Hierarchical RLE Level Set: A Compact and Versatile Deformable Surface Representation

BEN HOUSTON
Exocortex Technologies, Frantic Films
MICHAEL B. NIELSEN
University of Århus
CHRISTOPHER BATTY
University of British Columbia, Frantic Films
OLA NILSSON
Linköping Institute of Technology
and
KEN MUSETH
Linköping Institute of Technology and University of Århus

This article introduces the Hierarchical Run-Length Encoded (H-RLE) Level Set data structure. This novel data structure combines the best features of the DT-Grid (of Nielsen and Museth [2004]) and the RLE Sparse Level Set (of Houston et al. [2004]) to provide both optimal efficiency and extreme versatility. In brief, the H-RLE level set employs an RLE in a dimensionally recursive fashion. The RLE scheme allows the compact storage of sequential nonnarrowband regions while the dimensionally recursive encoding along each axis efficiently compacts nonnarrowband planes and volumes. Consequently, this new structure can store and process level sets with effective voxel resolutions exceeding $5000 \times 3000 \times 3000$ (45 billion voxels) on commodity PCs with only 1 GB of memory. This article, besides introducing the H-RLE level set data structure and its efficient core algorithms, also describes numerous applications that have benefited from our use of this structure: our unified implicit object representation, efficient and robust mesh to level set conversion, rapid ray tracing, level set metamorphosis, collision detection, and fully sparse fluid simulation (including RLE vector and matrix representations.) Our comparisons of the popular octree level set and Peng level set structures to the H-RLE level set indicate that the latter is superior in both narrowband sequential access speed and overall memory usage.

Categories and Subject Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Curve, surface, solid and object representations*; *physically based modeling*; I.6.8 [**Simulation and Modeling**]: Types of Simulation—*Animation*

Ben Houston and Christopher Batty were supported in part by National Research Council Industrial Research Assistance Program Grant #482564, and Michael B. Nielsen was partly supported by Center for Interactive Spaces under ISIS Katrinebjerg, Aarhus. Ken Museth would also like to acknowledge support from the Swedish Research Council (Grant #617-2004-5017) as well as research donations from Alias. This 2-year effort would not have been possible without the kind support of Frantic Films.
Authors' addresses: B. Houston: Exocortex Technologies, 6405 Sugar Creek Way, Ottawa, Ont., Canada, K1C 1X9; email; ben@exocortex.org; M. B. Nielsen, O. Nilsson, and K. Museth: Department of Science and Technology, Linköping Institute of Technology, Linköping University, 601 74 Norrköping, Sweden; email: bang@daimi.au.dk, olani@itn.liu.se, museth@acm.org; C. Batty: UBC Department of Computer Science, ICICS/CS, 201-2366 Main Mall, Vancouver, BC, Canada, V6T 1Z4; email: cbatty@cs.ubc.ca.

General Terms: Algorithms, Performance
Additional Key Words and Phrases: Level set methods, implicit surfaces, deformable surfaces, adaptive distance fields, computational fluid dynamics, geometric modeling, shape, morphology, mesh scan conversion

1. INTRODUCTION

Modeling and animation of geometry lie at the very heart of many computer graphics applications. As such, the field of geometric modeling has been the subject of many research articles in recent years. Efficient, high-quality representation of deforming surfaces, however, remains a major challenge. In this article, we offer a solution to this challenge by introducing a novel data structure and algorithms that allow for implicit but compact representations of deforming geometry at very high resolutions. We demonstrate the usefulness of this data structure with a series of important graphics applications.

While polygonal meshes are very popular in many graphics applications, when they are used to represent deformable surfaces they are hampered by issues like self-intersections and degeneracy of the polygons. These limits are most apparent in the animation of fluid and fire as well as during shape metamorphosis. For such dramatic surface deformations, level set models have been increasingly popular [Foster and Fedkiw 2001; Breen and Whitaker 2001; Enright et al. 2002; Museth et al. 2002; Losasso et al. 2004]. Level sets, originally developed by Osher and Sethian[1988] for use in interface tracking, are essentially time-dependent isosurfaces (typically Euclidian distance fields) sampled on a three-dimensional (3D) grid. The surface deformations are described implicitly by solving an associated time-dependent partial differential equation (PDE). Several robust mathematical schemes exist for solving this PDE, which together allow for virtually any type of surface deformation. As such, it would seem advantageous to adapt this implicit geometry representation for many graphics applications. Most current level set implementations, however, suffer from either poor computational or memory efficiency. Consequently, the majority of applications have been limited to effective voxel resolutions no greater than 512^3. Two exceptions are the recently and independently proposed RLE Sparse Level Set [Houston et al. 2004] and DT-Grid [Nielsen and Museth 2006]. In this article, we combine these level set representations into the Hierarchical Run Length Encoded (H-RLE) level set representation and hence form a novel, versatile, compact, and fast data structure and toolbox of algorithms specifically suited for computer graphics. The Hierarchical RLE level set allows for very high-resolution level sets without compromising computational or memory efficiency and at the same time offers a very high level of flexibility, which is a necessity for computer graphics applications. We stress that to the best of our knowledge no other existing data structure offers all of these advantages.

Our general approach is to use a compact run-length-encoding (RLE) of sampled level sets by means of hierarchical dimensional decomposition. This approach generalizes to any dimensions through recursive definitions and allows for the H-RLE level set to scale proportionally to the area of the geometric surface. Furthermore, our versatile grid representation has the advantage that it is truly dynamic; it is unbounded and can expand indefinitely (within machine precision). We finally note that our data structure can be used to store any volume-defined fields such as volume textures, fluid velocity fields, elastic tensor fields, and sparse matrices. We illustrate the significant value of this data structure with several different challenging graphics applications.

2. RELATED WORK

The powerful *level set method* is due to Osher and Sethian [1988]. However, both time and storage complexity of this original formulation is $O(n^3)$, where n is the side length of a bounding volume for

the corresponding surface deformation. Adalsteinsson and Sethian [1995] introduced a *narrowband method* which restricted most computations to a thin band of active voxels immediately surrounding the interface, thus reducing the time complexity to $O(n^2)$. However, the storage complexity for this narrowband scheme was still $O(n^3)$. Additionally, periodic updates of the narrowband structure required an $O(n^3)$ operation in which voxels over the entire volume were accessed to rebuild the list of active voxels. This $O(n^3)$ time complexity was eliminated in the approximate *sparse field level set method* of Whitaker [1998]. The sparse field level set method employs a set of linked lists to track the active voxels around the interface. This allows incremental extension of the active region as needed without incurring any significant overhead. While consistently efficient in time, $O(n^3)$ storage space is still required by the sparse field level set method. Peng et al. [1999] next introduced a narrowband scheme tailored for more accurate finite difference schemes and without the need for slow linked lists. However, their method did not overcome the $O(n^3)$ storage requirements and the $O(n^3)$ periodic update.

These narrowband methods [Adalsteinsson and Sethian 1995; Whitaker 1998; Peng et al. 1999] all require $O(n^3)$ storage even though computations are restricted to the $O(n^2)$ subset of the voxels around the interface. In fact additional storage is required in order to track the actual grid poins present in the narrowband. The presence of this dichotomy has thus pushed research toward reducing the storage requirements of level sets without significantly affecting the $O(n^2)$ computational efficiency. Two such recent approaches are sparse block grids and octrees.

The *sparse block grid* method of Bridson [2003] divides the entire bounding volume of size n^3 into small cubic blocks of m^3 voxels each. A coarse grid of size $(\frac{n}{m})^3$ then stores pointers only to those blocks that intersect the narrowband of the level set. Block allocation and deallocation occur as the surface propagates to accommodate to the deformations. This method has a suboptimal storage complexity of $O((\frac{n}{m})^3 + m^3n^2)$, but retains the constant time access inherent to dense grids.

Octrees have, in recent years, been applied to narrowband level sets [Strain 1999; Bridson 2003; Guendelman et al. 2003; Losasso et al. 2004, 2005; Enright et al. 2004] and signed distance fields [Frisken et al. 2000]. The octree level set method uses a tree of nested cubes of which the leaf nodes contain signed distance values. This representation is efficient in terms of storage, $O(n^2)$, and relatively efficient in terms of access queries, $O(\log n)$. However, as stated in Bridson [2003], there are hidden costs associated with the pointer structure of the octree, and a single random access can result in $O(\log n)$ cache misses. Implementing most finite difference-based level set methods using octrees requires uniform refinement along the interface in order to obtain sufficient precision [Losasso et al. 2005]. In particular, it is difficult to accurately solve hyperbolic advection with high-order methods like ENO [Osher and Shu 1991] or WENO [Liu et al. 1994] and parabolic diffusion problems like curvature based flow on a non-uniform octree. Hence one of the most advantageous features of octrees, that is, adaptiveness, is rarely exploited in level set simulations. However, octrees have proven very efficient for adaptive representations of velocities in computational fluid simulations [Losasso et al. 2004]. It should be noted, though, that these solutions of the Navier-Stokes equations on an adaptive octree grid ignore parabolic diffusion (i.e., they are inviscid) and use simple semi-Lagrangian integration techniques for the self-advection term.

The work most related to ours is the *RLE sparse level set* representation of Houston et al. [2004] and the *Dynamic Tubular Grid* (DT-Grid) of Nielsen and Museth [2006].

In the past, Run-Length-Encoding (RLE) has been successfully applied to volumes, for example, Curless et al. [1996]. While Bridson [2003] mentioned the potential value of applying RLE to level set volumes, Houston et al. [2004] were the first to implement an RLE compressed level set volume data structure. Using a two-dimensional look-up table, their RLE sparse level set also provided a very fast, $O(logr)$, random access method, where r is the number of runs in a level set cross section.

The DT-Grid of Nielsen and Museth [2006] is a level set representation that employs a hierarchical (with respect to dimension) compression scheme inspired by the "compressed row storage" technique known from sparse matrix representations. Their data structure and algorithms focus on space- and time-efficient level set representations of deforming closed interfaces. The memory consumption of the DT-Grid is proportional to the surface area, and it is verified to be near optimal in practice. Furthermore, their experiments show that it is faster than previous methods.

Our novel data structure, the *Hierarchical RLE level set* (H-RLE level set) representation, presented in this article and previewed as the *Compact RLE Level Set*[1] in the sketch of Houston et al. [2005], combines ideas from both the RLE sparse level set and the DT-Grid to obtain a general scalar field representation and set of algorithms that make the H-RLE level set tailored for computer graphics and level set applications. Contrary to previous work, it offers both a high level of flexibility and good performance with respect to both memory consumption and running times.

