Dataflow and Remapping for Wavelet Compression and View-dependent Optimization of Billion-triangle Isosurfaces

Mark A. Duchaineau[1], Serban D. Porumbescu[1,3], Martin Bertram[1,2], Bernd Hamann[3], and Kenneth I. Joy[3]

[1] Center for Applied Scientific Computing (CASC), Lawrence Livermore National Laboratory, P.O. Box 808, L-561, Livermore, CA 94551, USA, duchaineau1@llnl.gov

[2] AG Computergraphik und Computergeometrie, Universitt Kaiserslautern, Postfach 3049, D-67653 Kaiserslautern, Germany, bertram@informatik.uni-kl.de

[3] Center for Image Processing and Integrated Computing (CIPIC), Department of Computer Science, University of California at Davis, Davis, CA 95616-8562, USA, {hamann,joy,porumbes}@cs.ucdavis.edu

Summary. Currently, large physics simulations produce 3D discretized field data whose individual isosurfaces, after conventional extraction processes, contain upwards of hundreds of millions of triangles. Detailed interactive viewing of these surfaces requires (a) powerful compression to minimize storage, and (b) fast view-dependent optimization of display triangulations to most effectively utilize high-performance graphics hardware. In this work, we introduce the first end-to-end multiresolution dataflow strategy that can effectively combine the top performing subdivision-surface wavelet compression and view-dependent optimization methods, thus increasing efficiency by several orders of magnitude over conventional processing pipelines. In addition to the general development and analysis of the dataflow, we present new algorithms at two steps in the pipeline that provide the "glue" that makes an integrated large-scale data visualization approach possible. A shrink-wrapping step converts highly detailed unstructured surfaces of arbitrary topology to the semi-structured meshes needed for wavelet compression. Remapping to triangle bintrees minimizes disturbing "pops" during realtime display-triangulation optimization and provides effective selective-transmission compression for out-of-core and remote access to extremely large surfaces. Overall, this is the first effort to exploit semi-structured surface representations for a complete large-data visualization pipeline.

1 Background

Two formerly distinct multi-resolution methods—subdivision-surface wavelet compression, and realtime display through view-dependent optimization—must be integrated to achieve a highly scalable and storage-efficient system

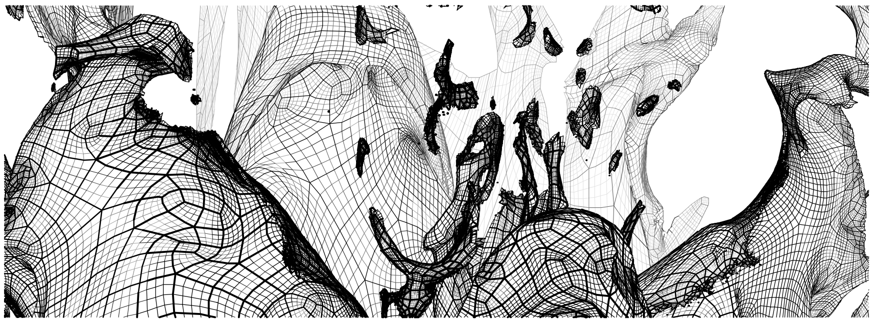

Fig. 1. Shrink-wrap result for .3% of a terascale-simulation isosurface. This conversion from irregular to a semi-regular mesh enables us to apply wavelet compression and view-dependent optimization.

for visualizing the results of recent large-scale 3D simulations. These semi-structured methods have been shown individually to have the highest performance of other respective approaches, but have required a number of missing re-mapping and optimization steps to be effectively scaled to huge data sets and to be linked together. This paper fills in these gaps and demonstrates the potential of the semi-structured approach.

Terascale physics simulations are now producing tens of terabytes of output for a several-day run on the largest computer systems. For example, the Gordon Bell Prize-winning simulation of a Richtmyer-Meshkov instability in a shock-tube experiment [12] produced isosurfaces of the mixing interface consisting of 460 million unstructured triangles using conventional isosurface extraction methods. New parallel systems are three times as powerful as the one used for this run, so billion-triangle surfaces are to be expected. Since we are interested in user interaction, especially sliding through time, surfaces are pre-computed; if they were not, the 100 kilotriangle-per-processor rates for the fastest isosurface extractors would result in several minutes per surface on 25 processors. Using 32-bit values for coordinates, normals and indices requires 16 gigabytes to store a single surface, and several terabytes for a single surface tracking through all 274 time steps of the simulation. This already exceeds the compressed storage of the 3D fields from which the surfaces are derived, and adding additional isolevels or fields per time step would make this approach infeasible. With the gigabyte-per-second read rates of current RAID storage, it would require 16 seconds to read a single surface. A compression factor of 100 with no loss of visual quality would cleanly solve both the storage and load-rate issues. This may be possible with new finite bicubic subdivision-surface wavelets that we introduced in [1]. Methods based on non-finite wavelet decomposition filters [10] are not likely to scale well to the massively parallel co-processing setting that we find critical for the future.

