Interactively Visualizing 18-Connected Object Boundaries in Huge Data Volumes

Robert E. Loke and Hans du Buf

Vision Laboratory, University of Algarve, 8000-810 Faro, Portugal
{loke,dubuf}@ualg.pt
http://w3.ualg.pt/∼dubuf/vision.html
tel: +351 289 800900 ext. 7761, fax: +351 289 819403

Abstract. We present a multiresolution framework for the visualization of structures in very large volumes. Emphasis is given to an in the framework embedded, new algorithm for triangulating 18-connected object boundaries which preserves 6-connectivity details. Such boundaries cannot be triangulated by standard 6-connectivity algorithms such as Marching Cubes. Real sonar imaging results show that the framework allows to visualize global subbottom structure, but also high-resolution objects, with a reduced CPU time and an improved user interactivity.

Keywords: Boundary triangulation, Marching cube, Voxel connectivity, Visualization.

1 Introduction

Visualization facilitates the analysis, modeling and manipulation of scalar data volumes. Visualization can be done by direct volume rendering (DVR) and surface rendering [1, 2]. In surface rendering, object boundaries are visualized by first extracting a geometric model of the volume (iso)surfaces and then by rendering the model. Advantages are that it is fast and that memory requirements are low if compared to DVR, because the geometric model has to be extracted only once and rotations etc. deal with the model only, and are not again affected by the entire data volume, like in DVR. Furthermore, realtime shading algorithms and hardware support are available for surface graphics.

In this paper we describe our visualization framework which has in part already been published before, see e.g. [3]. However, here we accurately define and extend the embedded boundary triangulation. Below, we describe the framework and triangulation algorithm used to build surfaces for detected object boundaries (Sections 2 and 3), apply them to a real sonar dataset (Section 4) and give conclusions and directions for future work (Section 5).

2 Interactive Visualization

Similar to other approaches [4, 5], we render surfaces in an octree, aiming at quick (multiresolution) processing and fast user interactivity. Octrees are representations of volumes in which different spatial resolution levels are computed

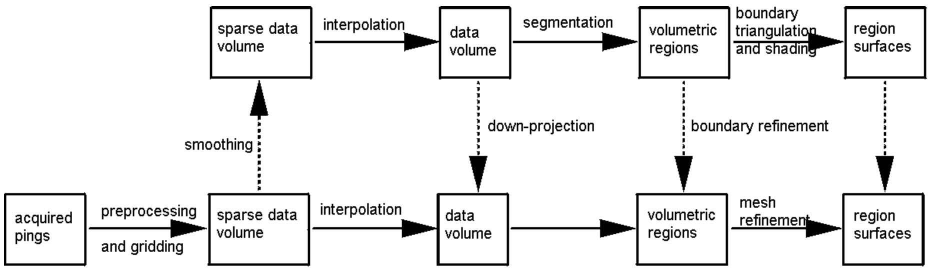


Fig. 1. Visualization at different resolution levels in the octree spatial data structure.

by sampling or filtering data in blocks of 2×2×2 [6]. They are hierarchical data structures with explicitly defined parent-child relationships: a parent represents 2×2×2 voxels at the lower level and $2^2 \times 2^2 \times 2^2$ voxels at the next lower level etc. We use an octree in which low resolution data at the higher tree levels are determined by spatially smoothing the available data at the lower tree levels. Voxel values at a higher level are the average of all values of non-empty data voxels in non-overlapping blocks of size 2×2×2 at the lower level. This simple processing results in a fast tree construction and facilitates quick data visualizations at low resolutions, because in the tree both the signal noise and the size of gaps decrease, even for huge volumes with a large number of gaps or volumes reconstructed from very noisy data. The loss in spatial resolution at the higher tree levels is compensated using adequate down-projection techniques. In particular, once the data have been classified at a high tree level, the boundaries of the segmented regions are refined by filtering the available data at the lower levels [7].