The next section outlines the main contributions of this article, including the novel features of the H-RLE level set, and points out the key differences between this work and the DT-Grid.[2]

3. CONTRIBUTIONS

Our H-RLE level set representation combines ideas from the RLE sparse level set [Houston et al. 2004] and the DT-Grid [Nielsen and Museth 2006]. In particular, our H-RLE level set uses the idea of hierarchical compression from the DT-Grid. Instead of applying it to a compressed-row storage scheme, we integrate it with the explicit RLE compression scheme introduced with the RLE sparse level set. More specifically, the H-RLE level set differs from the DT-Grid via the following four contributions:

—*Run codes.* As will be explained in Section 4, the H-RLE data structure employs RLE to encode the domain into a series of runs, each associated with a specific *run code*. The run code categorizes a run as being either defined (inside the narrowband), or undefined. An undefined run can be identified by any of the possible run codes[3] and hence offers great flexibility in encoding undefined regions. This feature is not shared by the DT-Grid, which only stores what corresponds to defined runs and distinguishes inside regions from outside regions based solely on the sign of adjacent level set values. One of the unique benefits of the H-RLE data structure implied by using run codes and having an explicit bounding box is that it allows us to easily store, access, and manipulate *unenclosed level sets* (see Section 6.5 and Figure 9 for more details) and in general unenclosed scalar or vector fields. (In contrast, the DT-Grid requires that the narrowband forms a closed domain.) The usability of this feature shows up frequently in computer graphics as it can be used to perform collision tests, do CSG operations, and so on with open level sets that are in fact part of a more complex level set model, hence speeding up these applications. Finally, note that a DT-Grid can be represented by an H-RLE level set, but that the opposite is not the case.

—*Flexible encoding.* The H-RLE level set is able to both represent narrowbands of varying width (i.e., the width of the narrowbands varies along the surface of an implicitly represented object) as well as nonsymmetric narrowbands (narrowbands that are wider on one side of the zero-set than the other). These two abilities are derived from the H-RLE grid's support for the encoding of arbitrary run configurations—the location and length of runs can be whatever one desires. We discuss our practical application of this feature to level set metamorphosis in Section 6.4.

[1]After submission of this sketch, we decided to change the name of the structure from the original Compact RLE level set name to the more descriptive Hierarchical RLE level set. We apologize for any confusion this may cause.

[2]Since [Houston et al. 2004] is an ACM SIGGRAPH sketch, it should be interpreted as a preview, and thus we will refrain for the remainder of this article from making comparisons between our current work and the RLE sparse level set.

[3]Any of the possible run codes, that is, within the bit-precision of the run code pointer, that is, $(2^{31} - 1)$.

—*Decoupling.* Contrary to the DT-Grid, the H-RLE data structure decouples the RLE-compressed grid structure from the array specifying the values in the narrowband. This can save storage (compared to the DT-Grid approach) and gives optimal flexibility in storing and combining several scalar fields as part of an augmented level set representation (see Section 6.1) using the same compressed H-RLE structure. It is also possible to not store an array of values at all, but simply use the H-RLE compression to efficiently store regions of constant values.

—*Explicit bounding box.* The H-RLE level set has an explicitly defined bounding box. In contrast, the DT-Grid only has the implicit bounding box of the defined narrowband. Because of this, the H-RLE level set allows for $O(1)$ culling of occlusion queries that fall outside of the explicit bounding box. We exploit this fast culling of occlusion queries for detecting collisions between our FEM tetramesh and our level set-based occlusions (see Section 6.5). Note that the existence of the explicit bounding box still allows for level set simulations to go "out-of-the-box" on the H-RLE level set, as the bounding box is recomputed dynamically by the dilation algorithm described in Section 5.

This article introduces several applications of the H-RLE level set in computer graphics and adds the following contributions to the new features outlined above:

—a robust and a fast technique for polygonal mesh scan conversion into our new level set data structure; both methods are memory-efficient, and the robust method handles both open and self-intersecting meshes with grace.

—a level set-based unified implicit object representation that provides a simplifying abstraction with wide applicability;

—an efficient RLE data structure for fluid simulation in which all computation and storage are restricted to the volume of the fluid, not to its bounding volume.

In addition, the H-RLE level set representation inherits the following features from combining the RLE sparse level set and the DT-Grid:

—fast, sequential traversal of the narrowband elements with $O(1)$ access time to elements within the finite difference stencil, thus leading to efficient level set propagation;

—rapid logarithmic time random access (and neighbor access) to any voxel within the bounding volume, thus leading to efficient ray tracing, collision detection, and other similar algorithms;

—both the encoding computational complexity and the overall memory consumption proportional to and scaling with the area of the geometric surface;

—compatibility with all existing mathematical finite difference schemes used to solve the level set partial differential equation on regular (i.e., equidistant) grids;

—"out-of-the-box" dynamic grid representation that frees surface deformations from dealing with any fixed boundaries of the computational domain, hence allowing surfaces to expand and contract freely without encountering boundaries and without any extra bookkeeping.

4. DATA STRUCTURE

Run-Length Encoding (RLE) is a popular lossless data compression algorithm based on the simple idea of replacing sequences (runs) of identical data values by a count number and a single value (run code). A sequence of level set values, $\phi_1, \ldots, \phi_n$, can be considered as a stream of data to which one can apply RLE. To facilitate efficient compression of a narrowband level set of width β, we introduce the following three run codes: *negative* (interior region with $\phi < -0.5\beta$), *positive* (exterior region with $\phi > 0.5\beta$), and *defined* (within the narrowband, $|\phi| \leq 0.5\beta$). Each continuous sequence of adjacent values not within the narrowband is compressed to just a single run, whereas values within the narrowband are stored

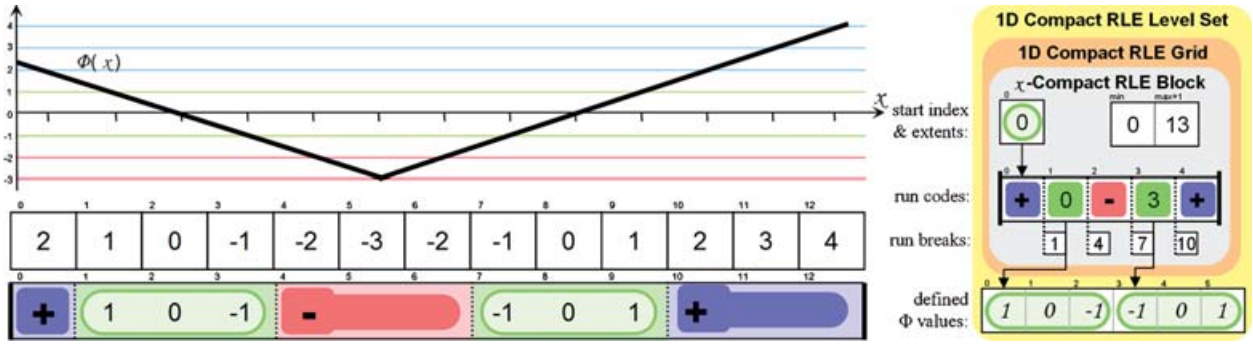

Fig. 1. Left: 1D RLE encoding of a level set function with two zero crossings. The width of the narrowband is $\beta = 3$ and corresponds to the region of the green guidelines. Right: The corresponding one-level hierarchical RLE level set data structure. Note the 1D RLE block illustrated by the gray box. See Section 4 for details.

explicitly. In our implementation, the *defined* run code for grid points inside the narrowband will be an index into a separate array storing the corresponding values of ϕ.

4.1 Taxonomy of the RLE Data Structure

As a prelude to the presentation of our dimensionally hierarchical data structure, it is convenient to introduce the following terms. A *run* is a sequence of connected values with the same *run code*. An *RLE segment* is a collection of adjacent runs corresponding to a single axis-aligned scan-line through the level set. An *RLE block* represents the encoding of a collection of segments, not necessarily immediately adjacent to each other, and serves as a fundamental building block in our hierarchical data structure. A *hierarchical RLE grid* is a recursive entity composed of n RLE blocks in R^n, and, finally, a *hierarchical RLE level set* is a hierarchical RLE grid together with the separate array of defined values (see Figure 1). Within the hierarchical RLE grid, it is convenient to think of the hierarchy of RLE blocks as beginning with the *top* RLE block and proceeding to *lower* RLE blocks until the *bottom* RLE block is encountered. In 3D, the top and bottom RLE blocks are, respectively, the z-axis and the x-axis RLE blocks.

Let us first consider the simple 1D level set function, $\phi(x)$, illustrated on the left in Figure 1. The rectangle at the bottom shows the RLE segment containing five (colored) runs corresponding to $\beta = 3$. The right-hand side of Figure 1 illustrates the corresponding hierarchical RLE level set and its two components: an RLE block that encodes the topology of the segment (gray), and an array with discrete values of $\phi(x)$ within the narrowband (bottom). This decoupling of values and topology is essential in our data structure as it facilitates great flexibility. In fact, this separation is somewhat analogous to the separation of topology and geometry common in mesh data structures.

Each RLE block consists of the following four separate arrays:

—*Start indices* for each segment are stored as pointers into the run-codes array (in 1D a single pointer since the block only has one segment).

—*Extents* of the absolute coordinates along the encoding axis are stored as a [min, max +1) pair.

—*Run codes* define how each segment is divided into different runs. *Negative* and *positive* run codes denote compressed runs while *defined* run codes are pointers into an associated defined data array or the start indices array of the next lower RLE block if it exists.

—*Run breaks* contain, in an order mirroring that of the run codes, the absolute coordinates along the encoding axis at which each run starts, except for the first run in each segment. The start coordinates of the first run of each segment are determined by the above-defined minimum extents.

