

University of Groningen

Certified meshing of families of isosurfaces

Plantinga, Simon; Vegter, Gert

Published in:
IEEE International Conference on Shape Modeling and Applications 2007, Proceedings

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2007

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):
Plantinga, S., & Vegter, G. (2007). Certified meshing of families of isosurfaces. In *IEEE International Conference on Shape Modeling and Applications 2007, Proceedings* (pp. 261-268). IEEE (The Institute of Electrical and Electronics Engineers).

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

Certified meshing of families of isosurfaces

Simon Plantinga
University of Groningen
The Netherlands

Gert Vegter
University of Groningen
The Netherlands

Abstract

Level sets are isosurfaces of an implicit function $F: \mathbb{R}^3 \rightarrow \mathbb{R}$, that is the set of points satisfying $F(x, y, z) = \theta$. In this paper we introduce an algorithm to move at interactive speed through the different level sets. Furthermore the meshes of the level sets are isotopic to the isosurface itself, as long as the surface stays away from singularities. When the surface moves close to singularities, the algorithm indicates arbitrarily small boxes where the topology is not certified. In this case, the user can decide to decrease the size of the boxes by further refinement. For special classes of functions, such as algebraic surfaces, other methods could be used to determine the topology inside the singular boxes.

1 Introduction

Recently a number of algorithms for regular isotopic meshing of implicit surfaces has appeared [10, 11, 1, 2, 4]. Due to singularities for generic level sets, no such results exist for examining level sets of implicit functions. Topology of isosurfaces is used to examine the clustering of matter in the universe [12], by computing topological properties of level sets of density functions. The display of level sets is also a fundamental technique for visualization of scalar fields [9]. In medical MRI scans isosurfaces indicate the separation between bones and soft tissues. In this case the topology is known, and can be used to determine an appropriate isovalue.

In this paper we extend our algorithm for isotopic meshing of implicit surfaces to the meshing of level sets. As far as we know this is the first meshing algorithm for level sets giving topological guarantees for the resulting mesh. Furthermore, the meshes generated for different isovalues are disjoint. The algorithm has been implemented and is fast enough to examine level sets interactively. The static isotopic meshing algorithm was first presented in [10]. In this paper we will use an improved meshing scheme based on tetrahedral subdivision [11], resulting in less complex mesh

generation.

Other meshing algorithms with topological correctness have been suggested. The algorithm in [1] is based on Morse theory and uses information about the critical points of the function to determine the topology.

Morse theory is also used in [13] and [3], where the topology of a mesh is determined by starting an isovalue with trivial mesh topology, and updating the mesh as the isosurface passes through singularities.

It is not clear whether these algorithms can be extended to meshing of level sets. In particular, details about the singular points are required, and there is no fixed spatial subdivision that can be maintained and reused for the different level sets.

Sampling based algorithms for isotopic meshing of static implicit surfaces construct a sufficiently dense sample of points on the surface, such that surface reconstruction results in a homeomorphic approximation. To construct such a sample, in [2] bounds on the distance to the medial axis are needed, while [4] requires the critical points of height-functions on the surface and on intersection curves. Even if these requirements are met, the point sample cannot be reused for different isovalues.

In Section 2 we summarize the isotopic meshing algorithm for static surfaces. In Section 3 we show how the same data structure can be used for meshing level sets with topological guarantees. Results of this algorithm are shown in Section 4. Finally, in Section 5 we discuss possible improvements to the current implementation.

2 Isotopic meshing

In this section we give an overview of the isotopic meshing algorithm for an implicit surface $F(x, y, z) = 0$. For full details see [11].

Enumeration methods for approximation subdivide space into smaller regions where the surface is ‘simple enough’ to create a local mesh for this region. The union of these local meshes forms the final approximation. A famous example is Marching Cubes [8]. These enumeration algorithms usually depend on a user-defined resolution to

trade off accuracy and speed. Marching Cubes uses a fixed size for all cells. For implicit curves quadrees have been used, where the level of subdivision depends on the local curvature [7]. However, the user has to predetermine what amount of curvature is allowed for a good approximation.

The isotopic meshing algorithm [11] uses an octree based subdivision to allow for local refinement where required, in particular in regions with high curvature. Interval arithmetic is used to test which leaves need further subdivision to guarantee an isotopic approximation. The main idea is firstly to test which regions do not contain part of the surface. These regions do not need further subdivision. Secondly, the variation in the gradient is tested. A test based on interval arithmetic determines if this variation is small enough to determine the local topology of the implicit surface.

