DOI: 10.1111/j.1467-8659.2011.01945.x

Eurographics / IEEE Symposium on Visualization 2011 (EuroVis 2011)
H. Hauser, H. Pfister, and J. J. van Wijk
(Guest Editors)

Volume 30 (2011), Number 3

Fast Extraction of High-quality Crease Surfaces for Visual Analysis

S. Barakat[1], N. Andrysco[1], and X. Tricoche[1]

[1]Purdue University, USA

Abstract

We present a novel algorithm for the efficient extraction and visualization of high-quality ridge and valley surfaces from numerical datasets. Despite their rapidly increasing popularity in visualization, these so-called crease surfaces remain challenging to compute owing to their strongly nonlinear and non-orientable nature, and their complex boundaries. In this context, existing meshing techniques require an extremely dense sampling that is computationally prohibitive. Our proposed solution intertwines sampling and meshing steps to yield an accurate approximation of the underlying surfaces while ensuring the geometric quality of the resulting mesh. Using the computation power of the GPU, we propose a fast, parallel method for sampling. Additionally, we present a new front propagation meshing strategy that leverages CPU multiprocessing. Results are shown for synthetic, medical and fluid dynamics datasets.

Categories and Subject Descriptors (according to ACM CCS): I.3.3 [Computer Graphics]: Picture/Image Generation—Line and curve generation

1. Introduction

Ridge and valley manifolds have recently seen their popularity increase noticeably in visualization. Their traditional application in computer vision and medical image analysis has found a natural extension in scientific visualization where they offer a compelling means to succinctly describe the geometric core structure of complex three-dimensional fields. Conceptually, ridges (and equivalently valleys) can be seen as dual to isosurfaces, which are typically used to capture the outer shell of objects embedded in a domain. These surfaces are generally non-orientable and they possess a complex set of boundaries. Hence, their extraction from a 3D scalar field sampled on a grid is significantly more demanding than a standard level set computation. As a result, algorithms that have proved successful for the extraction of isosurfaces yield disappointing results, in terms of accuracy, geometric quality, and computational efficiency when applied to ridge and valley surfaces.

Despite the important contributions made in the visualization literature to address this problem, significant limitations remain that prevent the broad adoption of ridge extraction techniques in the toolbox of visualization practitioners. First, the techniques proposed to date are rather slow and they typically trade approximation quality for speed, which complicates the interpretation of the resulting models. Second, the quality of the extracted meshes has been mostly neglected in previous work. This stands in contrast to the significant research effort dedicated to isosurface (re)meshing. Moreover, a mapping of ridge surface extraction methods to the graphics hardware is missing so far, due in part to the difficulties posed by an efficient GPU implementation of the reconstruction kernels needed to compute derivatives during the extraction.

This paper presents a fast method that addresses these limitations through a novel hybrid CPU/GPU approach for both the sampling and meshing of crease surfaces. We combine a generalized front propagation strategy that shares similarities with mesh reconstruction techniques with a carefully managed parallel organization of the computation that leverages the GPU's compute power. Specifically, our parallel front propagation meshing algorithm exploits the multithreading and multiprocessing available both on the GPU and the CPU to advance multiple fronts at once while merging them into high quality meshes. This solution is suitable for arbitrarily complex surfaces, it ensures the approximation and geometric quality of the produced triangulation and

it achieves a significant performance increase compared to previous techniques. As such, we believe that beyond ridges the proposed approach will benefit the extraction of other types of complex surfaces, such as level sets in nonlinear fields. Indeed, our experimentation with synthetic, medical, and engineering datasets confirms that our new technique significantly outperforms existing methods in terms of quality and efficiency. In addition, our implementation accounts for the scale space nature of ridge surfaces thus producing high quality meshes even in noisy or highly convoluted datasets.

2. Previous Work

Ridges are fundamental features in image analysis [EGMP94, PEMF98] that generalize the notion of point-wise extrema (where the gradient vanishes) of smooth scalar fields to objects of higher dimensions (e.g., curves and surfaces). Ridges are typically characterized as a set of points where a scalar function f in dimension N is maximized in a set of $p < N$ independent directions, thus forming a set of dimension $n = N - p$. The *height ridge* definition proposed by Eberly [EGMP94] defines the local coordinate frame in terms of the eigenvectors of the Hessian matrix $\mathbf{H} = \nabla^2 f$ (second-order derivative of f) associated with the p smallest eigenvalues $\lambda_1 < ... < \lambda_p$ with the additional requirement that these eigenvalues be negative. *Valleys* of f are similarly defined as ridges of $-f$. Spurious structures can be filtered out based on the value f and the *crease strength* $|\lambda_p|$.