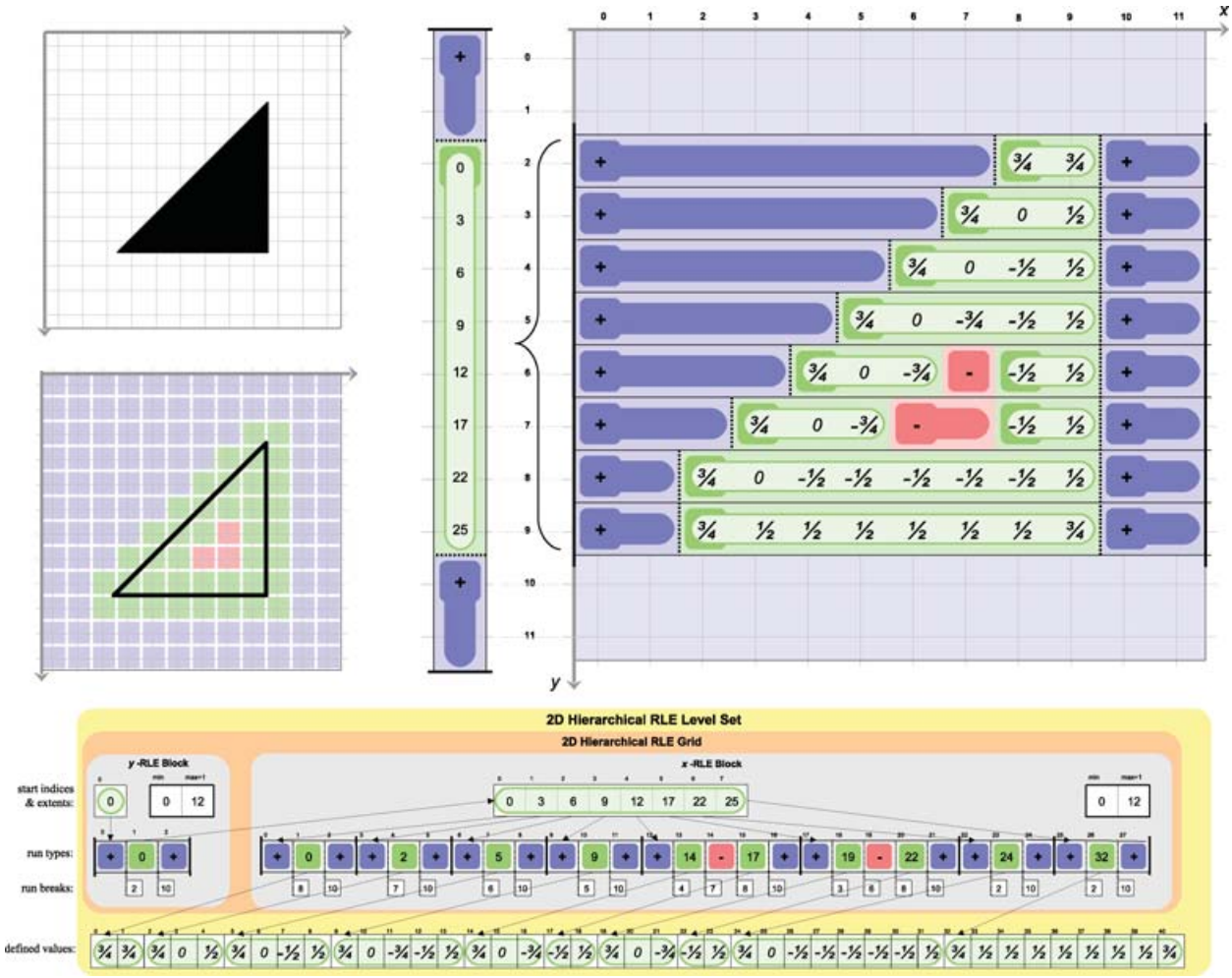

Fig. 2. Top left: A triangle shape overlaid on a coarse voxel grid. Middle left: The triangle interface is now indicated. The voxels are categorized based on a narrow bandwidth of Δx into narrowband (green), interior (red) and exterior (blue) voxels. Top right: A schematic representation of a 2D H-RLE level set. The top-level y-axis encoding is given in the 1×12 grid and the x-axis encoding is the 12×12 grid. Along the y-axis encoding, the ranges 0–1 and 10–11 are categorized as fully exterior, while the range 2–9 is defined. The defined values of the y-axis encoding indicate the offset of the corresponding x-axis encoding segment. The x-axis encoding contains no information for regions not encoded by the y-axis as defined. The values of the defined ranges of the x-axis encoding are the level set values of the narrowband. Bottom: The layout and contents of the H-RLE level set data structure corresponding to the schematic representation above. The defined values of the y-RLE grid's defined runs are stored in the x-RLE grid's start indices table. The defined values of the x-RLE grid's defined runs are stored in the decoupled defined values arrays. See Section 4 for further explanation.

Note that although the hierarchical RLE grid coincides with the RLE block in 1D this is not the case for higher dimensions (see Figure 2.)

4.2 The Hierarchical Data Structure

Our data structure generalizes to any dimensions through dimensionally hierarchical encoding [Nielsen and Museth 2006] based on the taxonomy introduced above. We stress that doing a hierarchical

encoding, as opposed to only encoding in a single direction (as in Houston et al. [2004]), ensures that the memory requirements of the data structure will be proportional to the area of the geometric surface. Let us consider the 2D example in the top of Figure 2 illustrating a 12×12 bounding box containing a subset of a level set function.

In the bottom of Figure 2 is the corresponding 2D H-RLE level set. It is composed of two RLE blocks related hierarchically, the top RLE block encoded along the y-axis and the bottom RLE block encoded along the x-axis. First, each linear traversal of the 2D level set function along the x-axis is encoded resulting in the x-block shown in the bottom-right part of Figure 2. Note that traversals resulting in a single nondefined run are not stored but rather offloaded to the next higher RLE block. The higher RLE block is encoded along the y-axis leading to the y-RLE block shown in the bottom-left part of Figure 2.

As shown in Figure 2 (bottom), the RLE blocks store the run-codes sequentially. Run-codes in the x-RLE block reference into the single array of defined values and indicate where that particular run of defined values begins. The run-codes of defined runs in the y-RLE block reference into the x-RLE block start index array. This is related to the fact that the lowest level RLE block encodes level set values, whereas all higher RLE blocks are actually encoding the results of the previous RLE encoding.

The storage format for the H-RLE level set readily generalizes to higher dimensions, in which case it stores one RLE block for each axial direction as well as an array of defined values. As a result, the storage requirement of our data structure is $O(D)$, where D is the number of defined voxels inside the narrowband. We point out that the bounding box associated with the H-RLE block extents is very useful for many graphics applications and is fully dynamic. In fact, the rebuild algorithm described in Section 5 applies a dilation algorithm that dynamically expands or shrinks this bounding box as the interface moves. We finally note that our data structure is versatile and allows for the representation of both closed and open surfaces, which has important practical applications for graphics, such as collision detection with fluids and geometric modeling.

5. ALGORITHMS

In this section we describe a set of efficient algorithms for the use of our H-RLE level set in graphics applications. To ease the description, we introduce the following notational conventions:

—n: the side length in voxels of a dense cubical bounding volume,
—N: the total number of voxels in the dense volume,
—D: the total count of defined voxels within the narrowband,
—s_i: the number of segment indices,
—r_i: the number of runs, and
—d_i: the number of defined values in the ith dimension RLE block.

5.1 Random Access

A fast random access operation is essential in many graphics applications, such as ray tracing, collision detection, signed distance function generation using fast marching [Sethian 1996], and so on. The random access algorithm of the H-RLE level set operates recursively:

(1) The random access operator begins as a procedural call on the top RLE block with two parameters: A vector of query coordinates, $Q = (q_1, \ldots, q_n)$, and a segment index, s_n, initialized to zero. The vector of query coordinates is split to isolate the first query coordinate, q_n.
(2) From the given segment index, s_i, one can determine the indices of both the first run and the first run break as well as the total number of runs in the segment (the *segment length*). The run code of the run containing q_i can be determined via a binary search within the run breaks array in the

region of the current segment (defined by the first run, first run break and the segment length). If the run code is *negative* or *positive*, then return the corresponding run code value. Otherwise, compute the *defined data index* by adding together the run code with the offset of q_i from the start coordinate of the run.

(3) If a lower RLE block exists, then this procedure is recursively called on it (go to Step 2 above), using the remaining vector of query coordinates and the defined data index as the respective values of its vector of query coordinates and segment index parameters. If no lower RLE block exists, return the defined data index or the corresponding value in the associated defined data array.

The computational complexity of the random access algorithm is as follows: step 1 above takes constant time. Step 2 takes time logarithmic in the number of runs (as opposed to the number of elements *within* the runs) in a single segment in an RLE block. Note that a binary search is possible since the run breaks are sorted in increasing order within each segment as part of the encoding. Step 3 calls recursively on a lower level RLE block of which there are at most k, where k is the dimension of the H-RLE grid. The total time required amounts to $O(\sum_{i=1..k} \log_2 r_{s_i})$, where r_{s_i} is the number of runs in the segment identified by segment index s_i and contained in the ith-dimension RLE block.

Also note that the time complexity depends only on the number of runs within a total of k segments in the compression—one segment at each level of the encoding. Hence the efficiency of the random access operation is not affected, even if a model projects to a large lower-dimensional area. In the worst case, the number of runs will be equal to the number of elements in the grid and, in that case, the random access operation will be logarithmic in the total number of elements stored in the grid, but in all practical cases it will be better.

5.2 Neighbor Access

Efficient neighbor search is of great importance in, for example, the fast marching method. When doing neighbor search in an H-RLE, the structure of the encoding can be exploited to make the neighbor search more efficient than its random access alternative. Imagine doing a neighbor search in the X-direction in the H-RLE depicted in Figure 2. In this case, neighbor search can be done in constant time, since the X direction is at the lowest level of compression. Finding a neighbor in the Y-direction would require a binary search (as described in step 2 of the random access operation above) in the neighboring segment in the Y-direction ($Y \pm 1$). In general, the time complexity of a neighbor search in the jth direction (where X is the first direction, Y is the second, and so on) is $O(1 + \sum_{i=1..(j-1)} \log_2 r_{s_i})$, where again r_{s_i} is the number of runs in the segment identified by segment index s_i and contained in the ith-dimension RLE block.

5.3 Sequential Access

Most existing surface deformation schemes access the level set sequentially, and the H-RLE level set has been specifically optimized for this. This algorithm also operates recursively.

The sequential access iterator also begins as a procedure call on the top RLE block with two parameters: a segment index initialized to zero, and an empty vector of parent coordinates. The procedure begins by traversing the runs of the specified segment. Nondefined runs are ignored when encountered. When a defined run is encountered, both its length and its start coordinate along the current encoding axis are determined. Then an iteration along the individual coordinates of this run commences. At each coordinate, the defined data index is computed by adding together the run code with the current coordinate's offset from the start of the run. Also, a vector of current coordinates is created by concatenating the input vector of parent coordinates with the current coordinate. If the RLE block is the bottommost, then append both the current coordinate vector and the defined data index

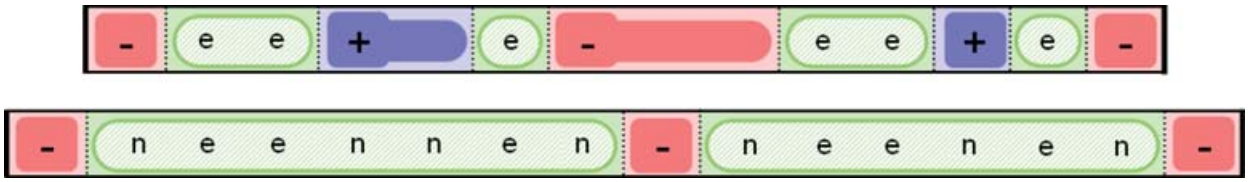

Fig. 3. Dilation of an RLE segment by one grid point. Top: original segment with existing defined grid points labeled e. Bottom: dilated segment with new grid points labeled **n**.

to the resulting enumeration. Otherwise, the above described procedure is called recursively on the next lower RLE block using the defined data index and the constructed current coordinate vector as the respective values of its segment index and parent coordinates parameters. This sequential access iterator traverses each defined run in each of the RLE blocks in addition to enumerating each of the defined data indices. The resulting computational complexity is $O(D)$ and hence $O(1)$ per defined data element.