Another bottleneck occurs with high-performance graphics hardware. The fastest commercial systems today can effectively draw around 20 million triangles per second, i.e., around 1 million triangles per frame at 20 frames per second. Thus around a thousand-fold reduction in triangle count is needed. This reduction is too aggressive to be done without taking the interactively-changing viewpoint into account. As the scientist moves close to an area of interest, that area should immediately and seamlessly be fully resolved while staying within the interactive triangle budget. This can be formulated as the optimization of an adaptive triangulation of the surface to minimize a view-dependent error measure, such as the projected geometric distortion on the screen. Since the viewpoint changes 20 times per second, and the error measure changes with the viewpoint, the million-element adaptation must be re-optimized continuously and quickly. The fastest time in which any algorithm can perform this optimization is O(Δoutput), an amount of time proportional to the number of element changes in the adaptation per frame. The Realtime Optimally Adapting Meshes (ROAM) algorithm achieves this optimal time using adaptations built from triangle bintrees [6], implemented for height-map surfaces. Using this approach in a flight-simulation example, it was found that around 3% of the elements change per frame, which resulted in a 30-fold speedup in optimization rates. This is critically important since the bottleneck in fine-grained view-dependent optimizers is processing time rather than graphics hardware rates. In this work we extend the ROAM optmizer to work on general surface shapes formulated as a kind of subdivision-surface.

Wavelet compression and view-dependent optimization are two powerful tools that are part of the larger dataflow from 3D simulation to interactive rendering. These are by no means the only challenging components of a terascale visualization system. For example, conversion is required to turn the irregular extracted surfaces into a semi-structured form appropriate for further processing, see Figure 1. Before turning to our focus on the two remapping steps that tie together the overall dataflow, we provide an overview of the complete data pipeline for surface interaction.

2 End-to-end Multiresolution Dataflow

The processing required to go from massive 3D data to the interactive, optimal display triangulations may be decomposed into six steps:

Extract: Obtain unstructured triangles through accelerated isosurface extraction methods or through material-boundary extraction from volume-fraction data [2]. The extraction for our purposes actually produces two intermediate results, 1) a signed-distance field near the desired surface, along with 2) a polygonal mesh of the surface ready for simplification.

Shrink-wrap: Convert the high-resolution triangulation to a similarly detailed surface that has subdivision-surface connectivity (i.e., is *semi-regular*), has high parametric quality, and minimizes the number of structured blocks. This step has three phases: (1) compute the complete signed-distance transform of the surface from the near-distance field obtained in the Extract step; (2) simplify the surface to a base mesh; and (3) iteratively *subdivide*, *smooth* and *snap* the new mesh to the fine-resolution surface until a specified tolerance is met (see Section 3).

Wavelet compress: For texture storage, geometry archiving, and initial transmission from the massively-parallel runs, use bicubic wavelets based on subdivision surfaces for nearly lossless compression [1]. Good shrink-wrap optimization can make a significant improvement in the wavelet compression performance of this step.

Triangle bintree remap: Remap the shrink-wrap parameterization to be optimal for subsequent view-dependent optimization (this is different than being optimal for high-quality wavelet compression). The bintree hierarchies use piecewise-linear "wavelets" without any vanishing moments, but where most wavelet coefficients can be a single scalar value in a derived normal direction, and where highly localized changes are supported during selective refinement (see Section 4).

Selective decompress: Asynchronously feed a stream of compressed detail where the view-dependent adaptation is most likely to find missing values during selective refinement in the near future. This stream can be efficiently stored in chunks for efficient I/O or network transmission (further details are provided later in this Section).

Display-list optimization: Perform the realtime optimization of the adaptive display-list triangulation each frame, taking maximal advantage of frame-to-frame coherence to accelerate frustum culling, element priority computation, and local refinement and coarsening to achieve the optimum per-view geometry [6].

These six processing steps occur in the order listed for terascale surface visualization. The first three steps—extraction, shrink-wrapping and wavelet compression—typically occur in batch mode either as a co-process of the massively parallel physics simulation, or in a subsequent parallel post-processing phase. Given the selective access during interaction, and given the already existing wavelet hierarchy, the remapping for ROAM interaction can be supported on demand by modest parallel resources. Because the ROAM remapping works in time O(output) in a coarse-to-fine progression, the amount of computation per minute of interaction is independent of the size of the physics grid or the fine-resolution surfaces. The ROAM remapper is envisioned as a runtime data service residing on a small farm of processors and disks, that reads, remaps and feeds a highly compact data stream on demand to the client ROAM display-list optimizer. This server-to-client link could be across a LAN or over high-speed WAN to provide remote access. The ROAM algorithm works at high speed per frame to optimize the display triangulation, and thus will reside proximate to the graphics hardware.

The decompression and remapping services can be abstracted into an asynchronous client-server interface between the ROAM optimizer client running at high frame rates, and a *loader* server performing the reads, remote transmission, decompression, remapping and local transmission. In the ROAM algorithm, refinement and coarsening of the displayed geometry occurs on triangle bintrees as depicted in Figure 2. The unit operation is the *simple split*, wherein two triangles T and its base neighbor T_B are replaced with four children T_0, T_1, T_{B0} and T_{B1}. The key change to the ROAM algorithm is that the simple split operation is allowed to fail in the case that the decompressed/remapped information for the four kids, such as geometry and textures, is not available yet. In that case, the priority of the split becomes temporarily set to zero until an event handler notes the arrival of the required information. Similarly, the chains of simple split operations resulting from a *force split* (depicted in Figure 3), require that the force split fail if any simple split in the chain fails.

