Volumetric Model Repair for Virtual Reality Applications

Andreas Kolb and Lars John

Faculty of Media-Information Science, University of Applied Sciences Wedel, Wedel, Germany

Abstract

Repairing Virtual Reality (VR) models is a challenge for productive applications. This paper describes a fast implementation of Nooruddin and Turk's ray-stabbing *method* [14] *based on standard graphics hardware. Ray-stabbing is used to convert a polygonal model into a volume model (also called* voxelization*). The volume model is back-converted into a polygonal model using the marching cubes (MC) algorithm* [12] *and the QSlim algorithm* [7] *for reducing the extracted polygon model. The overall process yields a properly closed polygonal model with no visual unimportant features like nested or overlapping geometries or unwanted cracks.*

The voxelization process is the key part of the reparation process. We discuss implementation details and essential problems of ray-stabbing not addressed by Nooruddin and Turk [14]*. We focus on the generation of the volume model utilizing OpenGL hardware support.*

The current implementation is a snapshot of an ongoing work at EADS Airbus, Europes leading commercial aircraft company. The final goal is a fast model repair and reduction workflow for generating VR-models and various levels of detail. Problems erase from the fact, that the polygonalization of the volume model using the MC-algorithm generates a far too fine tessellated model which then has to be reduced again. We also discuss possible approaches to overcome this drawback.

1. Introduction

This paper focuses on implementation and optimization aspects of model repair for Virtual Reality (VR) applications. Repairing VR-models is an essential step of generating VR-models from CAD-models.

One VR-focus of EADS-Airbus (Europe's leading aircraft manufacturing company) is rapid prototyping for customizing commercial aircrafts. A typical high-end VR system is used to present customized aircrafts using a three-sided CAVE, coupled with different interaction devices and driven by a SGI ONYX2 with three graphics pipelines.

Being able to present the latest aircraft technology, EADS focuses on the optimization of the generation process of the VR-model from the CAD-model. This process starts with a polygon model exported from the CAD-system (usually CADDS 5 or CATIA). These exported model shows several troublesome properties, especially holes and nested geometries. Additionally, the model is far too fine tessellated. In the past the model optimization process at EADS incorporated tremendous manual effort to construct the final VR-model.

To reduce the model, the holes and nested or overlapping geometries have to be removed first. Especially CAD models contain potentially a large amount of this visually unimportant geometry. Furthermore, holes in the initial model cause severe problems for polygon reduction and also for further interactive processing, e.g. for collision detection. It is the task of the model repair step to remove all these problems. Therefore a robust model repair technique is most essential to the whole optimization process.

Nooruddin and Turk [14] presented a volumetric approach called *ray-stabbing* to handle all these problems. Their approach is very similar to implicit surface techniques used for geometric fusion of multiple range images (for instance Hilton et al [9]). Unfortunately Nooruddin and Turk's [14] description of the technique is rather superficial, giving no implementation details. Furthermore the results given by Nooruddin and Turk indicate a rather time-consuming implementation, which can hardly be integrate in our new workflow, which is still semi-manual.

This paper describes implementation details of the ray stabbing technique and discusses further challenges to the whole model repair workflow.

The fundamentals for volumetric model repair are described in Section 2. Section 3 describes the details of our advanced implementation of the ray-stabbing technique. Results are given in 4. Section 5 discusses the current state and various problems of the ray-stabbing method. Promising approaches to solve these problems are presented in Section 6.

2. Volumetric Model Repair

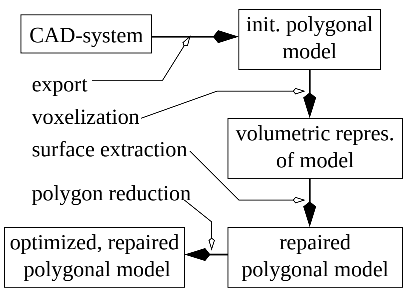
In this Section general approaches for repairing VR-models are described. Section 2.1 summerizes very briefly techniques for model repair and volumetric representation techniques for polygonal models. Afterwards the ray-stabbing algorithm as given by Nooruddin and Turk [14] is presented in Section 2.2.

