Data Structures for Interactive High Resolution Level-Set Surface Editing

Manolya Eyiyurekli* Drexel University

David Breen† Drexel University

Abstract

This paper presents data structures that enable interactive editing of large-scale level-set surface models. The new approach utilizes spatial hashing to store a narrow band of voxels around the level-set interface, as well as a k-d tree to hold the model's display points that lie on the surface itself. This sparse representation of voxels and surface points lets us create and modify high resolution level-set models with modest memory requirements, while allowing fast data access/modifications and interactive graphics updates. The data structures also support out-of-the-box editing, i.e. no bounding box limits the surface editing region, a restriction common when utilizing 3-D arrays. We formally define the level-set representation and demonstrate its interactive performance and scalability through manipulation of high-resolution level-set surface models.

Index Terms: Computer Graphics [I.3.6]: Methodology and Techniques—Graphics data structures and data types

1 Introduction

Level-set models are an important and powerful type of implicit model that offer numerous benefits. They combine a low-level volumetric representation with the mathematics of deformable implicit surfaces, which is based on formulating and solving a partial differential equation (PDE) [27, 34]. They have been devised within a well-developed mathematical framework that provides robust numerical techniques for evaluation and evolution [26]. They are guaranteed to define simple (non-self-intersecting) and closed surfaces, a property critical for analysis, computer-aided manufacturing and rapid prototyping. Level-set models easily change topological genus, making them ideal for representing complex structures of unknown or transforming genus. Additionally, they provide the advantages of implicit models, e.g. supporting straightforward solid modeling operations and calculations, while simultaneously offering a surface modeling paradigm. Because of these advantages a number of level-set modeling techniques have been developed, including volume sculpting [2], free-form surface editing [7, 8], CSG operations with automatic blending and curvature-based smoothing [21, 22].

A major limitation to editing a level-set model is the memory required to store its underlying volumetric representation. Space complexity and evaluation costs have prevented these models from being utilized in interactive modeling systems. An additional problem common to interactive systems for editing large-scale models is the cost of rendering the surfaces. A current state-of-the-art volumetric modeling system should support models containing at least one billion voxels and provide 25-30 frames-per-second (fps) evaluation and rendering rates at these resolutions. Even the most advanced data structures and algorithms developed to date for level-set models are not able to meet both of these requirements simultaneously. This paper describes a novel approach to storing a level-set model in a compact spatial hash table, as well as the utilization of k-d trees to optimize rendering times. Our work closes the gap between level-set methods and interactive modeling applications by providing new techniques that allow these models to be incorporated into current modeling systems. A level-set model interactively created with our new data structures is shown in Figure 1.

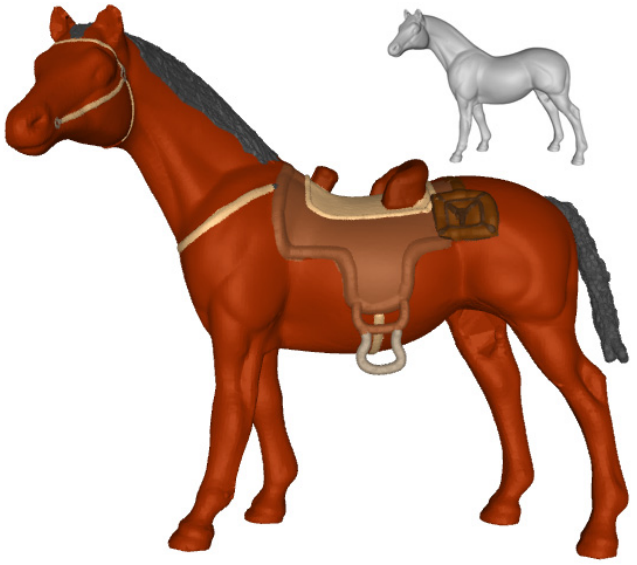


Figure 1: A scan converted level-set model of a horse (upper right) is edited to add surface details.

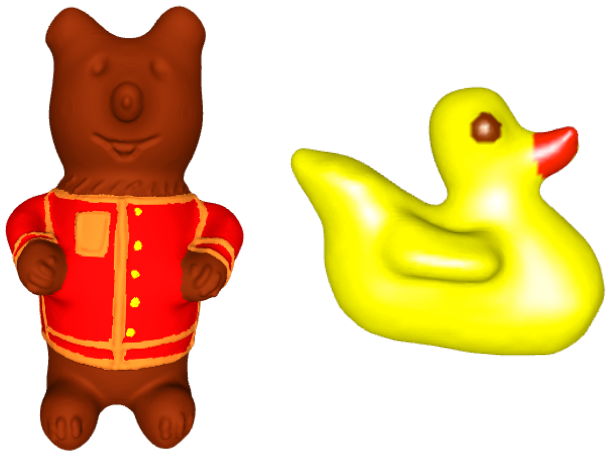
We recently developed a suite of free-form editing operators [7, 8] for modifying level-set models. They provide a user with numerous expressive editing capabilities when designing level-set surfaces. With these operators a user may directly pull a point or curve on a level-set surface or sketch cross-sectional curves above the surface to which the surface moves and conforms. Two level-set models generated with some of these operators are presented in Figure 2. The resolution of the models created from this earlier work was limited by the memory required to store the volume dataset that represented the surface.