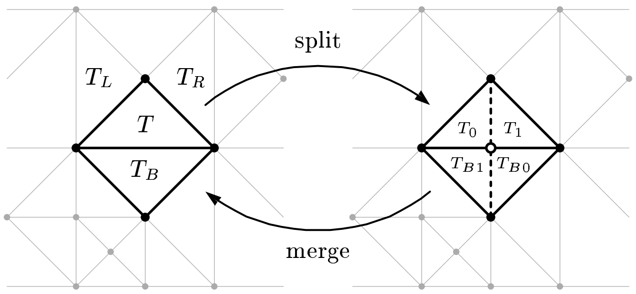

Fig. 2. Simple split and merge operations on a bintree triangulation. A typical neighborhood is shown for triangle T on the left. Splits can fail if the required details have not arrived from the loader.

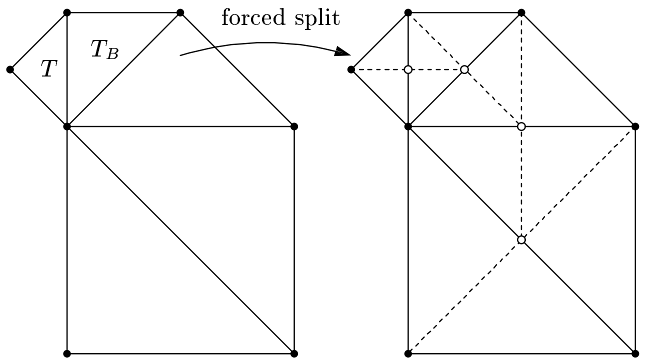

Fig. 3. A chain of simple splits, known as a *force split*, is required during mesh optimization to ensure continuity. Any simple split failure results in failure for the remainder of the chain.

A cache is kept of the remapped/decompressed information (the geometry and textures). A hash scheme is used to access the remapped information, keyed on the unique split-center vertex index. If the required information for the four kids exists, the split operation will succeed in finding them in the cache. Else the split-center key is put on the loader's decompress/remap request queue and the split fails. After a simple split succeeds, the information record for the four kids is taken off the re-use queue. On merge, the four-kid record is placed on the "most recent" end of the re-use queue. To conserve limited memory or graphics hardware resources, the least-recently-used records may be re-used.

The loader performs remapping on any awaiting remap requests in the order received. Upon completion, it generates an event to the ROAM optimizer that the four-kid record should be stored in the cache and made available for future simple splits. The priority of the associated splits are reset to their normal values at this time.

In order for the loading scheme to avoid causing delays to the per-frame processing, the display optimizer does not stop and wait for loads to occur, but continues to perform splits and merges in response to viewpoint changes. The delayed (failed) splits will eventually succeed after some frames, resulting in brief sub-optimal display accuracy but at full frame rates. However, due to frame-to-frame coherence these inaccuracies are expected to be minor and quickly corrected. To avoid the inaccuracies altogether, a form of eager evaluation is included that anticipates future split requirements. It works by putting the loader requests associated with the grandkids on the request queue each time a split occurs. These requests should be given lower priority than the ones immediately needed by currently-attempted split operations. Finally, to avoid stale entries in the loader request queue from accumulating, queue entries may be removed if the request was not repeated on the current frame.

The remainder of this paper focuses on the two remapping steps in this dataflow. The first step prepares a surface for wavelet compression, the second converts the parameterization for ROAM triangle bintree optimization.

3 Shrink-wrapping Large Isosurfaces

Subdivision surfaces originated with the methods of Catmull-Clark [3] and Doo-Sabin [4], with greatly renewed research activity in the last few years. Wavelets for these types of surfaces were first devised by Lounsbery and collaborators [11,7,14], with additional work on wavelets for Loop and Butterfly subdivision surfaces and compression by Khodakovsky *et al.* [10]. Work on finite filters with Catmull-Clark subdivision-surface wavelets was performed by us in Bertram *et al.* [1], which also introduced an early variant on the shrink-wrap procedure described in this section, which is applicable to isosurfaces. The shrink-wrapping method we introduced in [1] was used to demonstrate wavelet transforms of complex isosurfaces in a non-parallel, topology-preserving setting. Here we elaborate on extensions to allow parallel and topology-simplifying shrink-wrapping. An alternate approach to remapping for isosurfaces was presented by Wood *et al.* [15], but appears to be difficult to parallelize or to extend for topology simplification.

Before subdivision-surface wavelet compression can be applied to an isosurface, it must be remapped to a mesh with subdivision-surface connectivity. Minimizing the number of base-mesh elements increases the number of levels in a wavelet transform, thus increasing the potential for compression. Because of the extreme size of the surfaces encountered, and the large number of them, the remapper must be fast, work in parallel, and be entirely automated. The compression using wavelets is improved by generating high-quality meshes during the remapping step. For this, highly smooth and nearly square parameterizations result in the smallest magnitude and count of wavelet coefficient magnitudes, respectively, and yields small outputs after entropy coding. In addition, we would like to allow the new mesh to optionally have simplified topology, akin to actual physical shrink-wrapping of complex 3D objects with a few connected sheets of plastic. The strategy pursued here is to first produce a highly simplified version of the mesh, followed by the re-introduction of detail in a semi-regular refinement through a coarse-to-fine iterative optimization process.