2.1. General Concepts

Current methods for model repair can be separated into automatic and interactive techniques. Interactive techniques (e.g. Morvan and Fadel [13]) are not appropriate for large VR-models. Most automatic techniques incorporate the detection of the specific surface defect and the explicit reparation of these defect (e.g. Barquet and Sharir [2]). Such techniques mainly address surface holes and cracks and do not provide a unified approach to solve all problems stated earlier.

The basic idea of volumetric model repair is shown in Figure 2.1. The key-step of this process is the voxelization, i.e. the conversion of the polygonal model into a volumetric model. The volumetric model is given by a tri-variate scalar-valued *field-function* d_G describing the shortest distance of a given point $\mathbf{P} \in \mathbb{R}^3$ from the original polygonal model G. d_G is commonly called *distance map*. After voxelization standard methods for surface extraction (e.g. marching cubes [12] or marching tetrahedra [20]) are used to reconstruct G as a closed polygonal model.

Defining the distance map d_G as distance to the outer parts of G solves the problems of overlapping and nesting geometries.



To compute the distance map techniques such as 3D-filtering (Wang and Kaufman [21]) and explicit distance calculation (Schroeder et al [17]) have been developed. Nooruddin and Turk [14] use a scanline conversion algorithm to compute the distance map (see below).

2.2. Ray Stabbing

Nooruddin and Turk [14] describe a new volumetric technique called *ray stabbing* to compute the distance map for a given polygonal model. They define the distance map d_G for point $\mathbf{P} \in \mathbb{R}^3$ to the given geometry G as:

$$\left\{ \begin{array}{c} d_G(\mathbf{P}) = 0 \\ d_G(\mathbf{P}) = 1 \\ d_G(\mathbf{P}) \in [0,1] \end{array} \right\} \Longleftrightarrow \mathbf{P} \text{ resides} \left\{ \begin{array}{c} \text{outside } G \\ \text{inside } G \\ \text{near } G\text{'s surface} \end{array} \right\}$$

The ray-stabbing technique can be summarized as follows:

1. send parallel rays through G
2. a ray r defines as *valid vote*, if there is an even number of intersections with G, otherwise it is regarded as invalid. Nooruddin and Turk use orthographic projections and a polygon scan-conversion algorithm to determine depth values for each ray-object intersection, the so-called *depth maps*.
3. a voxel on r is classified interior if r is valid and the voxel lays within the first and last object intersection
4. parallel rays are send from different directions through G to handle holes, which cause an odd number of intersections
5. a voxel is finally classified as exterior if it has been classified as exterior by any direction

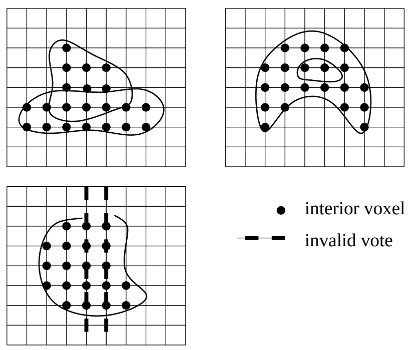


Figure 1: *Ray-Stabbing applied to overlapping (top-left), nested (top-right) and non-closed geometries (bottom). Rays are sent in the two major directions.*

This approach handles overlapping, non-closed and nested geometries from a visible point of view (see Figure 1).

Nooruddin and Turk use 13 projections to compute 13 depth maps that are finally combined to the distance map d_G. Unfortunately Nooruddin and Turk do not explicitly state how d_G is calculated.

Having an appropriate distance map d_G, Lorensen and Cline's *marching cubes* algorithm [12] is performed. This algorithm constructs a closed polygonal approximation of the volume model. Since the MC algorithm usually constructs a huge amount of triangles, Garland and Heckbert's *QSlim-algorithm* [7] is used to reduce the triangle-mesh afterwards.

Nooruddin and Turk additionally apply morphological operators, such as opening and closing to the volume model in order to remove small details.

3. Our Implementation

In this Section we give a detailed description on a fast implementation of the ray stabbing algorithm to compute the distance map d_G.

We define an oriented distance map, where $d_G(\mathbf{P}) > 0$ if $\mathbf{P}$ is exterior to G and $d_G(\mathbf{P}) < 0$ if $\mathbf{P}$ is interior to G.