Ridge surfaces in 3D ($n = 2$, $p = 1$) are often extracted as 0-level sets of the scalar product between gradient ∇f and smallest eigenvector $\mathbf{e}_2$ of $\mathbf{H}$ [TG92, FP01]. An alternative (piecewise) level-set description of ridge and valley manifolds based on a signed scalar field was proposed by Peikert and Sadlo [PS08] who also proposed a filtered ridge extraction based on adaptive mesh refinement [SP07]. The detection of the zero crossings of this scalar product requires careful inspection of the edges and faces of the voxels since the considered eigenvector field has neither norm nor intrinsic orientation [SP99, KTW07]. Similar ideas form the basis of the work by Li *et al.* [LLP*10] who introduced a grid-based algorithm for constructing polygonal approximations of extremal surfaces. A set of topological principles to improve both correctness and performance of voxel based ridge surface extraction methods was proposed by Schultz et al. [STS10]. Observe that these various methods can produce artifacts and low quality meshes, due to the geometric and topological complexity of crease surfaces. Increasing the resolution of the input volume alleviates some of these problems but it does so at the expense of high computational and memory requirements that add to the limitations of these techniques. Remarkably, none of these algorithms takes advantage of the GPU.

Ridges were recently applied to vector [TSW*05, SWH05] and tensor visualization [KTW06, KTW07, TKW08, STS10] problems through their computation in scalar invariants of vector and tensor fields. Following a different strategy, Kindlmann et al. [KSSW09] introduced a particle-based framework that optimally samples locations in scale space. The method produces a quasi uniform sampling of crease manifolds in output. However, it does not produce a mesh and is computationally involved.

Finally various sampling and meshing algorithms have been proposed over the last decade. Discussing all these contributions is beyond the scope of this paper and we focus on front propagation techniques with both error and quality control. Amenta and Bern [BO05] introduced the notion of $\varepsilon - sample$, a key concept in the theory of sampled surfaces, which is used by different meshing algorithms. Amenta *et al.* [ABK98] proposed a technique for the reconstruction of closed surfaces from unorganized sample points with bounds on errors provided that a sufficient number of samples are available. Alexa at el. [ABCO*01] presented a technique for the construction and rendering of smooth manifold using point sets. Scheidegger, Schreiner, and their coauthors [SFS05, SSFS06] contributed a bounded error triangulation algorithm able to filter noisy input data and provide good quality triangles independent of the sampling density. These authors applied this algorithm for high-quality extraction of isosurfaces [SSS06]. Note that all these methods are primarily concerned with the meshing aspect of the problem. They assume a pre-existing set of sample points, and they typically have limited scalability.

3. Challenge and Proposed Solution

Surface meshing requires careful sampling, whereby the sampling rate is usually selected conservatively based on the maximum expected curvature of the surface [BO05]. A subset of these points are subsequently connected to form a mesh. Mesh generation research has devised a theory that provides a relationship among the minimum number of mesh vertices, the curvature of the surface, and the resulting mesh quality. To accurately compute properties (*e.g.* smoothness and boundaries) of a complex surface, considerably more samples are required than those that will actually contribute to the final mesh geometry. As the accuracy becomes more of a concern (*e.g.* fast variation in curvature, thinner gaps between boundaries, *etc.*), the difference between the number of required samples widens. In previous meshing techniques, these large number of property samples would typically be passed to the triangulation algorithms even though most of the samples would not appear in the final mesh. Maintaining this burdensome number of samples requires a significant computation and memory overhead, something we are able to avoid with the ability to discard unneeded samples on the fly.

When crease surfaces are considered, a different strategy is therefore needed that ties together sampling and mesh-

ing more naturally. We propose a progressive cycle of sampling and meshing, with an error bounding criteria. Dense sampling, while still required by our approach, poses fewer challenges, in both computation (due to our massively parallel algorithm) and memory (the ability to efficiently discard unneeded samples on GPU), compared to past approaches. This strategy lends itself to an efficient implementation on modern architectures, especially on GPUs, where computation is cheap while memory access can often become an algorithm's bottleneck.

4. Algorithm

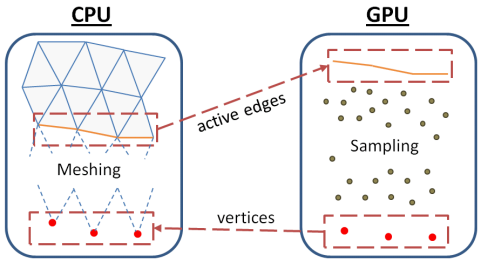

Figure 1: *Interaction diagram between CPU and GPU.*

Our method starts with a scale space analysis that selects at each spatial location the scale value that maximizes a measure of crease strength along the scale axis, as suggested by Lindeberg [Lin98]. Practically, we use the reconstruction framework proposed by Kindlmann *et al.* [KTW07, KSSW09] to create a smooth 4D continuum in which we carry out these one-dimensional searches, similar to a method recently described in our previous work [BT10]. The gradient, Hessian, and all other relevant quantities are then computed in pre-processing at each domain position at that optimal scale.

Next, our method identifies a set of seed triangles forming the initial fronts of the mesh as described in section 4.1. The mesh propagation algorithm iteratively selects a set of front edges to advance, sends a request for new vertices to the GPU, and adds the returned vertices to the mesh according to the strategy explained in section 4.2 (see figure 1). The GPU's role is to perform dense one-dimensional surface sampling along the front edges until the needed vertices are found according to the process described in section 4.3. The main purpose of the sampling is to track the distance between the surface and the mesh as a measure of error.