The algorithm uses as input a scalar field on a 3D mesh and an isolevel, and provides as output a surface mesh with subdivision-surface connectivity, i.e., a collection of logically-square patches of size $(2^n+1)\times(2^n+1)$ connected on mutual edges. The algorithm, on a high level, is organized in three steps:

Signed distance transform: For each grid point in the 3D mesh, compute the signed-distance field, i.e., the distance to the closest surface point, negated if in the region of scalar field less than the isolevel. Starting with the vertices of the 3D mesh elements containing isosurface components, the transform is computed using a kind of breadth-first propagation. Data parallelism is readily achieved by queueing up the propagating values on the boundary of a block subdomain, and communicating to the block-face neighbor when no further local propagation is possible.

Determine base mesh: To preserve topology, edge-collapse simplification is used on the full-resolution isosurface extracted from the distance field using conventional techniques. This phase is followed by an edge-removal phase (edges but not vertices are deleted) that improves the vertex and face degrees to be as close to four as possible. Parallelism for this mode of simplification is problematic, since edge-collapse methods are inherently serial using a single priority queue to order collapses.

To allow topology reduction, the 3D distance field is simplified before the isosurface is extracted. Simplification can be performed by using wavelet low-pass filtering on regular grids, or after resampling to a regular grid for curvilinear or unstructured 3D meshes. The use of the signed-distance field improves simplified-surface quality compared to working directly from the original scalar field. To achieve the analog of physical shrink-wrapping, a min operation can be used in place of the wavelet filtering. For each cell in an octree, the minimum of all the child cells is recursively computed. Assuming the solid of interest has values greater than an isolevel, the isosurfaces of the coarse octree levels will form a topologically-simplified surface that is guaranteed to completely contain the fully-resolved solid. This form of simplification is easily parallelized in a distributed setting. It is not clear at this time how to achieve topology preservation and significant distributed-data parallelism.

Subdivide and fit: The base mesh is iteratively fit and the parameterization optimized by repeating three phases: (1) subdivide using Catmull-Clark rules; (2) perform edge-length-weighted Laplacian smoothing; and (3) snap the mesh vertices onto the original full-resolution surface with the help of the signed-distance field. Snapping involves a hunt for the nearest fine-resolution surface position that lies on a line passing through the mesh point in an estimated normal direction of the shrink-wrap mesh. The estimated normal is used, instead of (for example) the distance-field gradient, to help spread the shrink-wrap vertices evenly over high-curvature regions. The signed-distance field is used to provide Newton-Raphson-iteration convergence when the snap hunt is close to the original surface, and to eliminate nearest-position candidates whose gradients are not facing in the directional hemisphere centered on the estimated normal. Steps 2 and 3 may be repeated several times after each subdivision step to improve the quality of the parameterization and fit. In the case of topology simplification, portions of the shrink-wrap surface with no appropriate snap target are left at their minimal-energy position determined by the smoothing and the boundary conditions of those points that do snap. Distributed computation is straightforward since all operations are local and independent for a given resolution.

The distance-field computation uses a regular cubic grid (for unstructured finite element input, a fast 3D variant of triangle scan conversion is used to resample to a regular cubic grid). Each vertex in the 3D grid contains the field value and the coordinates and distance to the closest point found so far, where the distances are initialized to a very large number. There is also a circular array of queued vertex indices (a single integer per vertex), of size no larger than the number of vertices in the 3D grid. This circular array is used as a simple first-in, first-out queue used to order the propagation of distance values through the domain. It is initialized to be empty.

The propagation is started at the isosurface of interest. For the purposes of this paper, material-fraction boundaries are treated one material at a time, using the same formulation as for isosurfaces by considering that material's volume fraction as a field and thresholding it at 50%. All the vertices of all the cells containing the surface are put onto the queue, and their closest points and distances are determined directly from the field values of their cell. These nearest points are computed by approximating the field as a linear function in the neighborhood of the cubic-grid vertex v:

$$\hat{f}(v) = f(v_0) + \nabla f(v_0) \cdot (v - v_0)$$

where we define the gradient at a vertex using central differences:

$$\nabla f(v_0) = \begin{bmatrix} (f(v_0 + d_x) - f(v_0 - d_x))/2 \\ (f(v_0 + d_y) - f(v_0 - d_y))/2 \\ (f(v_0 + d_z) - f(v_0 - d_z))/2 \end{bmatrix}$$

for $d_x = [1,0,0]^T$, $d_y = [0,1,0]^T$ and $d_z = [0,0,1]^T$. The closest point in the local linear approximation will be

$$\hat{v}_c = v_0 + (f_0 - f(v_0)) \frac{\nabla f(v_0)}{\|\nabla f(v_0)\|_2^2}$$

where $f(v)$ if the trilinearly-interpolated field for the cubic grid, and f_0 is the desired isolevel. Optionally, the precise distance to the trilinear field isosurface is obtained by performing Newton-Raphson iterations starting at $\hat{v}_c$, using the gradient and field values from the trilinear interpolant.