In order to obtain constant time access to all elements within the finite difference stencils typically used in level set deformations, one can iterate an entire stencil of iterators over the narrowband. It is possible to exploit the movement of the center of the stencil to further improve the practical access time. All iterators within the stencil pass over the grid once, so—on average—access to elements within the stencil is $O(1)$ if the entire grid is visited.

5.4 Rebuilding the Hierarchical RLE Level Set

This ensures that it contains, at all time steps, the grid points defining the zero-level set and a surrounding band wide enough to support the finite difference stencils used in the level set computations. Logically, rebuilding the H-RLE level set is comprised of two steps: first, grid points that are too far from the interface are removed. Next, the remaining subset of the original H-RLE level set is *dilated* by effectively adding grid points that pass under a hypercube-shaped stencil being iterated over the subset. Any existing method for reinitializing a level set function can then be used to reinitialize the resulting H-RLE level set to a signed distance function. While the dilation may seem trivial, we note that a direct implementation is very inefficient and slow. Hence, we have developed a much faster algorithm that dilates a narrowband in $O(D)$ time and space. This algorithm is recursive in the dimensionality of the H-RLE level set and takes advantage of its hierarchical RLE definition.

The dilation of a 1D H-RLE level set simply corresponds to dilating the corresponding 1D RLE segment as illustrated in Figure 3. It consists of two steps: first, each defined run in the segment is dilated independently. Next, a new RLE segment is constructed by forming the union of each dilated run and keeping the remaining inside/outside runs consistent with the original segment.

The dilation algorithm of a 2D H-RLE level set operates in each coordinate direction independently and proceeds as follows:

(1) Dilate the y-axis RLE block as a 1D RLE segment. The y-axis RLE block corresponds to the projection of the 2D H-RLE level set onto the y-axis, and the dilated y-axis RLE block corresponds to the projection of the dilated 2D H-RLE level set.

(2) To dilate the x-axis RLE block, the defined elements of the dilated y-axis RLE block are processed iteratively: let the current element have y-coordinate y_m. The corresponding dilated x-axis RLE segment is computed by first dilating all defined x-axis RLE segments in the original x-axis RLE block with y coordinate in the range $y_m - n$ to $y_m + n$, where n is the number of dilation grid points. Next, the union is taken of these individually dilated x-axis RLE segments.

```
typedef struct {
    HRLEGrid    interfaceHRLEGrid       /* required and shared among all allocated value arrays */
    float[]     interfaceLevelSetValues /* required, remember this is an augmented _level set_ */
    Vec3[]      interfaceVelocities     /* optional: centered velocites useful for representing occlusions */
    RGB[]       interfaceColors         /* optional: can be used for "mixing" differently colored liquids */
    UVW[]       interfaceTextureCoords  /* optional: used to create the Tar Monster effect */
} AugmentedHRLELevelSet
```

Fig. 4. A simple *augmented level set* data structure supporting three optional fields in addition to the shared H-RLE grid and required level set values. The augmented level set structure, a unified implicit object representation, greatly simplifies the design and implementation of algorithms that involve both level sets and other related data fields in areas such as level set-based fluid simulation, object deformation, collision detection and especially rendering.

(3) Allocate storage for the decoupled value array of the dilated 2D H-RLE level set. In a single pass, the value of grid points also existing in the original 2D H-RLE level set can be set to their original value and the value of grid points added as a result of the dilation can be initialized, if necessary.

We finally note that the dilation algorithm readily generalizes to higher dimensions.

6. APPLICATIONS

We now present a series of applications in order to fully demonstrate the usefulness of the techniques we have outlined. These examples include mesh conversion, rendering, morphing, and simulation.

6.1 Unified Implicit Object Representation

It is appropriate to begin the applications section by describing the compatibility of the H-RLE level set with the *augmented level set* (Figure 4), our unified implicit object representation. As the name augmented level set suggests, it is a level set (not necessarily an H-RLE level set) defining a implicit surface that is augmented with a set of subordinate fields of various types. The primary benefit of the augmented level set is that it provides an abstraction layer on top of many common level set operations such that the subordinate fields can be automatically handled appropriately. The H-RLE level set is well suited to the augmented level set because of the decoupled nature of the H-RLE grid and the array storing the level set distance values. It is a trivial matter to associate other arrays representing other data fields with the H-RLE grid of the level set. Besides efficient storage, this configuration also allows efficient access to the subordinate fields since the defined data index, resulting from an iteration or random access, can be reused to retrieve the corresponding value or values of any of the other fields. In addition to subordinate fields that align exactly with the level set grid structure, our augmented RLE level set can handle, with some loss of storage and access efficiency, fields with more complex lattice structures such as the staggered velocities of the MAC-grid (see Foster and Fedkiw [2001]) or the tensor fields required for viscoelastic fluids (see Goktekin et al. [2004].)

We have applied the augmented level set representation to our mesh-to-level set scan converter (to optionally capture texture coordinates, surface velocities and arbitrary map channels), constructive solid geometry (such as the union, subtraction, and intersection operators), advection (we use semi-Lagrangian methods to advect the subordinate fields where necessary), outward extrapolation of quantities from the level set interface (which we use for free surface fluid velocity extrapolation, and occlusion constrained velocity extrapolation), and rendering (where shading often requires texture coordinates or color data and motion blur requires the surface velocities). In addition, in our simulation systems (see Figures 10, 11 and 13) we represent both the fluid and the occlusions as augmented level sets—a choice which significantly reduced both the design and implementation complexity.

6.2 Mesh To Level Set Conversion

6.2.1 *Rapid Scan Conversion of Closed Meshes.* The scan conversion technique of Mauch [2000] for computing the signed distance transform to a closed and non-self-intersecting mesh has gained wide acceptance. It sequentially scan converts a set of characteristic polyhedra, resembling a set of widened Voronoi cells, derived from the mesh. In theory it has a near optimal speed, $O(cD + F)$ (when ignoring the $O(n^3)$ initialization time), where D is the number of defined voxels, F is the number of faces and c is a constant that depends on the overlap of the characteristic polyhedra (see Mauch [2000]). In practice the method is very fast and compares favorably with the fast marching method [Sethian 1996]. Furthermore, the result is accurate to within machine precision. The method as originally proposed has memory requirements that are $O(n^3 + F)$. Here we describe how to augment the algorithm to only use $O(D + F)$ memory, and how to use it in the context of H-RLE level set representations. Furthermore, this new technique has a total time complexity of $O(cD + F)$, whereas the original formulation has a time complexity $O(n^3 + F)$ when including the initialization time. As a prelude to introducing our H-RLE scan converter, let us first consider a modified "dense-grid" scan converter of a triangular mesh into a clamped signed distance function ($|\phi| \leq \beta/2$):

(1) Partition the bounding volume of the mesh into $P \times Q \times R$ (nonoverlapping) rectangular axis-aligned subvolumes and initialize the values in each of these to $\beta/2$.
(2) Partition the mesh into $P \times Q \times R$ sub-meshes, where the (p, q, r)th submesh contains all polygons of the original mesh that lie within a distance of $\beta/2$ from the (p, q, r)th subvolume.
(3) Scan convert each submesh into the corresponding subvolume.

Our approach is similar to the one above, except that, in order to ensure the $O(cD+F)$ time-complexity and the $O(D+F)$ memory complexity, only one representative subvolume of size $O(D)$ is kept in memory. In addition, this sub-volume is only initialized once. When scan converting the (p, q, r)th submesh into the representative subvolume, the coordinates of each grid point—for which a distance is computed—are stored in a data structure that compresses the coordinates on-the-fly. When the submesh has been scan converted, the values of the grid points that were updated are saved and the corresponding grid points in the representative subvolume reinitialized to $\beta/2$. This procedure is then repeated for the remaining submeshes, after which three linear time bucket sorts are performed on the compressed coordinates of the recorded grid points. This ensures they are lexicographically ordered. Finally, the sorted grid points are stored in an H-RLE level set. The outlined scan conversion algorithm retains the speed of the original method. In practice, the relative time usage of the partitioning and bucket sorting steps is negligible, for example, scan converting the Happy Buddha with a tight-fitting bounding box into a $600 \times 1445 \times 600$ volume with $\beta = 6$ takes only 59 s on a 1.5-GHz Pentium Mobile.

6.2.2 *Disambiguating Imperfectly Closed Meshes.* To robustly resolve meshes with holes or self-intersections, we adopt the efficient ray casting-based visibility sampling approach of Houston et al. [2003]. Our method more closely resembles the ray casting parity-counting method of Yngve and Turk [2002] rather than the local surface reconstruction method of Ju [2004] or the methods cited therein. Unlike Yngve and Turk [2002], we do not use parity-counting to determine interior regions but rather we integrate in a additive fashion surface orientation (i.e., normal) information. Our approach is adapted to the way that CG artists create their models. CG artists do not think about parity- counts, or whether their models are solid, but rather concentrate on how good a model appears in the viewport of their modeling tool. We thus embraced a visibility-based approach to determining surfaces, interiors, and exteriors that attempts to mimick the quick, context-based, "best-fit" judgments of artists.

From each voxel, we cast a number of rays, and used the normals of the nearest ray-mesh intersections to determine whether the voxel was inside the mesh. For imperfectly closed geometry, some voxels

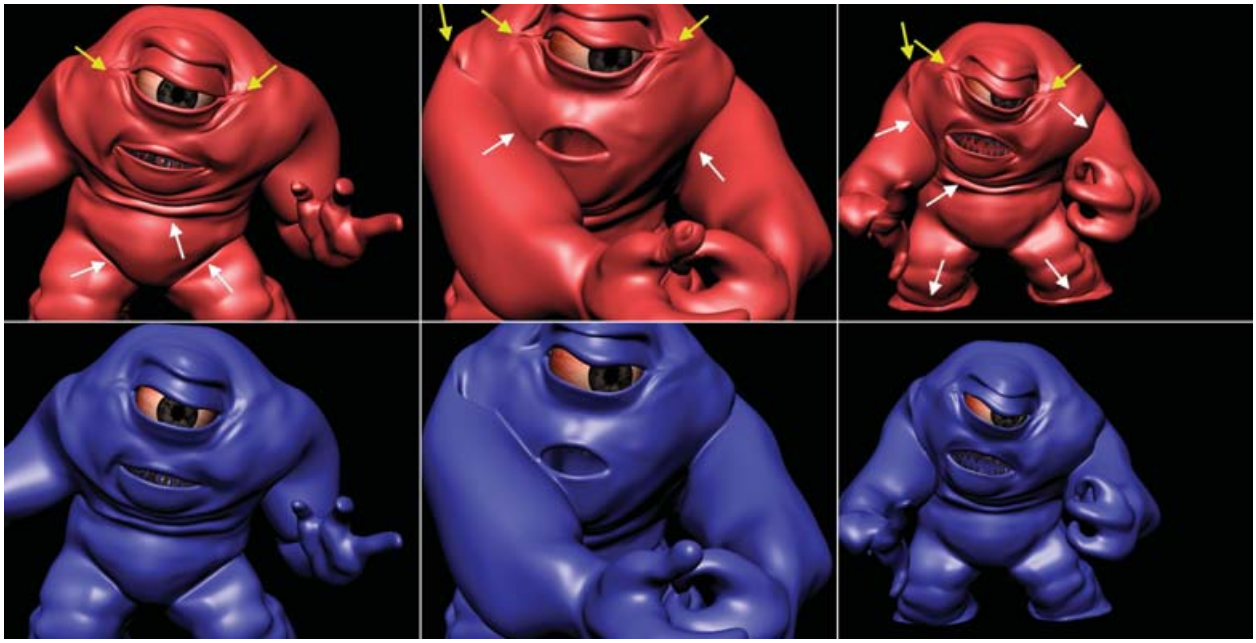

Fig. 5. A demonstration of our robust scan conversion algorithm described in Section 6.2. Top: The original *Tar Monster* character mesh (red) was often ill-posed. The white arrows denote regions of self-intersections. The yellow arrows point at crevasses or sharp ridges. Bottom: The RLE level sets (blue) resulting from the scan conversion of the *Tar Monster*'s 69,120 faces and 69,153 vertices. The original conversion, at a resolution of $624 \times 554 \times 488$, was able to resolve the mesh and its regions of self-intersections into a closed level set. The subsequent down-sampling, to a resolution of $312 \times 277 \times 244$, reduced the temporal aliasing artifacts caused by the original mesh's moving sharp ridges and crevasses. The whole process of scan conversion, down-sampling, and subsequent rendering took an average of 7 min frame.