After first selecting a region of interest (ROI), and registering the selected data to a regular 3D grid, we do all critical processing in an octree. Because the tree construction and the processing at the highest tree levels is very fast, initial coarse visualizations are quickly obtained, such that the ROI can be immediately adjusted. The initial coarse visualizations already give much insight in the structures which are being studied and are only refined, i.e. the data at the lower tree levels are only processed, if the ROI has been correctly set. A first, coarse visualization at the lowest resolution is obtained by interpolating data around gaps, segmenting the volume into regions, and constructing shaded, colored and/or transparent surfaces for all region boundaries. See Fig. 1 (pings are specific underwater acoustic signals which represent vertical columns in the volume). Next visualizations at higher resolutions are obtained by down-projecting and interpolating the available data into gaps, and refining the segmented structures and the constructed surfaces. Importantly, once the data have been visualized, the processing can be stopped at any moment in order to select a new ROI. The processing proceeds only if, according to the user, the ROI has been correctly set. If not, the processing is stopped and a tree is built for another ROI.

The octree provides a computational framework in which the following techniques can be employed: (A) the construction of a quadtree that allows to fill empty voxel columns [8], (B) a first but coarse visualization at a high tree level in order to rapidly adjust the ROI, and (C) a very efficient triangulation (mesh reduction) that allows for a fast interactivity even at the highest detail level. By using one single octree all this processing can be combined because (1) gaps can be filled by interpolation because they are smaller at higher tree levels, (2) connected components can be projected down the tree and refined using the data available there and (3) triangulations at higher tree levels can be used to steer those at lower levels to fill efficiently large and smooth surface areas. After the segmentation (and possibly a connected-component labeling) in the visualization framework, the object boundaries are visualized using surface rendering. Software libraries such as OpenGL (Open Graphics Library) or VRML (Virtual Reality Modeling Language) provide interfaces which enable an interactive analysis of structures by "flying" through and around the structures of interest. Thus, apart from using an octree, we use two extra techniques for improving interactivity: the selection of a ROI and the use of VRML/OpenGL.

3 Triangulation

The well-known Marching Cubes (MC) algorithm, as well as topology improved [9] and efficiency enhanced – in terms of a reduced triangle count – versions are all based on locally triangulating cuberille (2×2×2 voxel) configurations. Other surface construction algorithms decompose the cuberilles into voxels or tetrahedra, use boxes instead of cuberilles, use polyhedra or polygonal volume primitives instead of triangles, use rules instead of a look-up table for cuberille configurations, use heterogeneous grids to guarantee topologically coherent (closed, oriented) surfaces, or optimize the search of relevant cuberilles. In contrast to all these algorithms we:

1. Triangulate object boundaries by mapping complete 3×3×3 neighborhoods to polygons. This allows to optimize the polygons locally.
2. Interpolate between the coordinates of boundary voxels. This improves the smoothness of the built surfaces.
3. Allow 18 connectivity for objects (like in [9]; unlike 6 connectivity in MC)[1]. This allows to construct surfaces for boundaries which are not connected according to a 6-connectivity model, e.g. an object boundary which is tilted and thinned, see Fig. 2 (left).

Our algorithm is based on a property of non-intersecting surfaces, excluding the edges: for such surfaces, each point on the surface has exactly four neighboring

[1] Here we note that two voxels are n-connected ($n = 6, 18, 26$) if there exists a path between the voxels such that all subsequent voxels on the path are maximally n-adjacent one to another. Two voxels are n-adjacent if they are n-neighbors. The 6-neighborhood (respectively, 18-, 26-neighborhood) of a voxel at (x, y, z) is comprised by these voxels for which $|x - a| + |y - b| + |z - c| = 1$ (2, 3), with (a, b, c) arbitrary voxel coordinates. Thus, 6-connected voxels are also 18-connected and 26-connected, but 18-connected ones not 6, and 26-connected ones not 18 nor 6.

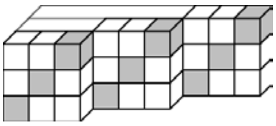
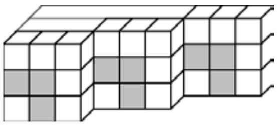


Fig. 2. Two examples of 3×3×3 voxel neighborhoods. Boundary voxels are grey. On the left, the boundary is 6-connected in z (into the paper), and 18-connected in the (x, y)-plane. On the right, it is 6-connected in the (x, y)-plane or, put differently, 18-connected with 6-connected shortcuts. The 2nd and 3rd layer have been shifted to the right in order to show their contents.

