Mesh Generation From Layered Depth Images Using Isosurface Raycasting

Steffen Frey, Filip Sadlo, and Thomas Ertl

Visualization Research Center, University of Stuttgart

Abstract. This paper presents an approach for the fast generation of meshes from Layered Depth Images (LDI), a representation that is independent of the underlying data structure and widely used in image-based rendering. LDIs can be quickly generated from high-quality, yet computationally expensive isosurface raycasters that are available for a wide range of different types of data. We propose a fast technique to extract meshes from one or several LDIs which can then be rendered for fast, yet high-quality analysis with comparatively low hardware requirements. To further improve quality, we also investigate mesh geometry merging and adaptive refinement, both for triangle and quad meshes. Quality and performance are evaluated using simulation data and analytic functions.

1 Introduction

Raycasting is available for a wide range of higher-order volumetric data representations, including classical polynomials [1] or radial basis functions [2] from smoothed-particle hydrodynamics, which are not defined on grids. Cell-based fields featuring piecewise polynomial representation resulting from higher-order finite element or discontinuous Galerkin simulations can also be visualized directly using raycasting [3]. Rendering this data interactively on a desktop computer can be impracticable due to storage and computational costs. Layered Depth Images (LDI) [4] of isosurfaces can be used as a replacement for the actual data to drastically lower hardware requirements, e.g., for preview rendering. An LDI is a view-dependent representation that stores several depth values per pixel and can easily be generated with slight modifications of existing raycasting code. However, common LDI rendering methods (warping or splatting) suffer from quality or performance issues in a number of scenarios, and many analysis operations (e.g., distance measurement) are not applicable. Our integrated technique allows for quick generation of LDIs and subsequently for fast extraction of a mesh from one or more LDIs to serve as a high-quality stub for interactive rendering (Sec. 3). These meshes can further be enhanced by surface-based refinement (Sec. 4) with advantages over traditional volume-based techniques (e.g, using octrees), and we also present a technique for the geometric merging of meshes from several LDIs (Sec. 5). In contrast, extracting meshes directly from the original data using common mesh-based isosurface extraction tools (like Marching Cubes [5], dual contouring [6], or others [7]) depends on the structure of the data, and

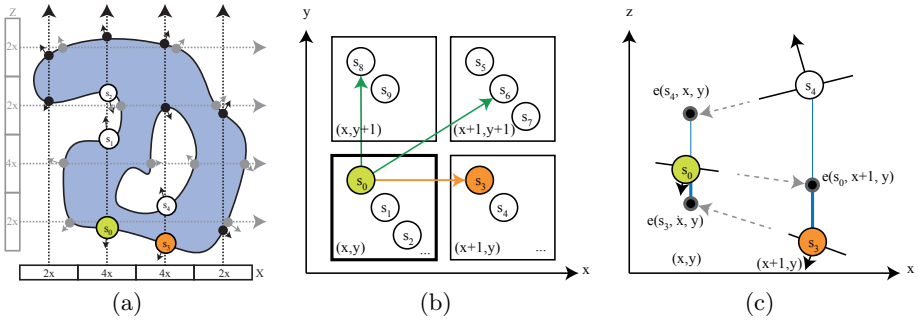
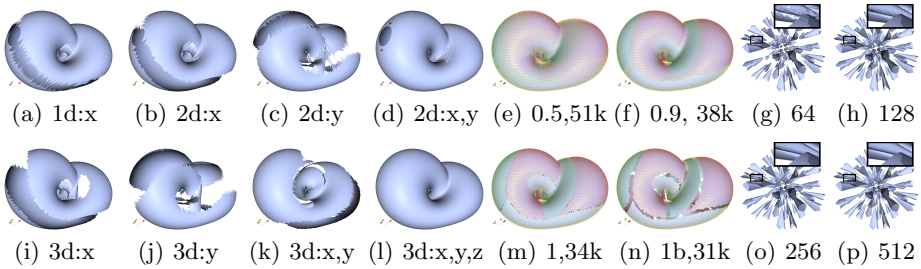