will return contradictory results; we therefore employed a "majority wins" rule to decide the sign of the voxel value. Each ray also provides the distance from the voxel to the geometry surface in the direction it is cast. This information was used to determine the magnitude of the voxel value, by selecting the minimum magnitude distance from the set of rays corresponding to the "winning" sign. In our implementation, we employed a structured sampling approach that utilizes the results from a single ray cast to provide sample results for all voxels that it intersects.

Employing the above algorithm directly may result in undefined regions of opposite sign being immediately adjacent, with no defined narrowband in between. This can occur in regions where a mesh has a significant hole or opening. In these situations, we simply assumed that the zero level set passes between the two adjacent signed regions and created an initial narrowband having a width of one or two voxels at this location. Finally, in order to produce a smooth signed distance field with a uniform defined narrowband width, we dilated the defined narrowband region as necessary and performed a reinitialization.

This method will always extract accurately the implicit surface of a closed mesh in addition to producing a guaranteed closed implicit surface for imperfectly closed or self-intersecting meshes.

Both the spider model in Figure 6 and the character meshes used in Figures 5, 13 and 14 were produced using this algorithm. In the case of the spider model, 724 hand-crafted trimesh components, a large number of which were nonclosed or highly self-intersecting, were converted at a bounding resolution of $1691 \times 1223 \times 839$ (1.74 billion voxel volume) in 7.5 h on a single-processor Athlon 1600

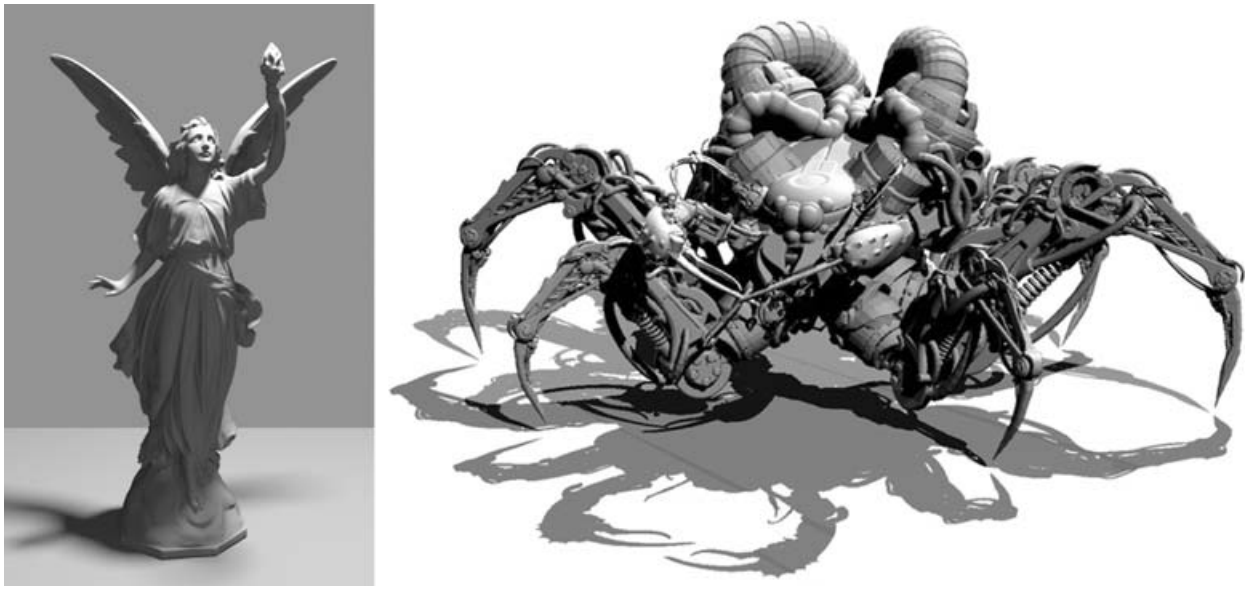

Fig. 6. Two large H-RLE level sets produced by the scan converters of Section 6.2 and rendered via direct ray tracing as described in Section 6.3. Left: The Lucy Statue level set was rapidly scan converted from a 14-M vertex and 23-M triangle mesh retrieved from the Stanford 3D Scanning Repository. The resulting level set had a minimum bounding voxel resolution of $3000 \times 1726 \times 5144$ (26-B voxels) and consumed 738 MB of memory. Right: The Spider Mech level set was robustly derived from a Frantic Films 0.5-M triangle mesh composed of 724 unclosed components. The resulting level set had a minimum bounding voxel resolution of $1691 \times 1223 \times 839$ (1.74-B voxels) and consumed 229.6 MB of memory. The defined interface contained 44.7 million voxels.

computer. The resulting defined interface consisted of 44.7 million voxels. The complete H-RLE level set representation of the spider model required only 229.6 MB.

6.2.3 *Extra Fields.* Often applications require additional properties of a mesh to be captured. For example, interface velocities are required when integrating level set-based occlusions into soft-body (see Section 6.6) or fluid simulations (see Section 6.7.) Also, the creation of the Tar Monster effect (see Section 6.5) required capturing both the grayscale vertex channel and the UVW texture coordinates of the artist animated character mesh.

We captured additional properties of a source mesh, such as the ones listed above, as a postprocess to the mesh to H-RLE level set scan conversion. First, a new defined value array, appropriate for the new fields data type, was allocated and paired, as an augmented field, with with H-RLE grid of the H-RLE level set resulting from the scan conversion. Then the narrowband was traversed. For each voxel immediately adjacent to the mesh surface (i.e., $\phi_i \leq \Delta c$, where c is the voxel width), the relevant property of the mesh at its surface location closest to the current voxel was determined. If the relevant property of the mesh was only defined on the vertices, the subface value could be calculated via Barycentric interpolation. After completion of the narrowband traversal, we then extrapolate newly set quantities off the immediate interface voxels into the remainder of the narrowband.

6.2.4 *Antialiasing.* Many meshes, especially those of traditional animated characters, often have subvoxel features such as sharp folds or edges. When animated across a regular grid of insufficient resolution, these features can result in visually displeasing aliasing artifacts. To remove these artifacts without artist intervention, a basic supersampling approach can be employed. First, the mesh is scan

converted to an H-RLE level set at twice the desired resolution and then subsequently down-sampled with a low-pass filter to remove the smaller features that otherwise would cause aliasing (see Figure 5).

6.3 Direct Ray Tracing

When rendering the surface embedded in the H-RLE level set, we prefer to use direct volumetric ray tracing over mesh extraction. Standard ray tracing techniques are directly applicable, with two major advantages:

—The H-RLE grid is not only a data representation, but an acceleration structure as well; it offers fast random access without the additional build (and memory consumption) of external acceleration structures (e.g., Kd-tree, octree).

—Voxel values are samples of a signed distance function that allows for efficient ray leaping.

To abstract the ray tracing algorithm from the details of the defined and undefined regions, we render the results of all (random access) queries as scalar level set values, which are then interpreted without additional discrimination to extract the surface profile. If a query is to a defined region, the actual level set value is used. When a query is to an undefined region, an approximate level set value is constructed by multiplying the sign of the undefined region by the narrowband's maximum allowable level set value—usually half of the interface bandwidth used during dilation.[4]

Our ray tracing algorithm achieves its efficiency by adaptive stepping based on the magnitude of the level set values encountered along the ray's path. Given an accurate signed distance field, it is possible to take discrete steps along the ray's path as large as the magnitude of the level set values encountered until either a voxel immediately adjacent to the interface is discovered (which can be inferred from the level set value being less than the voxel width) or the ray exits the bounding domain of the level set. When near the interface, an analytic solution to the ray-interface crossing is found with a cubic polynomial root finder. The normal is computed as the derivative of the cubic interpolation function. If no surface is found in the current narrowband region, ray tracing can once again begin taking large steps through the $+/-$ regions.

A crude efficiency estimate can be written based on random access to voxel values. For a grid of width n, with narrowband width β, approximately n/β such operations are needed per ray. This makes tracing of R rays $O((n/\beta)R \log_2 r)$. Memory footprint is $O(C_r + D)$ where C_r is the ray tracer and D is the number of defined voxels in the grid.

In practice, this allows us to efficiently ray trace very large models on ordinary desktop machines. We have scanned converted and ray traced models with effective grid resolutions up to $5000 \times 3000 \times 3000$. This corresponds to a full grid with more than 45 billion voxels, but required less than 1 GB of RAM. For examples, see Figures 6, 7 and 8.

6.4 Deformation of High-Resolution Interfaces

6.4.1 *Shape Metamorphosis.* Morphological operations with level sets have previously been restricted to relatively low-resolution models (see Breen and Whitaker [2001].) This is primarily due to the unfavorable memory requirements of the applied level set schemes. However, using our scalable H-RLE level set, we can easily perform high-resolution morphs between geometric models with fine surface details.

[4]These approximate undefined region level set values, while useful for both this ray tracing algorithm and other applications, are not useful for calculating gradients since they are constant within contiguous undefined regions and cannot properly account for situations where the narrowband is of widely varying bandwidth (such as the situation described in Section 6.4.)

Fig. 7. A level set shape metamorphosis. The human figure (left), courtesy of Frantic Films, has a bounding box of $512 \times 797 \times 145$. The Thai Statuette (left), retrieved from the Stanford 3D Scanning Repository, has a bounding box $1000 \times 1676 \times 865$.