points which are also located on the surface. In discrete terms, this means that between a boundary voxel a and another boundary voxel in its neighborhood, say b, two other adjacent boundary voxels c and d are needed to form a surface patch $a - c - b - d$. Below, we will distinguish between two types of voxels: face voxels and non-face voxels. Figure 3 (a) shows the definition of *face* voxels in the 3×3×3 neighborhood N of a boundary voxel B. Since we assume up to 18-connectivity for objects, a boundary voxel in N is a face if it is 6-connected to B, but also if it is 18-connected to B and no other boundary voxel can be found which 6-connects the voxel and B. If a boundary voxel in N is not a face, we call it a *non-face*. Furthermore, we will model the boundary topology using a very small set of configurations with, in each configuration, varying connectivity paths between a and b. In these configurations, a boundary voxel is 18-connected and sometimes 6-connected to each other boundary voxel in its 26-neighborhood. Then, by defining a surface patch for each configuration, object boundaries can be mapped to surfaces. Finally, we will extend the set of configurations in order to correctly model and map non-thin boundaries, i.e. boundaries with additional 6-connectivity paths between a and b.

3.1 Boundary Definition

In order to triangulate the boundaries in a volume, we first must determine all boundary voxels. Here, we define a voxel at (x, y, z) to be part of a component's boundary if at least one of the values of the voxels at $(x+1, y, z)$, $(x-1, y, z)$, $(x, y+1, z)$, $(x, y-1, z)$, $(x, y, z+1)$ and $(x, y, z-1)$ differs from its own value. However, the triangulation is not restricted by this definition, i.e. other boundary definitions may be used, employing for example 18- and 26-neighborhoods. Obviously, the resulting boundaries are not necessarily smooth, e.g. they may contain sharp corners/edges: in neighborhoods, boundaries may be both 6- and 18-connected, see Fig. 2 (right). Thinning can be used to remove boundary voxels which do not contribute to the connectivity of the boundary. This normally decreases the triangle counts of the resulting surfaces. However, in some applications this leads to undesired information loss or deformations. We note that for a correct application of our algorithm, thinning may be done but is not required.



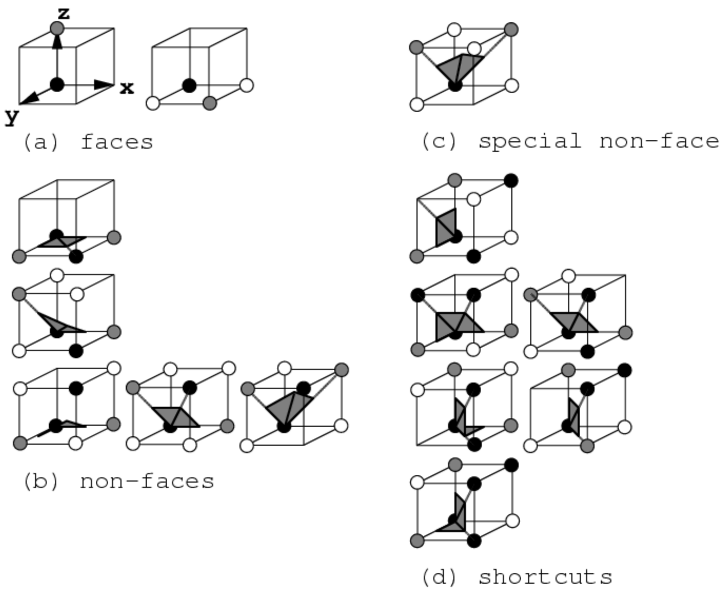


Fig. 3. Triangulation look-up table for boundary configurations with varying connectivity between voxel pair $a = (0, 0, 0)$ and $b = (1, 1, 0/1)$, or triplet a, $c = (0, 1, 1)$ and $d = (1, 0, 1)$. Only some configurations for one octant in the 3×3×3 neighborhood around a boundary voxel are shown; the other configurations for the same octant and those for the other octants are obtained by mirroring. Boundary voxels are either black or grey; background voxels white. Grey spheres denote face voxels. Corners on the cubes without spheres are positions which do not affect the connectivity.

3.2 Boundary Matching

We triangulate the volumetric boundaries locally, in the 3×3×3 neighborhood N around each boundary voxel B. We independently map the boundary in each of the eight octants in N to multiple vertex lists, such that in each list the coordinates and the order of the vertices of a matched boundary configuration are defined. This decomposition into octants allows to: (A) reduce the total number of configurations, and (B) correctly map neighborhoods at edges of boundaries.