The general steps to compute d_G are (see Figure 2):

1. compute the bounding sphere S^1 of G and define a regular $N \times N \times N$ voxel-grid around S^1
2. initiate the voxels distance value to `-LARGE` (interior)
3. define the *outer* bounding sphere S^2 around the voxel-grid
4. for each direction in a given set of directions:
   a. compute minimal and maximal depth maps with a predefined resolution using orthographic projections utilizing OpenGL hardware support
   b. update distance map according to current projection

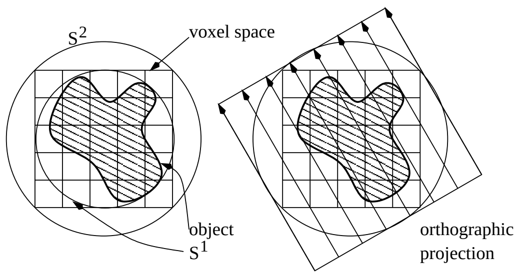


Figure 2: *The general setup using the object's and the voxel-grid's bounding sphere (left) and illustrating an orthographic projection (right).*

In our situation, we can guarantee the proper orientation of all polygonal faces of the initial model. Thus the decision on a vote's validity is equivalent to the visibility of back-facing polygons.

To compute the depth map for direction `dir` we use a standard OpenGL-renderer, especially OpenGL's culling facilities and it's stencil buffer (see OpenGL literature [15, 16] for detailed description of OpenGL functionality).

First we give a compact C-like code-fragment for extracting the necessary depth and stencil buffer:

```
void getBuffers(float depth_buf[],
                float stencil_buf[]
                int   cullDir ){
  glClearStencil(0x0);
  glClear(GL_COLOR_BUFFER_BIT |
          GL_DEPTH_BUFFER_BIT |
          GL_STENCIL_BUFFER_BIT);

  glCullFace(cullDir);
  glStencilFunc(GL_ALWAYS, 0x1, 0x1);
  glStencilOp(GL_REPLACE,
              GL_REPLACE,GL_REPLACE);
  renderScene();

  cullDir = ( cullDir == GL_BACK ) ?
            GL_FRONT : GL_BACK;
  glCullFace(cullDir);
  glStencilFunc (GL_ALWAYS, 0x0, 0x0);
  renderScene();

  readPixels(GL_DEPTH_COMPONENT, depth_buf);
  readPixels(GL_STENCIL_INDEX,  stencil_buf);
}
```

`readPixels` is a simplified version of OpenGL's `glReadPixels` function. `cullDir` indicates which viewing direction is assumed as backwards. This gets more clear further this section.

Utilizing this `getBuffers` function, the C-like pseudo-code for the extraction of the depth map looks as follows (z-Buffer range is assumed as $[0,1]$, where 1 indicates the far clipping plane):

```
glDepthFunc(GL_LESS);
getBuffers(dm_min, stencil_front, GL_BACK);

glDepthFunc(GL_GREATER);
getBuffers(dm_max, stencil_back, GL_FRONT);

foreach pixel (x,y) {
  valid(x,y) = stencil_front(x,y) &
               stencil_back(x,y);
}
```

`dm_min, dm_max` are the minimum and maximum depth map values.

Switching the depth-buffer function from `GL_LESS` to `GL_GREATER` yields the maximum depth values. The culling direction has to be switched as well, since for the maximum depth values, front-facing polygons indicate holes.

In very rare cases when a ray passes through two holes this algorithm marks an invalid vote as valid. This has not been observed in any realistic situation.

Obviously, the depth values are given in raster coordinates. In order to update the distance map d_G, we must associate the discrete voxel coordinates and the discrete raster coordinates from the various orthographic projections. Since we use orthographic projections that fit the outer bounding sphere S^2, the map between world- and raster coordinates is an affine map having only a variable rotational term (the direction). Thus the depth values from the different depth maps are comparable, i.e same depth values represent same euclidian distances.

In detail the transformation $T_{voxel\to rast}$ to map the voxel- to the raster coordinates is composed using $T_{voxel\to unit}$ (maps to the unit cube $[-1,1]^3$), R (rotation about the origin) and $T_{unit\to rast}$. R is a simple rotation matrix, whereas $T_{voxel\to unit}$ and $T_{unit\to rast}$ involve only fix scale and translational terms. The resulting transformation is

$$T_{voxel\to rast} = T_{unit\to rast} \cdot R \cdot T_{voxel\to unit}$$

After transforming a voxel $\mathbf{V}$ to raster coordinates using $T_{voxel\to rast}$ yielding $\mathbf{V} = (v_x, v_y, v_z)$, the current depth map is used to update the distance map. This is done by bilinear interpolating the minimum and maximum depth-map values of the four surrounding pixels of (v_x, v_y) and selecting the shortest distance (see Figure 3). Using nearest neighbour sampling instead results in visible aliasing artifacts.