Fig. 8. Extreme level set deformation of a high-resolution model exhibiting very thin walls. The Stanford bunny is advected by a divergence-free and periodic vector field [LeVeque 1996; Enright et al. 2002]. The maximum bounding grid is 1024^3, but memory requirements never exceed 97 MB. From left to right the simulation times are (in units of the period) 0, 1/15, 1/2, 1.

The morphing process from *source* to *target* can elegantly be achieved by a propagation of the *source* level set according to

$$\frac{\partial \phi_{source}}{\partial t} = (\phi_{source} - \phi_{target})|\nabla \phi_{source}|. \tag{1}$$

This PDE is propagated in time until steady state, when the two level sets completely overlap. Figure 7 shows the result of morphing a human model $512 \times 797 \times 145$ into a statuette at resolution $1000 \times 1676 \times 865$. This morph uses an H-RLE level set to represent both the source and the target distance field, resulting in a clamped speed function. Since the target level set is static, it could also be encoded by the adaptive distance field structure of Frisken et al. [2000].

Alternatively, it is possible to exploit the H-RLE level set's flexible encoding to optimize morphing between objects with similar profiles. This avoids the "speed limit" effect, visible in Figure 7, for morphs between level sets whose defined narrowbands do not overlap. If the gaps between the narrowbands are not prohibitively large, one can adjust the narrowbands via reencoding and subsequent reinitialization, of one or both level sets, to ensure complete overlap. This is possible since the H-RLE level set allows, unlike the DT-Grid, for variable and nonsymmetric narrowbandwidths. Once the region through which the morph will occur is defined, and the newly defined voxels are reinitialized to proper signed distance values, the morph can proceed without any "speed limiting" clamping artifacts. This method of

reencoding the source and target level sets to ensure their respective narrowbands overlap was employed twice in the final cut of *Scooby Doo 2*.

6.4.2 *3D Deformation Test.* As a benchmark test with dramatic surface deformations, we advected the Stanford bunny in the following time-dependent velocity field (from LeVeque et al. [1996] and Enright et al. [2002]):

$$\begin{aligned}\mathbf{v}_t(x, y, z) &= (2\sin(\pi t)\sin^2(\pi x)\sin(2\pi y)\sin(2\pi z) \\ &\quad - \sin(\pi t)\sin(2\pi x)\sin^2(\pi y)\sin(2\pi z) \\ &\quad - \sin(\pi t)\sin(2\pi x)\sin(2\pi y)\sin^2(\pi z)). \end{aligned} \tag{2}$$

Since it is periodic and divergence free (i.e., without sources or sinks), a geometric surface should return to its original shape after one period. However, very high grid resolutions are required to faithfully capture the thin level set surface resulting from this deformation. Figure 8 shows results from a simulation on an effective bounding grid of 1024^3 using a third order accurate TVD Runge-Kutta [Shu and Osher 1988] for time integration and a fifth order accurate HJ-WENO [Liu et al. 1994] for space discretization. Using our compact scalable data structure, this high resolution was easily achieved with a dynamic memory footprint never exceeding 97 MB.

6.5 Collision Detection

Guendelman et al. [2003] described a rigid body system that employs a dual level set-triangle mesh representation. This dual representation allows for quick collision detection by querying the level set. The same collision detection paradigm is also encountered in the cloth simulations of Bridson et al. [2003] and the finite element simulations of Irving et al. [2004]. Guendelman et al. [2003] recommended using octrees where memory is an issue, and dense level sets otherwise due to their faster access speeds. Our experience using RLE level sets for collision detection, as we relate below, indicate that they are a preferrable alternative to octree level sets when working with large and detailed occlusions.

For these collision detection tests, we used the RLE sparse level set of Houston et al. [2004]. The RLE sparse level set can be understood as a variant of the H-RLE level set in which the recursive encoding is halted after a single dimension. This choice offers faster random access times by eliminating two of the three binary searches, at the cost of more memory. The tradeoff between space and time for a particular problem can thus dictate the level of recursion that is appropriate.

In situations where large occlusions only have portions of their surface in close proximity to the body under simulation, it is advantageous to clip the occlusions to only these simulation-relevant regions, especially if the whole occlusion object requires significant memory or requires per-frame scan conversion because of its dynamic properties. When clipping the level sets of occlusion objects, the RLE level set's ability to store unenclosed level set domains (which is not shared by the DT-Grid) significantly reduces memory consumption (see Figure 9).

Additionally, both the H-RLE level set and the RLE sparse level set allow $O(1)$ culling of occlusion queries that fall outside of the structures' explicit bounding boxes. Similar occlusion queries into the DT-Grid, because it does not contain a bounding box, must rely on the logarithmic random access operation.

Figure 10 shows a finite element sphere going through a set of gears represented as RLE sparse level sets. This simulation was created using the method of Irving et al. [2004].

6.6 Liquid Simulation

Using the particle level set method-(PLSM-) based fluid simulation system presented in Enright et al. [2002] as a theoretical foundation, we have created our own fully sparse RLE-based system. Our RLE-based fluid simulator employs the RLE structure for both the fluid and occlusion surfaces (represented

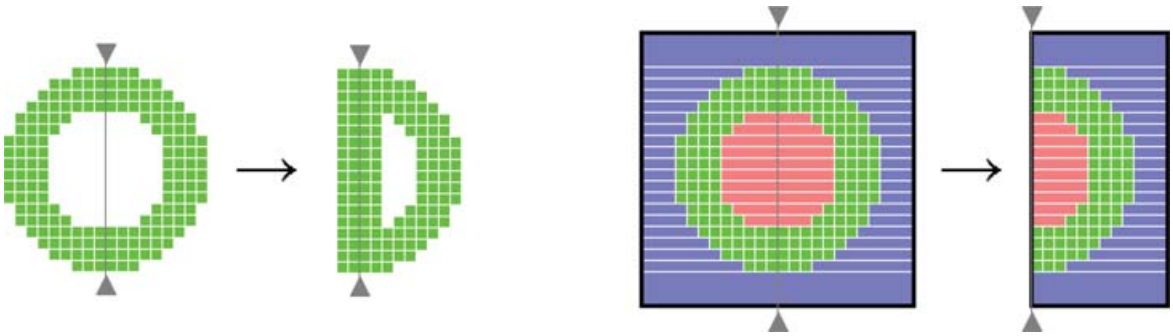

Fig. 9. Left: A sphere DT-Grid level set is depicted before and after clipping by a plane oriented to the right and aligned along the vertical axis. Note the need to allocate additional defined (green) voxels along the vertical axis in order to capture the effect of the clipping plane. Right: An equivalent sphere H-RLE level set is depicted before and after clipping by a similarly oriented plane along the same vertical axis. The resulting level set does not require the addition of any newly allocated defined (green) voxels because it is able to represent "unenclosed" level sets via the combination of its bounding box and its explicit categorization of non-narrowband regions as either positive or negative.

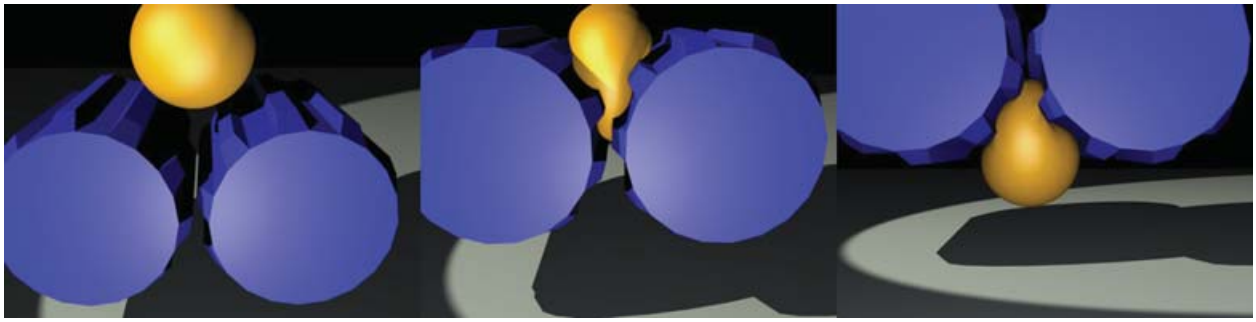

Fig. 10. A sequence of frames from a FEM soft body simulation of a deformable sphere (orange) interacting with two animated hard-body gears (blue.) The collision detection between the deformable sphere and the gears was performed using an augmented RLE level set. The augmented level set (which included a velocity field to capture the rotational motion of the gears) was generated from the gear meshes only within the immediate domain of the deformable sphere.

as level sets), the interior physical properties of the fluid (such as velocity and pressure), and the properties of the occlusions (such as slip coefficients and surface velocities). In this way, the memory and computational requirements of our RLE fluid simulation scale proportional to the volume of the fluid even though we maintain a regular grid structure. This is in contrast to the fixed uniform grids of [Stam 1999; Foster and Fedkiw 2001; Enright et al. 2002], the dynamic uniform grids of Shah et al. [2004] and Rasmussen et al. [2004], in which the cost scales with the bounding volume of the simulation grid, and the scalable, but nonregular grid fluid simulation method of Losasso et al. [2004]. Additionally, our RLE-based fluid simulator, because of the H-RLE level set's "out-of-box" nature, does not limit the domain of the fluid in terms of bounding volume or location. As best as we can determine, the scalable octree level set method described in Losasso et al. [2004] is still limited by the bounds of the root octree node. And, even though the methods of Shah et al. [2004] and Rasmussen et al. [2004] do permit something similar to our "out-of-box" feature via the ability to translate and resize the simulation domain, both of them still suffer from the fundamental volume limitations of a bounding uniform grid. Using our method, a moving, resizing, and unbounded grid is an inherent element of the data structure, and requires no additional code.

Our method involves the following:

—*Fluid surface.* We employ the RLE level set to represent the deformable fluid surface. Computational and memory requirements of this representation scale with the area of the fluid surface. In some situations, we have further augmented the level set of the fluid surface with auxiliary fields storing for example texture coordinates or RGB color values. The fluid surface is rendered using direct ray tracing, or if polygon representations of the fluid level set are required by third-party renderers or other tools, we use the adaptive polygonization method of de Araujo and Jorg [2004].

—*Fluid volume.* The properties of the fluid volume, such as those of the standard MAC grid, are also stored in a sparse RLE representation. Centered, per-voxel quantities such as fluid pressure, external forces, or RGB color values all share a single RLE grid that is defined across both the narrowband of the fluid surface and the interior volume of the fluid. The RLE MAC grid, used to store the fluid velocities, is constructed from three appropriately offset RLE scalar fields, one field for each of the three orthogonal velocity components. Like the RLE grid for fluid pressure, the RLE scalar fields of the MAC grid are defined throughout the fluid volume in addition to being defined in the surface narrowband region. The computational complexity and memory usage of the fluid simulation based on these RLE representations are proportional to the volume of the fluid.

