding to the duplicated vertices are equal and triangulated identically, we can keep any of them.

Things become more complicated if the model includes differently triangulated surfaces. In this case, to achieve the best result we have to try all the surfaces, rank how well they fit in the model, and then choose the best one to be kept and discard the remaining ones. This requires suitable data structures and adequate ranking algorithm.

After the duplicated vertices are deleted, we must run a connecting phase again as the edges that referenced to the shifted duplicated vertices are now in correct positions but still not connected to their neighbors (still boundary edges). This leads to idea of correcting the duplicated vertices before the face connecting phase.

2.7. Self intersections

Among self intersections [Bar98] we distinguish different parts of one model penetrating each other

 ISBN 978-80-86943-02-2

(Fig.6a) from one complex object intersecting itself (Fig.6b).

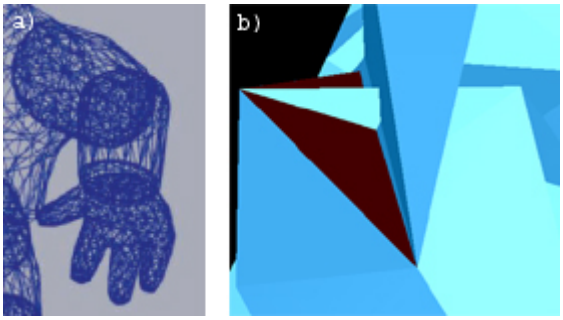

Fig.6 Examples on self intersections. a) [Bar98] illustrates mutually interpenetrating parts of the model; b) [Fel06] shows a self-intersecting object.

2.7.1. Origin

There are several sources of self intersections. First, self intersections may arise due to round-off errors. Result of such an error is a vertex shifted into a different position. In some cases, this may lead to self-intersections of incident faces.

Second, on concave objects, self intersections may be caused by using a wrong tessellation algorithm. Such an algorithm is unable to triangulate the concave parts correctly and twists the faces so that they intersect with each other.

And third, self intersections might be introduced into the model by the designer who does not notice them, for example because of a small resolution.

2.7.2. Detection

As mentioned in [Bar98], self intersecting geometry is also proximate in Euclidian space. Therefore, *k*d-tree can be efficiently used for its detection.

2.7.3. Correction

One technique for correction of self-intersections is voxelization [Noo03].

Converting a model into volumetric representation, if performed correctly, abstracts from the interior of the model and leaves only the surface. Thus, also the self intersections are left behind. However, the voxelization is suitable only for self-interpenetrating parts of one model because it corrects neither the badly positioned vertices nor the wrongly triangulated surface. It only turns the model into a 2-manifold (after the isosurface extraction).

The shifted vertices that cause the self intersections should be repositioned into a correct location and the concave parts should be retriangulated using a proper tessellating algorithm. This might be time consuming for a vast number of intersections but it is the proper solution.

2.8. Inconsistent normal orientation

Based on the origin, we tell apart cases caused by the surface reconstruction [Var05] from cases caused by improperly implemented modeling tools [Bor04].

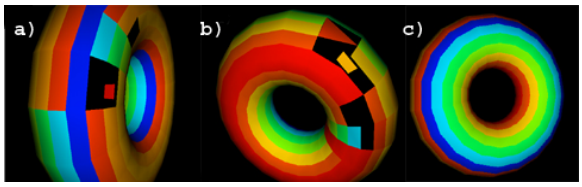

Fig.7 Inconsistent normal orientation. a) and b) show a torus with swapped normals and c) shows a correctly displayed torus

2.8.1. Origin

As mentioned above, there are two origins: surface reconstruction and modeling tools.

To reconstruction, problematic are models sampled either nonuniformly or differently in different directions.

Incorrect normal orientation that originates in modeling tools occurs randomly all over the model and depends on the software and its current version. Fig.7 above shows a VRML model exported from the 3ds format by MultiGen Creator [Mul06].

2.8.2. Detection

The detection of face orientation (vertices given CW or CCW) is possible only in 2D, so we have to find another technique. The straightest way is probably to iterate through boundary edges and seek for couples of edges with identical start and end vertices, i.e., for edge "1-->2" (starting in vertex 1 and ending in vertex 2) find another edge "1-->2". This means either that the edge is duplicated or that one of the two edges (and thus also the face belonging to that edge) has a swapped orientation.

2.8.3. Correction

There are two different aims of the correction: either to have all model normals oriented consistently or to have a model whose faces are visible from as many viewpoints as possible.