2.1 Interval arithmetic

One way to prevent rounding errors due to finite precision numbers is the use of interval arithmetic. These intervals can be considered as a value, together with an error bound.

An *inclusion function* $\square f$ for a function $f: \mathbb{R}^m \rightarrow \mathbb{R}^n$ computes for each m -dimensional interval I (i.e., an m -box) an n -dimensional interval $\square f(I)$ such that

$$x \in I \Rightarrow f(x) \in \square f(I)$$

An inclusion function is said to be *convergent* if the width of the output interval converges to 0 when the (largest) width of the input interval shrinks to 0. For example, if $f: \mathbb{R} \rightarrow \mathbb{R}$ is the squaring function $f(x) = x^2$, then a convergent inclusion function $\square f([a, b])$ is given by

$$\square f([a, b]) = \begin{cases} [\min(a^2, b^2), \max(a^2, b^2)], & a \cdot b < 0 \\ [0, \max(a^2, b^2)], & a \cdot b \geq 0 \end{cases}$$

When using interval arithmetic to prevent rounding errors, the width of the intervals should be as small as possible in order to give accurate results. We will use another application of interval arithmetic, and compute function values over large intervals. For example, if the function value computed over a large box results in a strictly positive interval, we can conclude that the function has no zeroes within that interval. In other words, the implicit manifold does not intersect this box.

Convergent inclusion functions exist for the basic operators and functions. To compute an inclusion function it is often sufficient to replace the standard number type (e.g., `double`) by an interval type (`Interval`), using an appropriate interval library, such as `Filib++` [6].

2.2 Meshing algorithm

Our isotopic meshing algorithm is based on an octree subdivision. Leaves of the octree are subdivided until we have determined that either it does not contain part of the surface ($0 \notin \square F(C)$), or the variation in the gradient is small enough to determine the local topology. The latter condition is given by the interval inequality:

$$\langle \square \nabla F(C), \square \nabla F(C) \rangle > 0.$$

It implies that for all $x, y \in C$ we have $\langle \nabla F(x), \nabla F(y) \rangle > 0$, so the gradient variation is less than $\pi/2$.

To facilitate the meshing of the resulting octree, we balance it first, and then tetrahedrize it. Balancing means that we perform extra subdivision until neighbouring leaves differ at most one level in depth, and therefore at most a factor of two in size. Balancing does not increase the complexity of the octree size. For the tetrahedrization we triangulate the faces of the octree in a non-ambiguous way. A vertex is added at the centre of each leaf, and tetrahedra are constructed by connecting this centre vertex with the triangles on the boundary of the leaf.

Algorithm APPROXIMATE SURFACE(F, B)

Input. An implicit function F , and a cubic bounding box B .

Output. A piecewise linear approximation of the implicit surface $F = 0$.

1. Initialize an octree \mathcal{T} to the bounding box B of $F = 0$.
2. Subdivide \mathcal{T} until for all leaves C we have $0 \notin \square F(C) \vee \langle \square \nabla F(C), \square \nabla F(C) \rangle > 0$.
3. BALANCE OCTREE(\mathcal{T})
4. **for** each face of the octree \mathcal{T}
5. **do** triangulate the face according to the subdivision of its edges
6. **for** each leaf of the octree
7. **do** add a centre vertex and create tetrahedra by connecting it to the triangles on the boundary of the leaf
8. **for** each tetrahedron
9. **do** determine the function sign of its vertices and mesh the tetrahedron using linear interpolation

The sides of each face of the octree can consist of either one octree edge or of two octree edges in case of a subdivided side. We walk around an octree face in a fixed direction, for example counterclockwise as seen from the positive axis perpendicular to that face, and starting at the corner vertex with largest coordinates. By checking whether the four sides are subdivided, this results in one of 16 unique sign patterns. We then triangulate the face according to the table in Figure 1.