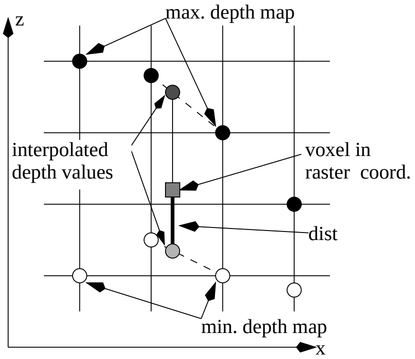


Figure 3: *Computing the distance value for a specific projection direction in raster coordinates.*

Additionally we check for invalid votes or too steep depth variation, which indicate pixels near the object's boundary. This is done to prevent interior voxels from getting incorrect positive distance values, since one outside ray would cause $\mathbf{V}$ to be declared exterior (see Figure 4).

The computation of the minimum and maximum depth map values for $\mathbf{V}$ as code fragment:

```
x = floor(v_x);
y = floor(v_y);

if ( !valid(x,y)   || !valid(x+1,y+1) ||
     !valid(x+1,y) || !valid(x,y+1)  ) {
  depth_min = 0.0;   /* set interior */
  depth_max = 1.0;
}
else {
  f1 = dm_min(x,y);   f2 = dm_min(x+1,y);
  f3 = dm_min(x,y+1); f4 = dm_min(x+1,y+1);
  min_low  = minimumOf(f1, f2, f3, f4);
```

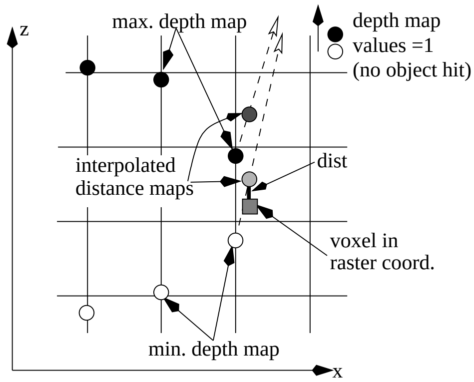


Figure 4: *The boundary situation: One ray near the voxel misses the object. The voxel's is placed interior, i.e.* `depth_min=0` *and* `depth_max=1`.

```
  min_high = maximumOf(f1, f2, f3, f4);

  b1 = dm_max(x,y);   b2 = dm_max(x+1,y);
  b3 = dm_max(x,y+1); b4 = dm_max(x+1,y+1);
  max_low  = minimumOf(b1, b2, b3, b4);
  max_high = maximumOf(b1, b2, b3, b4);

  if( (min_high-min_low > MAX_GRADIENT) ||
      (max_high-max_low > MAX_GRADIENT) ) {
    depth_min = 0.0;   /* set interior */
    depth_max = 1.0;
  }
  else {
    depth_min = interpolate(v,x, v,y,
                            f1, f2, f3, f4);
    depth_max = interpolate(v,x, v,y,
                            b1, b2, b3, b4);
  }
}
```

Finally, the distance value for $\mathbf{V}$ is calculated. Initially $\mathbf{V}$ is marked as interior, i.e. `dist_map(V) = -LARGE`. Once a positive distance is found, $\mathbf{V}$ is marked as exterior and it has be keep this property.

The relevant code fragment:

```
midpoint = ( depth_min + depth_max ) / 2.0;
if ( v_z < midpoint ) {
  dist = depth_min - z;
}
else {
  dist = z - depth_max;
}

if ( ( distance_map(V) < 0 &&
       dist > distance_map(V) ) ||
     ( dist > 0 &&
       distance_map(V) > dist ) ) {
  dist_map(V) = dist;
}
```

The value `dist` represents the approximated voxel's distance for the current direction in raster coordinates (see Figure 3).