Figure 3 (b), (c) and (d) show configurations for the octant in N with positive x, y and z coordinates, together with the triangles which are to be applied. The configurations for the other octants are obtained by mirroring about the planes $x = 0$, $y = 0$ and $z = 0$, about the x, y and z axes, or about B. The total number of configurations has been reduced using mirroring about the diagonal planes $x = y$, $x = z$ and $y = z$. We[2] obtained the configurations by: (A) determining the set of all valid (i.e., 6- and/or 18-connected) $a-c-b-d$ boundary voxel patterns in N (yielding Fig. 3 (b) and (c)); (B) extending the resulting set by increasing

[2] Similar configurations have also been obtained from a theoretical approach [10], providing a topological validation of the object surfaces built by our algorithm.

the boundary connectivity in all patterns (yielding Fig. 3 (d)). There are totally 12 configurations which are divided into three different types: (1) Configurations with four boundary voxels in which the non-face is 6- and/or 18-connected to B using exactly two faces. (2) A special configuration with three boundary voxels in which the non-face is (assumed to be) located outside N, which is 18-connected, again using exactly two faces. This configuration corresponds to the case in which the position of the center is (x, y, z), and two faces exist at $(x+1, y, z+1)$ and $(x, y+1, z+1)$. Then the voxel which is adjacent to both faces may be positioned outside N, at $(x+1, y+1, z+2)$. (3) Configurations with more than four boundary voxels, in which the 18-connected voxels in (1) are now also 6-connected. The latter configurations we call *shortcuts*, because they add an extra 6-connectivity to an already existing 18-connectivity.

For each configuration a vertex list is defined, which specifies the coordinates and the order of the vertices which are to be applied in the triangulation. Vertex coordinates are determined for each boundary voxel in N (apart from B, whose voxel *and* vertex coordinates are (0,0,0)) in one of two ways, dependent on whether it is a face of B or not. The vertex position computed for each face is the average of the voxel coordinates of B and the face. The vertex position of each non-face is the average of the neighboring four voxel coordinates, except for the one in the special non-face configuration, for which the coordinates are $(0.33, 0.33, 0.67)$, and the additional non-faces in the shortcut configurations. All vertex coordinates can be derived from Fig. 3, e.g., for the second shortcut (column 1, row 2) the vertex list is $\{(0.5, 0, 0), (0.5, 0.5, 0.5), (0, 0.5, 0.5), (0, 0.5, 0)\}$.

We match each octant in N with all (mirrored) configurations. If a configuration matches an octant, the (mirrored) vertex list of the configuration is stored. By using "don't care" voxels, i.e. voxels which may belong to either the boundary or the background, multiple configurations can match the same octant. This allows to correctly map "sharp" boundaries to surfaces. We note that this even allows to map intersecting boundaries, but that for intersections the linking algorithm [11] is not trivial. The neighborhood matching results in a number of vertex lists, which must be stored for all positive matches, in each of the eight octants. The order of the vertices in each list is implicitly defined in Fig. 3. After the matching, the order of the vertex lists is determined by linking all vertex lists, and the final patch can be triangulated and optimized [11]. Also, a normal vector is attributed to the patch for surface shading.

Figure 4 shows surface patches obtained by triangulating the boundaries of a cube of size 16×16×16 and a sphere of radius 14, without any patch optimization.

4 Visualization Results

We obtained several 3D datasets by maneuvring vessels mounted with bottom-penetrating sonar in shallow water areas. Dataset sizes may range up to several GBs per seabed, and this will further grow due to increasing demands on sampling rate and trace size. Obviously, it is impossible to conduct a vessel such that an entire site is scanned, which implies that a lot of 3D data are missing. Com-

