Adaptively Sampled Distance Fields: A General Representation of Shape for Computer Graphics

Sarah F. Frisken, Ronald N. Perry, Alyn P. Rockwood, and Thouis R. Jones
MERL – Mitsubishi Electric Research Laboratory

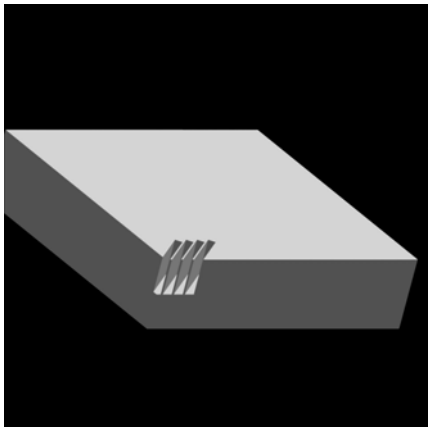


Figure 1. An ADF showing fine detail carved on a rectangular slab with a flat-edged chisel.

Figure 2. Artistic carving of a high order surface with a rounded chisel.

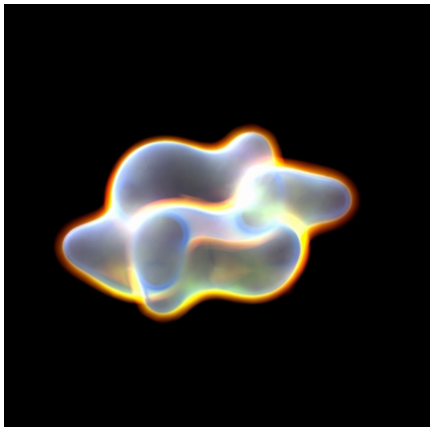


Figure 3. A semi-transparent electron probability distribution of a cyclohexane molecule.

ABSTRACT

Adaptively Sampled Distance Fields (ADFs) are a unifying representation of shape that integrate numerous concepts in computer graphics including the representation of geometry and volume data and a broad range of processing operations such as rendering, sculpting, level-of-detail management, surface offsetting, collision detection, and color gamut correction. Its structure is uncomplicated and direct, but is especially effective for quality reconstruction of complex shapes, e.g., artistic and organic forms, precision parts, volumes, high order functions, and fractals. We characterize one implementation of ADFs, illustrating its utility on two diverse applications: 1) artistic carving of fine detail, and 2) representing and rendering volume data and volumetric effects. Other applications are briefly presented.

CR Categories: I.3.6 [Computer Graphics]: Methodology and techniques – Graphics data structures; I.3.5 Computational Geometry and Object Modeling – Object modeling

Keywords: distance fields, carving, implicit surfaces, rendering, volume rendering, volume modeling, level of detail, graphics.

{frisken,perry,rockwood,jones}@merl.com

SIGGRAPH 2000, New Orleans, LA USA

1. INTRODUCTION

In this paper we propose adaptively sampled distance fields (ADFs) as a fundamental graphical data structure. A distance field is a scalar field that specifies the minimum distance to a shape, where the distance may be signed to distinguish between the inside and outside of the shape. In ADFs, distance fields are adaptively sampled according to local detail and stored in a spatial hierarchy for efficient processing. We recommend ADFs as a simple, yet consolidating form that supports an extensive variety of graphical shapes and a diverse set of processing operations. Figures 1, 2, and 3 illustrate the quality of object representation and rendering that can be achieved with ADFs as well as the diversity of processing they permit. Figures 1 and 2 show fine detail carved on a slab and an artistic carving on a high order curved surface. Figure 3 depicts an electron probability distribution of a molecule that has been volume rendered with a glowing aura that was computed using a 3D noise function.

ADFs have advantages over several standard shape representations because as well as representing a broad class of forms, they can also be used for a number of important operations such as locating surface points, performing inside/outside and proximity tests, Boolean operations, blending and filleting, determining the closest points on a surface, creating offset surfaces, and morphing between shapes.

It is important to note that by *shape* we mean more than just the 3D geometry of physical objects. We use it in a broad context for any locus defined in a metric space. Shape can have arbitrary dimension and can be derived from measured scientific data, computer simulation, or object trajectories through time and space. It may even be non-Euclidean.