Fig. 1. (a) Generation of LDIs from different directions. (b) For each LDI independently, sample s_0 at (x, y) potentially forms a quad with samples from $(x + 1, y), (x, y + 1)$, and $(x + 1, y + 1)$. (c) Normal-based depth prediction for s_0 to find its best match (in this case s_3), normals by black arrows.

often cannot be applied directly. Further, the resampling of volumes either leads to inaccurate results or, as in the case of direct rendering, is very expensive in terms of computation time and memory. LDIs allow for the decoupling of mesh generation and the actual data representation. They provide a view-dependent region-of-interest selection that is represented in high quality. The compact representation of LDIs, their fast generation, as well as the quick extraction and rendering of meshes therefrom make our approach useful in many scenarios. For instance, they can be used for local or remote preview or in situ visualization.

2 Related Work

For isosurface extraction from higher-order data, quad mesh generation techniques [8], contouring [9], and approximate isocontouring [10] have been proposed. For uniform grids based on trilinear interpolation, classical Marching Cubes (MC) [5] and variants are most popular (e.g., dual MC [11] among others [12,13]). MC adaptations were further introduced for tetrahedral meshes [14, 15], also supporting adaptive reconstruction [16, 17]. However, in contrast to our technique, they refine the volume and not the surface directly, and preventing cracks requires additional effort. Other approaches use Voronoi diagrams [18], advancing front techniques [19], and meshing from point clouds [20]. For combining disjoint meshes, as done with our approach (Sec. 5), several approaches have been proposed, including sewing [21], volumetric methods [22], zipping overlapping meshes [23], laser range images from different views [24], and polygon triangulation [25]. Various techniques have been presented for rendering cell-based higher-order fields [26], including a raytracer for cut-surfaces [27] and point-based visualization [28]. LDIs represent one camera view with multiple pixels along each line of sight [4], their size growing linearly with the observed depth complexity of the scene. In volume rendering, layer-based representations (like LDIs) have been used to defer lighting or transfer function changes (e.g., [29–31]).



(a) 1d:x (b) 2d:x (c) 2d:y (d) 2d:x,y (e) 0.5,51k (f) 0.9, 38k (g) 64 (h) 128

(i) 3d:x (j) 3d:y (k) 3d:x,y (l) 3d:x,y,z (m) 1,34k (n) 1b,31k (o) 256 (p) 512

Fig. 2. (a)–(d),(i)–(l) Preview meshes of the KleinBottle data with a resolution of 128×128 per direction, showing the differences in trimming for the same $r = 0.5$ but a varying set of directions (subcaption depicts the total number of directions as well as the ones that are rendered). (e),(f),(m),(n) Varying r with meshes from x, y and z-direction colored in red, yellow and green respectively (subcaption depicts r (plus additional boundary trimming) and the number of triangles). (g),(h),(o),(p) Barth data set with $r = 0.5$ for x, y, and z-directions and an increasing number of samples per direction, resulting in higher quality with a higher number of triangles: 64^2 : 24k, 128^2 : 136k, 256^2 : 625k, 512^2 : 2646k.

3 Mesh Generation and Trimming

For LDI generation, we restrict ourselves to raycasting with parallel projection in the following for the sake of simplicity (perspective projection works accordingly with slight adjustments). Depth and gradient information are stored not only for the first, but for all hits occurring along a ray (Fig. 1(a)). This is the only required, typically straight-forward, modification to existing raycasting codes.

To form a quad, every sample (s_0, located at (x, y)) selects one sample (its best match) each from the right $(x + 1, y)$, top $(x, y + 1)$ and the top right $(x + 1, y + 1)$ image position (Fig. 1(b)). The best match is determined by depth predictions based on normals. In detail, s_0 estimates the depth value $e(s_0, x+a, y+b)$ at the image position of the neighbor set $(a, b \in \{0, 1\}$, $(x+1, y)$ in Fig. 1(c)). Likewise, all candidate samples s (s_3 and s_4 in Fig. 1(c)) estimate a depth value $e(s, x, y)$ at image position (x, y). The best match for s_0 is then the sample s from the respective neighbor set with the smallest sum of distances $|e(s, x, y) - d(s_0)| + |e(d(s_0), a, b) - d(s)|$ where $d(\cdot)$ returns a sample's depth.