4.1. Seed Triangles

Seed triangles are used as an initial front from which we grow new triangles that approximate the underlying surface. These initial triangles are automatically computed by meshing in screen space the points obtained by ray-surface intersections. These intersections are computed using the ray casting approach we proposed in [BT10]. Depth discontinuities are used to determine patch boundaries. Applied along each coordinate direction and combined with depth peeling [Eve01], this scheme produces a set of (typically redundant) piecewise descriptions of each connected component of the surface. Next, our algorithm randomly selects a number of seed triangles from each connected component of that mesh based on its size. We verify that each triangle belongs to a patch of the surface by taking additional samples inside that triangle to ensure that it does not bridge two different components. An alternative approach is to let the user select triangles by clicking on features (*e.g.* bones or skin in medical datasets) in the ray casting interface.

4.2. Advancing Front Mechanism

We adopt the definition of the meshing problem proposed by Schreiner *et al.* [SSFS06], which we introduce as follows. Given an input surface S defined through a projection operator $P : \mathbb{R}^3 \to S$, we want to construct a triangulation Σ such that the Hausdorff distance between Σ and S is bounded by ε. We also want to control the number of outputted triangles and their geometric quality. As different fronts are grown on the surface, they will eventually encroach and be connected to one another. Therefore, our front propagation algorithm gives priority to accuracy over triangle quality in order to minimize artifacts in between these fronts. Note that many meshing algorithms provide estimated error bounds based on local curvature under the assumption of a smoothly varying surface normal, an assumption rarely valid for highly curved crease surfaces. Indeed, these methods construct edges that yield high-quality triangles rather than track the distance between the mesh and the actual surface satisfied with the smoothness assumption.

Unlike conventional front propagation meshing schemes (*e.g.* [SSFS06]), our method selects a large number of front edges to grow simultaneously in order to exploit the power of massively parallel machines. However, this strategy introduces several challenges from a meshing perspective since mesh decisions are no longer made on a one by one basis but rather as a batch. As seen in figure 2 (b) and (c), propagating fronts in parallel may result in two triangles overlapping as they try to simultaneously approximate the same area of the surface, also known as a *conflict*. The detection of a conflict is simplified knowing that the distance between any edge and the surface should be less than the provided error bound, ε. In short, two triangles are said to conflict if they do not share a common vertex and their edges are less than ε apart. When this problem occurs, we simply discard one of the triangles and create an active edge between the closest pair of vertices from both fronts, as can be seen in figure 2 (c).

Active edges are the edges that are used to grow new triangles. We keep a pool of all active edges as a representation of our fronts. Different triangles are produced at the same time in parallel on the GPU. Every edge is sent to the GPU ac-

companied with an *advancing direction vector*. The search for the new vertex starts at the midpoint of the active edge following the direction of that vector. An iterative process takes place to gradually advance the vertex and correct the direction.

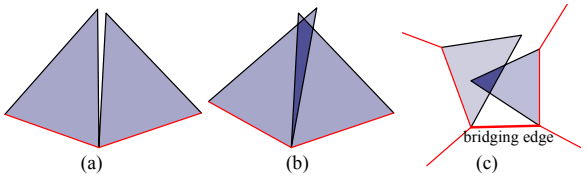

Figure 2: *In (a) and (b), we show cases that are avoided by edge locking, while (c) shows a conflict between triangles of different fronts.*

We first find an advancing direction perpendicular to the edge. If the active edge is part of an existing triangle, we use the perpendicular to the active edge in the triangle plane but in the opposite direction of that triangle. This is illustrated by the blue vector in figure 3 (b). Note, this ensures the search for the new vertex starts at the furthest direction from the existing triangle. Figure 3 (a) highlights the case when an edge is not part of any triangle (*i.e.* a bridging edge between two different fronts). For this, we simply use any perpendicular from a non-shared vertex of a neighboring active edge.

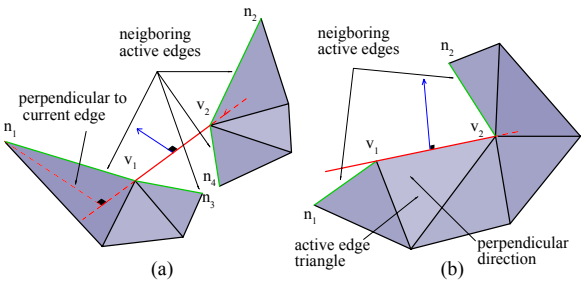

Figure 3: *(a) A bridging active edge and (b) an active edge that is part of an existing triangle. Neighbor active edges are shown in green. The dashed red lines are examples of possible perpendicular advancing directions.*