2. BACKGROUND

Commonly used shape representations for geometric design include parametric surfaces, subdivision surfaces, and implicit surfaces. Parametric representations include polygons, spline patches, and trimmed NURBs. Localizing (or generating) surface points on parametric surfaces is generally simpler than with other

representations and hence they are easier to draw, tessellate, subdivide, and bound [3]. Parametric surfaces typically need associated data structures such as B-reps or space partitioning structures for representing connectivity and for more efficient localization of primitives in rendering, collision detection, and other processing. Creating and maintaining such structures adds to the computational and memory requirements of the representation. Parametric surfaces also do not directly represent object interiors or exteriors and are subsequently more difficult to blend and use in Boolean operations. While subdivision surfaces provide an enhanced design interface, e.g., shapes are topologically unrestricted, they still suffer from many of the same limitations as parametric representations, e.g., the need for auxiliary data structures [8], the need to handle extraordinary points, and the difficulty in controlling fine edits.

Implicit surfaces are defined by an implicit function $f(\mathbf{x} \in \mathbf{R}^n) = c$, where c is the constant value of the iso-surface. Implicit functions naturally distinguish between interior and exterior and can be used to blend objects together and to morph between objects. Boolean operations defined for implicit functions provide a natural sculpting interface for implicit surfaces [3, 18]; however, when many operations are combined to generate a shape the computational requirements for interactive rendering or other processing become prohibitive. Furthermore, it is difficult to define an implicit function for an arbitrary object, or to chart points on its surface for rendering and other processing.

Volumetric data consists of a regular or irregular grid of sampled data, frequently generated from 3D image data or numerical simulation. Object surfaces can be represented as iso-surfaces of the sampled values and data between sample points can be reconstructed from local values for rendering or other processing. Several systems have been developed for sculpting volumetric data using Boolean operations on sample density values [1, 2]. However, in these systems, iso-surfaces lack sharp corners and edges because the density values are low-pass filtered near object surfaces to avoid aliasing artifacts in rendering. In addition, the need to pre-select volume size and the use of regular sampling force these systems to limit the amount of detail achievable. Sensable Devices™ has recently introduced a commercial volume sculpting system [21]. To create detailed models, very large volumes are required (a minimum of 512 Mbytes of RAM) and the system is advertised for modeling only "organic" forms, i.e. shapes with rounded edges and corners.

Additional representations of shape for computer graphics include look-up tables, Fourier expansions, particle systems, grammar-based models, and fractals (iterated function systems), all of which tend to have focused applications [10].

The ADF representation, its applications, and the implementation details presented in this paper are new. Sampled distance fields have, however, been used previously in a number of specific applications. They have been used in robotics for path planning [12, 13] and to generate swept volumes [20]. In computer graphics, sampled distance fields were proposed for volume rendering [11], to generate offset surfaces [4, 17], and to morph between surface models [7, 17]. Level sets can either be generated from distance fields or they can be used to generate sampled distance fields [15, 22]. As with regularly sampled volumes, regularly sampled distance fields suffer from large volume sizes and a resolution limited by the sampling rate. These limitations are addressed by ADFs.

3. ADAPTIVE DISTANCE FIELDS

A *distance field* is a scalar field that specifies the minimum distance to a shape, where the distance may be signed to distinguish between the inside and outside of the shape. As simple examples, consider the distance field of the unit sphere S in $\mathbf{R}^3$ given by $h(\mathbf{x}) = 1 - (x^2 + y^2 + z^2)^{\frac{1}{2}}$, in which h is the Euclidean signed distance from S, or $h(\mathbf{x}) = 1 - (x^2 + y^2 + z^2)$, in which h is the algebraic signed distance from S, or $h(\mathbf{x}) = (1 - (x^2 + y^2 + z^2))^2$, in which h is an unsigned distance from S.

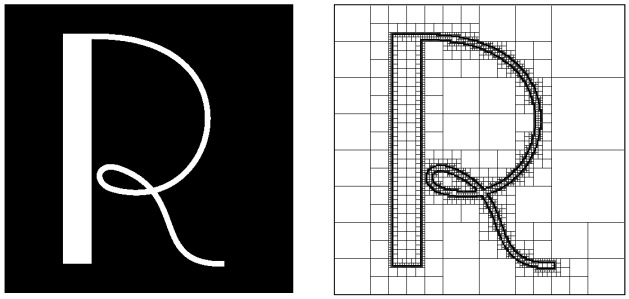
The distance field is an effective representation of shape. However, regularly sampled distance fields have drawbacks because of their size and limited resolution. Because fine detail requires dense sampling, immense volumes are needed to accurately represent classical distance fields with regular sampling when *any* fine detail is present, even when the fine detail occupies only a small fraction of the volume. To overcome this limitation, ADFs use adaptive, detail-directed sampling, with high sampling rates in regions where the distance field contains fine detail and low sampling rates where the field varies smoothly. Adaptive sampling permits arbitrary accuracy in the reconstructed field together with efficient memory usage. In order to process the adaptively sampled data more efficiently, ADFs store the sampled data in a hierarchy for fast localization. The combination of detail-directed sampling and the use of a spatial hierarchy for data storage allows ADFs to represent complex shapes to arbitrary precision while permitting efficient processing.

