A Survey of Implicit Surface Rendering Methods, and a Proposal for a Common Sampling Framework

Aaron Knoll
SCI Institute, University of Utah
IRTG, University of Kaiserslautern

Abstract: We consider several applications of implicit surfaces in visualization, and methods for rendering them. In particular we focus on geometry processing techniques for mesh extraction; and ray tracing methods for direct rendering of implicits. Given that both methods rely on sampling the implicit function in question, we design a software framework that could accomodate both algorithms. We conclude by evaluating the time complexity and performance of existing systems, and discuss the long-term potential of both methods for rendering and computational goals.

1 Introduction

An *implicit* is a multivariate function over an $\mathbf{R}^N$ domain. In 3-space, it is given by a function

$$f : \mathbf{R}^3 \rightarrow \mathbf{R}$$

such that, for an isovalue v, the implicit surface, or isosurface, is defined where $f(x,y,z) = v$, or

$$f(x,y,z) - v = 0$$

This formulation defines a level set, specifically a level surface.

In scientific computing and graphics, implicits generally serve to filters a scalar field of discrete data. As such, implicits are the *lingua franca* of data analysis, at least for data that does not lend itself to an immediate explicit visualization modality.

To render implicits in 3D, one is principally given three choices:

- *Extract explicit polygonal (triangle mesh) geometry, and rasterize that trivially on graphics hardware.*
- *Re-sample the implicit into some other proxy geometry, such as points or slices of a volume, and render that proxy geometry.*
- *Ray-trace the implicit directly.*

This paper will survey methods for rendering implicits. Noting that these rendering systems share common algorithmic components, we propose a common framework architecture for sampling implicits and comparing performance. Finally, we compare the implicit sampling performance of several existing systems, and make predictions concerning the long-term suitability of these various methods to various applications.

2 Applications of Implicits

Implicits are used across scientific disciplines such as mathematics, physics, biology and engineering. As computer scientists, our goal is to determine which visualizations of implicits are most meaningful and practical for their specific application. At the core of this issue lies the question of *what* the implicit itself is modeling.

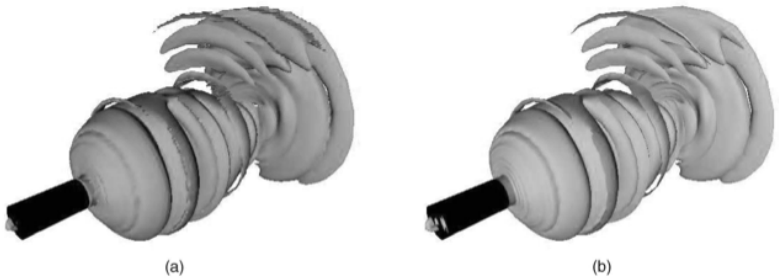

Figure 1: *Marching cubes (a), compared to ray-traced 13th-order Legendre polynomials approximating the computed implicits from the FE reference domain (b). From Nelson & Kirby [NK05].*

In some scenarios, the implicit function itself models natural or physical phenomena, and data is computed to approximate it discretely. This is the case for much volume data in simulation. Theoretically, it is desirable to visualize the same implicit employed in actual computation, as opposed to some arbitrary interpolant of the resulting data. As an example, for first-order finite elements, piecewise-linear planar interpolants of a tet mesh accurately represent the computed implicit surface. For higher-order spectral finite elements, however, this is not the case, and it is desirable to visualize higher-order implicit surfaces over the computational domain, or approximations thereof [NK05] (Figure 3).

For structured data, correct visualization is less clear. Finite differences computations approximate the differential operator and not the solution itself, thus no single implicit is strictly "correct" in visualizing them. Instead, we choose a convolution filter, the simplest being a first-order Lagrangian trilinear interpolant. This yields piecewise-smooth isosurfaces; but as the choice of filter is arbitrary, rendering it correctly is a dubious proposition (Figure 2). Scanned medical data, such as MRI and reconstructed CT scans, pose a similar conundrum of which filter is "best". This has partially been addressed [ML94, MMMY97] for volume rendering.

In the case of raw point sets, an implicit is used to interpolate data points from a scanner, allowing us to render a smooth surface. Here, the choice of implicit is secondary compared to how well it reproduces features (particularly high-curvature regions, and singularities) on the surface, and how efficient it is to render [CBC$^+$01, OBA$^+$03, AA03].

Finally, the implicit function itself can be of interest, for example when rendering superquadrics or blobby objects [Bli82], or graphing arbitrary algebraic [KB89] and even non-algebraic [Mit90, PLLdF06] expressions.