While using a single mesh can deliver a good approximation for slightly varying camera positions, meshes from multiple directions substantially improve the result for larger surface variations (Fig. 2). To determine the number of meshes (views) to use, independent from the actual data, we employ a quality measure based on the largest possible angle between the normal of a surface patch and the mesh direction, resulting in the following: 1 view: 90°, 2: 90°, 3: 58°, 4: 55°, 6: 49°, 6: 41°, 7: 39°, and 8: 37°. While the computation cost increases linearly with further directions, the quality of surface coverage does not. Accordingly, although our approach works with an arbitrary number of directions, we restrict ourselves to three as a reasonable trade-off in the following.

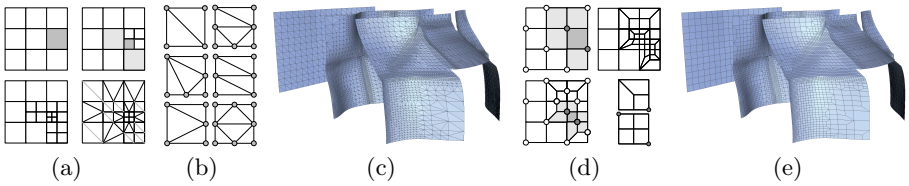


Fig. 3. Mesh refinement for triangles and quads demonstrated with the higher-order Shock Channel data [3] ($t = 3.0$, isovalue 4.6) showing the density of the flow around a box obstacle (c), (e). *Triangles:* (a) The subdivision of a cell (dark gray) forces further subdivisions (light gray) according to the 1-level difference rule. (b) Triangles are generated using six triangle templates (up to rotation and inversion). *Quads:* (d) Subdivision of a cell (dark gray) marks two respective edge nodes (dark circles) and thus cells (light gray). New edge nodes are introduced and previously marked edge nodes are removed. Another cell is chosen for subdivision (dark gray), respective edge nodes are marked and transition cells selected (light gray). Finally, quads are created using two templates.

Meshes from multiple directions cover some surface areas multiple times. Trimming them according to their quality of coverage—defined by means of the view direction and the surface normal (Sec. 3)—both avoids issues when the meshes are overlayed for rendering and creates distinguished boundaries for merging (Sec. 5). For every quadrilateral primitive $q \in Q_d$ from view direction $d \in D$, we compute the normal vector n of each of its four vertices $v \in q$ as the cross product with its two neighboring vertices: $n_v = (v - v_{prev}) \times (v - v_{next})$. Whether a vertex v is valid or not is determined by testing for $|n_v \cdot d_q| / \max(|n_v \cdot d|, \forall d \in D) > r$, with d_q being the view direction from which q was generated. The user-adjustable trimming parameter r specifies the extent of surface reduction with respect to its normal and view direction as well as all other view directions. The larger r, the more vertices are classified as invalid and the more primitives are eventually discarded (Fig. 2). The choice of r is application-dependent as discussed below. Finally, only quads remain that feature no invalid vertex. Overlaying meshes from different directions for rendering simply requires enabled depth testing. Optimally, r should thus be chosen such that there are neither low quality primitives occluding fine details, nor holes in the geometry.

4 Mesh Refinement

Templates are employed for refining the obtained meshes from Sec. 3 to better represent areas of high curvature without generating hanging vertices (i). This requires new samples for the LDI (ii). Refinement not only happens within meshes, but also at boundaries for growing the mesh toward silhouettes (iii).