The next step is to adjust the advancing direction based on the expected triangle. Our strategy consists of passing a search direction to the GPU that heuristically reduces the possibility of a conflict between active edges. Fig. 4 illustrates the various cases we may encounter, where the blue vectors show the result of computing the advancing direction. The ideal case is shown in Fig. 4 (a), where one of the neighboring active edges (green) forms a sharp angle with the current edge (red) and both of the neighboring edges' far vertices can be connected with a third edge (pink) to form an edge that is shorter than the maximum edge length η. For this situation, we simply set the advancing direction to be the direction toward the corresponding neighbor vertex. If both neighboring active edges satisfy these conditions, then we choose the neighboring vertex having the closest distance to the center of the current edge. Fig. 4 (b) demonstrates when both neighboring active edges (green) meet the sharp angle criteria, but neither edge (pink) can form a triangle due to the maximum edge length constraint. For this case, we use the direction toward the middle point of the neighbor vertices. In (c), where only one angle is sharp and the corresponding neighbor vertex cannot be reached with an edge shorter than η, we split the angle between the edges with an edge of length η (orange). We then set the direction toward the point at the far end of that edge. If none of the previous situations apply, we simply use the perpendicular direction as the search direction. The rationale for these decisions is that smaller angles leave little room for overlapping with other triangles. They also have limited connection options. If our attempt failed and the new returned triangle produced a conflict, we simply discard it.

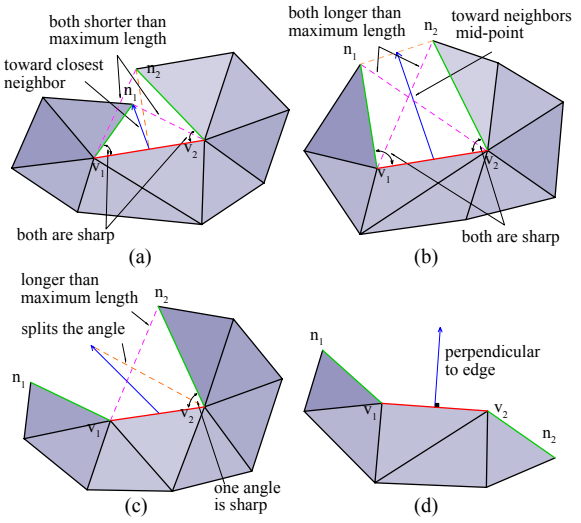

Figure 4: *Examples of various cases that are encountered while computing the advancing direction.*

To further prevent conflicts we do not send two neighboring active edges forming an angle less than 180° in the same GPU batch. We do that by marking the vertices of each edge added to the batch. Thin triangles also might occur if locking was not used (see figure 2). When new vertices are returned, we give priority to active edges that had sharp angles with its neighboring edges. Their corresponding triangles are added first to the mesh in order to avoid thin triangles. Notice that if the new vertex failed to reach a neighboring target vertex we simply use the new vertex favoring accuracy over triangle quality.

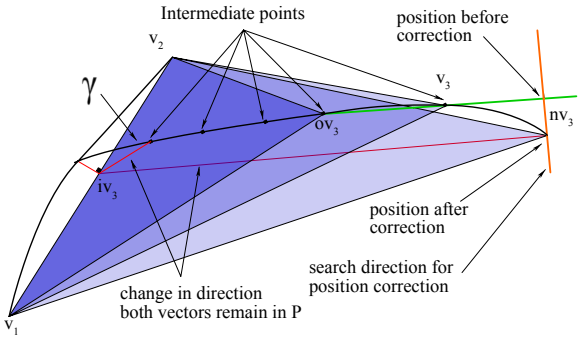

Figure 5: *A triangle growing over successive iterations.*

4.3. Triangle Generation

Every active edge is sent to the GPU in the form of two vertices, v_1 and v_2, in addition to the initial search direction, *dir*. The midpoint of the edge is iv_3. Let T be the triangle plane and P be a plane s.t. $P \perp T$, $iv_3 \in P$, and $P \| dir$. The new vertex nv_3 should satisfy $nv_3 \in P$ and $nv_3 \in S$.

As an initialization step, the vertex nv_3 is set to iv_3. An iterative process is then used in order to gradually advance the point until the maximum edge length η or maximum error ε is reached. In every iteration we first advance the point along the line joining its last two positions for a small step τ. τ is a fraction of ε to ensure that we are sampling the distance between the surface and the triangle (*i.e.* sampling the error) at the appropriate rate.

We next correct the new point position to guarantee that it belongs to S as well. The correction of the vertex position is done along the normal of the triangle plane using the search technique mentioned in [BT10]. The search is however bounded to a distance equivalent to the voxel edge length around the point. Once the corrected location of nv_3 is found, the error is evaluated and the error map is updated according to the process described in section 4.4. The process of acquiring a new vertex is computationally expensive as it requires searching in two directions.