In summary, ADFs consist of adaptively sampled distance values organized in a spatial data structure together with a method for reconstructing the underlying distance field from the sampled values. One can imagine a number of different instantiations of ADFs using a variety of distance functions, reconstruction methods, and spatial data structures. To provide a clear elucidation of ADFs, we focus on one specific instance for the remainder of this paper. This instance is simple, but results in efficient rendering, editing, and other processing used by applications developed in this paper. Specifically, we demonstrate an ADF which stores distance values at cell vertices of an octree data structure and uses trilinear interpolation for reconstruction and gradient estimation. The wide range of research in adaptive representations suggest several other ADF instantiations based on, for example, wavelets [5] or multi-resolution Delaunay tetrahedralizations [6].

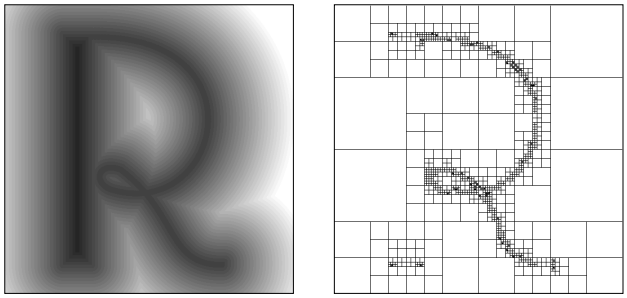
3.1 Octree-based ADFs

Octree data structures are well known and we assume familiarity (see [19]). For purposes of instruction, we demonstrate the concepts in 2D (with quadtrees), which are easily generalized to higher dimensions. In a quadtree-based ADF, each quadtree cell contains the sampled distance values of the cell's 4 corners and pointers to parent and child cells.

Given a shape as in Figure 4a, subdivision of a cell in the quadtree depends on the variation of the distance field (shown in Figure 4c) over the parent cell. This differs from 3-color quadtrees [19] which represent object boundaries by assigning one of three types to each cell in the quadtree: interior, exterior, and boundary. In 3-color quadtrees, all boundary cells are subdivided to a predetermined highest resolution level. In contrast, boundary cells of ADFs are only subdivided when the distance field within a cell is not well approximated by bilinear interpolation of its corner values. Hence, large cells can be used to represent edges in regions where the shape is relatively smooth, resulting in significantly more compression than 3-color quadtrees. This is illustrated in Figures 4b and 4d where the ADF of 4d requires only 1713 cells while the 3-color quadtree of 4b requires 23,573 cells. In the ADF quadtree, straight edges of the "R" are represented by large cells; only corners provoke repeated



Figures 4a "R" and **4b** 3-color quadtree containing 23,573 cells.



Figures 4c Distance field of "R" and **4d** ADF containing 1713 cells.

Error Tolerance	*Triangle Count*	*ADF Cell Count*	*ADF Sample Count*
6.25×10^{-5}	32,768	16,201	24,809
3.13×10^{-5}	131,072	44,681	67,405
1.56×10^{-6}	2,097,152	131,913	164,847

Table 1. Comparison of triangle count for a sphere (r = 0.4) to ADF size.

subdivision.

Figure 4d also shows that even highly curved edges can be efficiently represented by ADFs. Because bilinear interpolation represents curvature reasonably well, cells with smoothly curved edges do not require many levels in the ADF hierarchy. Cells that do require many levels in the hierarchy are concentrated at corners and cusps.

These are typical statistics for 2D objects. As another indication of ADF size, Table 1 compares the number of triangles required to represent a sphere of radius 0.4 to the number of cells and distance (sample) values of the corresponding ADF when both the triangles and the interpolated distance values are within a given error tolerance from the true sphere.

Higher order reconstruction methods and better predicates for subdivision might be employed to further increase compression, but the numbers already suggest a point of diminishing returns for the extra effort.

3.2 Generating ADFs

The generation of an ADF requires a procedure or function to produce the distance function, $h(\mathbf{x})$ at $\mathbf{x} \in \mathbf{R}^n$, where distance is interpreted very broadly as in Section 3. Continuity, differentiability, and bounded growth of the distance function can be used to advantage in rendering or other processing, but are not required. Some of the images in this paper utilize distance functions that are non-differentiable (Figure 8) and highly non-Euclidean with rapid polynomial growth (Figures 2 and 3).