(i) Refinement Templates. Templates differ for triangle and quad output. For triangles, a classical quadtree approach is used (Fig. 3) with the maximum subdivision level difference between neighboring cells being restricted to one

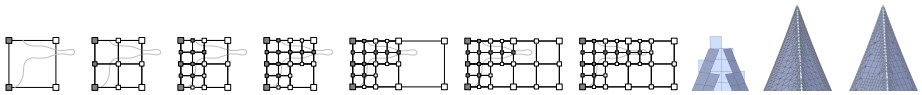


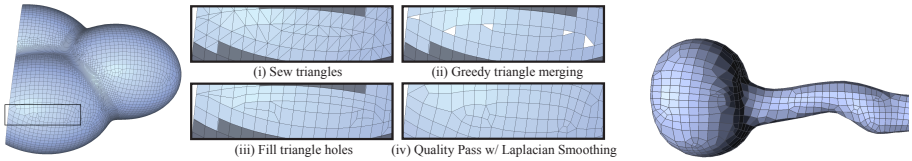
Fig. 4. Refinement at silhouettes. (a) Initial quad and true silhouette (gray curve). Cells containing true (gray) and invalid (white) samples are refined. (b) After two levels of refinement (c), the edge growing rule marks the right upper quad because it contains a valid sample on its left edge. (d) To resolve the hanging node at the right edge of the original quad, a new quad to its right is generated (e) and subdivided (f)–(g) to the respective level. Exemplary refinement of (h) toward edges (and corners) with triangles (i) and quads (j).

to achieve good quality triangles. After cells are marked for subdivision, an additional iterative *1-level difference* pass is employed that marks cells that have to be subdivided additionally to assure the constraint before generating triangles.

For quad mesh output the 1-level difference pass is substituted with the 2-refinement quad templates (Fig. 3(d)) [32] [33]. Two templates featuring "reference nodes" (dark circles in Fig. 3(d)) have to match so-called *edge nodes* inside the mesh. During subdivision, an edge node is created each time an edge is subdivided. For example, classical quadtree subdivision produces 5 new nodes, one in the center and one on each of the edges of the original cell. Ebeida et al. [33] propose to subdivide every cell of the initial mesh for producing an initial set of edge nodes. Instead, we identify edge nodes for the initial (unrefined) mesh from the pixel coordinates (x, y) of a vertex by testing if $x + y \bmod 2 = 0$, thus producing a "chessboard pattern" of initial edge nodes. First, edge nodes are identified that belong to cells marked for subdivision (*active edge nodes*, gray circles). Subsequently, all cells sharing such an active edge node are identified (*transition cells* marked by light gray quads) and templates are applied to all identified cells. Finally, previously active edge nodes are removed from the edge node set. A quad is marked for refinement, if any of its vertices is invalid, or the smallest dot product of a vertex normal with all neighboring vertex normals is below a certain value (0.99 proved successful in our experiments).

(ii) LDI Extension Sampling. Requests for additional LDI samples using the raycaster contain the direction and the new image position. All requests of one refinement pass are collected and processed at one go. Quad generation then works analogous to the initial mesh creation process (Sec. 3). If no new quad can be generated with the samples, an "invalid" vertex is used instead whose depth and normal are generated by interpolation from surrounding vertices. Invalid vertices are used for refining toward boundaries and silhouettes (iii). Primitives containing invalid vertices at the end of the refinement phase are removed.

(iii) Growing toward Boundaries and Silhouettes. The initial quads might not suffice as features can be missed that are smaller than the initial sampling distance. Thus boundaries and sharp tips of the isosurface might not be represented appropriately (Fig. 4). To refine toward these (Fig. 4(a)), a new top



(a) Triangle to quad conversion for sewing regions (light (i)–(iii)) and refinement of curved regions

(b) Quad mesh from CFD flow around sphere

Fig. 5. Merged and refined quadrilateral isosurface meshes from three LDIs.

level quad is added if one valid vertex exists on an open edge of an existing top level quad (quad before subdivision) (Figs. 4(b)–(d)). Such a vertex represents a hanging node that is resolved by quad subdivision (Figs. 4(f)–(g)). Adding a top level quad instead of a "small" (already refined) quad allows efficient recognition of quads growing from different boundaries to a sharing edge, thus preventing mesh overlaps. Furthermore, the refinement templates with their associated 1-level-difference criterion or marked vertices can be handled more consistently.