Borodin combines proximity with visibility technique to be able to achieve both. He connects the properly specified polygons into patches. These patches can touch each other only by vertices or *non-manifold edge*s [Bor04] or they do not connect with each other at all. That is, if two patches had common edges, they would be merged into one larger patch. Each pair of patches is ranked with a *boundary coherence coefficient*, which reflects how well do these patches fit together. Moreover, for each patch a front and back-face visibility is also computed. A greedy algorithm then gradually merges patch pairs with highest coherence ranking and updates their visibility ranking. The final normal orientation is

decided based on the coherence and visibility coefficients values.

2.9. Invisible polygons

Polygons become invisible [Vel06], e.g., when two walls in CAD are modeled separately and placed aside to each other (see Fig.8). If the two objects are not connected together (they just share the boundary vertices), it is not a real error and in fact, this situation occurs very often. However, invisible polygons are not needed in the model and moreover, they increase the complexity of the model.

If the invisible polygons are part of one object, it is an error because such a mesh is not 2-manifold. In this case, the invisible polygons should be removed from the model.

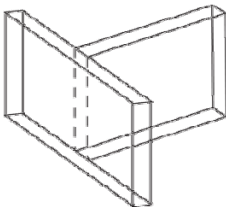

Fig.8 Invisible polygon at the connection of two walls (dashed line).

2.9.1. Origin

Generally, invisible polygons can be found on objects that stand side by side to each other. This way of placing objects is usual in building industry where the single components of the model (walls, panels) have to be separated.

2.9.2. Detection

The detection should iterate through the list of faces and look for face couples where one face overlaps the other and lies in the same plane or is coplanar and lies in the ε-distance from the other.

2.9.3. Correction

Correction of invisible polygons between two separate objects should include connection of these objects and subsequent retriangulation of the newly created object. But as we have mentioned, this is not always wanted.

Correction of a non-manifold object that contains invisible polygons comprises only of removing these polygons as there is nothing to connect or to retriangulate.

2.10. Degenerate faces

Degenerate faces [Vel06] can be subdivided into collapsed faces and non-planar faces. Among the collapsed faces, we differentiate 2D faces – lines and 1D faces – vertices. For details on faces not suitable for FEM see [Bot01].

Among the collapsed faces we count, for example, collinear vertices (Fig.9a), a set of identical vertices (AAA), or a face formed by two vertices (ABA).

In Fig.9b is an example of a non-planar face: one vertex has a different height from the others, so the four vertices do not lie in the plane. However, non-planar faces are not always considered to be errors and using them is sometimes a necessity.

2.10.1. Example

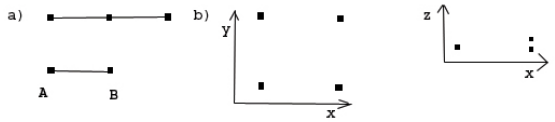

Fig.9 Invalid faces. a) A face formed by three collinear vertices is shown in the upper part of the panel and a face formed by two vertices one of which is used twice (ABA) is shown below. b) Four vertices forming a non-planar face from two points of view.

2.10.2. Origin

It must be pointed out that all the cases mentioned above may also be used intentionally. For example, MultiGen Creator exports only faces, neither it exports edges nor does it export vertices. As a consequence, designers who wish to export vertices create 1D faces, which are then depicted as a vertices.

All kinds of the errors discussed above arise during the export into VRML.

2.10.3. Detection

To cover all the above cases of degeneracies every face should be tested to be formed by more than 2 different vertices which must not be all collinear and must lie in the same plane.

Ideally, these errors should be tested and eliminated by the converter so that they do not originate at all.

2.10.4. Correction

Correction of these errors is almost impossible as we cannot find out what a correct face should look like.

2.11. Concavity errors

Exporting concavities [Vel06] (Fig.10a) brings problems too. Result of such an export is shown in Fig.10b, where the concavity is transformed into a convex object by connecting the two opposite corners of the windows. As a side effect, the new convex object overlaps the windows (marked red). This error is usual in models in the building industry; on a building frontage with concave polygons between windows, where the windows are often intersected by newly introduced mistaken edges.

Displaying concave surfaces is implementation-dependent and differs in every browser. For example a Cortona viewer displays the model shown in Fig.10

correctly while Xj3D [Xj06] has problems with displaying the concavities.

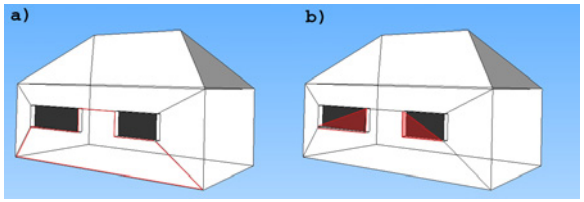

Fig.10 A misinterpreted concavity. a) shows the original concave surface (traced red) and b) shows how both windows will be overlapped by the improperly displayed concavity.