One example of a distance function is the implicit form of an object, for which the distance function can correspond directly to the implicit function. A second example includes procedures that determine the Euclidean distance to a parametric surface. For

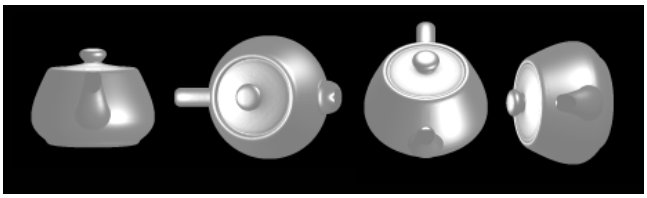


Figure 5. A thin walled version of the Utah Teapot rendered using sampled ray casting and Phong lighting from an ADF computed from a 32 bicubic Bezier patch model.

example, Figure 5 was rendered from the distance field computed for a 32 bicubic Bezier patch model of the Utah Teapot. Distances to Bezier patches were determined by solving 7^{th} order Bezier equations using the Bezier clipping algorithm described in [14]. To define an inside and outside for the teapot, the unsigned distance from the Bezier surface is biased to produce the signed distance field of an offset surface, resulting in a thin-walled teapot. Other distance functions include Euclidean distances for a triangle model that can be computed as the minimum of the signed distances to each of the triangles in the model and distance fields computed by applying Boolean operations to the distance fields of primitive elements in a CSG representation.

Given a distance function, there are a number of ways to generate an ADF. Two simple examples include a bottom-up and a top-down approach which are described briefly here. The bottom-up approach starts with a regularly sampled distance field of finite resolution and constructs a fully populated octree for the 3D data. Starting with the smallest cells in the octree, a group of 8 neighboring cells is coalesced if and only if none of the cells have any child cells and the sampled distances of all of the 8 cells can be reconstructed from the sample values of their parent to a specified error tolerance. After all cells are considered for coalescing at a given level in the hierarchy, groups of cells at the next level are considered. When no cells are coalesced at a given level or the root node is reached, the ADF generation is complete.

In the top-down approach, first the distance values for the root node of the ADF hierarchy are computed. ADF cells are then recursively subdivided according to a subdivision rule. For example, if the primary interest is the iso-surface represented in the field, the recursive subdivision would stop if the given cell is guaranteed not to contain the surface, if the cell contains the surface but passes some predicate, or if a specified maximum level in the hierarchy is reached. One can imagine many predicates to control the subdivision. In examples presented in this paper, we use a simple predicate that compares distances within a cell computed using the distance function to distances reconstructed from the cell's sampled values. In this predicate, the absolute differences between the computed and reconstructed distances are determined at the center of the cell and the centers of each of the cell's faces and edges (i.e. 19 differences per cell). If any of the differences are greater than a specified error tolerance, the cell is subdivided.

3.3 Reconstructing ADFs

Each ADF cell has an associated method for reconstructing distance values between sampled points. In the case of the 3D octree, distance values within a cell are reconstructed from the 8 corner distance values stored per cell using standard trilinear interpolation. In addition to distance values, many operations such as rendering, collision detection, or closest point localization require surface normals and hence, processing an ADF may also require a method for estimating surface normals from the sampled data. For distance fields, the surface normal is equal to the

normalized gradient of the distance field at the surface. There are several methods for estimating the gradient of sampled data. We use the analytic gradient of the trilinear reconstruction within each cell: $grad(x,y,z) = (h(x_r,y,z) - h(x_l,y,z), h(x,y_u,z) - h(x,y_d,z), h(x,y,z_f) - h(x,y,z_b))$, where (x_r,y,z), (x_l,y,z), (x,y_u,z), etc. are projections of (x,y,z) onto the right, left, up, down, front, and back faces of the cell, respectively. In theory, this cell-localized gradient estimation can result in C^1 discontinuities at cell boundaries but as can be seen from the figures, these artifacts are not noticeable with sufficient subdivision.

4. APPLICATIONS AND IMPLEMENTATION DETAILS

ADFs have application in a broad range of computer graphics problems. We present two examples below to illustrate the utility of ADFs and to provide some useful implementation details on processing methods such as rendering and sculpting ADF models. This section ends with short descriptions of several other applications to give the reader an idea of the diverse utility of ADFs.

4.1 Precise carving

Figures 1, 2, and 6 show examples of objects represented and carved as ADFs. Because objects are represented as distance fields, the ADF can represent and reconstruct smooth surfaces from sampled data. Because the ADF efficiently samples distance fields with high local curvature, it can represent sharp surface corners without requiring excessive memory. Carving is intuitive; the object is edited simply by moving a tool across the surface. It does not require control point manipulation, remeshing the surface, or trimming. By storing sample points in an octree, both localizing the surface for editing and determining ray-surface intersections for rendering are efficient.