5 Combining Meshes

First, we trim overlapping parts (Sec. 3) using $r = 1$ (i.e., keeping only vertices whose normal best matches its original view direction) and then remove the "boundary" layer of cells featuring an edge with no neighbor (e.g., Fig. 2(n)). The resulting parts are combined by inserting so-called bridges (i) that connect close meshes. The remaining holes in the connected meshes are filled with triangles (ii) which can be quality-improved (iii), and finally converted to quads *(iv)*.

(i) Bridge Generation. Bridges are quads with one edge e_a being connected to mesh a, one edge e_b connected to another mesh b, and two connecting open edges e_{ab} and e_{ba}. Bounding boxes are used to determine the distances of all mesh patch pairs a and b in order to identify the bridge to be inserted with the smallest edge lengths $|e_{ab}|$ and $|e_{ba}|$. The bridge is kept and thus the meshes are merged if neither $|e_{ab}|$ nor $|e_{ba}|$ exceed $l = |e_a| + |e_b|$. In our experiments, we stopped the search early when $|e_{ab}| + |e_{ba}| < l/2$ for faster results.

(ii) Hole Filling. Bridges reduce mesh combination to a hole filling problem. We employ an ear-cutting algorithm due to its simplicity and low computational complexity: iteratively the shortest possible edge is introduced that forms a triangle with two existing open edges until there are no open edges left. However, due to the intricacy of 3D meshing, these approaches are "heuristics" that typically cannot be proven to provide the correct result [25]. Nevertheless, refinement and trimming typically provide good-natured hole problems that are easy to fill.

(iii) Triangle Mesh Enhancement. First, all quads that are directly adjacent to hole-filling triangles are split into triangles (resulting in the sewing region Fig. 5(a)(i) (light)). Next, edge flips based on edge length are performed. 3-to-1 triangle merges then resolve configurations of three adjacent triangles that are almost coplanar, detected by a vanishing determinant of the spanned tetrahedron.

Table 1. Timing results in seconds for different data sets and steps of our approach (if executed) on a single core of an Intel Core i7 with 3.4 GHz.

Fig.	Mesh.	Ref.	Bridge	Sew	Qual.	Tot.
Klein Bottle [34] (128^2)						
2	0.04	–	–	–	–	0.04
Barth [34] ($64^2, 128^2, 256^2, 512^2$)						
2(g)	0.02	–	–	–	–	0.02
2(h)	0.11	–	–	–	–	0.11
2(o)	0.43	–	–	–	–	0.43
2(p)	1.84	–	–	–	–	1.84
Marschner-Lobb [35] ($64^2, 512^2$)						
6(g)	0.08	–	0.11	0.04	0.23	0.46
6(h)	0.08	0.73	0.62	0.1	0.79	2.32
6(i)	7.19	–	0.82	3.38	22.07	33.46

Fig.	Mesh.	Ref.	Bridge	Sew	Qual.	Tot.
Coulomb [34] 64^2						
5(a)	< 0.01	< 0.01	0.01	0.01	1.84	1.86
Sphere [3] 64^2						
5(b)	< 0.01	–	< 0.01	0.01	0.40	0.40
Shock Channel [3] 64^2						
3	< 0.01	0.01	–	–	–	0.01
Slices $16^2, 24^2, 32^2$ $x^2 + y^2 + z^2 + \sin(5x + 15y + 6z) - 1$						
6(j)	< 0.01	–	< 0.01	< 0.01	< 0.01	< 0.01
6(k)	< 0.01	–	0.01	0.01	< 0.01	0.02
6(l)	0.02	–	0.02	0.01	0.04	0.09