Our editing system utilizes the VISPACK level-set library [37] to represent and process level-set models. VISPACK contains a narrow-band implementation for efficiently solving the level-set PDE [36]. The technique localizes computation to only those voxels that lie within a narrow band around the level-set surface. While this implementation provides an efficient way to store and update the narrow band, it still relies on a dense volume (i.e. a 3-D array) to store the voxels in memory. The $O(n^3)$ size of this representation (where n is the number of samples in one of the spatial dimensions) quickly overwhelms the memory capacity of most computers even for models with limited resolution. For example in our previous work the largest level-set volume dataset that we were able to store and interactively manipulate on a computer with 14GB of memory contained approximately 80 million voxels.[1] This represents a cu-

*e-mail: me52@cs.drexel.edu

†e-mail:david@cs.drexel.edu

[1]Our editing operators also require additional memory for the complex narrow-band data structures needed to localize computation to only those



(a) (b)

Figure 2: A cartoon bear and a toy rubber duck created with level-set surface editing operators described in [7] and [8]. (a) Resolution: $320 \times 320 \times 600$. (b) Resolution: $79 \times 67 \times 37$.

bic volume dataset with n approximately equal to 430. Therefore, a sparse data structure that only stores the data associated with the voxels lying in the narrow band is the key to representing high resolution level-set models.

Higher spatial resolutions are needed in order to make detailed models with small, fine structures. Creating a volumetric model where n approaches 2,000 (8 billion voxels if stored in a 3-D array) would provide high-resolution capabilities and would be desirable for current volumetric modeling applications. The challenge when creating volumetric models of these sizes is to keep the model processing time *interactive*, i.e. providing model and display updates at a rate of 25-30 frames-per-second (fps). While current level-set data structures can represent models at these high resolutions [14, 23, 24], they do not support the rapid, arbitrary access and update operations required for interactive editing applications. To address this deficiency, this paper presents data structures that enable interactive editing of large-scale level-set surface models. The new approach utilizes spatial hashing to represent a narrow band of voxels around the level-set interface, as well as a k-d tree to hold the model's display points that lie on the surface itself. This sparse representation of voxels and surface points lets us create and modify high resolution level-set models with modest memory requirements, while allowing fast data access/modifications and interactive graphics updates (normally above 25 fps). It also supports out-of-the-box editing, i.e. no bounding box limits the surface editing region, a restriction common when utilizing 3-D arrays. As compared to previous PDE-based modeling work, our system provides significantly faster processing speeds on much larger volumetric models, even when considering the difference in processor power.

2 Previous Work

2.1 Advanced Level-Set Data Structures

The original level-set method as proposed by Osher and Sethian [27] has $O(n^3)$ time and storage complexity, where n is the side length of the bounding volume in which the deforming iso-surface is embedded. The computational complexity can be reduced to $O(n^2)$ with a narrow-band scheme [1, 28, 36] that solves the PDE in a narrow band only around the interface. In order to minimize the storage requirement of level sets while keeping the computational complexity low, octree-based approaches have been employed [2, 19]. Octrees have also been utilized in other volumetric modeling systems [11, 20, 29]. The storage required in these methods is $O(n^2)$ and the random access time to the values is $O(\log n)$. The major drawback of these methods, as with all level-set methods, is that they require a uniform refinement along the interface to use finite-difference based numerical methods; thus losing the adaptiveness benefit provided by the octrees.

Run-length encoding (RLE) is a simple form of lossless data compression in which runs of data (i.e. sequences in which the same data value occurs in consecutive data elements) are stored as a single data value and a count. The RLE sparse level-set data structure [15] assigns either a very large positive or negative value to all voxels outside of the narrow band, which are then compressed into runs on each side of the narrow band using RLE. It has $O(n^2 + R + D)$ storage requirements, where R is the number of runs and D is the number of voxels in the narrow band. The method uses a 2-D array to facilitate fast random access ($O(\log r)$, where r is the number of runs in a level set cross section). Nielsen and Museth [23] created the DT-Grid (Dynamic Tubular Grid) data structure, which uses a hierarchical representation of the data's dimensions to compress the volume and gain memory efficiency. DT-Grid provides constant access time to the grid's values and their immediate neighbors as long as all values are accessed sequentially. It also provides logarithmic ($O(\log n)$) random access to the sparse data by keeping the data lexicographically sorted. This data structure is unsatisfactory though for interactive applications because of the time required to keep the data sorted and densely packed during insertion and deletion operations. The RLE sparse level set and DT-Grid were combined to create an improved data structure, Hierarchical RLE (H-RLE) level sets [14]. This method employs an RLE in a dimensionally recursive fashion combined with a narrow-band scheme and provides $O(D)$ storage complexity, i.e. $O(n^2)$. H-RLE also keeps the data sorted in linear arrays and suffers from the same drawbacks as DT-Grid when performing arbitrary modifications to the level set.

While narrow-band schemes effectively address the problem of computational complexity in the original level-set formulation, they explicitly store a full Cartesian grid and use additional data structures to identify the narrow-band grid voxels. For example, the cartoon bear model in Figure 2, which is represented with a $320 \times 320 \times 600$ volume in Whitaker's VISPACK library [37], requires 3 GB of memory during editing. The advanced data structures described in this section all reduce the memory requirements from $O(n^3)$ to $\sim O(n^2)$. However, none of these data structures were designed for the rapid, random, local voxel accesses, updates and modifications that are essential for an interactive surface editing application.

Ferley et al. [9] present a sculpting metaphor using a uniformly sampled scalar field as the volume representation. Efficient data structures such as binary search trees are used for fast extraction of relevant scalar values to create an interactive environment. However, this framework still suffers from slow frame rates. Resolution adaptive volume sculpting [10] is a multi-resolution sculpting environment that achieved the goal of running at interactive speeds. The authors used an adaptive grid, instead of a uniformly sampled grid as in their previous work, that dynamically subdivides or merges for a more efficient data representation. They utilize a hashing structure in order to create an unbounded grid, as well as a narrow-band representation for the volume. However, the paper does not provide the details of the hash function used or discussions on the performance they achieved by utilizing spatial hashing for data storage.