Like implicit surfaces, ADFs can be sculpted using simple Boolean operations applied to the object and tool distance fields. Figures 1, 2, and 6 show carving using the difference operator, $h_{carved}(\mathbf{x}) = \min(h_{object}(\mathbf{x}), -h_{tool}(\mathbf{x}))$. Other operators include addition, $h_{carved}(\mathbf{x}) = \max(h_{object}(\mathbf{x}), h_{tool}(\mathbf{x}))$, and intersection, $h_{carved}(\mathbf{x}) = \min(h_{object}(\mathbf{x}), h_{tool}(\mathbf{x}))$. Blending or filleting can also be defined for shaping or combining objects (as was done for the molecules of Figures 3 and 7). While these Boolean operations apply to the entire distance field, for systems where only surfaces are important, application of the operations can be limited in practice to a region within a slightly extended bounding box of the tool.

The basic edit operation is much like a localized ADF generation. The first step in the editing process is to determine the smallest ADF cell, or set of cells, entirely containing the tool's extended bounding box (obvious consideration of the ADF boundaries apply). The containing cell is then recursively subdivided, applying the difference operator to the object and tool values to obtain new values for the carved ADF. During the recursive subdivision, cell values from the object are obtained either from existing sampled values or by reconstruction if an edited cell is subdivided beyond its original level. Subdivision rules similar to those of top-down generation are applied, with the exception that the containing cell must be subdivided to some minimum level related to the tool size.

The carving examples were rendered using ray casting with analytic surface intersection. In this method, a surface point is determined by finding the intersection between a ray cast into the ADF octree from the eye and the zero-value iso-surface of the ADF. Local gradients are computed at surface points using the gradient estimation described above (the figures were rendered with simple Phong lighting). When the traversing ray passes through a leaf node of the octree, intersection between the ray and the surface reconstructed from the 8 cell sample values is tested. We have used two different methods to find the ray-surface intersection; a cubic root solver that finds the exact intersection of the ray with the trilinear surface defined by the distance values at the cell corners (as in [16]), and a linear approximation which determines the distance values where the ray enters and exits the cell and computes the linear zero-crossing if the two values have a different sign. Both methods work well but the linear approximation has proven to be faster and its rendered images are not visibly different from those rendered with the cubic solver. When solving for intersections, we set the distance at the entry point of a cell to be equal to the distance at the exit from the previous cell. This avoids the crack problem discussed in [24] for rendering hierarchical volume data, preventing C^0 discontinuities in the surface where ADF cells of different size abut. Most of the images shown in this paper were rendered using a supersampling of 16 rays per pixel followed by the application of a Mitchell filter of radius 2.0.

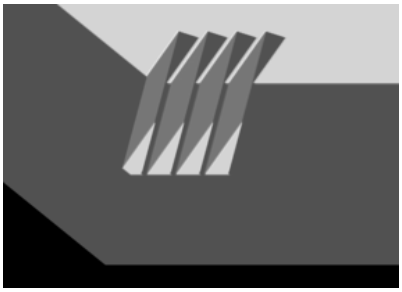


Figure 6. A close up of the carved slab in Figure 1.

The octree promotes efficient ray traversal even for very complicated scenes. Rendering the Menger Sponge (Figure 8) takes approximately the same amount of time as rendering less complex ADF models. As in most rendering methods based on spatial decomposition, rendering time is determined more by screen coverage than by model complexity. Current rendering rates are fast enough for interactive updating of the carving region during editing. Preliminary tests indicate that an order of magnitude improvement in the rendering speed of the entire image can be achieved by adaptive supersampling.

4.2 Volume data

ADFs are also amenable to volume rendering and can be used to produce interesting effects. For example, offset surfaces can be used to render thick, translucent surfaces. Adding volume texture within the thick surface in the form of variations in color or transparency is relatively easy. In addition, distance values farther away from the zero-valued iso-surface can be used for special effects. Figure 7 shows a cocaine molecule volume rendered in a haze of turbulent mist. The mist was generated using a color function based on distance from the molecule surface. To achieve the turbulence the distance value input to the color function is modulated by a noise function based on position [9].

We use a ray casting volume renderer to demonstrate some of these effects. Colors and opacities are accumulated at equally spaced samples along each ray using a back-to-front rendering algorithm. Sample points that lie near the zero-value iso-surface are shaded with Phong lighting.