3 Rendering Implicits

In general, four options exist for rendering implicits in 3-space: direct volume rendering; extraction and rasterization; ray tracing; and point-based rendering. Volume rendering and point-based methods are proxy methods. In practice these work quite

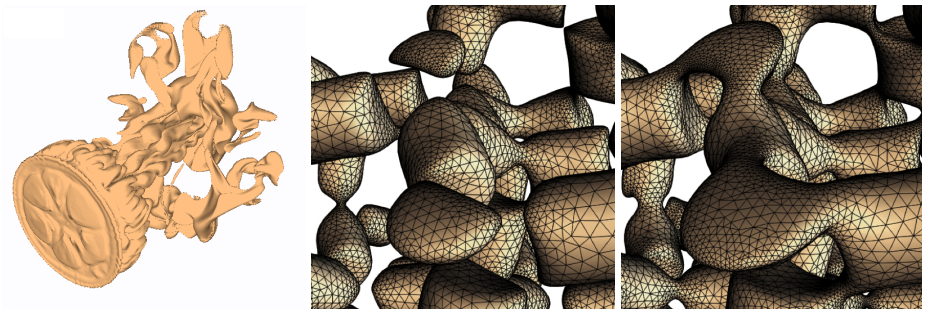

Figure 2: *Left to right: a finite-differences CD simulation ray-traced using a piecewise C^0 trilinear interpolant filter, and centrally differenced gradient normals [KHW07]; Advancing front extraction of structured data using piecewise-C^1 Catmull-Rom spline interpolants; and the same technique using piecewise-C^2 B-spline filtering [SS06]. Aesthetics aside, it is questionable which filter, if any, is "correct".*

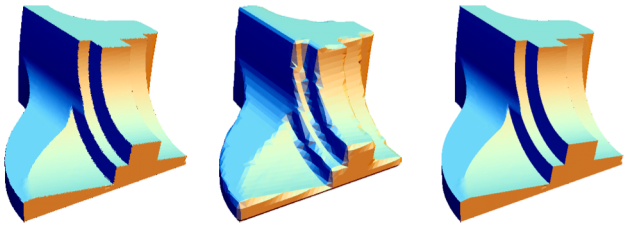

Figure 3: *Piecewise-smooth partition-of-unity implicit reconstruction of raw point data, reconstructed with high polygonization, coarse polygonization, and sphere traced (e.g. Hart [Har96]). From Ohtake et al. [OBA+03].*

well, due to efficient mapping on graphics hardware. However, they can vary widely in actual implementation, depending on desired fidelity and type of data. Partly for this reason we shall focus primarily on extraction and ray tracing, as they represent algorithmic extremes that are either designed to accurately construct a mesh, or render a view-dependent scene.

3.1 Volume Rendering

Direct volume rendering, samples the scalar field (usually regularly) and convolves these samples via a transfer function. When a transfer function is sufficiently sharp, or singular, at a desired value, volume rendering can approximate a surface. However, volume rendering is computationally demanding, and only interactive when implemented efficiently on graphics hardware. For rendering discrete isosurfaces, a near-singular transfer function is imperfect, particularly for large, entropic, data. In practice however, volume rendering of isosurfaces is useful, and worth mentioning as a feasible alternative to methods that render surfaces directly. An efficient implementation combining GPU volume rendering with isosurface rendering is given in [HSS+05].

3.2 Mesh Extraction

For rendering implicit surfaces, the most common method is to extract a mesh from the given data, and rasterize that mesh trivially on Z-buffer graphics hardware. The best-known method for surface extraction is Marching Cubes (MC), developed independently by Cleary & Wyvill [WMW86] and Lorensen & Cline [LC87]. The general idea in extraction is to subdivide the scalar field into convex "cell" primitives, and solve where the isosurface intersects cell boundaries. In conventional marching cubes, cells are voxels of the volume, and the isosurface is approximated by solving where linear interpolation yields the desired isovalue along a voxel edge. By considering the values at vertices of the cell, the algorithm knows which edges will contain a surface; moreover for a trilinear interpolant implicit only 15 different configurations exist for piecewise-linear iso-polygons within a given voxel. As voxels are C^0 at their edges, the resulting mesh is guaranteed to be continuous. More generally, this method allows a mesh to approximate any implicit, as solving for arbitrary $f(x,y,z)$ reduces to a 1D problem when any two coordinate axes are constant.