2.2 Interactive Rendering of Large-Scale Dynamic Point Sets

The level-set model may be displayed either with point or polygon rendering. The VISPACK library can return a set of points lying on the level-set surface [37]. The surface may also be displayed as

regions of the model being modified [7].

a set of polygons, which can be generated from the level-set volume using a polygon extraction algorithm [18, 38]. Extraction of points, along with point rendering, is much faster than mesh generation and polygon rendering, providing more interactive editing feedback. We have found it useful to be able to switch between the two types of viewing, allowing the user to choose either responsiveness (displaying points) or quality (displaying a mesh) when rendering [7].

Hierarchical octree data structures are one of the most common choices to handle large point sets for interactive rendering [4, 32]. While octrees provide a simple hierarchical organization of space they can suffer from the fact that in general points on a 3D surface cannot be evenly partitioned into octants. This may lead to an unbalanced and suboptimal data structure. In contrast, k-d trees [25, 33] can guarantee a fully balanced hierarchical structure. Bounding volume hierarchies (BVHs) have also been used in rendering to efficiently support spatial queries such as visibility culling or ray-object intersections [3, 12, 31]. Unlike octrees and k-d trees, a BVH does not necessarily completely partition space; thus it allows for a more generic and efficient hierarchical organization of spatial data. While both BVHs and k-d trees are good candidates to evenly partition a set of 3D surface points into several VBOs, we found k-d trees more suitable due to the fact that the generality and increased partitioning efficiency of BVHs are not critical for our application, especially considering the additional storage needed to represent BVHs. K-d trees have proven to provide the interactive performance needed to meet our requirements.

2.3 Optimized Spatial Hashing

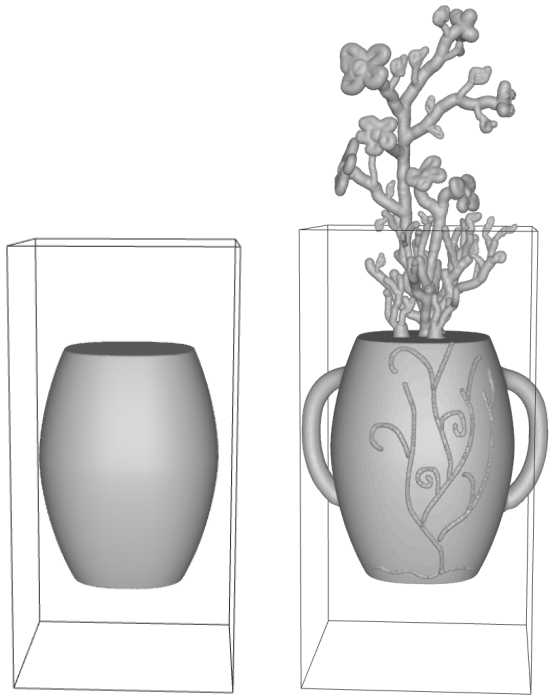
Spatial hashing is a process by which a 3-D or 2-D domain space is projected into a 1-D hash table. The hash function takes a 2-D or 3-D data point and returns an index that corresponds to a 1-D entry in the hash table. Points on an object may be hashed and the locations can then be quickly queried. Spatial hashing has been utilized by several fast collision detection algorithms [6, 17] and has garnered much interest from the gaming industry [13, 30].

A "collision" occurs when two distinct elements are assigned into the same position in the hash table. A perfect hash function for a set S is a hash function that maps the elements of S into unique integers, with no collisions. Perfect hash functions are rare in the space of all possible functions. Thus one cannot expect to construct a perfect hash function for an arbitrary and dynamic set of points. Instead, a hash function must be developed that minimizes collisions. Lefebvre and Hoppe [16] define a GPU-compatible minimal perfect hash function that is pre-computed on a static set of points. Using this hash function they can pack sparse data into a compact table while retaining efficient random access. While their GPU algorithm is efficient, it is not suitable for storing dynamic content. Teschner et al. [35] proposed an algorithm for (self-)collision detection of dynamically deforming objects. Their algorithm employs a hash function for compressing a potentially infinite regular spatial grid. Although the hash function does not always provide a unique mapping of points to hash table positions, it can be generated very efficiently and does not require complex data structures, such as octrees or BSP trees.

3 Efficient and Dynamic Data Structures for High-Resolution Level-Set Models

3.1 Voxel Representation

For our level-set editing system, we wish to pack sparse data into a dense 1-D hash table using a hash function $H(P)$ for a set of data with 3-D coordinates P. The hash values should be uniformly distributed in order to minimize collisions and to guarantee adequate performance. Furthermore, the evaluation time for the hash function should be compatible with the frame rates of the interactive application. To meet these requirements, we use spatial hashing to



(a) (b)

Figure 3: (a) Scan converted initial model with its bounding box. (b) Level-set editing operators create a new model that extends outside of the bounding box.

store data associated with narrow-band voxels in a 1-D hash table of size T_S. The data includes the 3-D position of the voxel, distance values, a gradient vector, a color index, a pointer into the narrow-band data structure, and indices into the arrays storing display data. Hash values are computed for all discrete vertex positions P using a hash function $H(P)$. We use the hash function described in [35]

$$H(P) = (P_x * C_1 \hat{} P_y * C_2 \hat{} P_z * C_3) \,\%\, T_S,$$

where P_x, P_y, and P_z are 3-D coordinates, and C_1, C_2, and C_3 are three constants, $\hat{}$ is the *bitwise exclusive or* operator and % is the *modulus* operator. The three constants, $C_1 = 73,856,093, C_2 = 19,349,663$, and $C_3 = 83,492,791$, are large prime numbers. We assume that the modulus operator always returns positive numbers, i.e. $-1 \,\%\, 5 = 4$. Teschner et al.'s analysis [35] concludes that the function can be evaluated very efficiently and produces a comparatively small number of collisions with large hash tables.