—*Occlusions.* For occlusions, we use an augmented level set representation that captures the occlusion surface as well as the instantaneous surface velocities in an auxiliary RLE velocity field. The occlusions are integrated into the fluid simulation using the *constrained velocity extrapolation* approach introduced in Houston et al. [2003] and further developed in Rasmussen et al. [2004]. In addition, the use of "unenclosed" level sets allowed us in many cases to store only the regions of occlusions in close proximity to the fluid, reducing the memory overhead and processing time required by large occlusions.

—*Linear systems.* Two steps of the stable fluids algorithm of Stam [1999] require the solution of large sparse systems of linear equations: The viscosity diffusion step and the pressure projection step. Our approach is premised on the three following attributes of the linear system in our context: First, as a result of our uniform 3D simulation grid, the coefficient matrix $\mathbf{A}$ is both symmetric and heptadiagonal (i.e., it contains just six nonzero off-center diagonals.) Second, directly mapping from our uniform simulation grid to the system of linear equations results in a large nullspace—the pressure in voxels outside of the fluid domain are irrelevant to our free surface fluid calculations. Third, unless calculating the variable viscosity solution as discussed in Carlson [2004], the coefficients in the six nonzero off-center diagonals are restricted to two values, either 0 or -1. Because of the symmetric, heptadiagonal nature of the matrix, we need only store the central and three upper (or lower) diagonals. Fourth, in simulations that do not require complex occlusion or surface boundaries, the central diagonal elements are restricted to the integral range of 0 to 6 and is the negative sum of the row's diagonal elements. This results in reducing the required matrix storage to $O(n)$ from $O(n^2)$, where n is the total volume of the fluid domain bounding box. Our matrices and vectors also exclude the nonfluid volume null-space elements via undefined runs using the RLE scheme. This reduces the storage further to $O(v)$, where v is proportional to the fluid volume, from $O(n)$. Additionally, since the null space is a property of the linear system as a whole, in our system, all vectors and matrices in a time step share the same RLE run data structure (i.e., the H-RLE grid.) When not computing a variable viscosity solution, we dispense with the full storage of the three noncentral diagonals. Instead, we use a single byte per defined (i.e., non-null-space) central diagonal element whose first 6 bits indicate whether the corresponding noncentral diagonal element is 0 or -1. In the situation where neither complex occlusion or surface boundaries are required, the integer central and off-center diagonal elements can be compressed into runs. While this optimization precludes sharing the common H-RLE grid with the matrix diagonals, the storage of the matrix by itself is reduced to $O(v^{1/2})$

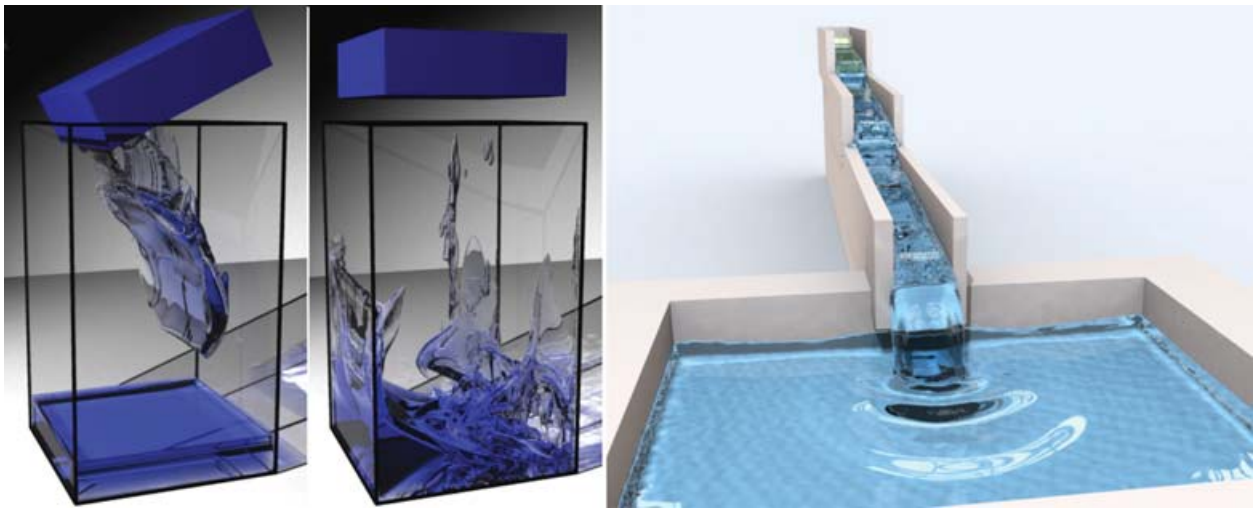

Fig. 11. Two fluid simulations using RLE level sets to achieve a large simulation bounding volume without incurring a similarly large memory burden. Left: The bounding volume is $136 \times 136 \times 320$ (6 million voxels) while the memory usage is only 20 MB, 77% less than the 90 MB of memory required by a corresponding dense field fluid simulation. Right: The bounding volume is $444 \times 178 \times 102$ (8 million voxels) while memory usage is only 30 MB, 75% less than the 130 MB of memory required by a corresponding dense field fluid simulation. See Section 6.6 for details.

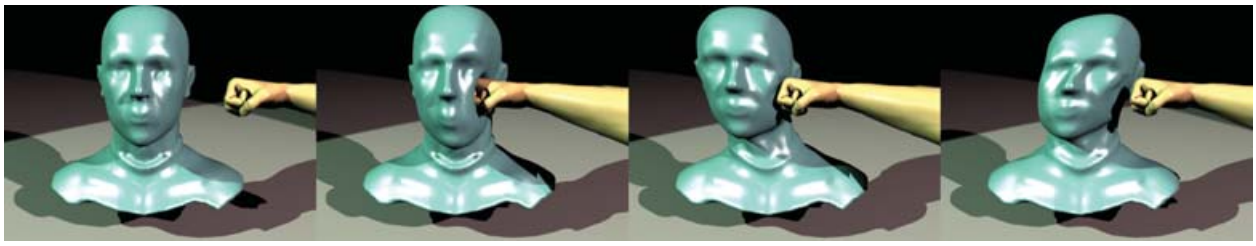

Fig. 12. Four frames of an animation featuring a fully sparse RLE-based elastic fluid interacting with an augmented RLE level set-based occlusion. The simulation of fluid with elastic or viscoelastic properties necessitated development of an RLE-based tensor field to compliment our RLE-based MAC grid. See Section 6.6 for full explanation.

in most cases. Last, whether or not the last two conditional optimizations are implemented, all vector and vector-matrix operations employed by the standard conjugate gradient algorithm (see Shewchuk [1994]) require only $O(v)$ computation since all operations can be computed via a single-pass linear traversal through the defined regions of the involved RLE-vector and/or RLE-matrix data structures.

This RLE-matrix and RLE-vector approach is roughly equivalent to that of Carlson [2004] in terms of efficiency and storage. It should also be noted that since the memory improvements discussed above require the use of multiple classes of nondefined runs they cannot readily be implemented with the DT-Grid structure.

It should be mentioned that, while we are considering implementing an octree fluid velocity grid similar to Losasso et al. [2004] to pair with our RLE fluid surface level set representation, our current regular RLE fluid velocity grid has the advantage of allowing trivial solution of the parabolic PDE viscosity equation which has proved problematic (according to Losasso et al. [2004]) on octree velocity grids such that octree fluids are limited to inviscid fluids.

Table I. Results for Smoothing Away the Stanford Bunny (Using Mean Curvature Flow) on a Grid of Size 256 × 256 × 256 (The octree stores only a uniformly refined narrowband. Peng-I refers to the full rebuild method described in Peng et al. [1999], and Peng-II refers to the recent sparse rebuild optimization of Breen et al. [2004]. The Sparse RLE LS is the Sparse Run Length Encoded Level Set of Houston et al. [2004] and DT-Grid is the Dynamic Tubular Grid of Nielsen and Museth [2006]. The benchmarks were performed on a 2.4-GHz AMD Athlon 4000+).

	H-RLE	Sparse RLE LS	DT-Grid	Octree	Peng-I	Peng-II
Min mem (MB)	0.00	0.00	0.00	0.00	80.0	80.0
Max mem (MB)	4.20	4.39	4.00	8.60	85.1	85.1
Avg. time/iteration (s)	0.727	1.050	0.600	1.47	3.22	1.41

Table II. A Comparison of the Memory Required by Various Level Set Representations to Represent the Larger Models of the Stanford 3D Scanning Repository (A narrowbandwidth of 6 voxels was used. The octree stores only a uniformly refined narrowband. Peng refers to the method described in Peng et al. [1999].)

Model	Bounding Box	H-RLE	Sparse RLE LS	DT-Grid	Octree	Peng
Thai Statue	1800 × 3023 × 1556	402 MB	436 MB	378 MB	836 MB	41.0 GB
Lucy	3000 × 1726 × 5144	738 MB	769 MB	700 MB	1.60 GB	128 GB
Buddha	1800 × 4367 × 1802	816 MB	847 MB	765 MB	1.63 GB	68.0 GB
Asian Dragon	5000 × 2781 × 3329	750 MB	843 MB	700 MB	1.48 GB	222 GB

On one frame of the 136 × 136 × 320 scene, shown on the right in Figure 11, the level set and velocity field consumed roughly 20 MB, whereas a comparable dense grid simulation would require 90 MB. The bounding resolution of the fluid simulation depicted in Figure 10(b) was 448 × 178 × 102. The storage required for the fluid level set and the MAC grid velocity field was only 30 MB, a savings of 75% over the 100-MB storage required by a comparable dense grid simulation. As Figure 12 demonstrates, it is not a difficult matter to further extend our RLE-based representations to tensor fields, thus allowing for the simulation of elastic or viscoelastic fluids (for more information on viscoelastic fluid simulation see Goktekin et al. [2004]).

7. BENCHMARK RESULTS

Below we compare the performance of the H-RLE data structure to other existing state-of-the art level set representations with respect to time and memory usage. The performances of the H-RLE and the DT-Grid are comparable, although the H-RLE sacrifices a bit of the efficiency due to the high level of flexibility of the data structure.

7.1 Memory and Time Usage for Level Set Propagation

Table I lists the time usage and the minimum and maximum memory consumption for smoothing away the Stanford bunny (using mean curvature flow) on a 256 × 256 × 256 grid using several different level set representations. The relevance of this benchmark test is justified by for example, the use of smoothing in blending of geometric models (as demonstrated in Museth et al. [2002].) As can be seen from this test, the performance of the H-RLE level set is comparable to the performance of the DT-Grid. Due to the unique and extreme flexibility of the H-RLE, a bit of the efficiency is sacrificed, but it still uses considerably less memory and time than the other state-of-the-art representations. Even though the H-RLE level set operations are more complex, the good results are attributable to the cache coherent memory layout of the data structure and the ability of the associated algorithms to exploit this.