The distance-field propagation now proceeds by repeatedly processing the output end of the queue until no queue entries are left. The vertex index i is removed from the queue, and its 26 vertex neighbors are examined. If the current closest point $v_c(i)$ to vertex i is closer to neighbor $v(j)$ than j's current closest point $v_c(j)$, then we set $v_c(j) \leftarrow v_c(i)$, update the current distance $d(j)$, and put j on the input end of the queue if it is not already on the queue. Overall our tests show that a vertex will be updated between about 1.2 and 1.8 times during the propagation, meaning that the algorithm is observed empirically to run in time O(N) for N vertices. The distance transform on the $256 \times 256 \times 384$ volume used for the large example in Figure 1 took 63 seconds to compute on a 1.8GHz Pentium 4 processor with 1GB of 800MHz RDRAM memory.

Note that closest points by definition travel perpendicular to the isosurface in straight lines. This, combined with the fact that the propagation updates are highly localized, leads to a simple and effective means of parallelization. Each processor is assigned a random subset of small blocks (e.g. 64^3 cells) as a passive load-balancing mechanism. Each processor initializes the queue for the vertices near the isosurface on all its blocks. The processing then proceeds as follows. The processor chooses a block with a nonempty queue. It processes all entries of the queue until the queue is empty, and accumulates a list of updated vertices on each of the six block faces. For each face that has updates, a message containing the list of updates is sent to the processor that owns the neighboring block. The processor receives any available update messages, puts the updated vertices on the indicated block queue, and sends an acknowledgement reply message. Also the processor receives any available acknowledgements to its previous update transmissions. If at any time the processor transitions to having all empty block queues and no pending acknowledgements to receive, then it increments a global completion counter. If it transitions out of the completion state (by receipt of further updates), then it decrements the global counter before sending acknowledgements. The entire distance-transform computation is complete when the global counter equals the number of processors. The processor continues with another block if available, otherwise it waits idle for acknowledgements, update messages or global completion.

Once the signed distance field has been computed, a coarse base mesh can be constructed using the octree minima values obtained through a simple, parallelizable, coarse-to-fine recursion on the octree cells. The octree adaptation is determined by the average deviation from the trilinear approximation that the full-resolution data takes on over an octree cell. For example, in a top-down recursive splitting, any octree cell where the average deviation is more than 0.2 times the cell width is split further. Octree cell neighbors are recursively split if they are more than one level coarser than the current cell. After the recursion completes, the faces from the leaf octree cell boundaries form the base mesh. In the case of edges that are on the boundary of octree leaves of different resolutions, the mutual refinement of the edge is used, leading faces to have between four and eight edges (each edge may be split if any edge-neighbor is at finer resolution; neighbors are, by construction, at most one level finer). This base mesh is now ready for the iterative application of smoothing, snapping and refinement to create an optimized fit of the detailed surface.

Smoothing is defined by a simple edge-weighted local averaging procedure, with an optional correction to improve processing in high-curvature regions. The simple weighting formula to update a vertex p is

$$p' = (1 - \alpha)p + \alpha\bar{p}$$

where p is the old vertex position, $\alpha = 0.2$ (approximately), and

$$\bar{p} = \frac{\sum_i r_i p_i}{\sum_i r_i}$$

where p_i are the edge-neighbor vertices of p, and $r_i = \|p_i - p\|_2$ are the edge lengths to them. The smoothing neighborhood is depicted in Figure 4.

This smoothing tends to shrink spherical shapes progressively to a point, which can lead to stability and quality problems in high-curvature regions. An option to correct for this is to project p' in the normal direction n onto a local least-squares-optimal quadratic height map. This map is computed with respect to the estimated normal at p and its tangent plane, passing through p, and otherwise minimizing the least-squares error with respect to the p_i's.

Snapping involves tracing a ray from the current vertex position p, in the normal direction n estimated from the current shrink-wrap mesh neighborhood, and updating the vertex position if a suitable intersection is found on

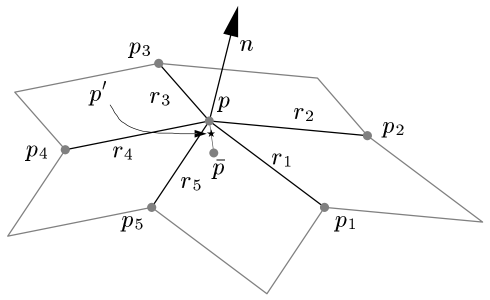

Fig. 4. A neighborhood of vertex p undergoing smoothing. The simple version of smoothing involves edge-weighted averaging with neighbors. An optional correction projects to a local quadratic height map where the height direction is given by the local normal n.

the high-resolution isosurface. As with the smoothing procedure, the normal estimate is computed by averaging the set of cross products of successive edges neighboring p, and normalizing to unit length. Intersections are found in two stages, first by using binary search for zeros of the signed distance field, and later by using Newton-Raphson iterations that again use the distance field. The transition to the Newton-Raphson procedure is attempted at each step of the binary search, and if it fails to converge in five steps then the binary search is continued.