Once all faces of an octree cell are triangulated, we can tetrahedrize a cell by constructing a tetrahedron for each triangle on its boundary, by connecting it to the centre of the

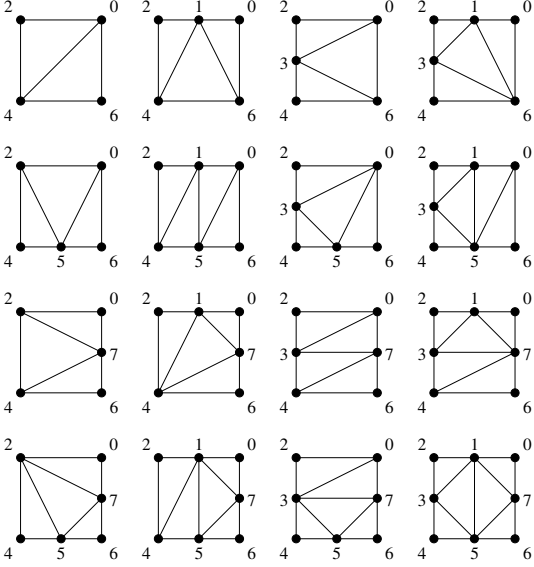


Figure 1. Triangulation of octree faces. The eight possible vertex positions are labeled 0 through 7 in counterclockwise direction around the positive axis.

cell. Since the triangulation of octree faces is determined uniquely, this results in a tetrahedrization of the entire octree (Figure 2).

To generate the mesh, we construct at most two triangles for each tetrahedron. If one of the four vertices has an opposite sign we construct one triangle, if two vertices have the opposite sign of the other two, we construct two triangles forming a 4-gon (Figure 3). The vertices of these triangles are placed using linear interpolation of the function values at the endpoints of the corresponding octree edges.

For the proof of isotopic correctness of an octree based mesh we refer to [11].

3 Level sets

Introduction In this section we extend the isotopic approximation algorithm to mesh level sets of an implicit function. Changing the isovalue means that the isosurface moves through the singularities of the implicit function. Our static algorithm only works for regular surfaces.

For the isotopic approximation algorithm, we required as little information about the implicit function as possible. We used a black box approach, where we can only compute function values and derivatives at points and over intervals. With this approach it is not possible to determine when and where exactly singularities occur. Note that the surface passing through a singularity of F only depends on the value of F at that point. Higher order derivatives do

not give any more information. This implies that we would need to know the exact location and function value of singularities points. With the black box approach this is not possible. However, we can construct arbitrarily small boxes around the singularities, by terminating the subdivision at a predetermined maximum level. The leaves at the maximum level for which the gradient variation bound does not hold could contain singularities of the implicit function. We call these *singular boxes*, although they do not necessarily have to contain a singularity of F . Outside these singular boxes the topology is already guaranteed. For the singular boxes and a particular isovalue θ , we can test whether $\theta \notin \square F(C)$. If this is the case, then also inside the singular box we have correct topology, since the level set does not intersect this box. If not, we do not know what happens inside this box. We could subdivide further until the isotopy class is determined, but since we do not know if the isosurface for the current value of θ is regular there is no guarantee that this process terminates. Instead, we choose to simply mark the box as *red*. For visualization purposes, this means that we can easily identify for which isovalues we have small areas with uncertain topology, and where in space those singularities occur. Depending on the requirements of the user, we could choose to do further octree refinement of red boxes, which might remove the topological uncertainty. For specific classes of surfaces, such as algebraic surfaces, we could also examine the function itself to study the possible singularities inside a red box.

Algorithm Instead of looking at the zero set $F = 0$ we now consider the level set $F = \theta$. For the subdivision process this means we have to examine the interval condition

$$\theta \notin \square F(C) \quad \vee \quad \langle \square \nabla F(C), \square \nabla F(C) \rangle > 0.$$

The lefthand clause is dependent on the specific level set θ . We therefore subdivide the initial octree until the righthand clause holds for each leaf, or until a predetermined maximum octree depth is reached. For a better Hausdorff accuracy we could also perform extra subdivision until all leaves have a given minimum depth. As in the static case we balance the tree for convenience. For the resulting octree we now precompute $\square F(C)$ for all leaves for which the gradient variation bound does not hold. This way we can easily check whether $\theta \notin \square F(C)$, and therefore determine quickly if the topology inside a cell is guaranteed.

If we also store the function values at the octree vertices, the resulting data structure contains all the information we need to create the meshes for different level sets. Also, if the isovalue changes only by a small amount we can locally update the structure of the mesh.