For a level-set model with $O(n^3)$ voxels, where n is the resolution in one dimension, the number of voxels on the surface are $O(n^2)$. We need at least three neighboring voxels on each side of the interface for the finite difference methods that calculate surface normals and curvature. We set the size of the hash table to be $7 * n^2$, which is approximately equal to the number of voxels inside the narrow band. This drastically reduces the memory complexity for large n, i.e. $n \sim 2^{10}$, while keeping the number of entry collisions low. For non-square volumes, the hash table size is set, at the beginning of an editing session, to $7 * I * J$, where I and J are the two largest dimensions of the bounding box around the model to be edited. In addition to the benefits of low memory requirements and fast access/modification times, spatial hash tables provide the extra advantage of implementing an unbounded grid; thus enabling "out-of-the-box" computing. In the past storing level-set values in a 3-D array has constrained the model to lie within the bounds of the array. Similar to the data structures of Houston et al. [14] and Nielsen and

Museth [23], spatial hash tables allow level-set models to evolve outside of the bounds of their initial model. Figure 3 presents the bounding box of a scan-converted model and demonstrates how the editing operators can create a new model that extends out of this bounding box.

3.2 Display Representation

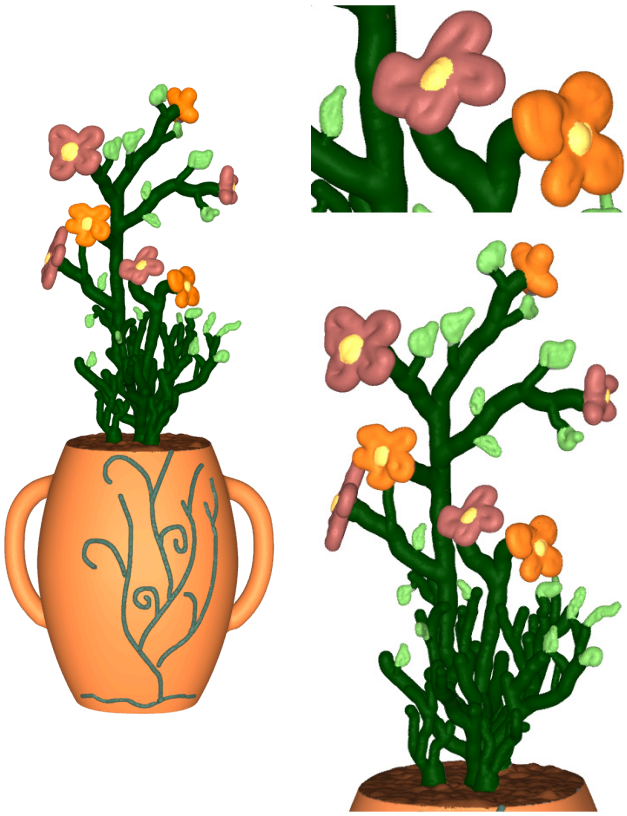
We employ OpenGL Vertex Buffer Objects (VBOs) to display the points lying on the dynamic level-set surface. Every time the surface changes the buffer holding the point data is remapped and transfered to the GPU. This remapping process may slow down an editing application if a large-scale model is displayed with one buffer, and the entire buffer must be updated for small changes in the model occurring every frame. Since our editing operators usually modify the surface locally we spatially divide the surface amongst several VBOs. Only the VBOs associated with the modified portion of the model are remapped during an editing operation. This distribution of the surface over several VBOs and the selected updates of a subset of the VBOs significantly improve display performance for small, localized surface modifications.

It is important to partition the surface evenly between the VBOs to optimize update times. One can utilize binary space partitioning to accurately distribute the vertices between VBOs. This method ordinarily requires that the data first be sorted when building the partition, which requires $O(N \log N)$ time for N data points. However, the surface is likely to change between each frame, thus requiring that the data structure be completely recomputed in order to keep the data sorted and the binary tree balanced. Therefore, it is more advantageous to use a simpler algorithm with lower computation costs that subdivides the set of vertices into approximately equal segments. We have loosened the requirement of having a strictly balanced BSP tree in order to minimize the amount of time spent subdividing and balancing the tree that holds the display vertices. Therefore, we utilize a k-d tree and a separation plane calculation that does not require sorting to produce an approximately balanced tree. We employ a divide-and-conquer method that recursively finds the centroid of the display vertices and subdivides the set into two parts around an axis-aligned plane passing through the centroid. Finding the centroid is linear in both space and time complexity.

Display vertices are added to and dropped from the surface as it changes from application of the editing operators. Enhancements to the VISPACK library return the lists of voxels that are added to and removed from the narrow band. Once a vertex is added, the VBO that should display this vertex is located by traversing the k-d tree and inserting the vertex into the vertex array associated with the particular VBO. If a vertex is dropped because it is no longer on the surface, its location in the VBO arrays can be retrieved from a hash table, and then removed from the VBO's vertex array. Only the VBOs associated with the modified nodes of the tree are remapped at each frame.

The algorithm does not guarantee to evenly partition a set of points on a level set, however, it is straightforward to calculate and does not require sorting of the vertices. As seen from Figure 7, it creates a fair partitioning where each VBO will be assigned a set of vertices to display. Being a binary tree, it also answers vertex location queries in $O(L)$ time, where L is the level of subdivision, providing a fast way to update the vertex data and identify which VBOs need to be remapped at every frame.