Mesh Extraction Variants Marching cubes is popular due to its simplicity and ease of implementation. However, it possesses several limitations. Some MC configurations cause ambiguities in orientation that must be resolved to preserve continuity, but this has been addressed [NH91]. Fundamentally, marching cubes is limited in that it samples regularly over a uniform grid. While this is adequate for small structured volumes, it is ill-suited for irregular point sets and unstructured data. Fortunately, a similar algorithm, Marching Tetrahedra (MT), was developed by Doi & Koide [DK91] to operate natively on unstructured tet meshes, though it can equally be applied to structured data due to the dual relationship between tetrahedra and voxels. Though MT addresses some sampling issues of marching cubes with unstructured data, both algorithms generate poor-quality, irregular meshes, and fail to capture topological features such as singularities. Dual contouring [JLSW02] and dual marching cubes [Nie04, SW04] alleviate these problems, reducing overall triangle counts, and effectively capturing singularities. However, "thin" features below a geometric subdivision criteria can still be lost using these techniques, as evident from Paiva et al. [PLLdF06], and meshes generating using dual marching cubes still suffer from irregularity (Figure 4). For generating high-quality triangle meshes, Scheidegger et al. [SFS05] proposed an advancing front meshing algorithm using a curvature-based guidance field. Schreiner et al. [SS06] extended this method to structured and unstructured scalar volume fields, and modified the guidance field to minimize both geometric error and number of required samples.

Dynamic Extraction and Visualization For larger data, it becomes necessary to manage overall scene complexity, measured in the number of cells or tetrahedra visited by the extraction algorithm. As long as this is controlled, it is generally trivial to render the resulting mesh interactively on graphics hardware. To reduce the visited set of cells, Wilhelms & Van Gelder [WV92] proposed a min/max hierarchy embedded in a branch-on-need octree for trivially ignoring empty regions of a volume. Livnat et al. refined the interval tree concept in creating a span space tree [LSJ96]. Livnat & Hansen [LH98] extended the min-max octree technique with view-dependent frustum culling using a shearing transformation visibility test. Westermann et al. [WKE99] implemented adaptive marching cubes on multiresolution octree data. Pesco et al. [PLPS04] proposed an occlusion test for the implicit itself, further reducing the cost of isosurface extraction at render-time. With few exceptions these techniques enable interactive rendering of data up 512^3. For larger data, interactive visualization is difficult, and extraction techniques often rely on topologically-guided simplification [LMM06] into Morse-Smale complexes. Though this does not render

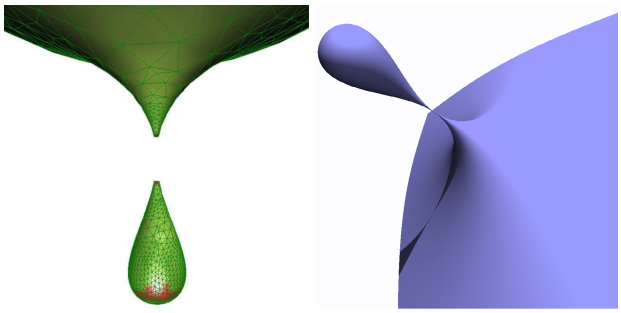

Figure 4: *Left: a robust meshing algorithm employing interval arithmetic and dual marching cubes fails to determine the correct connectivity of a surface, likely due to insufficiently fine geometric sampling [PLLdF06]. Right: ray tracing using a similar interval arithmetic technique can also experience difficulty sampling thin features, but samples in a view-dependent fashion and thus better recovers features for closer views [KHH+06].*

the entire original data set, topological simplification and segmentation is arguably more useful for data analysis.

3.3 Ray Tracing

While extraction methods focus on sampling the full scalar field, ray tracing samples the implicit directly at the intersection point of a viewing ray and the implicit surface. For rendering large data, this potentially entails far fewer samples than an adequate extraction method, and "better" samples in the sense that we sample exactly (and only) the regions required to fill a 2D uniform grid in screen space. Thus, ray tracing performs both view frustum and object-occlusion culling automatically. With an efficient acceleration structure, it performs spatial subdivision as well and renders objects in $O(logN)$ time as opposed to $O(N)$. For large volume data, this is extremely important. In structured volumes, the number of voxels multiplies eight-fold every time dimensions double; large data can easily overwhelm an extraction/rasterization method, even one with comprehensive subdivision and occlusion culling algorithms. The main drawback of ray tracing is that, particularly when not optimized, it can be slow. However, the inherent scalability of ray tracing, combined with increasing power of multicore CPU's and flexibility of GPU's, suggests the algorithm will increase in feasibility and popularity.