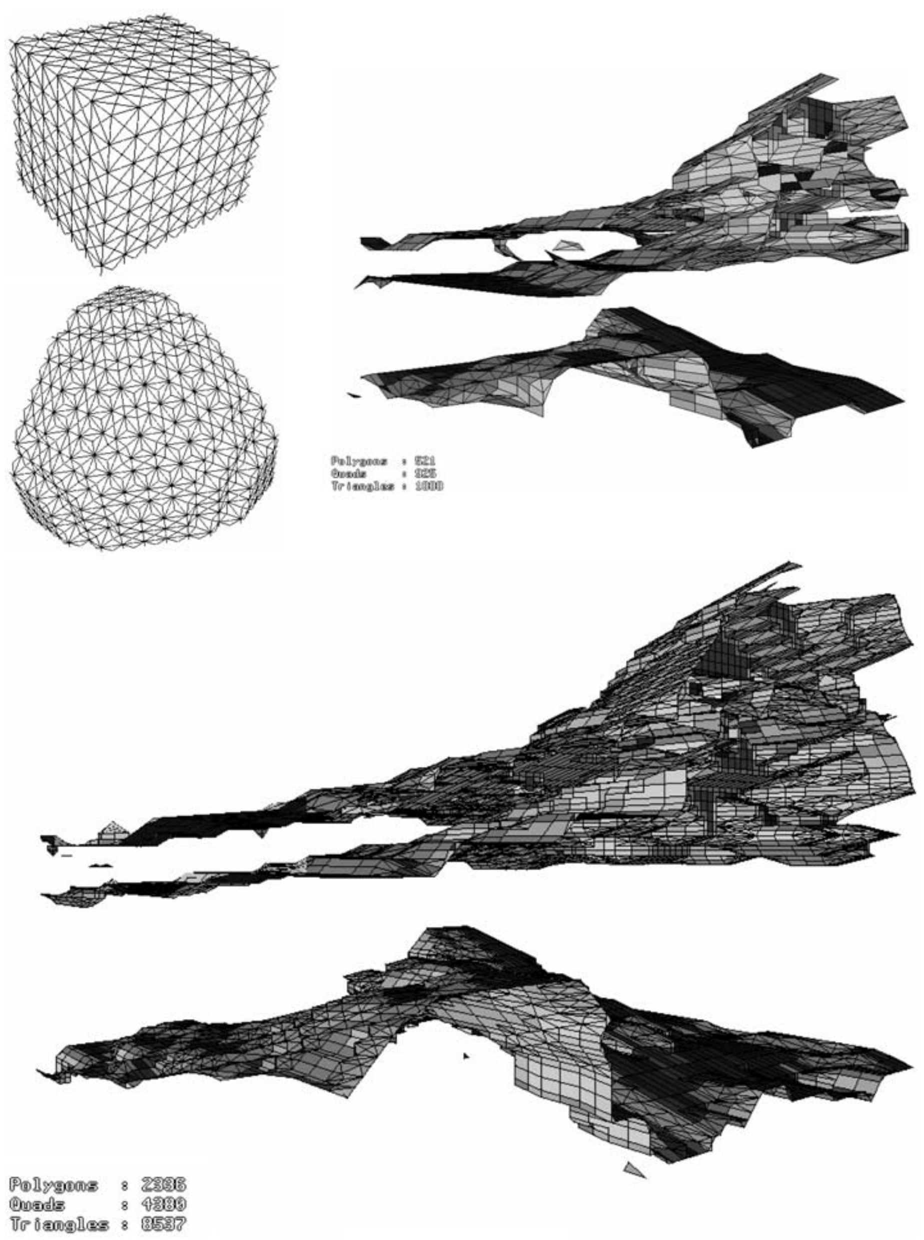


Fig. 4. Wireframes of 1/8th part of a cube and a sphere (top left), and shaded and optimized surfaces of detected subbottom structures in the large ROI at octree level 2 (top right) and 1 (bottom). Note the improvement in detail when refining structures from level 2 to 1.

monly, sonar operators need to explore data at different scales. They may want to visualize a large area of a seabed, but also a small part, for example when they look for objects. The analyses which are required demand for different sampling rates. Here, we will show volumetric reconstructions of a seabed at two different scales, in two different ROIs: a large region in which the size of the voxels equals $3.8 \times 4.5 \times 0.6\ m^3$ and a small region with a voxel size of $0.5 \times 0.5 \times 0.08\ m^3$.

Figure 4 shows shaded surfaces with wireframes for boundaries of the structures found in the large ROI. The images were obtained by mapping all data from one site to a regular grid of size 32×128×128. Thereafter, 29% of the volume was filled. The octree consisted of 4 levels. For these images, we did not apply any interpolation. We directly constructed the tree and projected the segmented boundaries from the highest tree level to the lower levels using a robust planar filtering on the boundary data [7]. CPU times on an SGI Origin 200QC server, using all 4 processors and including disk IO, were 1.6, 1.3, 4.4 and 19.7s at octree level 3, 2, 1 and 0. We note that the Origin has MIPS R10000 processors at 180 MHz, and that a normal Pentium III computer at 733 MHz is faster than one Origin processor by a factor of 2.2. Using the latest GHz processors, the total time, about 27s, can be reduced to less than 4s. Hence, our visualization framework can be applied in realtime for routine inspection and interpretation.