2.11.1. Origin

Concavities arise e.g. during export into a VRML format. They might be caused either by triangulation errors or by the exporter program.

2.11.2. Detection

Every face should be tested for a concavity. One of the possible ways is constructing a normal vector for each vertex of the face. This is done by multiplying the vectors representing the two edges connected to the vertex. Once we have computed all the normals, we check whether all of them have the same orientation. If not, the face is concave.

2.11.3. Correction

The concave face should be split it into two or more separate faces of which every face will be convex.

3. CONCLUSION

We gave the survey on the most typical mesh errors that often arise in CAD systems. For each error, we described algorithms for its detection and correction. We concentrated on mesh processing algorithms as they can separate the detection and the correction steps. We prefer a clear separation of detection and correction steps as it gives the user a better control over the mesh correction process. Corrected meshes should then contain no new errors, simplifications, or deformations.

We pointed out problems of two approaches: direct mesh processing and processing of a voxelized mesh. We find a combined approach (such as of Bischoff [Bis05]) as the most promising for the future research.

4. REFERENCES

[Bar98] Barequet et. al.: RSVP: A Geometric Toolkit for Controlled Repair of Solid Models, *IEEE Vis98*, 1998.

[Bis05] Bischoff, S., et. al., Automatic Restoration of Polygon Models, *In ACM Transactions on Graphics,* Vol. 24, No. 4, pages 1332–1352. 2005.

[Bøh95] Bøhn, J.H. Removing Zero-Volume Parts from CAD Models for Layered Manufacturing, *IEEE Computer Graphics and Applications,* pages 27-34, 1995.

[Bor02] Borodin, P., et. al., Progressive gap closing for mesh repairing. In *Advances in Modelling, Animation and Rendering*, J. Vince and R. Earnshaw, Eds. Springer Verlag, pages 201–213. 2002.

[Bor04] Borodin, P., *Consistent Normal Orientation for Polygonal Meshes*, Institute of Computer Science II, University of Bonn, Germany, 2004.

[Bot01] Botshc, M. and Kobbelt, L. *A Robust Procedure to Eliminate Degenerate Faces from Triangle Meshes*, CGG RWTH Aachen, 2001.

[Duc97] Duchaineau M. et al.. ROAMing Terrain: Real-time Optimally Adapting Meshes. *IEEE Visualization*, pages 81-88, 1997.

[Fel06] Felkel, P. and Obdrzalek, S. Improvement of Oliva's Algorithm for Surface Reconstruction from Contours. *SCCG'99*, pages 254-263, 1999.

[Gar97] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *ACM SIGGRAPH Computer Graphics Proceedings*, pages 209–216, 1997.

[Ju04] Ju, T., Robust repair of polygonal models. *ACM Trans. Graph. 23*, 3, pages 888–895. 2004.

[Ken98] McKenney, D., Model Quality: The Key to CAD/CAM/CAE Interoperability, International TechneGroup Incorporated, Milford, OH, 1998.

[Lie03] Liepa, P., Filling holes in meshes. In *Proceedings of the Symposium on Geometry Processing 03*. pages 200–205. 2003.

[Los04] Losasso, F., Hoppe, H., „Geometry Clipmaps: Terrain Rendering Using Nested Regular Grids." *ACM Transactions on Graphics (SIGGRAPH 2004)*, pp. 769-776, 2004.

[Mul06] *MultiGen official homepage*, http://www.multigen.com/. Last visit: 10.09.2006.

[Mur97] Murali T. M., Funkhouser T. A. Consistent solid and boundary representations from arbitrary polygonal data. *In Symposium on Interactive 3D Graphics*, pages 155-162, 196, 1997.

[Nie99] Nielson, M., Cracking the Cracking Problem with Coons Patches, *IEEE Vis99*, pages 91-106, 1999.

[Noo03] Nooruddin, F., Simplification and Repair of Polygonal Models Using Volumetric Techniques, *IEEE Vis03*, 2003.

[Pop97] Jovan Popović and Hugues Hoppe. Progressive simplicial complexes. In *ACM SIGGRAPH Computer Graphics Proceedings*, pages 217–224, 1997.

[Var05] Varnuška, M., *Surface reconstruction of geometrical objects from scattered points*, Doctoral Thesis, University of West Bohemia, 2005.

[Vel06] Veleba, D., Correction of surface representation, Bachelor's thesis, Computer Science, Czech Technical University, 2006.

[Xj06] *The official site of Xj3D project*, http://www.xj3d.org. Last visit: 10.09.2006.