Our sampled ray caster is not optimized for speed. However, properties of the ADF data structure can be used to greatly increase the rendering rate. For example, the octree allows us to quickly skip regions of the volume that are far from the surface. In

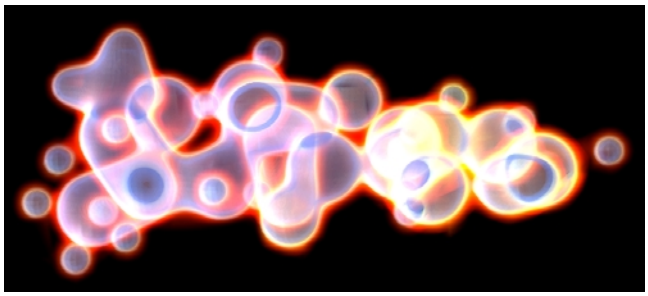


Figure 7. An ADF cocaine molecule volume rendered in a haze of turbulent mist. The mist was generated using a color function dependent on distance from the molecule surface.

addition, because distances to the closest surface are available at each sample point, space-leaping methods can be used to speed up rendering [26].

4.3 Other application areas

4.3.1 Representing complexity

Complexity may be considered from several viewpoints. Firstly, the *visual complexity* of an object might include factors such as surface variation and topology. Secondly, the *representation complexity* is determined by the size and intricacy of the data structure needed to represent the object. The third measure of complexity considers the *algebraic complexity* of the object, which includes such factors as polynomial degree, transcendental functions, and numerical routines required to define the object's shape. Such routines are pertinent especially when algebraic distance is employed for the distance field.

Figure 8 shows a good example of the first two types of complexity, the Menger Sponge, which is a fractal created recursively by subtracting smaller and smaller crossing cuboids from an initial cube. In the limit there is no neighborhood of the surface that is not punctured regardless of how small the neighborhood is chosen. It is an infinite perforation, a 3D version of the famous Cantor set.

After each level of subtraction there are 20 self-similar subcuboids generated. An artless approach to maintaining the data structure would generate order $O(20^n)$ faces for n iterations. Even if shared faces were combined and interior faces culled, an approach that keeps a boundary representation (B-rep) without troublesome T-junctions would have $O(12^n)$ faces. To be more exact, after seven iterations there would be 26 million+ faces in a B-rep data structure. Consider the difficulty of performing proximity tests, collision detection, or inside/outside tests with such a representation. In contrast, these tests are much simpler using ADFs. Far from being a contrived case, the complexity of the distance field of the Menger Sponge is representative of the distance fields of many naturally occurring shapes which would present similar problems for traditional methods.

Figures 2 and 7 both demonstrate ADFs' ability to handle algebraic complexity. While Figure 7 reconstructs an approximate probability density field for a molecule of 43 atoms ($C_{17}H_{21}NO_4$), the vase in Figure 2 is defined first as a rotation of a quintic Bezier curve. Mathematically, it is posed as a rational implicit function with a square root of a (total) degree 16 over 2. Cubic Bezier curves are then mapped onto the surface as paths for the carving tool. In this case, the carver is a curved chisel, resulting in a very high degree tubular surface on the vase. This carving path and vase create an algebraically very complex distance field, which is nevertheless cleanly reconstructed and rendered.

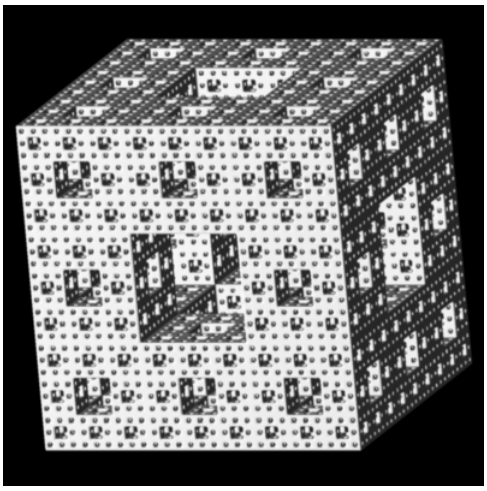


Figure 8. An ADF of the Menger Sponge, a fractal created recursively by subtracting smaller and smaller crossing cuboids from an initial cube. Four levels of recursion are shown.

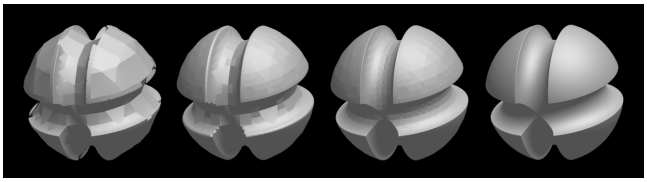


Figure 9. Four LOD models with varying amounts of error rendered from an ADF octree.

4.3.2 Level-of-detail models

There are at least two approaches for representing ADF models at different levels of detail for rendering and progressive transmission of models. The simplest approach is to truncate the ADF at fixed levels in the octree. The truncation can either be done during rendering or transmission, during generation, or to an existing high resolution ADF. A second method uses the error generated in the test for cell subdivision during top-down generation of the ADF. By storing this error within the cell, an LOD model can be generated by truncating ADF cells with errors less than that specified for the given LOD. This provides a more continuously varying parameter for selecting the LOD and provides degradation of the object shape that is consistent for both smooth and highly curved portions of the surface as the level of the LOD model decreases. This second method is illustrated in Figure 9 where four LOD models with varying amounts of error are rendered from an ADF octree.