We have found that having the unbalanced distribution of vertices in the VBOs produced by our partitioning method does not significantly impact graphics performance when editing and displaying our models. During editing, it is possible that the imbalance of vertices can increase significantly and slow the interactive display when updating and drawing the model. Given this situation, which occurs infrequently, the partitioning algorithm should be run again to improve the distribution of vertices within the VBOs. We have explored a number of methods for automatically triggering the repartitioning of the display vertices, based on frame rates and on quantifying the imbalance using the difference between or the ratio of the minimum and maximum VBO sizes. If the difference or ratio changes significantly from the value calculated at initialization then the VBO data structure can be recomputed. Since graphics performance was normally not an issue when creating our models, we provide a manual method for repartitioning display vertices via user input in our system interface, which the user can execute if display times degrade during the editing session.

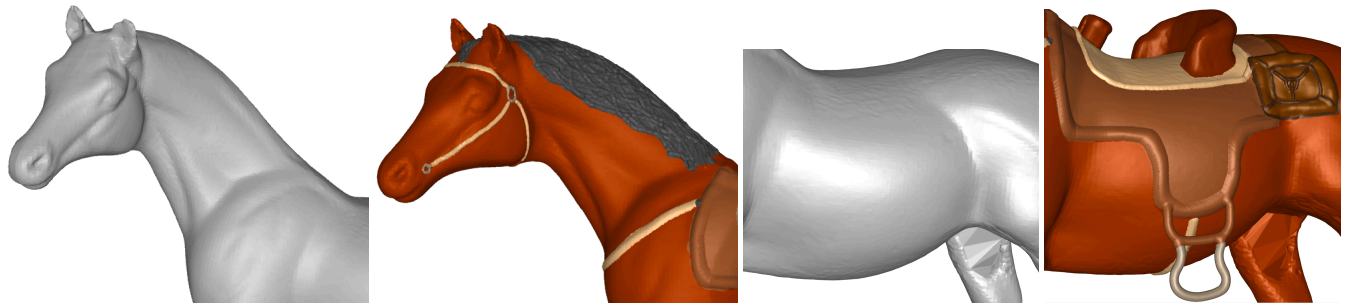


(a) (b)

Figure 4: A flower pot is modeled from a superellipsoid. (a) Handles and decorations on the surface are added to the initial model. Soil is added to the top of the pot. Stems, leaves and the flowers are then modeled above the soil. (b) Close-ups of the final painted model.

4 Results

We created three models to demonstrate the capabilities of our level-set editing system. Figures 3 and 4 show a flower pot created from a simple initial model. The pot for the model was defined as a scan-converted superquadric [22]. The stems, flowers and leaves are sculpted on top of the pot, and some soil and surface details are added to the pot itself with freeform surface editing operators [7]. The pot was scan-converted into a $600 \times 600 \times 1200$ volume with $7,384,242$ voxels in the narrow band and the final edited model moved out of this bounding box and has effective dimensions of $654 \times 600 \times 1794$, with $11,852,818$ voxels in the narrow band. Figures 1 and 6 show the editing of a horse model. We added a saddle, stirrups and a bridle to the model, as well as a tail and mane, using our editing tools. The initial and final models have the same effective dimensions, $944 \times 2048 \times 1709$. The initial model has $26,236,562$ voxels in the narrow band and the final edited model



(a) (b) (c) (d)

Figure 6: (a,c) Portions of the scan converted horse model. (b) A bridle, and mane are added. (d) A saddle, stirrups and saddlebag are added. (See Figure 1 for the final model)

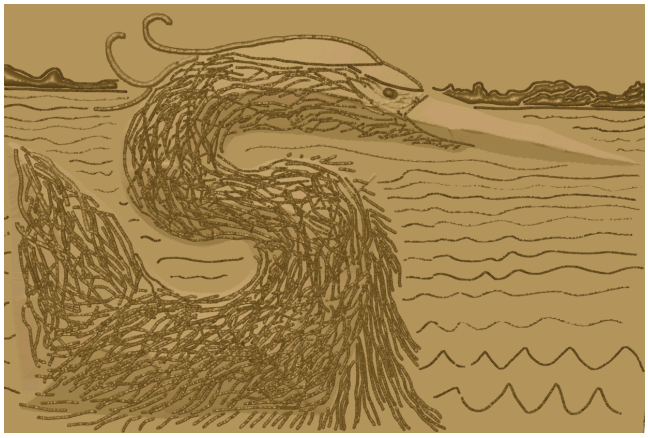


Figure 5: A bas relief model of a heron created with an open level-set model.

has $28,582,546$ narrow band voxels.

Figure 5 presents a design of a heron. It is created with an open level-set model that defines a single "flat" surface that can modified in regions away from the boundary. Similar to unenclosed H-RLE level sets [14], our level-set models do not need to be defined as closed solid objects. Given our flexible narrow-band representation, localization of PDE processing, and an explicit inside/outside categorization of the voxels, we are able to, for the first time, evolve a level-set thin sheet. We can set the bounding box of the object and the Regions of Influence (ROIs) of the editing operators to be smaller than the extent of the model, thus avoiding unwanted numerical problems along the sheet boundary. Just modeling a single sheet surface allows us to make high resolution bas relief type models. The resolution of the initial volume representing a plane is $3081 \times 2057 \times 60$ and the effective resolution of the final model is $3081 \times 2057 \times 69$. The initial model has $44,091,388$ voxels in the narrow band and the final edited model has $44,552,586$ narrow band voxels.