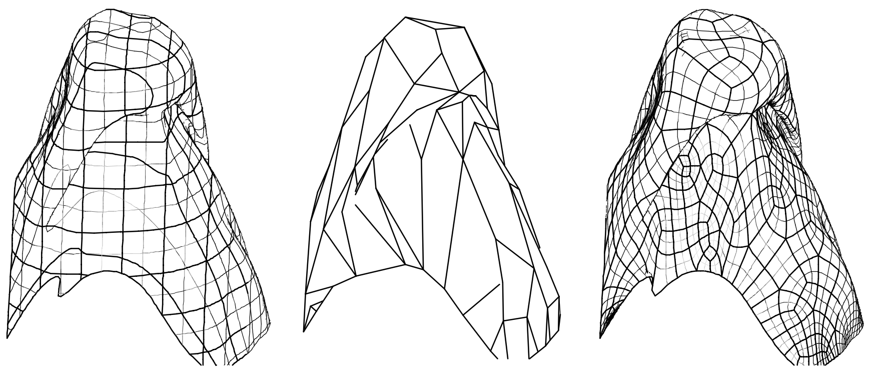

Fig. 5. Shrink wrapping steps (left to right): full-resolution isosurface, base mesh constructed from edge collapses, and final shrink-wrap with subdivision-surface connectivity.

The shrink-wrap process is depicted in Figure 5 for approximately .016% of the 460 million-triangle Richtmyer-Meshkov mixing interface in topology-preserving mode. The original isosurface fragment contains 37,335 vertices, the base mesh 93 vertices, and the shrink-wrap result 75,777 vertices. A larger portion of the same isosurface is shown in Figure 1. The extracted, full resolution unstructured mesh contains 976,321 vertices, the base mesh 19,527 vertices, and the fine-resolution shrink-wrap mesh 1,187,277 vertices. The shrink-wrap refinement process in both cases was continued until the fit was accurate to a tiny fraction of a cell width. Indeed, the fit surface takes on the artifacts of the isosurface extraction process, slight "staircase" patterns at the grid resolution.

The shrink-wrap processing succeeded in our goal of creating a parameterization that results in high compression rates. With only 1.6% of the wavelet coefficients, the root mean squared error is a fraction of the width of a finite element in the grid, implying effective compression even before considering improvements through quantization and entropy coding.

4 Remapping for ROAM

The ROAM algorithm typically exploits a piecewise block-structured surface grid to provide efficient selective refinement for view-dependent optimization. A triangle bintree structure is used, with split and merge operations for selective refinement and coarsening respectively, as described in Section 2 (see Figures 2-3). In this section we introduce a second re-mapping procedure that solves the problem of *tangential motion* that arises when performing selective refinement on the "curvy" parameterizations that result from the shrink-wrapping algorithm.

A simple demonstration of the problem is shown in Figure 6. Note that even for a perfectly planar region, significant "pops" will occur during selective refinement due to the nonlinear parameterization. A natural choice for remapping is to project the original surface geometry in an estimated normal direction onto the coarse bintree triangles acting as the new domain. Occasionally the projection causes excessive distortions or folds, but typically it can be applied without problems. When projection can be applied, it reduces the new split-point updates to all be in the local "height" or normal direction, and can thus completely avoid tangential motion. Indeed, this can be seen as a form of compression, since only a single scalar Δ-height need be stored instead of a 3-component vector.

Although it would be simpler to have a single type of mapping, the local-heightmap scheme for triangle bintrees is not optimal for bicubic subdivision-surface wavelet compression, and so we choose to keep two distinct remappings. The curved parameterizations are ideal for the higher-order wavelet compression. The ROAM-optimized mappings often appear jagged and less pleasing to a human observer *who is looking at the mesh lines*, but nevertheless are much better performing for actual use during view-dependent display, where the mesh lines are *not* shown.

Several lines of previous research serve as a backdrop for the methods we have devised. The earliest known local hierarchical displacement frames were introduced by Forsey and Bartels in the context of hierarchical B-spline

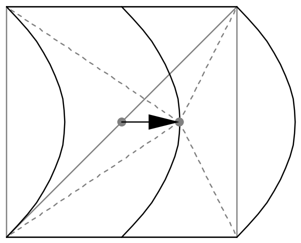

Fig. 6. The highly curved parameterization from shrink-wrapping is shown in black, whereas the natural parameterization for triangle bintrees is shown in gray. Without remapping, the large tangential motion (arrow) will occur when splitting.

editing [8]. A more general method, including integration with wavelets, was introduced in [5]. The recent work of Guskov *et al.* [9] has many similarities to our approach, but in the context of geometric modeling and compression rather than view-dependent display optimization.

The ROAM remapping works from coarse to fine resolutions, following the simple split operations in whatever order they appear during selective refinement. The vertices of the base mesh are left fixed at their original positions, and are stored as full three-component coordinates. For every edge-bisection vertex formed in one simple split step, estimated normals are computed by averaging the normals of the two triangles whose common edge is being bisected, and making the averaged normal unit length. All vertices carry with them a reference to the $2^n + 1$-squared patch that they reside in, along with the two parameter coordinates $[u, v]$ within that patch. During edge bisection, the *parametric midpoint* is computed by "topologically gluing" at most two patches together from the original mesh, computing the mid-parameter values in this glued space, then converting those parameters back to unglued parameters. The gluing and remapping is depicted in Figure 7.