4.3.3 Collision detection

Distance fields have been used for collision avoidance in robotics and for detecting collisions and computing penetration forces in haptics [12, 13]. Octrees or other hierarchies of bounding boxes have also been used successfully to accelerate collision detection algorithms. The combination of these two representations in the ADF as well as the ability to represent offset surfaces and surfaces at different levels of detail suggest that ADFs have significant potential for applications that require collision detection.

4.3.4 Color gamut representation

Devices such as color printers and monitors have unique color characteristics. Each can represent colors within their own particular color gamut, which is restricted, for example, by the types of dyes used by the printer. When an image is acquired or designed on one system and then displayed or printed on another, it is often important to match colors as closely as possible. This

involves correcting colors that fall outside of the device's gamut and sometimes requires a complicated mapping to warp the gamut of one system onto that of another [23].

Most color devices represent their color gamuts in large look-up-tables (LUTs). Usually, a binary table is used to test colors against the device's gamut to see if they fall in or out of gamut. If a color falls out of gamut, a set of model coefficients and look-up tables are used to map the color onto the 'closest' device color. Using ADFs to represent a device's gamut has several advantages over the LUT approach. First, out-of-gamut tests are easily performed with ADFs and edge-sampling errors that occur with the use of binary tables are avoided. Second, an ADF out-of-gamut test provides more information than is available with binary tables; the distance indicates how far out of gamut a color lies and the gradient indicates the direction to the nearest in-gamut color. Third, since ADFs use adaptive sampling, they should provide significant compression over LUT representations. Finally, since distance fields can be used to warp between shapes, ADFs may prove to be a useful representation for mapping between device gamuts.

4.3.5 Machining

ADFs provide powerful tools for computer aided machining. The use of a distance function for representing surfaces allows the representation of the surface, the interior of the object, and the material that must be removed. Knowledge of the object interior can be used for part testing (e.g., part thickness tests [25]). A representation of the volume outside of the surface as well as distances to the closest surface can be used for planning tool paths and tool sizes for the machining process. Offset surfaces can be used to plan rough cutting for coarse-to-fine machining or for designing part molds for casting. The size of cells at the surface and the object normal near the surface can be used to select tool size and orientation. Finally, as illustrated in Figure 6, ADFs can represent fine surfaces and sharp corners efficiently, making it possible to represent machining precision in the ADF model.

5. CONCLUSIONS

Although distance fields have been used in certain specific applications as mentioned above, the breadth and flexibility of their application to problems in computer graphics has not been appreciated, in part due to their large memory requirements. ADFs address this issue by adaptively sampling the distance field and storing sampled values in a spatial hierarchy. For 2D shapes, we typically achieve better than 20:1 reductions over straightforward boundary (3-color) quadtrees. Nevertheless, ADFs maintain the reconstruction quality of the original distance field as seen in the examples presented; shapes, even those with high frequency components such as edges or corners, are reconstructed accurately.

Distance fields can embody considerable information about a shape, not just the critical zero-valued iso-surface, but also information about the volume in which it sits, an indication of inside vs. outside, and gradient and proximity information.

Operations on a shape can often be achieved by operations on its distance field. For example, Boolean set operations become simple max/min operations on the field; edges and corners can be rounded by low-pass filtering; and so forth.

ADFs tend to separate generation of shapes into a preprocess step that may require complex and time-consuming methods, and a process for graphical operations that is fast and tolerant of various types of complexity. Indeed, fractals and mathematically sophisticated or carved shapes can be processed as quickly as much simpler shapes. The wide diversity of such manipulations include, for example, proximity testing (for collision detection, haptics, color gamut correction, milling), efficient ray-surface intersection for rendering, localized reconstruction, surface and volume texturing, blending, filleting, offset surfaces, and shape warping.

6. FUTURE WORK

The introduction of ADFs opens up a wide range of future directions. Considerable research is left to investigate the possible transformations between shape and its distance field. Different hierarchical structures and reconstruction methods await testing and experience. For example, wavelets show particular promise [5], and Delaunay tetrahedralizations have been successfully used for multiresolution representation of volume data [6]. The relative compactness for very complex shapes has implications for level of detail management and progressive transmission. Efficient conversion between ADFs and standard (e.g. triangle and NURB) models is a valuable undertaking. Finally, we look forward to combining ADFs with more powerful rendering methods; for example, we envision hierarchical radiosity using form factors based on the ADF cells.