Starting with an isovalue outside the range of function values at octree vertices, the initial mesh is empty. If we change the isovalue θ , the combinatorics of the mesh only

changes in cells for which θ passes the function value of one of the vertices on the boundary of that cell. Otherwise we only have to move existing mesh vertices to maintain linear interpolation. For the cells where the combinatory does change we simply remesh these octree leaves. Meshing of a single cell is done by tetrahedrizing the cell (if the tetrahedrization is not precomputed and stored), and subsequently meshing the tetrahedra.

After (or during) updating the mesh, we examine all singular boxes. Recall that these are the octree leaves of maximal depth for which the gradient variation bound does not hold. If for such a leave $\theta \in \square F(C)$, we mark the leave as 'red'. In our implementation we display small red boxes for these leaves, indication that there is an uncertainty in topology. For nonempty leaves having vertices with function values both larger and smaller than θ , we could also construct a local triangle mesh, displaying these triangles in another colour to indicate the topological uncertainty.

In practice we can now move interactively through the level sets of an implicit function. For regular surfaces the mesh will be isotopic to the implicit surface, except for small red boxes appearing if the surface moves close to or through a singular box of the octree. If the maximum subdivision level is large enough, these singular boxes will only appear in a small neighbourhood of singularities of the implicit function.

After further user input we could do a temporary refinement of the octree inside these red boxes, or perform some kind of analysis of the implicit function. For almost all θ the corresponding isosurface is regular, and a few extra levels of subdivision should remove most of the red boxes.

4 Results

For the implementation we computed the tetrahedrization immediately after balancing of the octree. We stored the information about the implicit function (such as function range, gradient variation bound and vertex values) in the tetrahedron structure. By storing this information of the octree leaves in the tetrahedron structure, the octree itself is not needed anymore. Unfortunately, storing the entire tetrahedrization of the octree takes up a large amount of memory space, resulting in fairly low maximum subdivision levels.

We tested the implementation on several implicit functions. In this section we show some results for the Tangle Cube, the Chair surface and for a non-algebraic surface (Figure 4).

In Figure 5 two isosurfaces of the tangle cube are shown. This surface is defined by:

$$F(x, y, z) = x^4 - 5x^2 + y^4 - 5y^2 + z^4 - 5z^2$$

Figure 6 shows a series of close-ups as the isosurface passes a singularity. With a maximum subdivision level of

9, the red boxes are small, and only appear when the surface is fairly close to the singular boxes. They could easily be removed by further subdivision, but this would require intervention by the user.

Figure 7 shows the 'chair' function:

$$F(x, y, z) = (x^2 + y^2 + z^2 - ak^2)^2 - b((z-k)^2 - 2x^2)((z+k)^2 - 2y^2),$$

with $k = 5$, $a = 0.95$ and $b = 0.8$. Due to the higher number of computations, the interval arithmetic converges slower, and the bound on the variation of the gradient is only satisfied at a higher level of subdivision. The maximum subdivision level used for Figure 7 is 6, resulting in large areas of red boxes. Although they could be removed with extra subdivision, it seems that in this case a higher initial subdivision level would perform better. However, due to the large tetrahedron structure this requires an implementation that does not store the entire tetrahedrization, or stores it on external memory.

The non-algebraic surface is given by the function:

$$F(x, y, z) = -4(\sin(5x) + \sin(5y) + \cos(5z)) + x^2 + 3y^2 + 2z^2$$

This surface consists of many components and passes through a lot of singularities. Approximation the entire surface requires a large initial subdivision level. Since most of the detail is too small on a computer screen to examine singularities, it makes sense to zoom in on the surface. We approximated the part of the surface defined by the bounding box $[2, 2.5] \times [2, 2.5] \times [2, 2.5]$ with a maximum subdivision level of 8. Results are shown in Figure 8, giving a good indication of the stages where red boxes appear and disappear. Notice in particular how red boxes appear just before a new component appears and again before it merges with the main surface component.