Considering polygon reduction, lots of techniques have been introduced in the last years. Rather than implementing and comparing the different methods we made our choice based on Cignoni et al's comparison [5]. They give a comparison on the following techniques: Multiresolution Decimation (Ciampalini et al [4]), Simplification Envelopes (Cohen et al [6]), Quadric Error Metrics – *QSlim* (Garland and Heckbert [7]), Mesh Optimization (Hoppe [10]) and Mesh Decimation (Schroeder et al [18]), .

Cignoni et al compare the algorithm's speed using three different polygonal models. Additionally distance errors are given. According to their results, the QSlim-algorithm [7] offers a good trade-off between speed and accuracy. Thus we use the same polygon reduction scheme as Nooruddin and Turk[14].

The setting of various parameters

We tried different ratios of rendering to voxel grid resolution. Our experience show good results for a resolution ratio of approximatly 5. Instead of 13 projections as Nooruddin and Turk [14], we found seven directions (the three major axis and the four cube diagonals) to be enough.

Concerning the rejection of a vote as invalid is case of a large gradient, we found a bound of 5 to give good results, i.e.

```
MAX_GRADIENT = 5 / RENDER_RESOLUTION
```

4. Results

To discuss the method described above, we use three parts of an aircraft as shown in Figure 5: A seat, an upperdeck part and a side-plate. All models exhibit small hole at the edges of the original surface patches.

Reducing these models with QSlim without taking care of the holes and the nested geometries yields undesired holes and cracks in the reduces model (see Figure 6).

Applying the model repair before reducing the polygonal model yields perfectly closed and well shaped results (see Figure 7)

Problems appear in regions with thin parts. This can be seen at the base of the seat, where many tee-like geometries reside. These thin object regions can not be reconstructed properly. Figure 8 compares the results of applying QSlim only and Ray Stabbing in combination with QSlim.

The problem with thin object regions is extremly obvious in case of the side plate. The side-plate model resembles a deformed thin sheet of metal. Thus even a relativ high voxel grid resolution of 200^3 can not represent the polygonal model, resulting in a only fragmentary reconstructed model (see Figure 9 left).

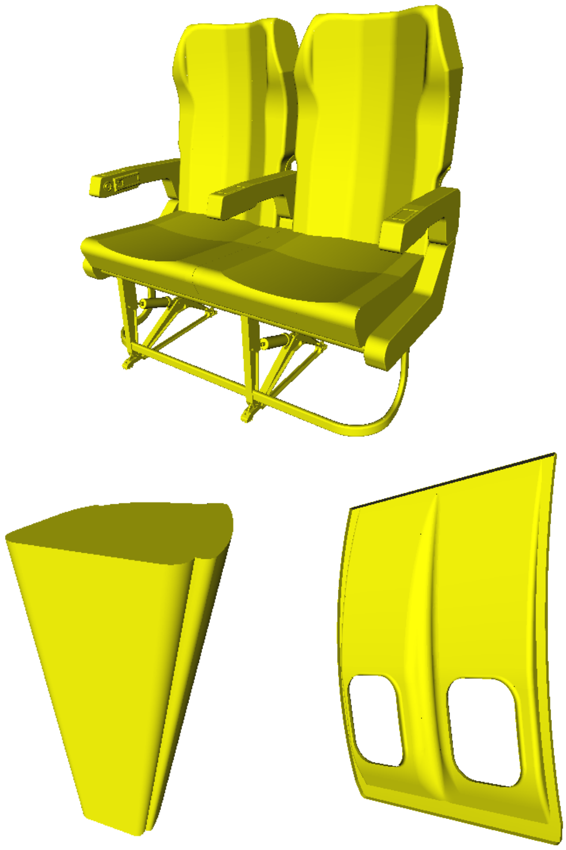


Figure 5: *The original models. Top: Seat model with 52639 triangles. Bottom left: Upperdeck model with 324 triangles. Bottom right: Side-plate model with 16896 triangles.*

A simple, but not very accurate approach to reduce this problem is the usage of an iso-value greater than zero (see Figure 9 right). The resulting objects are not perfectly shaped. This is due to fact, that the distance map's approximation of the euclidian distance is more inaccurate for point off the original object's surface.

5. Conclusion

A fast and robust implementation of a volumetic model repair technique has been introduced. This technique is based on Nooruddin and Turk's ray stabbing method. Many details of the hardware accelerated implementation have been described. Furthermore, essential problems caused by the marching cubes (MC) algorithm have been discussed and possible solution have been pointed out. Finally some results based on technical objects have been presented.