7. ACKNOWLEDGEMENTS

We gratefully acknowledge the help of Mars Brimhall, John Ford, and Stephan Roth in generating some of the images in this paper.

8. REFERENCES

[1] R. Avila and L. Sobierajski, "A haptic interaction method for volume visualization", Proc. IEEE Visualization'96, pp. 197-204, 1996.
[2] J. Baerentzen, "Octree-based volume sculpting", Proc. Late Breaking Hot Topics, IEEE Visualization'98, pp. 9-12, 1998.
[3] J. Bloomenthal, Introduction to Implicit Surfaces, Morgan Kaufman Publishers, 1997.
[4] D. Breen, S. Mauch and R. Whitaker, "3D scan conversion of CSG models into distance volumes", Proc. 1998 IEEE Symposium on Volume Visualization, pp. 7-14, 1998.
[5] M. Chow and M. Teichmann, "A Wavelet-Based Multiresolution Polyhedral Object Representation", Visual Proc. SIGGRAPH '97, p. 175, 1997.
[6] P. Cignoni, L. De Floriani, C. Montani, E. Puppo, R. Scopigno, "Multiresolution Modeling and Rendering of Volume Data based on Simplicial Complexes", 1994 ACM Volume Visualization Conference Proceedings, 1994, pp.19-26.
[7] D. Cohen-Or, D. Levin, and A. Solomovici, "Three-dimensional distance field metamorphosis", ACM Transactions on Graphics, 1997.
[8] T. DeRose, M. Kass, T. Truong, "Subdivision surfaces in character animation", Proc. SIGGRAPH '98, pp. 85-94, 1998.
[9] D. Ebert, F.K. Musgrave, D. Peachy, K. Perlin, S. Worley, Texturing and Modeling a Procedural Approach, Academic Press, 1998.
[10] J. Foley, A. van Dam, S. Feiner, and J. Hughes, Computer Graphics: Principles and Practice, Addison-Wesley, 1992.
[11] S. Gibson, "Using DistanceMaps for smooth surface representation in sampled volumes", Proc. 1998 IEEE Volume Visualization Symposium, pp. 23-30, 1998.
[12] R. Kimmel, N. Kiryati and A. Bruckstein, "Multi-valued distance maps for motion planning on surfaces with moving obstacles", IEEE Trans. on Robotics & Automation, 14, pp. 427-436, 1998.
[13] J. Lengyel, M. Reichert, B. Donald and D. Greenberg, "Real-time robot motion planning using rasterizing computer graphics hardware", Proc. SIGGRAPH '90, pp. 327-335, 1990.
[14] T. Nishita, T.W. Sederberg and M. Kakimoto, "Ray tracing trimmed rational surface patches", Proc. SIGGRAPH '90, pp. 337-345, 1990.
[15] S. Osher and J. Sethian, "Fronts propagating with curvature-dependent speed: algorithms based on Hamilton-Jacobi formulation", J. Computational Physics, 79, pp. 12-49, 1988.
[16] S. Parker, M. Parker, Y. Livnat, P. Sloan, C. Hansen, and P. Shirley, "Interactive ray tracing for volume visualization" IEEE Transactions On Visualization and Computer Graphics, Vol. 5 (3), pp. 238-250, 1999.
[17] B. Payne and A. Toga, "Distance field manipulation of surface models", IEEE Computer Graphics and Applications, pp. 65-71, 1992.
[18] A. Ricci, "A constructive geometry for computer graphics", Computer Journal, Vol. 16, No. 2, pp. 157-160, 1973.
[19] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1989.
[20] W. Schroeder, W. Lorensen, and S. Linthicum, "Implicit modeling of swept surfaces and volumes," Proc. Visualization '94, pp. 40-45, 1994.
[21] Sensable Devices' FreeForm modeling software. http://www.sensable.com/freeform.
[22] J. Sethian, Level Set Methods: Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Material Science, Cambridge University Press, 1996.
[23] M. Stone, W. Cowan, J. Beatty, "Color gamut mappings and the printing of digital color images", ACM Transaction on Graphics, Vol. 7, pp. 249-292, 1988.
[24] R. Westermann, O. Sommer, T. Ertl, "Decoupling polygon rendering from geometry using rasterization hardware", in Proc. Eurographics Rendering Workshop '99, pp. 45-56, 1999.
[25] R. Yagel, S. Lu, A. Rubello, R. Miller, "Volume-based reasoning and visualization of dicastability" In Proc. IEEE Visualization '95, pp. 359-362, 1995.
[26] K. Zuiderveld, A. Koning, and M. Viergever, "Acceleration of ray-casting using 3D distance transforms", in Proc. Visualization in Biomedical Computing '92, pp. 324-335, 1992.