Ideally, octrees can be used for visualizing large structures in huge data volumes at high tree levels and small ones at low levels. Here we have preferred to select another, much smaller ROI, and to reconstruct another volume (of size 384×64×700) at a much higher spatial resolution. The vertical spatial resolution in depth was increased by averaging and sampling each underwater acoustic signal with a mask of size 2 (for the large ROI a mask of size 40 was used). In order to automatically detect and visualize the sewage pipes which appear at this level of detail, and to cope with the increased data sparseness (this volume was filled for only 9%), we performed additional *inter*-slice interpolations. In these interpolations, we match/correlate voxel columns in order to correctly obtain single surface reflections and to avoid artificial double/multiple reflections [8]. Hereafter, an octree of three levels was built in order to interpolate remaining gaps, automatically detect the pipes and triangulate their boundaries. The CPU time was 228s for the inter-slice interpolation and 28, 127 and 241s for the octree processing at level 2, 1 and 0. These times are much bigger than those for the large ROI. However, again, using the latest GHz processors, the octree times can be reduced to less than 4, 19 and 35s, and the time for the extra interpolations can be reduced to less than 33s. The optimized time needed for a complete processing at the highest tree level, 37s, enables application of the framework for routine inspection and interpretation work, in near realtime. Figure 5 shows the seafloor and some semi-buried pipeline segments as well as a zoom-in of one segment, seen from different viewpoints. It is even possible to "look through" the pipe. Although a correct reconstruction of the seabottom is a very difficult task, due to the sparseness of and the noise in the data, these volumetric reconstructions allow for a detailed exploration and analysis of the seabed.