From the data presented in Tables 2 and 3 it can be seen that we have achieved the goal of implementing a system that allows us to interactively edit high resolution level-set models. Given that the operator functions and PDE processing takes more computation time than surface display, the timing results presented in these tables show that our system, with a spatial hash table for storing the level-set surface and a k-d tree for storing the display points, provides an acceptable interactive frame rate for most of the editing operations. These frame rates compare favorably with previous volumetric PDE-based modeling systems, which either take many seconds to minutes to perform a low resolution surface modification [5], or exhibit interactive rates approximately four times slower than ours [2]. The one model change that was not interactive is the offsetting operation performed on the heron body in Figure 5. This is a large-scale modification that affects nearly half of the high resolution surface. The body was created by first defining the ROI with a boundary curve. The ROI area was then lifted with a speed proportional to the distance to the boundary curve. To perform this editing operation required processing almost 27 million voxels; thus the excessive computation times.

5 Discussion

The goal of our work is to develop techniques that enable interactive editing of high-resolution level-set models. There are two potential bottlenecks that may interfere with the attainment of this goal and slow editing times. The first is the processing and evolution of the level-set PDE and the second is the display of the constantly-changing large-scale model. These processes run sequentially and we can only edit the model as fast as the slower of these two components. We present techniques for accelerating the former in [7] by localizing the editing operations and reducing the size of the PDE domain that must be solved. Fast data access also affects the speed at which the PDE can be solved. Spatial hash tables, as described in Section 3.1, furnish acceptable access times, while also providing for the efficient storage of high resolution models and supporting unconstrained out-of-the-box computing. Table 1 contains the average number and standard deviation of voxels per table entry with the given hash table sizes for the example models. The table shows that most of the entries in our hash tables contain 1 to 3 voxels, highlighting the effectiveness of the hash function to distribute data over the whole table and minimize data collisions. Table 1 also shows that the spatial hash tables are able to represent these models with an order of magnitude fewer voxels than a 3-D array. For example the full resolution model for Figure 1 would contain over 3.3 billion voxels, while our data structure only stores the approximately 28 million voxels of the narrow band.

During our research we implemented and investigated a number of potential data structures for our level-set modeling system, namely DT-grids [23] and run-length encoding (RLE) [15]. We found neither of them to provide the rapid random access and modification times required for an interactive editing application. DT-grids were developed for high resolution level-set applications that process the complete level-set surface, storing the narrow-band data in dense, lexicographically-sorted arrays. Therefore they do not adequately support random access, insertion and deletion of data elements, since they assume that the whole level-set surface is processed sequentially for every time step. For example, inserting a new element into the (sorted) DT-grid is an O(N) operation, where N is the number of elements in the data structure. Because of this, during development we abandoned DT-grids as a data structure for

Model	Dimensions	Table Size	Number of Voxels in Narrow Band	Mean	Standard Deviation
Horse, Figure 1	$944 \times 2048 \times 1709$	24,500,224	27,869,503	1.138	1.067
Flowers, Figure 4	$654 \times 600 \times 1794$	8,212,932	11,505,899	1.401	1.184
Heron, Figure 5	$3081 \times 2057 \times 69$	44,363,320	44,552,586	1.003	1.002

Table 1: Statistics for the spatial hash function and hash table. Given are the number of entries in the hash table (size), and the mean and the standard deviation of the number of voxels stored in each entry.

Model	#Vertices	1 VBO		8 VBOs		16 VBOs		32 VBOs		64 VBOs		128 VBOs	
		E	R	E	R	E	R	E	R	E	R	E	R
Fig. 1	3,749,988	.0656	-	.0083	5.86	.0036	7.13	.0022	7.87	.0012	8.74	.00063	9.71
Fig. 4	1,055,282	.0265	-	.0035	2.00	.0015	2.32	.00055	2.62	.00046	2.90	.00036	3.16
Fig. 5	6,291,456	.0994	-	.0121	7.96	.0065	9.96	.0034	12.1	.0016	14.2	0.0011	16.2

Table 2: Average frame times (in seconds) needed to remap, transfer graphics data and draw the VBOs after an editing operation (E). Times (in seconds) needed to rebuild (R) the VBO k-d tree. Times are given for rendering with 1, 8, 16, 32, 64 and 128 VBOs.

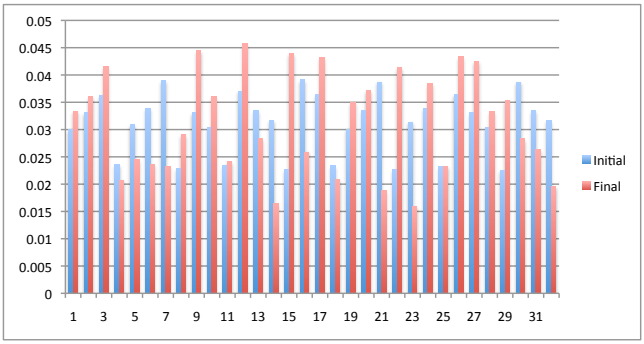


Figure 7: Percentage of vertices per VBO for the flower pot model shown in Figure 4. Blue bars represent the distribution for the initial "pot only" model and the red bars represent the vertex distribution for the final edited model.

our interactive level-set modeling system. RLE was then investigated as a potential method for storing our models.

Table 3 presents a comparison of execution times required to perform a variety of editing operations on the example models using two different data structures, one based on run-length encoding and the other on spatial hashing of the model's voxel data. In general using spatial hash tables provides a 1.5 to 2 times speed-up over a run-length encoded data structure, mainly because random access/insertion is faster with the hash tables. Another major drawback of an RLE implementation is that as the surface changes it is costly to keep the data structure condensed and simplified. As the model is modified, run-lengths need to be added, dropped and merged to keep the data structure compact and precise. Furthermore, we found that the RLE implementation used about 33% more memory than the spatial hash implementation in our studies, given the overhead of the run-length data and the pointers/indices needed to connect the run-lengths.