General implicit ray tracing In a sense, all primitives are formulated implicitly implicit in ray tracer, including the plane equation of a triangle. In general, ray tracing solves the implicit function for a parameter t along each ray. This is accomplished by substituting $P_i = O_i + tD_i$ for $i = \{X,Y,Z\}$. Then, $f(x,y,z)$ becomes a one-dimensional function $f_t(t)$, and finally $f_t(t) = v$ may be solved for t. Thus, ray tracing implicits is a root-finding problem; it solves for the first root along a given interval. Simple implicits of second degree and lower can be solved analytically. Blinn's blobbies [Bli82] were the first application of ray tracing higher order or rational implicit surfaces. Kalra & Barr [KB89] devised a general method for ray tracing algebraic surfaces with known Lipschitz-condition bounds for the gradient and curvature. The key issue with ray tracing arbitrary implicits is that globally convergent numerical methods fail to handle discontinuous or non-monotonic functions. One solution is to evaluate intersection by finding and minimizing a signed distance function for the implicit surface, as proposed

by Hart [Har96]. Yet more general is the use of an inclusion algebra such as interval arithmetic [Mit90] or affine arithmetic [dCJdFG99]. General methods for implicit ray tracing have historically proven slow, but recent work [KHH+06] using coherent optimizations suggests that pure interval bisection is surprisingly practical and interactive. However, this method has not been thoroughly tested on fitting and filtering point and scalar data, and it should be assumed to be slower than optimized special-case intersection for most third-degree and lower implicits. Higher-order and rational implicits, however, can be rendered efficiently using interval ray tracing methods.

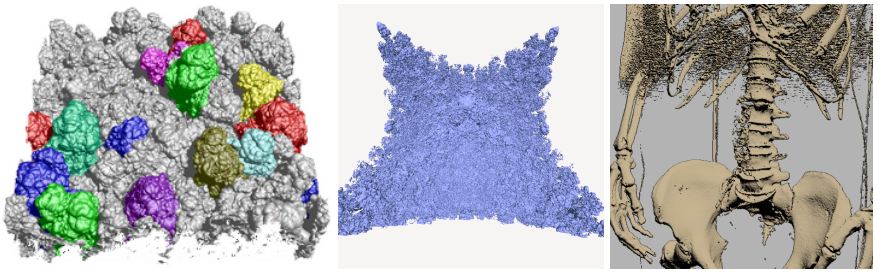

Figure 5: *Left: Morse-Smale topological segementation and simplification of large data [LMM06]. Center:* 2048*x*2048*x*1920 *Richtmyer-Meshkov instability field, losslessly compressed from 7.5 GB to 1.8 GB and rendered at multiple fps on a multicore laptop. [KWPH06]. Right: hybrid extraction and point-based rendering system, rendering the* 512*x*512*x*1024 *visible female dataset interactively [LT04].*

Structured data isosurface ray tracing Parker et al. [PSL+98] were the first to ray trace isosurfaces from structured volumes, employing these piecewise implicit interpolating patches. Though a small supercomputer was necessary to render them at the time, the method proved powerful in the size of volume data it could render at full resolution. The scalability of this method to large data was further demonstrated by DeMarle et al. [DPH+03] on clusters, rendering the 2048*x*2048*x*1920 Richtmyer Meshkov CFD data set in its entirety using a software distributed shared memory layer. More recently, Knoll et al. [KWPH06] implemented a lossless octree compression scheme for structured volumes, allowing both volume data and acceleration structure to be compactly represented in less than 25% the original volume footprint. In conjunction with an efficient ray-octree traversal algorithm, this enables rendering the Richtmyer Meshkov data on a laptop with 2 GB RAM at multiple frames per second, and interactively on a 16-core workstation.

Unstructured and point-set ray tracing Ray tracing is equally suited to visualizing implicit point-set surfaces using radial basis functions, as demonstrated by Wald et al. [WS05]. Large unstructured data has also recently been investigated [WFKH07]. With coherent BVH traversal and an optimized ray-isopolygon intersection similar to marching tetrahedra, CPU ray tracing is surprisingly competitive with GPU shader-based extraction techniques [WKME03, BCCS07], even on modest multicore desktop hardware.

3.4 Point-Based Rendering

Though CPU ray tracing tackles large data easily, its overall performance on moderate-size data is still underwhelming compared to GPU rendering methods. It is worth

mentioning the contributions of point-based rendering methods on the GPU, conceived by Levoy & Whitted [LW05] and proven for large point-set data by Rusinkiewicz et al.[RL00] with QSplat. Point-based rendering is spiritually similar to ray tracing in that acceleration structure traversal and primitive intersection are computed out-of-core on the CPU, but shading and actual rasterization occur on the GPU. Co et al. [CHJ03] extended the isosurface ray tracing technique of Parker et al. to point-based rendering on the GPU. Livnat & Tricoche [LT04] took this system one step further, implementing a hybrid isosurface render that combined extraction and point-based methods, and handled reasonably large data interactively. Zhou et al. [ZG06] develop a system for rendering higher-order finite element volumes very efficiently, though it requires presegmenting data into point sets.