***(iv)* Quad Mesh Generation.** The triangles introduced during hole filling and mesh enhancement (Fig. 5(a)(i)) can be converted to quads. The Catmull-Clark algorithm would generate a high primitive count by introducing three quads per triangle. Instead, we iteratively combine the pair of triangles that leads to the best quad according to a simple quality metric (ratio of minimum to maximum edge length) (Fig. 5(a)(ii)). Typically a small number of unmerged triangles remains. Then, we find the triangle pair with the shortest connecting path using Dijkstra's algorithm. This path is edge-connected and contains the triangle pair and the quads in between. Subsequently, we go from one triangle to the other, splitting traversed edges by introducing edge vertices (Fig. 5(a)(iii)). Passing a quad straight splits the quad in two, while passing adjacent edges splits the quad in three (using the quad refinement template from Fig. 3(d)). We repeat from identifying the shortest connecting path until all triangles are split into quads and finally merge quads sharing two edges, improve the vertex valence via quad edge flips toward four and apply Laplacian smoothing (Fig. 5(a)(iv)).

6 Results

For evaluation, we use the higher-order unstructured grid raycaster by Üffinger et al. [3] using CUDA, and the implicit surface raycaster by Knoll et al. [34] implemented in Cg. All data sets as well as timings for the presented results throughout the paper can be found in Tab. 1. Timings do not include raycasting times to generate the underlying LDI. Results were obtained using three viewing directions along the x, y, and z-axis unless otherwise noted. Fig. 2(e),(f), and (m) show that the larger r, the larger the mutually covered regions, reducing the risk of holes, but also potentially leading to invalid coverings. In our experience, $r = 0.5$ provides a good trade-off overall. The timings (Tab. 1) also suggest that r could be interactively adjusted to best fit the data set and the requirements of the user. Fig. 2(n) shows $r = 1$ with the additional boundary trimming prior to sewing. Preview meshes generated from different LDI resolutions (Fig. 2(g),(h),(o), and (p)) provide more details for higher resolutions with a better coverage of thin features and strong curvature with an approximately linear relation between

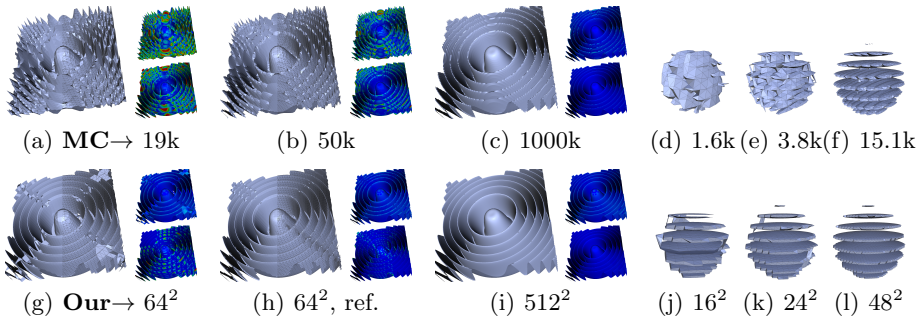


Fig. 6. Comparison of our approach (bottom row with LDI resolution, "ref." denotes refinement) to MC (top row with triangle count) using the Marschner-Lobb signal and the Slices data set. Vertical image pairs have approximately the same triangle count. For Marschner-Lobb, additionally forward and backward geometrical distances are depicted (top and bottom right) using a rainbow color map from 0 to 0.005 and 0.03, respectively (domain extent is 0.64 per direction).

primitive numbers and generation time. Even the generation of high-resolution meshes at interactive rates would be possible, considering the large improvement potential through parallelization. Only one LDI can already suffice depending on the nature of the isosurface (Fig. 3). Fig. 3 also demonstrates refinement to the surface structure for triangles and quads. Although the original mesh of a test function (Fig. 4) is very coarse (Fig. 4(h)), mesh growing still allows to nicely adapt to edges and corners, both for triangles and quads (Fig. 4(i) and (j)).