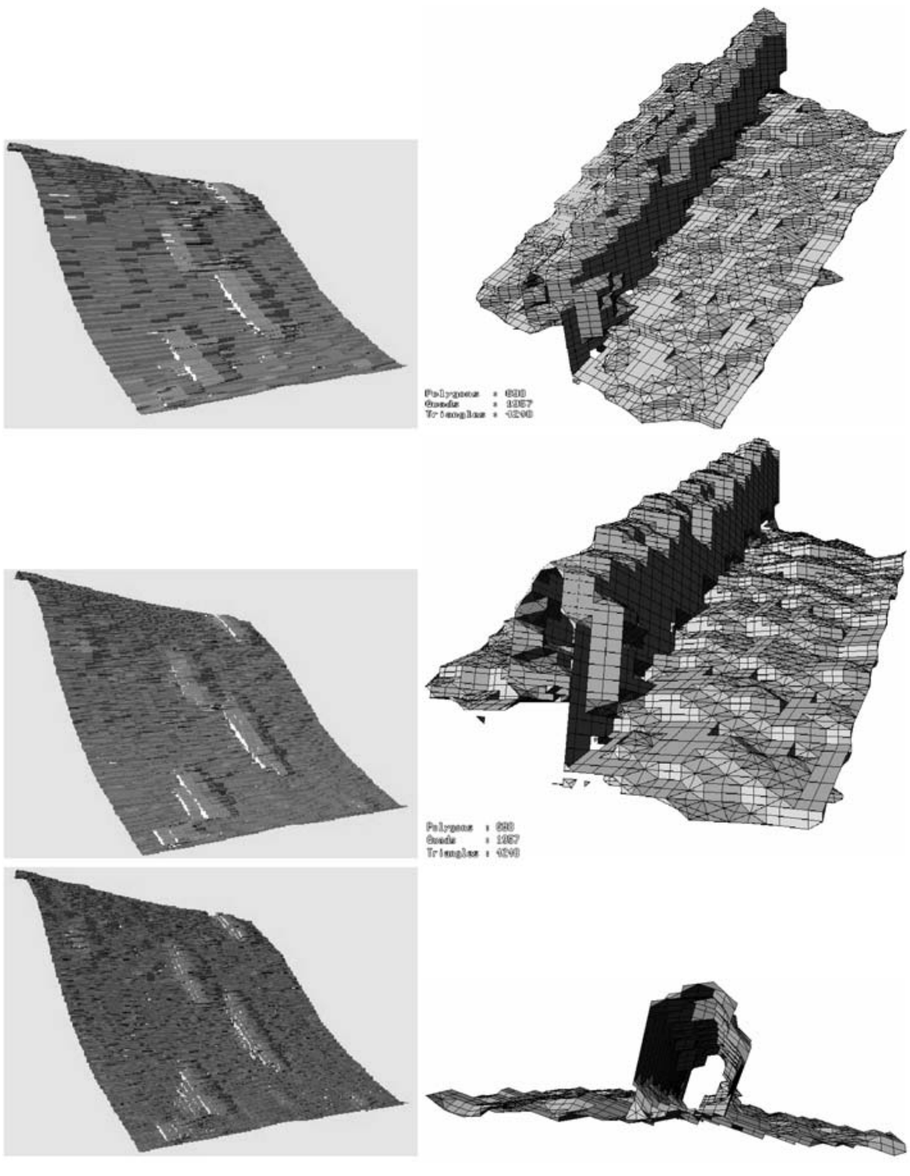


Fig. 5. Optimized and shaded seabottom surfaces in the small ROI at octree level 2, 1 and 0, and a sewage pipe seen from three viewpoints at octree level 1. These surfaces can be extracted from an incomplete volume of very noisy sonar data, sized 384×64×700, in near realtime.

5 Conclusions

We use multiresolution octrees for interactively visualizing large data volumes in (near) realtime. Application to volumes reconstructed from a very large sonar dataset showed that octree visualizations facilitate a fast seabottom analysis and/or a fast searching for objects in the *sub*bottom, even for volumes reconstructed from very noisy data and for volumes with a large number of unknown voxel values. In the future we will look for further applications, aiming at further finetuning and optimization of the embedded techniques, in order to enable a fast processing of huge datasets, thereby focussing on a fast user interactivity.

Acknowledgements

The data were obtained at *Åsgårdstrand,* a site in the North Sea close to the city of Horten, in the Oslofjord, in the European MAST-III ISACS Project (`http://w3.ualg.pt/isacs/`), contract MAS3-CT95-0046. The visualizations in Figs. 4 and 5 have been partly obtained with VRMLview, from Systems in Motion, Oslo, Norway. This work was partially supported by the FCT Programa Operacional Sociedade de Informação (POSI) in the frame of QCA III. Loke is currently involved in a Portuguese project on 3D modeling from video, contract POSI/SRI/34121/1999.

References

1. T. T. Elvins, "A survey of algorithms for volume visualization," *Computer Graphics*, vol. 26, no. 3, pp. 194–201, 1992.
2. A. Kaufman, *Volume Visualization.* Los Alamitos (CA), USA: IEEE Computer Society Press Tutorial, 1991.
3. R. E. Loke and J. M. H. du Buf, "Sonar object visualization in an octree," in *Proc. OCEANS 2000 MTS/IEEE Conf.*, Providence (RI), USA, 2000, pp. 2067–2073.
4. J. Wilhelms and A. Van Gelder, "Multi-dimensional trees for controlled volume rendering and compression," in *1994 ACM Symposium on Volume Visualization*, A. Press, Ed., Tysons Corner (VA), USA, 1994, pp. 27–34.
5. D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, 1982.
6. H. Samet, *Applications of Spatial Data Structures: Computer Graphics, Image Processing, and GIS.* Reading (MA), USA: Addison-Wesley, 1990.
7. R. E. Loke and J. M. H. du Buf, "3D data segmentation by means of adaptive boundary refinement in an octree," *Pattern Recognition*, 2002, subm.
8. ——, "Quadtree-guided 3D interpolation of irregular sonar data sets," *IEEE J. Oceanic Eng.*, 2003, to appear.
9. J. O. Lachaud and A. Montanvert, "Continuous analogs of digital boundaries: A topological approach to iso-surfaces," *Graphical Models and Image Processing*, vol. 62, pp. 129–164, 2000.
10. M. Couprie and G. Bertrand, "Simplicity surfaces: a new definition of surfaces in Z^3," in *Proc. SPIE Vision Geometry VII*, vol. 3454, 1998, pp. 40–51.
11. R. E. Loke and J. M. H. du Buf, "Linking matched cubes: efficient triangulation of 18-connected 3D object boundaries," *The Visual Computer*, 2003, to appear.