4 A Unified Framework for Extraction and Ray Tracing

Rasterization and ray tracing ultimately both sample the input data in world space, though their means of sample generation and processing differ. It would not be difficult to share a common spatial subdivision and data sampling framework between an extraction and ray tracing application. At the lower algorithmic level, ray tracing and extraction methods can vary (between themselves, as well as each other) in actual implementation, largely in whether they iteratively evaluate or analytically solve roots on the implicit to generate the sample.

4.1 Framework Overview

The general idea of our framework is to abstract data acquisition, subdivision structure generation, and implicit expression generation, which are used by multiple common extraction and ray tracing implementations. In short, our pipeline is designed to handle two common-case ray tracing and extraction systems with varying degrees of generality; and an unlimited number of fixed-function techniques for ray tracing or rasterization that can use other components as necessary. We require three inputs from the user: data; an implicit to filter that data; and a sampler (either a ray tracing or extraction method). After processing these inputs, the system outputs either a mesh or a ray-traced image, as shown in Figure 6.

4.2 Pairing Ray Tracing and Extraction

In the previous sections 3.2 and 3.3, we have discussed common techniques for extracting and ray tracing implicits. We note that the bounded spectral radius for the minimal guidance field in advancing front extraction [SS06] is similar to the geometric bounds of the L-G surfaces in Kalra & Barr [KB89]. Both techniques assume the filter implicit to be twice differentiable (or at least have constant gradient or curvature) with known bounds on gradient and curvature (Jacobian and Hessian, respectively). Hence, they could logically be paired together as techniques for C^2 algebraic implicits.

For rendering of arbitrary non-algebraic implicit functions, the interval arithmetic method proposed by Paiva et al. [PLLdF06] is roughly analagous to ray tracing using interval bisection [Mit90, KHH+06]. Extraction methods using this technique would require a spatial subdivision structure such as an octree or BVH over which to evaluate intervals, followed by application of dual marching cubes or a similar algorithm over these regions.

The term "subdivision structure" in Figure 6 is somewhat vague; particularly in the

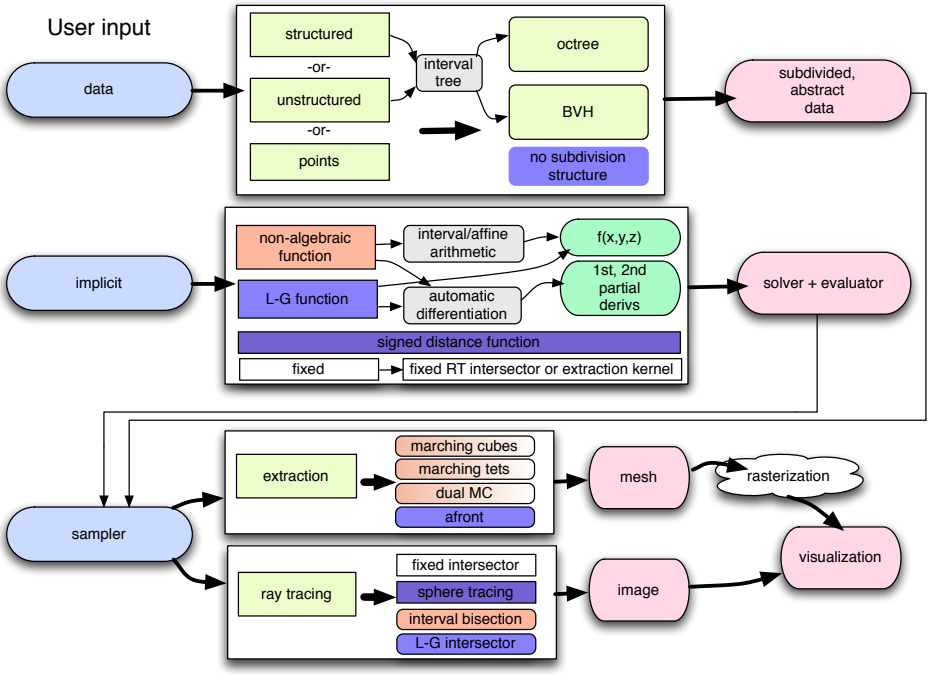

Figure 6: *Proposed framework for extraction and ray-tracing.*

context of rendering functions as in [KHH+06] and [PLLdF06]. Both techniques involve sampling within the implicit function domain, not merely the data domain. However, in Paiva et al. [PLLdF06] samples are constructed explicitly from dual marching cubes over this domain, whereas Knoll et al. [KHH+06] recommend no explicit spatial subdivision, and instead sample the implicit on-the-fly during intersection.