The Ray Stabbing algorithm proved to be a powerful tech-

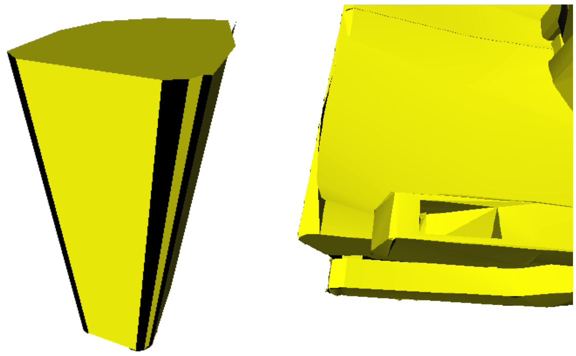


Figure 6: *Applying polygon reduction only. Left: The upperdeck with holes. Right: A close-up to the arm-rest of the seat with holes and cracks. Both models have been reduced by 90%.*

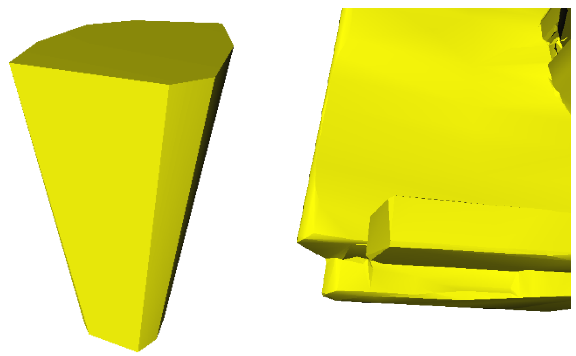


Figure 7: *Applying model repair and polygon reduction. Left: The upperdeck has been voxelized using a* 40^3 *voxel-grid. Right: The chair has been voxelized using a* 120^3 *voxel-grid. Both models have been reduced by 90%.*

nique for separating and deleting visually unnecessary geometry. Furthermore the resulting object surface is properly closed. Our implementation for computing the distance map reduces the calculation time from several minutes (see Nooruddin and Turk[14], table 1; samples computed on a multiprocessor system) to less than a minute. Additionally, the calculation time is relatively independant of the number of triangles in the initial model.

Still the process exhibits one major problem. The choice of the size of the voxel-grid determines the maximum frequency that can be represented by the grid's distance function. This frequency however can be very high, especially when working with technical object containing thin sections.

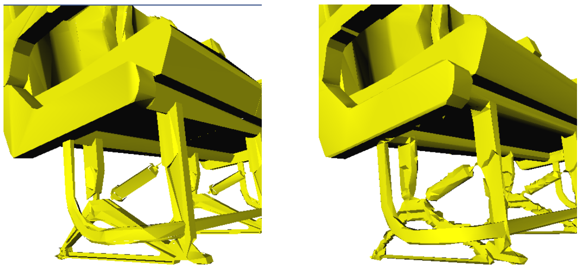


Figure 8: *The seat base. Left: Only QSlim was applied; the model exhibits several large holes. Right: Model repair and polygon reduction were both applied. The models have been reduced by 90%.*

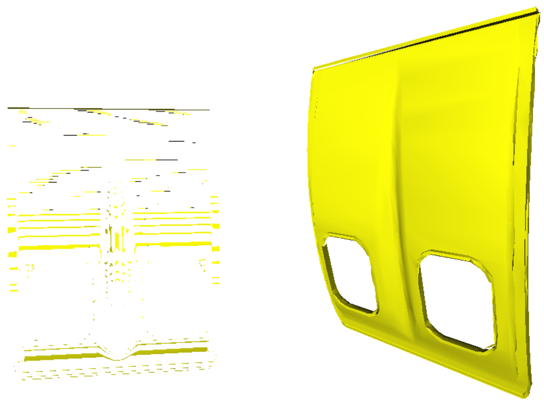


Figure 9: *The side plate. Left: Attempt to repair the very thin model. Right: Reconstructing an iso-surface for* $d_G \approx 1\%$ *of object's size. Both models have been reduced by 90%.*

This is a classical problem of sampling theory and the reconstruction of continuous objects in discrete spaces.