Given the constraints on our procedure, it is not possible for bisecting-edge endpoints to cross more than one patch boundary. A ray-trace intersection is performed from the midpoint of the line segment being bisected, in the estimated normal direction. Since we expect the intersection to be near the parametric midpoint in most cases, it is efficient to begin ray intersection tests there for early candidates.

Since the surface being ray-traced stays fixed throughout the remapping, the construction of typical ray-surface intersection-acceleration structures can be amortized and overall offer time savings. However, we have found that, overall, the hunt using the parametric midpoint as a starting point is very fast in practice and further acceleration methods are not needed. For the final precise intersection evaluation, we use interval-Newton iterations [13]. Intersections are rejected if they are not within a parametric window defined by the four remapped vertices of the two triangles being split, shrunk by some factor (e.g., reduced by 50%) around the parametric midpoint. The

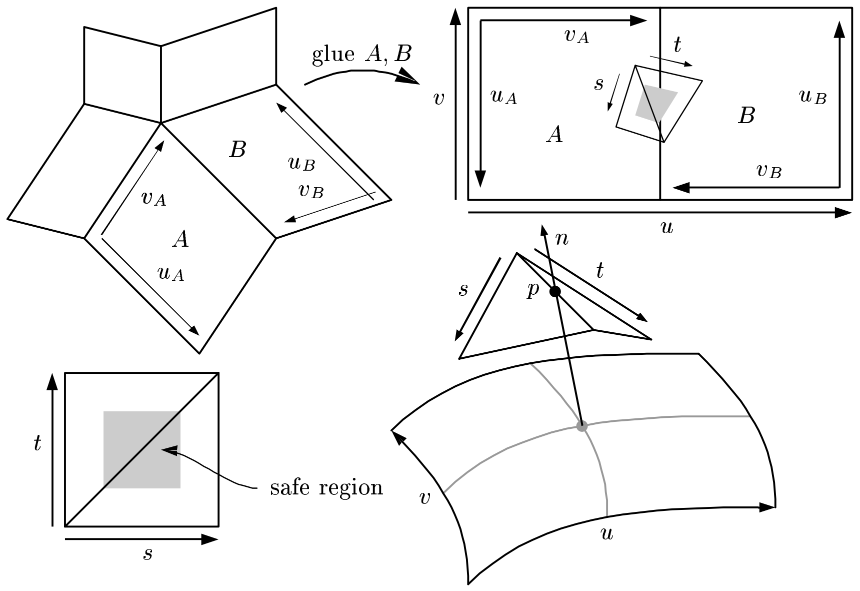

Fig. 7. Two patches parameterized by $[u_A, v_A]$ and $[u_B, v_B]$ are glued to form a single patch parameterized by $[u, v]$. A ray-tracing procedure finds the intersection of the ray through midpoint p in the estimated normal direction n. The $[u, v]$ of the intersection is acceptable if it appears in the image of the safe region of $[s, t]$-space in the glued space.

closest acceptable intersection is chosen (closest with respect to the distance from p in world space). If an acceptable intersection exists, then the single scalar value is given specifying the distance from p along the normal ray n. If no intersections exist or are acceptable, then parametric midpoint is chosen and is specified with a three-component vector of the displacement from edge midpoint p to the parametric midpoint.

The result of remapping is shown in Figure 8 for a test object produced by Catmull-Clark subdivision with semi-sharp features. The original parameterization on the left is optimal for compression by bicubic subdivision-surface wavelets, but produces extreme and unnecessary tangential motions during triangle-bintree refinement. The remapped surface, shown on the right, has bisection-midpoint displacements ("poor man's wavelets") of length zero in the flat regions of the disk, and displacements of minimized length elsewhere. We note that while the main motivation for this procedure is to increase accuracy and reduce the "pops" during realtime display-mesh optimization, the typical reduction to a single scalar value of the displacement vectors (wavelet coefficients) gives a fair amount of compression. This is desirable when the remap from high-quality wavelet parameterization and compression is too time-consuming to allow some clients to perform it themselves, so that a

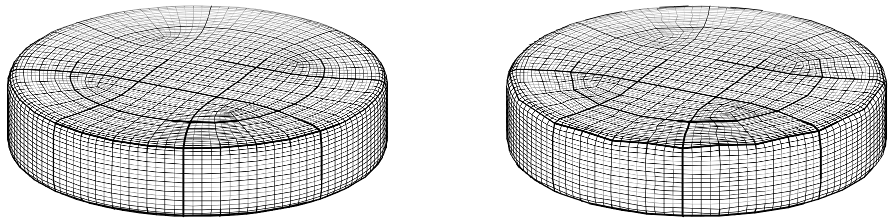

Fig. 8. Before (left) and after remapping for triangle bintree hierarchies. Tangential motion during subdivision is eliminated.

more capable server performs the remap yet there is still a fair amount of compression over the server-to-client communications link.

Generally the ROAM remapping decreases the error for a given level of approximation. However, tiny discrepancies exist in the finest resolution mesh due to remapping. It is certainly possible to continue remapping beyond the resolution of the input mesh, in effect fully resolving the exact artifacts and other details of the original mesh. In most applications this is not necessary.

The ROAM algorithm naturally requires only a tiny fraction of the current optimal display mesh to be updated each frame. Combined with caching and the compression potential of remapping, this promises to provide an effective mechanism for out-of-core and remote access to the surfaces on demand during interaction.