4.3 Fixed-Function Components

We recommend a fixed workflow for certain types of implicits, mostly for comparison purposes. For example, one could implement standard marching cubes using trilinear interpolation over voxels, or optimized ray-trilinear patch intersection using the Marmitt et al. [MFK+04] technique. Fixed functionality need not be absolute throughout the system; the Marmitt intersector could be used in conjunction with either a BVH or an octree, for example. Another practical option, used inherently by point-based implicits such as MPU [OBA+03] and sphere tracing [Har96] would be allowing the user to specify a signed distance equation as opposed to the implicit formulation itself.

4.4 Expected Quality and Performance

Using this framework, it would be fairly simple to benchmark the number of samples collected by ray tracing and mesh extraction techniques for similar views. These numbers would be somewhat unfair unless the rasterization system performed all the

appropriate culling optimizations [WV92, LSJ96, LH98, WKE99, PLPS04]. However, we can make some general assumptions about anticipated behavior. Assuming the implicit is an accurate representation of our data (and more specifically, what we seek to visualize), we expect the quality of the image or mesh to be dependent on the number of samples, and where these samples are spent.

With ray casting, samples are created from a camera by ray generation. For visualization purposes, it is hard to go wrong with this technique, as samples are spent exactly where the user wishes to see. Even when a sample misses the surface, it is still meaningful (and usually cheaper to compute). One could argue that this purely view-dependent sampling strategy is flawed when the projected object feature frequency exceeds the viewing ray frequency, causing aliasing. However, this could be solved either by supersampling rays or downsampling the object (via level-of-detail methods) over high object-frequency regions – both well-known techniques in computer graphics.

In extraction, mesh construction works best when sampling adapts to the curvature of the surface – specifically when regions of high curvature are allocated more samples. When the user only requires one mesh as output, it makes sense to generate the best mesh possible, and employ an adaptive and topologically sensitive method such as advancing front. For dynamic visualization purposes, however, the quality of the mesh is generally secondary to the extraction time; naive marching cubes (perhaps with some view-dependent adaptivity) is likely preferable in this case.

Empirically, existing literature suggests it is more difficult to extract a good mesh than to ray trace a good image. For example, Schreiner et al. [SS06] report the small 256^3 Aneurism data requiring 7 seconds to extract 134K triangles using conventional marching cubes, and over 5 minutes to extract 462K triangles using advancing front. On a single-core 2.4GHz Opteron CPU, using the Marmitt trilinear patch intersection method, Wald et al. [WFM+05] ray traced an isosurface of this data in over 6 fps at $640x480$. Assuming conservatively that only 1/20 of the samples hit the surface, and normalizing for number of processing cores on an Opteron 2.2 GHz, ray tracing processes 112K samples per second, whereas marching cubes generates 10K samples per second, and advancing front around 720 samples per second. The number of samples per second processed in point-based GPU methods are easily two orders of magnitude even than ray tracing, though point set data must be precomputed in advance [ZG06]. For higher-resolution data, ray tracing should perform similarly well due to its logarithmic acceleration structure traversal complexity. Advancing front should fare better compared to marching cubes, due to its adaptivity. While these comparisons are extremely rough and not completely fair, they are consistent with the findings of Livnat & Tricoche [LT04], in comparing their point-based isosurface rendering technique to full extraction. In their hybrid method, they find that view-dependent extraction works well for moderately near views, but for closeup or far away views point-based methods are more effective. The overall lesson is not that ray tracing is superior to extraction, but that view-dependent sampling is necessary for efficient, dynamic visualization.

5 Conclusion

Comparing ray tracing and extraction is an apples-and-oranges affair. The decision on which to perform should depend purely on the application. When the isosurface is ultimately used for modeling, for example with rigid-body mechanical simulations, mesh extraction is the most appropriate solution. For rendering animated blobby shapes in a computer game that can be deformed dynamically by in-game physics, extracting a mesh is all but necessary; for general-purpose rendering, rasterization is unlikely to be replaced anytime soon, if ever. When mesh generation is acceptable as an offline process, it is desirable to generate the cleanest mesh possible. For this

reason, and their topological and geometric adaptivity, methods such as dual marching cubes and advancing front are more useful than marching cubes, particularly for larger data. For the same reason of effective sample use, ray tracing and point-based methods are better suited to dynamic visualization than on-the-fly rasterization and extraction.

References

[AA03] Anders Adamson and Marc Alexa. Ray Tracing Point Set Surfaces. In *SMI '03: Proceedings of the Shape Modeling International 2003*, page 272, 2003.