The algorithm was tested on a Pentium 1.8 GHz computer running Linux. The timing results for the preprocessing are given in the following table. The time required for updating the mesh when the isovalue θ changes is dependent on the function and on the values of the previous and new θ . In our experiments the updating typically takes a few tenths of a second, fast enough for interactive examination of the level sets. The table shows the time for computing the octree and the tetrahedrization, and the size of the octree (number of leaves), the balanced octree and the number of tetrahedra. The numbers in parentheses indicate the maximum subdivision level used.

	tangle (9)	chair (6)	non-alg. (8)
Octree subdiv	0.43s	0.48s	0.47s
Tetrahedrization	3.04s	5.47s	6.98s
Octree size	45032	66592	34987
Balanced size	46544	66760	49253
Tetrahedra	696432	862800	792638

Our experiments show that a maximal subdivision level of 8 or 9 results in a small number of red boxes. Since

the tetrahedrization uses much more memory than the octree, an implementation that computes the tetrahedrization locally on demand should perform better. For small changes in the isovalue θ the number of octree leaves that require tetrahedrization is small. Comparing the time required for complete tetrahedrization with the size of the octree suggests that the time for updating the mesh would increase only slightly, since only a small fraction of the tetrahedra need to be reconstructed.

5 Conclusion and future work

Depending on the specific requirements there are many options in implementing the algorithm. For example, to move through an entire range of level sets, we could sort the vertices of the octree on function value. This list can be used to determine quickly where the next combinatorial changes in the mesh will occur.

From our results it is clear that we have to make a trade-off between speed and memory use. Storing only the octree requires much less memory space, allowing a greater maximum depth of the initial tree. However, updating the mesh will take slightly more time due to the tetrahedrization and recomputation of function values. Another option is to precompute the tetrahedrization, and store it in external memory.

It is not yet clear how to deal with the red boxes. The static algorithm for implicit functions terminates fairly quickly for regular surfaces. We expect that a few extra levels of subdivision would remove most of the red boxes. Again there is a trade-off between a lower initial subdivision combined with extra subdivision of red boxes, or starting with a higher initial subdivision level.

Another open problem is whether changing from interval arithmetic to affine arithmetic (AA) [5] will speed up the algorithm. AA computations are slower, but the results converge faster. Another option that we used in our implementation is to use standard interval arithmetic with extra levels of subdivision, to get more accurate inclusion intervals.

References

- [1] J. Boissonnat, D. Cohen-Steiner, and G. Vegter. Isotopic implicit surface meshing. In *Proceedings Thirty-Sixth Annual ACM Symposium on Theory of Computing*, pages 301–309, Chicago, 2004.
- [2] J. Boissonnat and S. Oudot. Provably good sampling and meshing of surfaces. *Graphical Models*, 67:405–451, September 2005.
- [3] A. Bottino, W. Nuij, and K. van Overveld. How to shrinkwrap through a critical point: an algorithm for the adaptive triangulation of iso-surfaces with arbitrary topology. In *Proc. Implicit Surfaces*, pages 55–72, 1996.
- [4] S. Cheng, T. Dey, E. Ramos, and T. Ray. Sampling and meshing a surface with guaranteed topology and geometry. In *Proceedings Symposium on Computational Geometry*, pages 280–289, 2004.
- [5] J. L. D. Comba and J. Stolfi. Affine arithmetic and its applications to computer graphics. In *Proc. VI Brazilian Symposium on Computer Graphics and Image Processing (SIB-GRAPI'93)*, pages 9–18, 1993.
- [6] M. Lerch, G. Tischler, J. W. von Gudenberg, W. Hofschuster, and W. Krämer. Filib++ interval library. <http://www.math.uni-wuppertal.de/wrswt/software/filib.html>.
- [7] H. Lopes, J. B. Oliveira, and L. H. de Figueiredo. Robust adaptive polygonal approximation of implicit curves. *Computers and Graphics*, 26(6):841–852, 2002.
- [8] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. In *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 163–169. ACM Press, 1987.
- [9] V. Pascucci and K. Cole-McLaughlin. Efficient computation of the topology of level sets. In *VIS '02: Proceedings of the conference on Visualization '02*, pages 187–194, Washington, DC, USA, 2002. IEEE Computer Society.
- [10] S. Plantinga and G. Vegter. Isotopic approximation of implicit curves and surfaces. In *Symposium on Geometry Processing*, pages 251–260, 2004.
- [11] S. Plantinga and G. Vegter. Isotopic meshing of implicit surfaces. *The Visual Computer*, 23(1):45–58, 2007.
- [12] J. V. Sheth, V. Sahni, S. F. Shandarin, and B. S. Sathyaprakash. Measuring the geometry and topology of large scale structure using surfgen: Methodology and preliminary results. *MON.NOT.ROY.ASTRON.SOC.*, 343, 2003.
- [13] B. T. Stander and J. C. Hart. Guaranteeing the topology of an implicit surface polygonization for interactive modeling. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 279–286. ACM Press/Addison-Wesley Publishing Co., 1997.