In Fig. 6, we demonstrate the accuracy of our approach using geometric distances in comparison to meshes from MC [36] with an approximately equal triangle count. We compare all vertices of the candidate mesh to a reference (MC with 5M triangles) (forward) and all vertices of the reference to the candidate (backward). In contrast to MC with a similar triangle count, our meshes deliver close to perfect results with forward comparison, and are also consistently better for backward comparison (particularly at signal peaks in comparison to the jagged coverage of MC). Much better results than MC (Fig. 6(a)) are also achieved for very low resolutions, despite some artifacts due to insufficient sampling (Fig. 6(g), incorrect bridges, and hole filling as both steps rely on sampling distance for finding correct matches). Our approach preserves the basic structure of data well, even if topologically incorrect bridges between the slices are introduced (Fig. 6(d)–(f), (j)–(l)). Refinement leads to smaller gaps between mesh patches that belong together topologically and thus reduces artifacts (Fig. 6(h)). However, although significantly decreasing, small holes in the peaks of the signal in boundary regions even persist with fine sampling (Fig. 6(i)) as connections across the peak are shorter than along the peak. This could be fixed by advanced bridging and hole filling heuristics considering normal variation in addition to distance. Overall, from low to high resolution, our approach was able to generate a more detailed approximation for approximately equal triangle counts.

7 Conclusion

We presented and evaluated a novel approach for view-dependent mesh extraction from LDIs independent of the underlying data structure. We demonstrated its usefulness in the context of existing raycasting-based techniques, providing fast generation of adaptively refined triangle and quad meshes, both for the purpose of preview rendering and further analysis. As a consequence, however, no topological guarantees can be made in contrast to commonly used isosurface extraction techniques (e.g., [5] [6]). For future work, we plan to work on a data-dependent LDI view selection and to conduct a more in-depth evaluation of our technique in comparison to other meshing techniques beyond classic MC, e.g., by using topology verification techniques [37].

References

1. Knoll, A.M., Wald, I., Hansen, C.D.: Coherent multiresolution isosurface ray tracing. Vis. Comput. **25** (2009) 209–225
2. Gamito, M.N., Maddock, S.C.: Ray casting implicit fractal surfaces with reduced affine arithmetic. Vis. Comput. **23** (2007) 155–165
3. Üffinger, M., Frey, S., Ertl, T.: Interactive high-quality visualization of higher-order finite elements. CGF **29** (2010) 337–346
4. Shade, J., Gortler, S., He, L.w., Szeliski, R.: Layered depth images. SIGGRAPH (1998) 231–242
5. Lorensen, W., Cline, H.: Marching cubes: A high resolution 3D surface construction algorithm. Computer Graphics **21** (1987) 163–169
6. Schaefer, S., Warren, J.: Dual marching cubes: primal contouring of dual grids. (2004) 70–76
7. Fryazinov, O., Pasko, A., Comninos, P.: Fast reliable interrogation of procedurally defined implicit surfaces using extended revised affine arithmetic. Comput. Graph. **34** (2010) 708–718
8. Remacle, J.F., Henrotte, F., Baudouin, T., Geuzaine, C., Bchet, E., Mouton, T., Marchandise, E.: A frontal delaunay quad mesh generator using the l norm. In: 20th Meshing Roundtable. (2012) 455–472
9. Wiley, D.F., Childs, H.R., Gregorski, B.F., Hamann, B., Joy, K.I.: Contouring curved quadratic elements. In: VisSym. (2003) –1–1
10. Pagot, C.A., Vollrath, J., Sadlo, F., Weiskopf, D., Ertl, T., Comba, J.: Interactive isocontouring of high-order surfaces. In: Scientific Visualization: Interactions, Features, Metaphors. (2011)
11. Nielson, G.M.: Dual marching cubes. VIS (2004) 489–496
12. Dietrich, C., Scheidegger, C., Schreiner, J., Comba, J., Nedel, L., Silva, C.: Edge transformations for improving mesh quality of marching cubes. TVCG **15** (2009) 150 –159
13. Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., Zorin, D.: Quad meshing. In: Eurographics, The Eurographics Association (2012) 159–182
14. Zhou, Y., Chen, B., Kaufman, A.: Multiresolution tetrahedral framework for visualizing regular volume data. (1997) 135 –142
15. Anderson, J., Bennett, J., Joy, K.: Marching diamonds for unstructured meshes. In: VIS 05. (2005) 423 – 429