To aviod this problem, the highest frequency has to determine the overall voxel-grid size. Thus, using the marching cubes algorithm, a unnecessary high amount of triangles is constructed, which, again, has be reduced in a following polygon reduction. Both steps, extracting the iso-surface and reducing the mesh-complexity, thus, consume much more time than necessary. Furthermore the calculation of the distance map has to be done for all voxels. Adaptive algorithms would evaluate the distance map only at points near the iso-surface, i.e. near the model itself.

In section 6 we discuss possible approaches to solve this problem.

6. Further Optimization

For most models the choice of the fixed voxel step-size results in a far too long runtime of the algorithm. The fix

	Upper-deck	Seat	Side-plate
Voxel-grid	40^3	120^3	200^3
Reduction	10%	10%	10%
depth-map	2”	23”	1’35”
dist.-map	2”	55”	5’05”
MC	4”	4’13”	2’15”
QSlim	13’	2’05”	57”

Table 1: *Timing information for the different models. The test have been made on a SGI O2 with R12K, 300 MHz Processor and 512 MB RAM. The runtime for the side-plate model is taken for the iso-value $d_G = 0$.*

choice of the step-size in combination with the simple application of the marching cubes algorithm potentially constructs far too many triangles. Furthermore the MC algorithm froces the evaluation of the distance map at each voxel, even far off the iso-surface.

One alternative is Hilton et al's [8] *Marching Triangle (MT)* algorithm. The MT-algorithm is a mesh-growing approach, utilizing a circumssphere technique similar to the construction of a Delaunay triangulation. The resulting triangulation exhibits better shaped triangles. The construction of new vertices starts with a point with fixed distance to a current boundary edge. Thus, the stated guarantee to preserve the models's local topology is not obvious to us. The generation of new vertices is supposed to be adaptive with respect to the local behavior of the field function.

Another alternative is a hierarchical MC-approach, using an adaptive step-size. Shekhar et al [19] use a hierarchical octree-approach to reduce the number of generated triangles. They apply a merging strategy, trying to combine small triangles to larger ones. This process is still more time consuming since it is a bottom-up approach.

Shekhar at al's technique could be reformulated as a top-down approach. Starting with a coarse grid, the refinement algorithm now consists of the following steps:

Refinement-Decision: Has the current cell (a cube with eight voxels as corners) to be subdivided?
In the case of Ray Stabbing finding the answer is simple: If the distance between the first and last object intersection for a ray direction is less than the current grid-resolution, the voxel should be subdivided.

Constrained MC: If a voxel has to be subdivided, we us a 2^3-subgrid. The MC algorithm potentially generates triangles on the finer grid. Thus, we have to guarantee that adjacent triangles in neighbouring coarser cells define a proper triangulation in conjunction with the subgrid triangles.

Assuming the refinement-decision guarantees that a cell is subdivided when the field function additionally changes sign along one of its edges or at the midpoint of its faces. This midpoint-criterion should also be used to handle the ambigouos cases of the MC algorithm. Now consider neighbouring cells at different levels of refinement. Edges of generated triangles for these cells can simply be matched, since no additional edges (or polylines) can appear (see Figure 10). Thus subdividing the triangles for the coarser cell according to the polyline of the finer cells closes the gaps between the triangulations of different level of refinement.

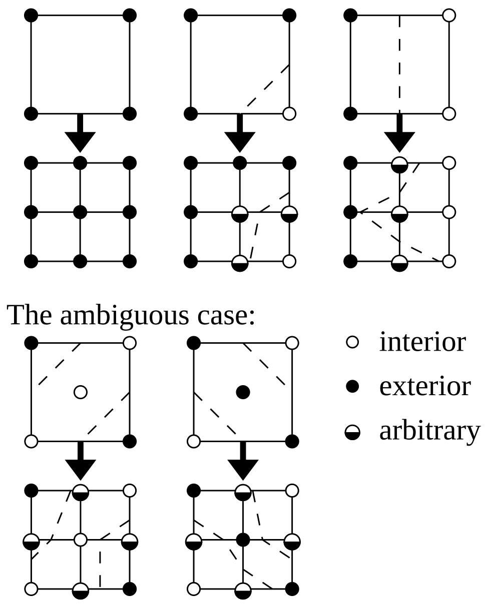


Figure 10: *Neighboring cell with different level of refinement. Edges on cell of the coarser grid correspond to polylines on the finer grid.*

Acknowledgment

We thank the stuff at EADS Airbus, Hamburg for their cooperative support. Especially we thank Dieter Kasch and Jörn Düring, the VR-system administrators, as well as Dr. Stefan Hiesener, responsible for R&D-activitices in cooperation with universities.