[BCCS07] F. F. Bernardon, S. P. Callahan, J. L. D. Comba, and C. T. Silva. An Adaptive Framework for Visualizing Unstructured Grids with Time-Varying Scalar Fields. *Parallel Computing*, 2007. to appear.

[Bli82] James Blinn. A Generalization of Algebraic Surface Drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[CBC+01] J. C. Carr, R. K. Beatson, J. B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 67–76, New York, NY, USA, 2001. ACM Press.

[CHJ03] C. S. Co, B. Hamann, and K. I. Joy. Iso-splatting: A Point-based Alternative to Isosurface Visualization. In J. Rokne, W. Wang, and R. Klein, editors, *Proceedings of Pacific Graphics 2003*, pages 325–334, October 8–10 2003.

[dCJdFG99] A. de Cusatis Junior, L. de Figueiredo, and M. Gattas. Interval Methods for Raycasting Implicit Surfaces with Affine Arithmetic. In *Proceedings of XII SIBGRPHI*, pages 1–7, 1999.

[DK91] Akio Doi and Akoi Koide. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE Trans Commun. Elec. Inf. Syst*, E-74(1):213–224, 1991.

[DPH+03] David DeMarle, Steve Parker, Mark Hartner, Christiaan Gribble, and Charles Hansen. Distributed Interactive Ray Tracing for Large Volume Visualization. In *Proceedings of the IEEE PVG*, pages 87–94, 2003.

[Har96] J. C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545, 1996.

[HSS+05] Markus Hadwiger, Christian Sigg, Henning Scharsach, Khatja Bühler, and Markus Gross. Real-Time Ray-Casting and Advanced Shading of Discrete Isosurfaces. *Computer Graphics Forum*, 24(3):303–312, 2005.

[JLSW02] Tao Ju, Frank Losasso, Scott Schaefer, and Joe Warren. Dual contouring of hermite data. *ACM Trans. Graph.*, 21(3):339–346, 2002.

[KB89] D. Kalra and A. H. Barr. Guaranteed ray intersections with implicit surfaces. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pages 297–306, New York, NY, USA, 1989. ACM Press.

[KHH+06] Aaron Knoll, Younis Hijazi, Charles Hansen, Ingo Wald, and Hans Hagen. Interactive Ray Tracing of Arbitrary Implicit Functions. Technical Report UUSCI-2007-002, SCI Institute, University of Utah, 2006.

[KHW07] Aaron Knoll, Charles Hansen, and Ingo Wald. Coherent Multiresolution Isosurface Ray Tracing. Technical Report UUSCI-07-001, University of Utah, School of Computing, 2007. (submitted for publication).

[KWPH06] Aaron Knoll, Ingo Wald, Steven G Parker, and Charles D Hansen. Interactive Isosurface Ray Tracing of Large Octree Volumes. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 115–124, 2006.

[LC87] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics (Proceedings of ACM SIGGRAPH)*, 21(4):163–169, 1987.

[LH98] Yarden Livnat and Charles D. Hansen. View Dependent Isosurface Extraction. In *Proceedings of IEEE Visualization '98*, pages 175–180. IEEE Computer Society, October 1998.

[LMM06] D. Laney, A. Mascarenhas, and P. Miller. Understanding the Structure of the Turbulent Mixing Layer in Hydrodynamic Instabilities. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1053–1060, 2006. Member-P. -T. Bremer and Member-V. Pascucci.

[LSJ96] Yarden Livnat, Han-Wei Shen, and Christopher R. Johnson. A Near Optimal Isosurface Extraction Algorithm Using the Span Space. *IEEE Transactions on Visualization and Computer Graphics*, 2(1):73–84, 1996.

[LT04] Yarden Livnat and Xavier Tricoche. Interactive Point Based Isosurface Extraction. In *Proceedings of IEEE Visualization 2004*, pages 457–464, 2004.

[LW05] Mark Levoy and Turner Whitted. The use of points as display primitives. Technical report, CS Department, University of North Carolina at Chapel Hill, 2005.

[MFK+04] Gerd Marmitt, Heiko Friedrich, Andreas Kleer, Ingo Wald, and Philipp Slusallek. Fast and Accurate Ray-Voxel Intersection Techniques for Iso-Surface Ray Tracing. In *Proceedings of Vision, Modeling, and Visualization (VMV)*, pages 429–435, 2004.

[Mit90] Don Mitchell. Robust Ray Intersection with Interval Arithmetic. In *Proceedings on Graphics Interface 1990*, pages 68–74, 1990.

[ML94] Stephen R. Marschner and Richard J. Lobb. An evaluation of reconstruction filters for volume rendering. In *Proceedings of Visualization '94*, pages 100–107. IEEE Computer Society Press, 1994.