Fig. 13. Right: The *Tar Monster* from *Scooby Doo 2* immediately after completing a RLE level set shape metamorphosis (see Section 6.4) from a large claw object. Left: The *Tar Monster* holding the actress Sarah Michelle Gellar captive while speaking and gesturing—making use of both the robust mesh to level set converter (see Section 6.2) and a surface deformation algorithm (see Wiebe and Houston [2004]) to ensure his facial features are properly resolved. *Scooby Doo 2* production images ©2004 Warner Brothers. Images provided by Frantic Films.

Fig. 14. A production still from the movie *Cursed* containing a prominent trail of wispy smoke. Augmented RLE level sets were employed to accurately integrate the contours and motions of the actors into the smoke simulation. See Batty and Houston [2005] for more details. *Cursed* production image ©2005 Dimension Films. Image provided by Frantic Films.

7.2 Memory Usage for High-Resolution Models

We have scan-converted several of the high-detail models from the Stanford 3D Scanning Repository. In Table II, we compare the memory required by various level set data structures to represent these models. Again, the memory usage of the H-RLE is comparable to that of the DT-Grid. The deviations are between 5 and 7% which, given the flexibility that comes with the H-RLE, is a reasonable price to pay. As also mentioned earlier, when storing unenclosed level sets, the performance of the H-RLE will be better than the DT-Grid, which is forced to close the level set by introducing articifical boundaries (see Figure 9). Clearly, the H-RLE level set is more memory efficient than the other state-of-the-art representations. It should be noted that an *adaptive distance field* of Frisken et al. [2000] probably would be more memory efficient than the representations listed here in some cases. However, as mentioned earlier, current octree-based level set implementations refine uniformly near the interface.

8. CONCLUSIONS AND FUTURE WORK

Due to their ability to describe arbitrary topological changes, level sets have gained increasing popularity not only in academia but also in real-life production settings. As a result, existing level set technology is constantly pushed to its limits as the demand for larger and more detailed simulations becomes ubiquitous. At the same time, a high level of flexibility is required of a level set data structure in order to be useful in a wide range of computer graphics applications. In this article, we addressed these issues by presenting the *hierarchical run-length encoded (H-RLE) level set* that combines ideas from two recently proposed level set representations, the RLE sparse level set and the DT-Grid. The H-RLE is a novel and very versatile data structure and toolbox of algorithms that enable very high-resolution level set deformations with a near optimal memory footprint. To the best of our knowledge, no other existing level set data structure has all of these features. Also, the performance of the method presented proved comparable to the performance of the DT-Grid and more time-efficient than other existing state-of-the-art approaches, due to its cache optimized memory layout and algorithms. The H-RLE level set and the H-RLE grid, due in large part to their versatile underlying RLE encoding scheme, are easily tailored to many applications in computer graphics, as we demonstrated: fluid simulation, collision detection, level set surface editing, morphing, and geometric modeling. We also focused on developing a level set pipeline in computer graphics by proposing the augmented level set unified implicit object representation, a fast and robust mesh to level set converter and an efficient method for direct ray tracing. This allowed us to scan-convert, deform, and render several very detailed models represented as level sets with effective voxel counts on the order of billions.

The move from theory to practice has been swift. Already, RLE level sets have played key roles in two Hollywood productions: in the creation of the CG *Tar Monster* effect, described in Wiebe and Houston [2004], for Warner Brothers' 2004 release of *Scooby Doo 2* (see Figure 13), and in the wispy smoke simulation technology, described in Batty and Houston [2005], developed for Dimension Films' 2005 release of *Cursed* (see Figure 14).

ACKNOWLEDGMENTS

The following individuals contributed to the modeling, simulation, and/or rendering of the images included in this article: Mark Wiebe, Liem Nguyen, Chris Pember, Jason Cobill, Blair Werschler, Kert Gartner, Jason Booth, and George Taylor. We appreciated the constructive and pertinent comments from the anonymous SIGGRAPH and ACM TOG reviewers.

REFERENCES

ADALSTEINSSON, D. AND SETHIAN, J. A. 1995. A fast level set method for propagating interfaces. *J. Computat. Phys. 118*, 2, 269–277.

BATTY, C. AND HOUSTON, B. 2005. The visual simulation of wispy smoke. In *Proceedings of the SIGGRAPH 2005 on Sketches & Applications*. ACM Press, New York, NY.

BREEN, D., FEDKIW, R., MUSETH, K., OSHER, S., SAPIRO, G., AND WHITAKER, R. 2004. *Level Sets and PDE Methods for Computer Graphics*. ACM SIGGRAPH '04 COURSE #27. ACM SIGGRAPH, Los Angeles, CA. ISBN 1-58113-950-X.

BREEN, D. E. AND WHITAKER, R. T. 2001. A level-set approach for the metamorphosis of solid models. *IEEE Trans. Visual. Comput. Graph. 7*, 2, 173–192.

BRIDSON, R. 2003. Computational aspects of dynamic surfaces. dissertation. Stanford University, Stanford, CA.

BRIDSON, R., MARINO, S., AND FEDKIW, R. 2003. Simulation of clothing with folds and wrinkles. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 28–36.

CARLSON, M. 2004. Rigid, melting and flowing fluid. Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA.

CURLESS, B. AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Computer Graphics 30*. Annual Conference Series. 303–312.

DE ARAUJO, B. R. AND JORGE, J. A. P. 2004. Curvature dependent polygonization of implicit surfaces. In *Proceedings of the XVII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI'04)*. IEEE, Computer Society Press, Los Alamitos, CA. 266–273.

ENRIGHT, D., LOSASSO, F., AND FEDKIW, R. 2004. A fast and accurate semi-Lagrangian particle level set method. *Comput. Struct. 83*, 479–490.

ENRIGHT, D., MARSCHNER, S., AND FEDKIW, R. 2002. Animation and rendering of complex water surfaces. *ACM Trans. Graph. 21*, 3, 736–744.

FOSTER, N. AND FEDKIW, R. 2001. Practical animation of liquids. In *SIGGRAPH '01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*. ACM Press, New York, NY, 23–30.

FRISKEN, S. F., PERRY, R. N., ROCKWOOD, A. P., AND JONES, T. R. 2000. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proceedings of SIGGRAPH 2000*. Computer Graphics Proceedings, Annual Conference Series. ACM Press, New York, NY/ Addison Wesley Longman, Reading, MA, 249–254.

GOKTEKIN, T. G., BARGTEIL, A. W., AND O'BRIEN, J. F. 2004. A method for animating viscoelastic fluids. *ACM Trans. Graph. 23*, 3, 463–468.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph. 22*, 3, 871–878.

HOUSTON, B., BOND, C., AND WIEBE, M. 2003. A unified approach for modeling complex occlusions in fluid simulations. In *Proceedings of the SIGGRAPH 2003 on Sketches & Applications*. ACM Press, New York, NY.

HOUSTON, B., NIELSEN, M. B., BATTY, C., NILSSON, O., AND MUSETH, K. 2005. Gigantic deformable surfaces. In *Proceedings of the SIGGRAPH 2005 on Sketches & Applications*. ACM Press, New York, NY.

HOUSTON, B., WIEBE, M., AND BATTY, C. 2004. RLE sparse level sets. In *Proceedings of the SIGGRAPH 2004 on Sketches & Applications*. ACM Press, New York, NY.

IRVING, G., TERAN, J., AND FEDKIW, R. 2004. Invertible finite elements for robust simulation of large deformation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. New York, NY, 131–140.

JU, T. 2004. Robust repair of polygonal models. *ACM Trans. Graph. 23*, 3, 888–895.

LEVEQUE, R. 1996. High-resolution conservative algorithms for advection in incompressible flow. *SIAM J. Numer. Anal. 33*, 627–665.

LIU, X., OSHER, S., AND CHAN, T. 1994. Weighted essentially nonoscillatory schemes. *J. Comput. Physat. 115*, 200–212.

LOSASSO, F., FEDKIW, R., AND OSHER, S. 2005. Spatially adaptive techniques for level set methods and incompressible flow. *Comput. Fluids*. In Press.

LOSASSO, F., GIBOU, F., AND FEDKIW, R. 2004. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. 23*, 3, 457–462.

MAUCH, S. 2000. A fast algorithm for computing the closest point and distance transform. Go online to `http://www.acm.caltech.edu/seanm/software/cpt/cpt.pdf`.

MUSETH, K., BREEN, D. E., WHITAKER, R. T., AND BARR, A. H. 2002. Level set surface editing operators. *ACM Trans. Graph. 21*, 3, 330–338.

NIELSEN, M. B. AND MUSETH, K. 2006. Dynamic Tubular Grid: An efficient data structure and algorithms for high resolution level sets. *J. Sci. Comput. 26*, 1, 1–39.

OSHER, S. AND SETHIAN, J. A. 1988. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Computat. Phys. 79*, 12–49.

OSHER, S. AND SHU, C. 1991. High-order essentially nonoscillatory schemes for Hamilton-Jacobi equations. *SIAM J. Num. Anal. 28*, 907–922.

PENG, D., MERRIMAN, B., OSHER, S., ZHAO, H., AND KANG, M. 1999. A PDE-based fast local level set method. *J. Computat. Phys. 155*, 2, 410–438.

RASMUSSEN, N., ENRIGHT, D., NGUYEN, D., MARINO, S., SUMNER, N., GEIGER, W., HOON, S., AND FEDKIW, R. 2004. Directable photorealistic liquids. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM Press, New York, NY, 193–202.

SETHIAN, J. A. 1996. A fast marching level set method for monotonically advancing fronts. In *Proceedings of the National Academy of Sciences of the USA 93*, 4 (Feb.), 1591–1595.

SHAH, M., COHEN, J. M., PATEL, S., LEE, P., AND PIGHIN, F. 2004. Extended Galilean invariance for adaptive fluid simulation. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM Press, New York, NY, 213–221.

SHEWCHUK, J. R. 1994. An introduction to the conjugate gradient method without the agonizing pain. Go online to `http://www.cs.cmu.edu/quake-papers/painless-conjugate-gradient.pdf`.

SHU, C. AND OSHER, S. 1988. Efficient implementation of essentially non-oscillatory shock capturing schemes. *J. Computat. Phys. 77*, 439–471.

STAM, J. 1999. Stable fluids. In *Proceedings of SIGGRAPH 99*. Computer Graphics Proceedings, Annual Conference Series. 121–128.

STRAIN, J. 1999. Tree methods for moving interfaces. *J. Computat. Phys. 151*, 2, 616–648.

WHITAKER, R. T. 1998. A level-set approach to 3d reconstruction from range data. *Int. J. Comput. Vision 29*, 3, 203–231.

WIEBE, M. AND HOUSTON, B. 2004. The Tar Monster: Creating a character with fluid simulation. In *Proceedings of the SIGGRAPH 2004 on Sketches & Applications*. ACM Press, New York, NY.

YNGVE, G. AND TURK, G. 2002. Robust creation of implicit surfaces from polygonal meshes. *IEEE Trans. Visual. Comput. Graph. 8*, 4, 346–359.

Received June 2005; revised October 2005; accepted October 2005