4.4. Error Tracking

As the triangles grow, we must ensure that they provide an accurate approximation of the surface by tracking the error between the actual surface, S, and the considered triangle. The maximum distance between S and the triangle must not exceed our set error tolerance ε. Solving the error tracking problem in the two dimensional plane P (from section 4.3) is a close approximation for the error between the triangle and the surface since we know that all three points of the resulting triangle reside on the surface and that the plane goes through the middle of one edge to the point facing that edge. Even if we assume that the surface is highly irregular, such that the error could be growing toward the other two edges,

Algorithm 1: Pseudocode for triangle generation

Input: Edge points v_1 and v_2, advancing direction *dir*
Output: Third point on triangle v_3, and *boundary* flag

```
dir := normalize(dir)
edge := v2 - v1
c := normalize(cross(edge, dir))
i := 0
while i < max do
    Step 1. Find the new vertex
    if i == 0 then
        nv3 := iv3 + τ . dir
    if i > 0 then
        nv3 := v3 + τ . normalize(v3 - ov3)
    Step 2. Correct the new vertex position
    boundary := CorrectVertex(Normal(v1,v2,v3), nv3)
    if boundary == 1 then
        return;
    Step 3. Find the new length and error
    ln := max(|nv3 - v1| , |nv3 - v2|)
    e := GetError(nv3, iv3, dir, c, distmap)
    Step 4. Check the constraints
    if ln > η then
        return;
    if e > ε then
        return;
    Step 5. Update the error distance map
    UpdateErrorMap(iv3, ov3, v3, nv3, dir, c, distmap)
    Step 6. Prepare for next Step
    ov3 := v3
    v3 := nv3
    i := i + 1
```

the neighboring triangles should compensate for this effect. Otherwise this high curvature feature would have a spatial scale smaller than the local size of the triangles. Additionally, we choose ε to be very small ($\frac{1}{6}$ of a voxel edge length in all our experiments) to make sure that variations inside a single triangle are limited [AB98].

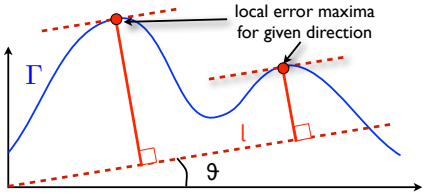

Figure 6: *A 2D slice of the surface is shown, Γ. The maximum distance between Γ and the line, l, can only occur where the tangent of Γ is parallel to l.*

We wish to evaluate the error continuously along a curve Γ, of the surface S. A straightforward, but inefficient, solution would be to save the points along the search direc-

tion. The error is then evaluated by finding the maximum distance between any point and the line l. However, saving and retrieving a large number of points along the way would cancel out any benefit gained from running the extraction in parallel on the GPU because of the GPU's limited memory throughput. It is important to note that the point of maximum distance on the curve Γ from the line l must occur where the tangent of Γ is parallel to l, as shown in Fig. 6. Using this idea, we can limit the amount of information to track.

The example provided is simply a 2D slice, but of course our problem is 3D. Because of this, we also need to keep a record of the maximum distances per angle. Using what we call an *error map*, we are able to discretize the angular domain. The error map is simply an array where each entry is the maximum error of each angular sample. For each sample point on the surface, we can approximate the error by interpolating between our entries in the error map. After we evaluate the error for our newly sampled point, we update the angular entries in the map that fall between the tangential slopes of this new point and the previous point. This will ensure that future interpolations between map entries will provided an accurate approximation. We found that 18 samples, or one sample every 10°, was enough to obtain sufficient accuracy.

During the extraction process of a new vertex, the search might hit the boundary of the crease surface. At this point, the search fails to correct the vertex position along the normal. The search will then backtrack to the previous point on the surface. We note that a point may falsely detect a boundary when it hits a tiny hole on the surface. This situation does not represent a discontinuity on the surface, but rather a noise consequence. For this scenario we exclude an edge from the front only if both of its vertices are on a boundary. This solution also helps straighten the boundaries.

In summary, the process of acquiring a new vertex to complete a triangle is a form of excessive surface sampling used to capture the surface's complex properties. In practice, the intermediate steps can exceed a few dozen iterations for each new vertex. Each iteration does not complicate the meshing process or increase the memory overhead, but rather the iterations are mainly used to draw conclusions about the surface's properties.

5. Implementation

In contrast to marching cubes methods, front advancing algorithms provide higher flexibility for the meshing decisions, which enables better quality. However, the existing front advancing schemes are poorly suited for parallel implementation due to their reliance on the linked lists to represent fronts [BMR*99, SFS05, SSS06]. These representations are also problematic in the case of complex surfaces since fronts might be numerous. Our implementation instead uses a queue of active edges and keeps track of corresponding connection information between vertices. Additionally, our method can be applied in batch to produce a mesh with a desired accuracy and quality.

Practically, we opted for a hybrid approach by dividing the data volume into blocks, each block handling the advancing cases occurring within its boundaries. Triangles crossing these boundaries are handled in a separate serial phase while preserving the priority and order of operations. This approach allows for high parallelism and is key to the overall performance of our method.

To detect conflicts, we use an axis aligned bounding box octree to keep track of all triangles in the domain. If a box intersection is detected, we perform more accurate checks to verify the existence of a conflict. As the number of the objects in the tree increase, the performance of the tree gradually decreases. Hence, we decided to use a forest of trees to cover the domain with one tree per block. Triangles crossing block boundaries are duplicated across the corresponding trees.