Figure 7 presents the distribution of display vertices (as a percentage) in the 32 VBOs used to display the flower pot model in Figure 4. The distribution is produced via the k-d tree subdivision described in Section 3.2. The average VBO size (percentage of vertices in a single VBO) is, of course, 1/32 (0.031), and the standard deviation of the initial model distribution is 0.0055 and the final model distribution is 0.0093, with the maximum and minimum sizes being 0.0457 and 0.0160. The distribution demonstrates that the fast, approximate technique used to assign vertices to VBOs creates a relatively even distribution, with most VBO sizes being

Model Detail	Time (secs) w/ RLE	Time (secs) w/ Hashing	fps w/ Hashing
Horse, Fig. 1. Dimensions: $944 \times 2048 \times 1709$			
Saddle	0.033	0.016	63
Seat	0.33	0.20	5
Mane/Tail	0.0090	0.0025	400
Tail	0.17	0.10	10
Bridle	0.010	0.0083	120
Stirrups	0.016	0.010	100
Flowers, Fig. 4. Dimensions: $654 \times 600 \times 1794$			
Pot handles	0.071	0.040	25
Pot details	0.011	0.0066	152
Plant roots	0.014	0.0059	169
Flowers	0.033	0.020	50
Leaves	0.0083	0.0040	250
Soil	0.033	0.018	56
Heron, Fig. 5. Dimensions: $3081 \times 2057 \times 69$			
Heron body	322	177	0.006
Feathers	0.0085	0.0020	500
Waves	0.18	0.083	12
Mountains	0.31	0.22	5

Table 3: Average execution times (in seconds) needed to compute an editing operation for one display frame during the creation of a variety of model details. Times are given for an implementation of RLE Sparse Level Sets [15] and our Spatial Hash method.

within 33% of the mean. The imbalance of the VBOs does increase after the model has been modified, but as stated earlier, this imbalance does not significantly affect the time needed to display the model. We found similar results with the horse model. Since the heron model is effectively a height field, we utilize an X-Y only partitioning to produce a nearly uniform distribution of display points over the VBOs. The distribution statistics for the three models is listed in Table 4.

Having the display vertices subdivided and distributed amongst multiple VBOs in order to improve graphics performance raises the question of what is the optimal level of subdivision. We performed a simple editing operation, a point-click and pull operation with a radius of 5 voxels that creates a 20 voxel protrusion, and gathered display time information for several levels of spatial subdivision (and therefore several total numbers of VBOs) to explore this issue. The editing operation was performed on the initial horse, pot and open level-set models. The results of this study are presented in Table 2, and include the number of vertices used to display the

Model	Std. Dev.	Max	Min
Fig. 1 Initial	0.0084	0.0556	0.0171
Fig. 1 Final	0.0081	0.0533	0.0179
Fig. 4 Initial	0.0055	0.0391	0.0226
Fig. 4 Final	0.0093	0.0457	0.0160
Fig. 5 Initial	8.27E-05	0.0313	0.0312
Fig. 5 Final	0.0093	0.0326	0.0305

Table 4: Statistics for the VBO k-d trees. Given are the standard deviation, minimum and maximum sizes of the 32 VBOs used to display the example models. The average VBO size (percentage of vertices in a single VBO) is 1/32 (0.031).

model, average times (in seconds) needed to display the vertices after editing (E) for each frame using a number of VBOs (from 1 to 128) produced via k-d tree partitioning. The VBO data includes the time needed to remap, transfer graphics data and draw the VBO. Also included are the times (in seconds) required to repartition the display vertices of the model (R). While rendering times go down with increasing VBO number, the repartitioning times increase.

These experiments led us to use 5 levels of subdivision with 32 VBOs when creating the example models. By comparing the editing operation times in Table 3 with the display times in the 32 VBOs column of Table 2 it can be seen that with this number of VBOs more time is needed to compute the editing operation than to display the resulting dynamic model. The one exception is the simple feather detailing operation used in Figure 5. Choosing to display the level-set model with 32 VBOs gives graphics frame rates of 300 fps and better, and makes the editing functions and PDE processing the computational bottleneck during interactive modeling. Of course higher levels of subdivision produce greater repartitioning (R) times. Since repartitioning is required so infrequently we believe that achieving interactive display performance that is not limited by rendering times (for our models/application) is worth the cost of the few extra seconds that may be needed occasionally for VBO restructuring.

6 Conclusions

We have described data structures that enable interactive editing of large-scale level-set surface models. The new approach utilizes spatial hashing to represent a narrow band of voxels around the level-set interface, as well as a k-d tree to hold the model's display points that lie on the surface itself. This sparse representation of voxels and surface points lets us create and modify high resolution level-set models with modest memory requirements, while supporting fast data access/modifications and interactive graphics updates. The data structures also support out-of-the-box editing, i.e. no bounding box limits the surface editing region. Through a number of experiments we have shown that the data structures have the properties necessary to meet our performance requirements. They allow us to interactively edit (at frame rates usual over 25 fps) high resolution level-set models (with voxel counts equivalent to a 1500^3 volume dataset). The spatial hash function of Teschner et al. [35] has been shown to satisfactorily distribute surface locations in the hash table, thus minimizing collisions and maximizing access/modification times. Storing display points in a k-d tree supports the localization of graphics processing, which minimizes the amount of data that needs to be transfered from the application to the GPU during editing of small regions of a high resolution model. Together, these data structures provide a new capability for the interactive modification of large-scale level-set models.

Acknowledgements

References

[1] D. Adalsteinsson and J. Sethian. A fast level set method for propagating interfaces. *Journal of Computational Physics*, 118(2):269–277, 1995.