16. Grosso, R., Ertl, T.: Progressive iso-surface extraction from hierarchical 3d meshes. CGF **17** (1998) 125–135
17. Westermann, R., Kobbelt, L., Ertl, T.: Real-time exploration of regular volume data by adaptive reconstruction of iso-surfaces. The Visual Computer **15** (1999) 100–111
18. Dey, T., Levine, J.: Delaunay meshing of isosurfaces. In: Shape Modeling and Applications, 2007. (2007) 241 –250
19. Schreiner, J., Scheidegger, C., Silva, C.: High-quality extraction of isosurfaces from regular and irregular grids. TVCG **12** (2006) 1205–1212
20. Scheidegger, C.E., Fleishman, S., Silva, C.T.: Triangulating point set surfaces with bounded error. In: EG symposium on Geometry processing. (2005)
21. Kobbelt, L.P., Botsch, M.: An interactive approach to point cloud triangulation. In: Proc. Eurographics. (2000)
22. Curless, B., Levoy, M.: A volumetric method for building complex models from range images. SIGGRAPH (1996) 303–312
23. Turk, G., Levoy, M.: Zippered polygon meshes from range images. SIGGRAPH (1994) 311–318
24. Rocchini, C., Cignoni, P., Ganovelli, F., Montani, C., Pingi, P., Scopigno, R.: The marching intersections algorithm for merging range images. The Visual Computer **20** (2004) 149–164
25. Held, M.: Fist: Fast industrial-strength triangulation of polygons. Technical report, Algorithmica (2000)
26. Sadlo, F., Üffinger, M., Pagot, C., Osmari, D., Comba, J., Ertl, T., Munz, C.D., Weiskopf, D.: Visualization of cell-based higher-order fields. Computing in Science and Engineering **13** (2011) 84–91
27. Nelson, B., Kirby, R.M., Haimes, R.: GPU-Based Interactive Cut-Surface Extraction From High-Order Finite Element Fields. TVCG **17** (2011) 1803 –1811
28. Rosenthal, P., Linsen, L.: Direct isosurface extraction from scattered volume data. In: EuroVis. (2006) 99–106
29. Ropinski, T., Prassni, J., Steinicke, F., Hinrichs, K.: Stroke-based transfer function design. In: SPBG, Eurographics Association (2008) 41–48
30. LaMar, E., Pascucci, V.: A multi-layered image cache for scientific visualization. In: PVG. (2003) 9–
31. Tikhonova, A., Correa, C., Ma, K.L.: Explorable images for visualizing volume data. In: PacificVis. (2010) 177–184
32. Schneiders, R.: Refining quadrilateral and hexahedral element meshes. In: 5th International Conference on Grid Generation in Computational Field Simulations, CRC Press (1996) 679–688
33. Ebeida, M.S., Patney, A., Owens, J.D., Mestreau, E.: Isotropic conforming refinement of quadrilateral and hexahedral meshes using two-refinement templates. International Journal for Numerical Methods in Engineering (2011)
34. Knoll, A., Hijazi, Y., Kensler, A., Schott, M., Hansen, C.D., Hagen, H.: Fast ray tracing of arbitrary implicit surfaces with interval and affine arithmetic. CGF **28** (2009) 26–40
35. Marschner, S.R., Lobb, R.J.: An evaluation of reconstruction filters for volume rendering. In: Vis. VIS (1994) 100–107
36. Cignoni, P., Rocchini, C., Scopigno, R.: Metro: Measuring error on simplified surfaces. CGF **17** (1998) 167–174
37. Etiene, T., Nonato, L.G., Scheidegger, C., Tierny, J., Peters, T.J., Pascucci, V., Kirby, R.M., Silva, C.T.: Topology verification for isosurface extraction. TVCG **18** (2012) 952–965