The cost of the ridge extraction on the GPU is primarily due to texture fetches to access gradient and Hessian values, and to the evaluation of the Hessian's eigensystem. Fortunately, our advancing front approach guarantees that these expensive operations are confined the direct vicinity of the surface, which contrasts with existing methods that carry out the extraction search throughout the domain.

6. Results

We tested our method on a high-end Quad core Intel Core2 Extreme QX9650 (12MB, 3.0GHz) machine where the GPU kernels are executed on NVIDIA GeForce GTX 280. The implementation was done in CUDA and C++. To document the performance of our method, we present here results obtained for several datasets corresponding to different application scenarios.

The selection of the accuracy parameter ε depends on the user need. However, we found empirically that a distance equivalent to one sixth of a voxel edge length is a good choice. The GPU step parameter τ is typically chosen to be $\frac{1}{4} \times \varepsilon$ for all our tests cases. This parameter needs to be sufficiently small to avoid missing important error variations. The maximum edge length parameter η can be set to a multiple of the ε value (4 to 10). A smaller η is more conservative and leads to better triangle edge ratios. However, it also leads to a large number of triangles.

We have considered four different test cases at different degrees of complexity. The first dataset that we consider is a synthetic dataset we created by computing the distance function for the faces of a rectangular volume. The ridge surface would then represent the skeleton of that volume. We have not applied scale space analysis for this test case to demonstrate the robustness of our method. As seen in figure 7,

the extracted mesh shows no artifacts even at the boundaries between the different branches. For non-manifold surfaces such as this one, our method do not attempt to handle branching explicitly through numerical checks. We assume that edges grow in a single direction and our front propagation strategy implicitly enforces a local manifold topology. However, we found that at the intersection of the branches, different triangles attempt to grow on different branches at random. In addition, the seeding prevents a branch from being missed.

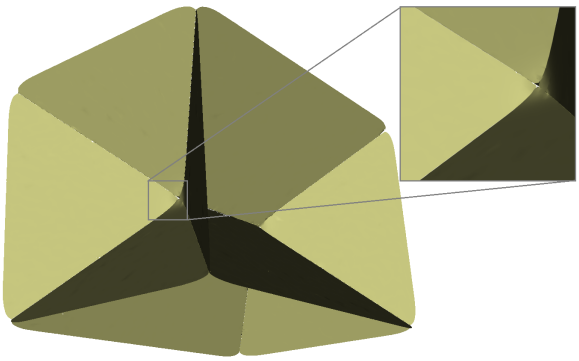

Figure 7: *Mesh extracted for a synthetic test case.*

The second dataset that we considered is a synthetic (ABC) fluid flow that was analyzed through its Lagrangian coherent structures (LCS) [Hal01]. We chose this dataset because of its smooth yet convoluted patterns that have been abundantly displayed in scientific visualization and fluid dynamics publications. We did not apply scale space analysis for this dataset as well in order to challenge the robustness of the extraction algorithm. Our method achieves smooth results for this dataset as shown in figure 8.

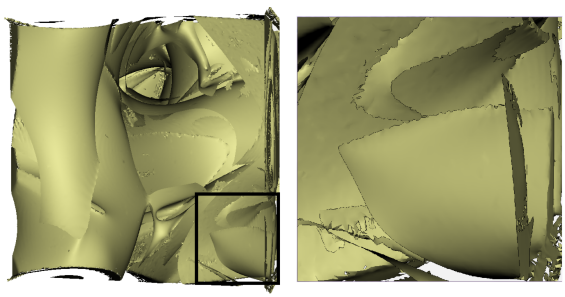

Figure 8: *Extracted mesh for the ridges of FTLE field in ABC flow [Hal01].*

As a second example related to fluid flows, we consider a CFD simulation of a turbulent flow at a high Reynolds number that corresponds to a streaming jet. The simulation is time dependent and we have selected an integration length in the FTLE computation that reveals the patterns of the turbulent mixing. This dataset is clearly a challenging one with high geometric complexity in addition to topological intricacy. Pre-processing for scale analysis was applied to this dataset. Even in this demanding configuration our method performed remarkably well. We illustrate this in figure 9 by rendering the resulting mesh colored by the distance to the mesh extracted by the method of Schultz *et al.* [STS10]. We notice that the differences are relatively small and are mainly concentrated at the boundaries of the surface. The complete result of our method for this demanding dataset is shown in Fig. 10.