[MMMY97] Torsten M oller, Raghu Machiraju, Klaus Mueller, and Roni Yagel. Evaluation and Design of Filters Using a Taylor Series Expansion. *IEEE Transactions on Visualization and Computer Graphics*, 3(2):184–199, 1997.

[NH91] Greg Nielson and Bernd Hamann. The Asymptotic Decider: Removing the Ambiguity in Marching Cubes. In G. Nielson and L. Rosenblum, editors, *Proceedings of Visualization '91*, pages 83–91. IEEE Computer Society Press, 1991.

[Nie04] Gregory M. Nielson. Dual Marching Cubes. In *VIS '04: Proceedings of the conference on Visualization '04*, pages 489–496, Washington, DC, USA, 2004. IEEE Computer Society.

[NK05] Blake Nelson and Robert M. Kirby. Ray-Tracing Polymorphic Multi-Domain Spectral/hp Elements for Isosurface Rendering. *IEEE Transactions on Visualization and Computer Graphics (Proceedings IEEE Visualization 2005)*, 12(1):114–125, 2005.

[OBA+03] Yutaka Ohtake, Alexander Belyaev, Marc Alexa, Greg Turk, and Hans-Peter Seidel. Multi-level partition of unity implicits. *ACM Trans. Graph.*, 22(3):463–470, 2003.

[PLLdF06] Afonso Paiva, Hlio Lopes, Thomas Lewiner, and Luiz Henrique de Figueiredo. Robust adaptive meshes for implicit surfaces. In *19th Brazilian Symposium on Computer Graphics and Image Processing*, pages 205–212, 2006.

[PLPS04] Sinesio Pesco, Peter Lindstrom, Valerio Pascucci, and Claudio T. Silva. Implicit Occluders. In *IEEE/SIGGRAPH Symposium on Volume Visualization*, pages 47–54, 2004.

[PSL+98] Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter-Pike Sloan. Interactive Ray Tracing for Isosurface Rendering. In *IEEE Visualization '98*, pages 233–238, October 1998.

[RL00] Szymon Rusinkiewicz and Marc Levoy. QSplat: A Multiresolution Point Rendering System for Large Meshes. In *Proc. of ACM SIGGRAPH*, pages 343–352, 2000.

[SFS05] Carlos Scheidegger, Sachar Fleishman, and Claudio Silva. Triangulating Point-Set Surfaces With Bounded Error. In *Proc. of 3rd Eurographics/ACM Symposium on Geometry Processing*, pages 63–72, 2005.

[SS06] John Schreiner and Carlos Scheidegger. High-Quality Extraction of Isosurfaces from Regular and Irregular Grids. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1205–1212, 2006. Member-Claudio Silva.

[SW04] Scott Schaefer and Joe Warren. Dual Marching Cubes: Primal Contouring of Dual Grids. In *Proceedings of Pacific Graphics*, pages 70–76, 2004.

[WFKH07] Ingo Wald, Heiko Friedrich, Aaron Knoll, and Charles D Hansen. Interactive Isosurface Ray Tracing of Time-Varying Tetrahedral Volumes. Technical Report UUSCI-2007-003, SCI Institute, University of Utah, 2007. (submitted for publication).

[WFM+05] Ingo Wald, Heiko Friedrich, Gerd Marmitt, Philipp Slusallek, and Hans-Peter Seidel. Faster Isosurface Ray Tracing using Implicit KD-Trees. *IEEE Transactions on Visualization and Computer Graphics*, 11(5):562–573, 2005.

[WKE99] Rüdiger Westermann, Leif Kobbelt, and Tom Ertl. Real-time Exploration of Regular Volume Data by Adaptive Reconstruction of Iso-Surfaces. *The Visual Computer*, 15(2):100–111, 1999.

[WKME03] Manfred Weiler, Martin Kraus, Markus Merz, and Thomas Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 333–340, Washington, DC, USA, 2003. IEEE Computer Society.

[WMW86] Geoff Wyvill, Craig McPheeters, and Brian Wyvill. Data structure for soft objects. *The Visual Computer*, 2:227–234, 1986.

[WS05] Ingo Wald and Hans-Peter Seidel. Interactive Ray Tracing of Point Based Models. In *Proceedings of 2005 Symposium on Point Based Graphics (PGB)*, page to appear, 2005.

[WV92] J Wilhelms and A Van Gelder. Octrees for faster isosurface generation. *ACM Transactions on Graphics*, 11(3):201–227, July 1992.

[ZG06] Yuan Zhou and Michael Garland. Interactive Point-Based Rendering of Higher-Order Tetrahedral Data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1229–1236, 2006.