5 Future Work

We have demonstrated, for the first time, the end-to-end use of semi-structured representations for large-data compression and view-dependent display. The primary lesson learned is that two remapping steps are needed for best performance, rather than one, and that a runtime on-demand caching scheme is needed to best perform the second remapping. Overall the software is still very much a research prototype with many challenges remaining to create a production-ready system:

1. For topology-preserving simplification, the inherently serial nature of the queue-based schemes must be overcome to harness parallelism.
2. Means such as transparent textures must be devised to handle the non-snapping shink-wrap regions that result from topology simplification.
3. The shrink-wrapping procedure can fail to produce one-to-one, onto mappings in some cases even when such mappings exist. Perhaps it is possible to revert to expensive simplification schemes that carry one-to-one, onto mappings only in problematic neighborhoods.
4. Shrink-wrapping needs to be extended to produce time-coherent mappings for time-dependent surfaces. This is a great challenge due to the complex evolution that surfaces undergo in time-dependent physics simulations.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by University of California Lawrence Livermore National Laboratory under contract No. W-7405-Eng-48. We thank LLNL for support through the Science and Technology Education Program, the Student-Employee Graduate Research Fellowship Program, the Laboratory-Directed Research and Development Program, the Institute for Scientific Computing Research, and the Accelerated Strategic Computing Initiative.

In addition, this work was supported by the National Science Foundation under contract ACI 9624034 (CAREER Award), through the Large Scientific and Software Data Set Visualization (LSSDSV) program under contract ACI 9982251, and through the National Partnership for Advanced Computational Infrastructure (NPACI); the Office of Naval Research under contract N00014-97-1-0222; the Army Research Office under contract ARO 36598-MA-RIP; the NASA Ames Research Center through an NRA award under contract NAG2-1216; the Lawrence Livermore National Laboratory under ASCI ASAP Level-2 Memorandum Agreement B347878 and under Memorandum Agreement B503159; the Lawrence Berkeley National Laboratory; the Los Alamos National Laboratory; and the North Atlantic Treaty Organization (NATO) under contract CRG.971628. We also acknowledge the support of ALSTOM Schilling Robotics and SGI. We thank the members of the Visualization and Graphics Research Group at the Center for Image Processing and Integrated Computing (CIPIC) at the University of California, Davis.

References

1. Martin Bertram, Mark A. Duchaineau, Bernd Hamann, and Kenneth I. Joy. Bicubic subdivision-surface wavelets for large-scale isosurface representation and visualization. *Proceedings of IEEE Vis00*, October 2000.
2. Kathleen S. Bonnell, Daniel R. Schikore, Kenneth I. Joy, Mark Duchaineau, and Bernd Hamann. Constructing material interfaces from data sets with volume-fraction information. *Proceedings of IEEE Vis00*, October 2000.
3. E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-Aided Design*, 10:350—355, September 1978.
4. D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10:356—360, September 1978.
5. Mark A. Duchaineau. *Dyadic Splines.* PhD thesis, University of California, Davis, 1996.
6. Mark A. Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich, and Mark B. Mineev-Weinstein. ROAMing terrain: Real-time optimally adapting meshes. *IEEE Visualization '97*, pages 81–88, November 1997. ISBN 0-58113-011-2.
7. Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In

Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 173–182. ACM SIGGRAPH, August 1995.

8. David R. Forsey and Richard H. Bartels. Hierarchical b-spline refinement. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, volume 22, pages 205–212, Atlanta, Georgia, August 1988.
9. Igor Guskov, Kiril Vidimce, Wim Sweldens, and Peter Schröder. Normal meshes. *Proceedings of SIGGRAPH 2000*, pages 95–102, July 2000.
10. Andrei Khodakovsky, Peter Schröder, and Wim Sweldens. Progressive geometry compression. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, Annual Conference Series, pages 271–278. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
11. Michael Lounsbery. *Multiresolution Analysis for Surfaces of Arbitrary Topological Type*. PhD thesis, Dept. of Computer Science and Engineering, U. of Washington, 1994.
12. Arthur A. Mirin, Ron H. Cohen, Bruce C. Curtis, William P. Dannevik, Andris M. Dimits, Mark A. Duchaineau, Don E. Eliason, Daniel R. Schikore, Sarah E. Anderson, David H. Porter, Paul R. Woodward, L. J. Shieh, and Steve W. White. Very high resolution simulation of compressible turbulence on the IBM-SP system. *Supercomputing 99 Conference*, November 1999.
13. Tomoyuki Nishita, Thomas W. Sederberg, and Masanori Kakimoto. Ray tracing trimmed rational surface patches. *Computer Graphics (SIGGRAPH '90 Proc.)*, 24(4):337–345, August 1990.
14. Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. *Wavelets for Computer Graphics: Theory and Applications.* Morgann Kaufmann, San Francisco, CA, 1996.
15. Zoë J. Wood, Mathieu Desbrun, Peter Schröder, and David Breen. Semi-regular mesh extraction from volumes. In T. Ertl, B. Hamann, and A. Varshney, editors, *Proceedings Visualization 2000*, pages 275–282. IEEE Computer Society Technical Committee on Computer Graphics, 2000.