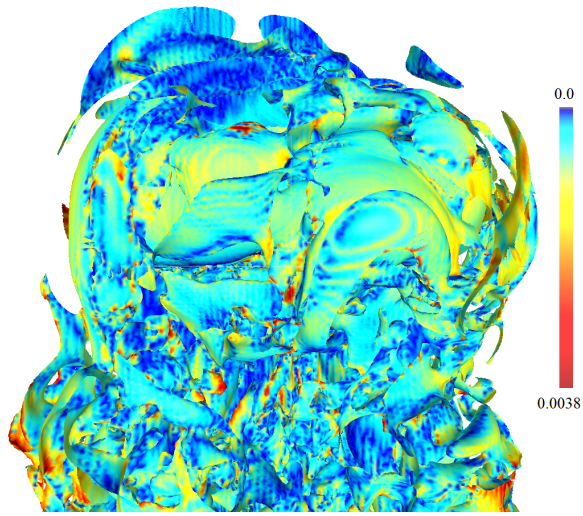

Figure 9: *Our extracted mesh for the jet dataset colored with the distance to the mesh extracted using the method described in [STS10]. The distance is relative to the longest volume side.*

Our third dataset corresponds to a medical imaging application scenario. A number of papers in recent years have investigated the visualization of ridge manifolds in the particular context of brain imaging. Following this approach we are presenting results obtained for the DTI dataset that was made available as part of the IEEE visualization contest 2010. Specifically, we applied our method to the second dataset which corresponds to the brain of a patient exhibiting a large tumor. Our results shown in Fig. 11 clearly capture the geometry of that salient feature of the dataset while properly resolving the scale space context of its spatial embedding. The latter can be easily seen through the color coding of the optimal scale that we have superimposed on the extracted geometry of the ridges.

While the geometric structure of the cortical surface is inherently convoluted, as illustrated by the finest scales that correspond to that region of the data we obtain smooth results in the vicinity of the tumor. It must be noted that this

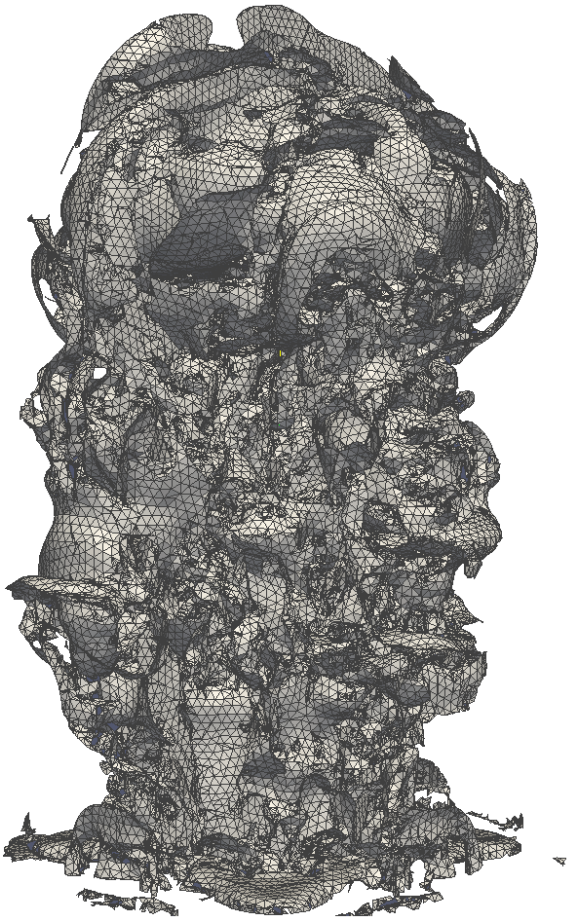

Figure 10: *Complete ridge surfaces (LCS) of the FTLE field in a turbulent jet. The maximum edge length is set to 10ε compared to 5ε for Fig. 9.*

surface, though geometrically pleasing, is in fact very subtle and difficult to obtain. Indeed the ridge strength measured on the surface is among the weakest observed in this particular data. Additionally, the surface contains a number of holes and boundaries that our method was able to properly resolve. In particular, our ray casting approach applied to a high resolution to the same region confirmed that the gaps that we identified on the surface did correspond to discontinuities of the ridge manifold.

We show the numerical results obtained with our datasets in table 1. In this table we compare our method with the current state of the art in ridge surface extraction that was recently described in a paper by Schultz *et al.* [STS10]. Our method yields both a smaller mesh size and a better mesh quality than previous approaches. The quality is evaluated in terms of both the average ratio between the minimum and maximum edge length incident at a vertex, and the average

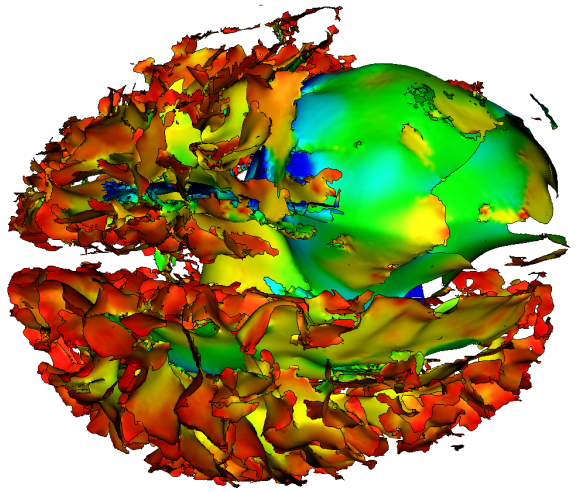

Figure 11: *Our extraction for the ridge in the IEEE Visualization 2010 Contest brain dataset colored by the scale information (blue:coarse, red:fine).*