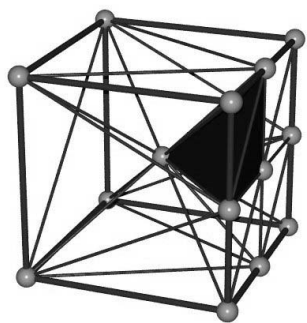
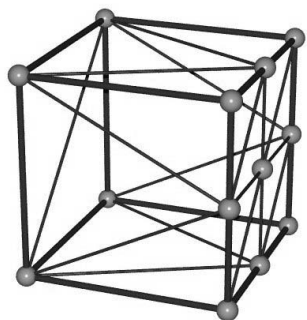


Figure 2. Tetrahedrization of a cell by triangulating its boundary. One of the 22 tetrahedra is shown in the bottom figure.

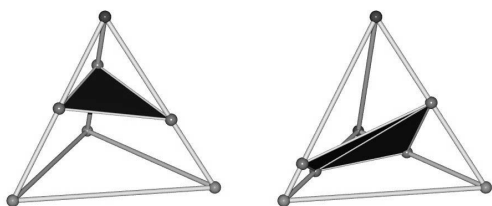


Figure 3. Meshing the tetrahedra.

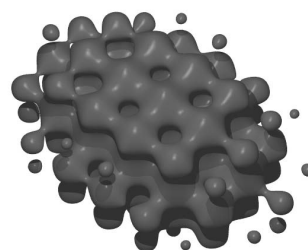
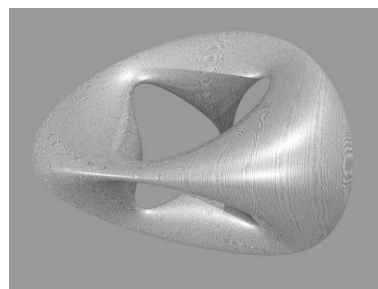
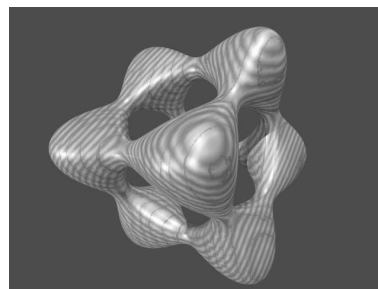


Figure 4. The tangle cube, the chair, and the non-algebraic surface.

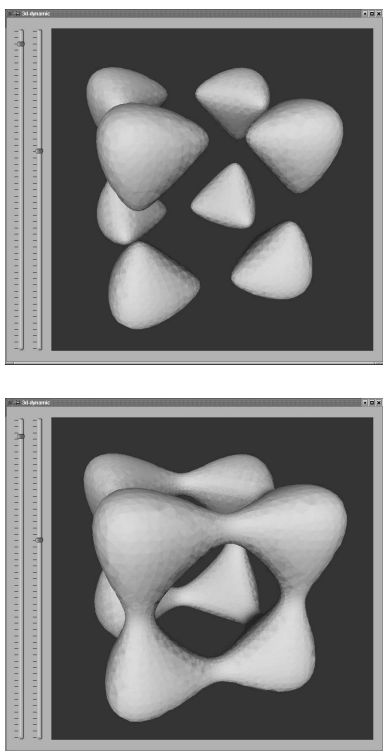


Figure 5. Level sets of the tangle cube.