References

1. M.-E. Algorri and F. Schmitt. Mesh simplification. In *Proc. EUROGRAPHICS*, volume 15, pages 77–86. Eurographics, 1996.

2. G. Barquet and M.Sharir Filling gaps in the boundary of a polyhedron. *Computer-Aided Geom. Design*, volume 12, pages 207–229, 1995.

3. S. Campagna, L. Kobbelt, and H.-P. Seidel. Efficient

decimation of complex triangle meshes. Technical Report, Universität Erlangen-Nürnberg, 1998.

4. A. Ciampalini, P. Cignoni, C. Montani, and R. Scopigno. Multiresolution decimation based on global error. Technical Report CNUCE: C96021, Istituto per l'Elaborazione dell'Informazione - Condsiglio Nazionale delle Richere, Pisa, ITALY, 1995.

5. P. Cignoni, C. Montani, and R. Scopigno. A comparison of mesh simplification algorithms. *Computers and Graphics*, volume 22, pages 37–54, 1998.

6. J. Cohen, A. Varshney, D. Manocha, G. Turk, H. Weber, P. Agarwal, F. Brooks, and W. Wright. Simplification envelopes. In *ACM Proceedings SIGGRAPH*, volume 30, 1996.

7. M. Garland and P.S. Heckbert. Surface simplification using quadric error metrics. In *ACM Proceedings SIGGRAPH*, volume 31, pages 209–216, 1997.

8. A. Hilton, A. Stoddart, J. Illingworth, and T. Windeatt. Marching triangles: Range image fusion for complex object modeling. In *International Conference on Image Processing*, pages 381–384, 1996.

9. A. Hilton, A. Stoddart, J. Illingworth, and T. Windeatt. Reliable surface reconstruction from multiple range images. In *Fourth European Conf. on Computer Vision*, pages 14–18, 1996.

10. H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *ACM Proceedings SIGGRAPH*, volume 27, pages 19–26. ACM, 1993.

11. H. Hoppe. Progressive meshes. In *ACM Proceedings SIGGRAPH*, volume 30, pages 99–108, 1996.

12. W.E. Lorensen and H.E. Cline. Marching cubes: A high resolution 3d-construction algorithm. In *ACM Proceedings SIGGRAPH*, volume 21, 1987.

13. S. Morvan and G. Fadel IVECS: An Interactive Virtual Environment for the Correction of .STL Files. Conference on Virtual Design, University of California at Irvine, Irvine, CA., 1996.

14. F. Nooruddin and G. Turk. Simplification and repair of polygonal models using volumetric techniques. Technical Report GITGVU -99-37, Georgia Institute of Technology, Atlanta, 1999.

15. OpenGL Architecture Review Board. *OpenGL Programming Guide*. Addion Wesley Longman, 1999.

16. OpenGL Architecture Review Board. *OpenGL Reference Manual*. Addion Wesley Longman, 1999.

17. W. Schroeder and W. Lorensen and S. Linthicum Implicit modeling of swept surfaces and volumes In Proceedings IEEE Conference on Visualization'94, pages 40–45, 1994.

18. W.J. Schroeder, J.A. Zarge, and W.E. Lorensen. Decimation of triangle meshes. In *ACM Proceedings SIGGRAPH*, volume 26, pages 65–70, 1992.

19. R. Shekhar, E. Fayad, R. Yagel, and F. Cornhill. Octree-based decimation of marching cubes surfaces. In *Proceedings IEEE Conference on Visualization*, pages 335–342, 1996.

20. P. Shirley and A.A. Tuchman. Polygonal approximation to direct scalar volume rendering. In *Proceedings San Diego Workshop on Volume Visualization, Computer Graphics*, volume 24:5, pages 63–70, 1990.

21. S. Wang and A. Kaufman Volume Sampled Voxelization of Geometric Primitives In Proceedings IEEE Conference on Visualization'93, pages 78–84, 1993.