[2] J. Baerentzen and N. Christensen. Volume sculpting using the level-set method. In *Proc. International Conference on Shape Modeling and Applications*, pages 175–182, 2002.

[3] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Communications of the ACM*, 19(10):547–554, 1976.

[4] L. Coconu and H.-C. Hege. Hardware-oriented point-based rendering of complex scenes. In *Proc. Eurographics Workshop on Rendering*, pages 43–52, 2002.

[5] H. Du and H. Qin. Free-form geometric modeling by integrating parametric and implicit PDEs. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):549–561, 2007.

[6] M. Eitz and G. Lixu. Hierarchical spatial hashing for real-time collision detection. In *Proc. International Conference on Shape Modeling and Applications*, pages 61–70, 2007.

[7] M. Eyiyurekli and D. Breen. Interactive free-form level-set surface-editing operators. *Computers & Graphics*, 34(5):621 – 638, 2010.

[8] M. Eyiyurekli, C. Grimm, and D. Breen. Editing level-set models with sketched curves. In *Proc. Eurographics/ACM Symposium on Sketch-Based Interfaces and Modeling*, pages 45–52, 2009.

[9] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Practical volumetric sculpting. *The Visual Computer*, 16(8):469–480, 2000.

[10] E. Ferley, M.-P. Cani, and J.-D. Gascuel. Resolution adaptive volume sculpting. *Graphical Models*, 63(6):459–478, 2001.

[11] S. Frisken, R. Perry, A. Rockwood, and T. Jones. Adaptively sampled distance fields: A general representation of shape for computer graphics. In *Proc. SIGGRAPH*, pages 249–254, 2000.

[12] M. Gross and H. Pfister. *Point-Based Graphics*. Morgan Kaufmann, San Francisco, 2007.

[13] E. J. Hastings, J. Mesit, and R. K. Guha. Optimization of large-scale, real-time simulations by spatial hashing. In *Proc. 2005 Summer Computer Simulation Conference, Vol. 37, No. 4*, pages 9 – 17, 2005.

[14] B. Houston, M. Nielsen, C. Batty, O. Nilsson, and K. Museth. Hierarchical RLE level set: A compact and versatile deformable surface representation. *ACM Transactions on Graphics*, 25(1):151–175, 2006.

[15] B. Houston, M. Wiebe, and C. Batty. RLE sparse level sets. In *ACM SIGGRAPH Sketches*, page 137, 2004.

[16] S. Lefebvre and H. Hoppe. Perfect spatial hashing. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3):579–588, 2006.

[17] X. Li, L. Gu, S. Zhang, J. Zhang, G. Zheng, P. Huang, and J. Xu. Hierarchical spatial hashing-based collision detection and hybrid collision response in a haptic surgery simulator. *International Journal of Medical Robotics and Computer Assisted Surgery*, 4(1):77–86, 2008.

[18] W. Lorensen and H. Cline. Marching Cubes: A high resolution 3D surface construction algorithm. In *Proc. SIGGRAPH*, pages 163–169, July 1987.

[19] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 23(3):457–462, 2004.

[20] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, June 1982.

[21] K. Museth, D. Breen, R. Whitaker, and A. Barr. Level set surface editing operators. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 21(3):330–338, 2002.

[22] K. Museth, D. Breen, R. Whitaker, S. Mauch, and D. Johnson. Algorithms for interactive editing of level set models. *Computer Graphics Forum*, 24(4):821–841, 2005.

[23] M. Nielsen and K. Museth. Dynamic tubular grid: An efficient data structure and algorithms for high resolution level sets. *Journal of Scientific Computing*, 26(3):261–299, 2006.

[24] M. Nielsen, O. Nilsson, A. Söderström, and K. Museth. Out-of-core and compressed level set simulations. *ACM Transactions on Graphics*, 26(4), 2007.

[25] J. Nievergelt and P. Widmayer. Spatial data structures: Concepts and design choices. In *Algorithmic Foundations of Geographic Information Systems*, volume 1340 of *Lecture Notes in Computer Science*, pages 153–197. Springer, Berlin, 1997.

[26] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer, Berlin, 2002.

[27] S. Osher and J. Sethian. Fronts propagating with curvature-dependent

speed: Algorithms based on Hamilton-Jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[28] D. Peng, B. Merriman, S. Osher, H.-K. Zhao, and M. Kang. A PDE-based fast local level set method. *Journal of Computational Physics*, 155:410–438, 1999.

[29] R. Perry and S. Frisken. Kizamu: A system for sculpting digital characters. In *Proc. SIGGRAPH*, pages 47–56, 2001.

[30] C. Reynolds. Big fast crowds on PS3. In *Proc. 2006 ACM SIGGRAPH Symposium on Videogames*, pages 113–121, 2006.

[31] S. Rusinkiewicz and M. Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH*, pages 343–352, 2000.

[32] M. Sainz and R. Pajarola. Point-based rendering techniques. *Computers & Graphics*, 28(6):869–879, 2004.

[33] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[34] J. Sethian. *Level Set Methods and Fast Marching Methods*. Cambridge University Press, Cambridge, UK, second edition, 1999.

[35] M. Teschner, B. Heidelberger, M. Mueller, D. Pomeranets, and M. Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proc. Vision, Modeling and Visualization*, pages 47–54, 2003.

[36] R. Whitaker. A level-set approach to 3D reconstruction from range data. *International Journal of Computer Vision*, 29(3):203–231, 1998.

[37] R. Whitaker. VISPACK. Technical Report UUCS 08-0011, School of Computing, University of Utah, 2008.

[38] G. Wyvill, C. McPheeters, and B. Wyvill. Data structures for soft objects. *The Visual Computer*, 2(4):227–234, 1986.