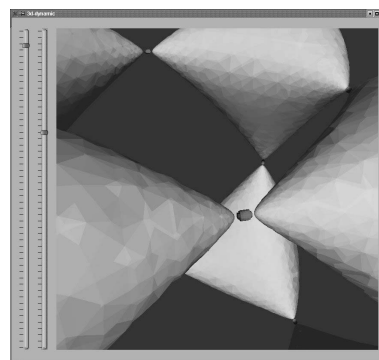
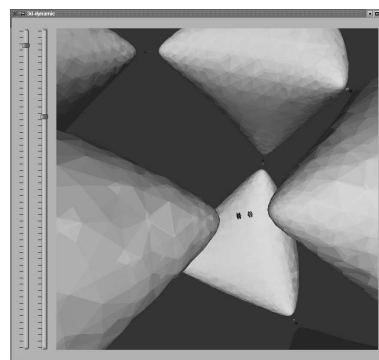
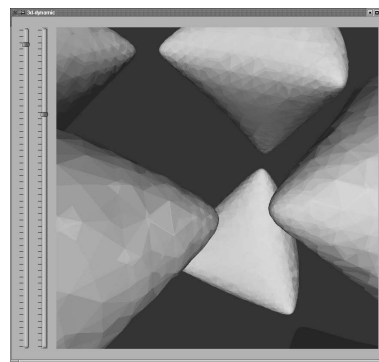


Figure 6. Close up of the 'red' boxes for level sets of the tangle cube. The maximum subdivision level is 9.

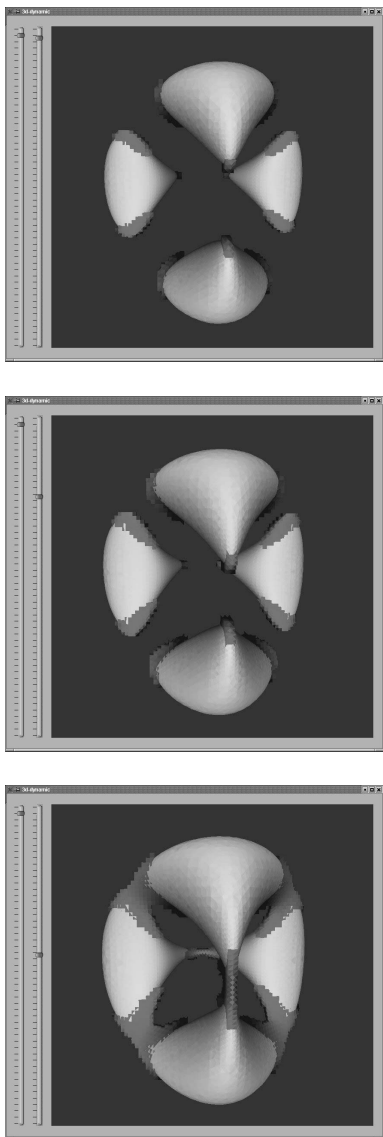


Figure 7. Level sets of the chair, with a maximum subdivision level of 6.

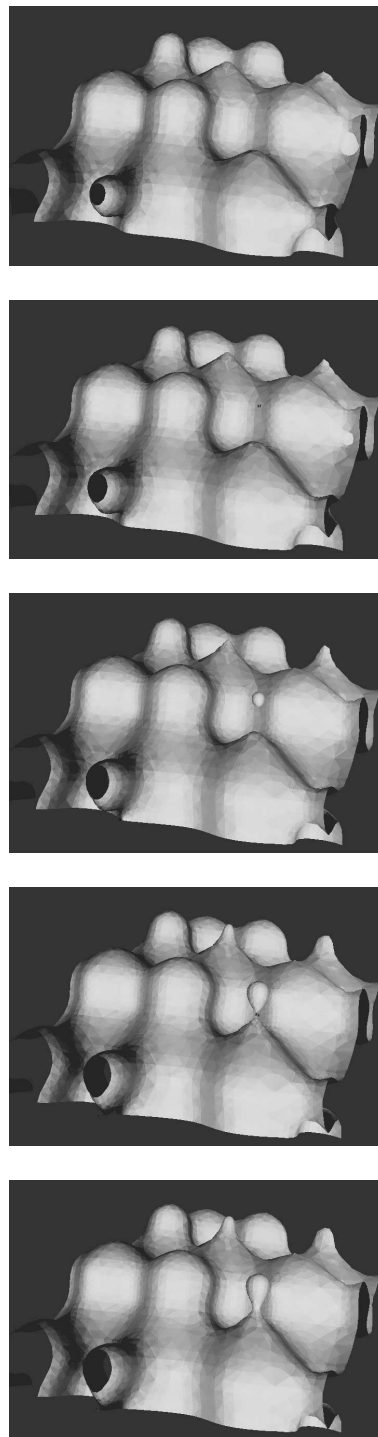


Figure 8. Level sets of part of the non-algebraic surface, with a maximum subdivision level of 8.