Table 1: *Results of our method compared to two variations of [STS10]. "S1" uses the same resolution as our method, while "S2" is doubled along each axis. The percentage of discarded triangles for our method was 8.6%, 32.7%, 29.4%, and 27.5% for the Cube, ABC, Jet, and Brain data sets, respectively.*

		Time (sec)	# Tri.	Edge-Vertex	Edge-Triangle
Cube	Our	4.73	55K	0.58	0.73
	S1	4.2	86K	0.20	0.41
	S2	24.14	350K	0.21	0.41
ABC	Our	52.3	632K	0.46	0.61
	S1	18.03	912K	0.37	0.54
	S2	227.6	3866K	0.39	0.56
Jet	Our	143.3	1761K	0.47	0.67
	S1	54.3	1790K	0.37	0.55
	S2	996.1	6284K	0.39	0.57
Brain	Our	32.75	405K	0.39	0.59
	S1	149.1	3101K	0.36	0.50

ratio between the minimum and maximum edge length in a triangle. In table 2, we show information about the mean and RMS distance between our meshes and the meshes extracted by Schultz's method. We have noticed that the distances are generally small and concentrated at the boundaries of the surfaces.

One limitation we noticed in our method is the presence of a small number of skinny triangles in the resulting mesh.

Table 2: *The mean and RMS distance between our meshes and the meshes extracted using the method described in [STS10]. The numbers are relative to the longest volume side.*

	Mean distance (e-3)	RMS distance (e-3)
Cube	0.132	0.478
ABC	2.57	5.9
Jet	0.731	1.058
Brain	0.438	0.684

These triangles are created when two triangles grow on non-neighbor active edges in a way that limits the space between them. This problem could be resolved in a postprocessing step through local re-meshing. However, this step would need to ensure the same accuracy. Our method is also limited by the trilinear interpolation imposed by the graphics hardware, which contrasts with the smooth reconstruction kernels that are available to other methods [KTW07, KSSW09]. However we did not find this aspect to penalize the quality of our results.

7. Conclusions and Future Work

We have presented a novel technique for the fast extraction of high quality ridges from 3D data sets. We have proposed an innovative strategy that intertwines sampling and meshing and exploits the compute power of both the CPU and GPU through a careful organization of massively parallel (sampling) and inherently sequential (mesh consistency checks) operations. We have also shown that our method produces higher quality meshes in terms of both approximation accuracy and geometric properties.

We find the observed collaborative performance between CPU and GPU very promising and we would like to pursue this avenue. We believe that utilizing new technologies such as OpenCL will be useful in this endeavor. Additionally we would like to see how our parallel approach performs on even larger data sets and investigates its scalability on many-core systems. This promises to be particularly interesting in the context of problems that far exceed the memory of modern day desktop computers, such as in the processing of high-performance computing simulations. Streaming strategies will need to be considered in this case.

Though our method performs reasonably well in ridge surface extraction and is fast compared to conventional front propagation techniques, we are confident that significant performance gains can be achieved by using a better spatial data structure for conflict detection. In particular we found that the collision detection method we used had limited scalability, thus it can adversely impact the performance of our method in the context of very large meshes.

8. Acknowledgments

This work was partially supported by a Intel PhD Fellowship (SB) and a gift by Intel Corporation "Exploring the Capabilities of the Larrabee Platform" (XT). We also want to thank Thomas Schultz at the University of Chicago for his help with his implementation of the crease extraction available through the Teem library. Finally, we want to thank the providers of the Teem library http://teem.sf.net

References

[AB98] AMENTA N., BERN M.: Surface reconstruction by voronoi filtering. *Discrete and Computational Geometry 22* (1998), 481–504. 5

[ABCO*01] ALEXA M., BEHR J., COHEN-OR D., FLEISHMAN S., LEVIN D., SILVA C. T.: Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01* (Washington, DC, USA, 2001), IEEE Computer Society, pp. 21–28. 2

[ABK98] AMENTA N., BERN M., KAMVYSSELIS M.: A new voronoi-based surface reconstruction algorithm. In *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), ACM, pp. 415–421. 2

[BMR*99] BERNARDINI F., MITTLEMAN J., RUSHMEIER H., SILVA C., TAUBIN G.: The ball-pivoting algorithm for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics 5*, 4 (1999), 349–359. 6

[BO05] BOISSONNAT J.-D., OUDOT S.: Provably good sampling and meshing of surfaces. *Graph. Models 67*, 5 (2005), 405–451. 2

[BT10] BARAKAT S., TRICOCHE X.: An image-based approach to interactive crease extraction and rendering. *Procedia Computer Science 1*, 1 (2010), 1703 – 1712. ICCS 2010. 3, 5

[EGMP94] EBERLY D., GARDNER R., MORSE B., PIZER S.: Ridges for image analysis. *Journal of Mathematical Imaging and Vision 4* (1994), 351–371. 2

[Eve01] EVERITT C.: *Interactive Order-Independent Transparency*. NVIDIA Corporation, 2001. 3

[FP01] FURST J. D., PIZER S. M.: Marching ridges. In *Proceedings of 2001 IASTED International Conference on Signal and Image Processing* (2001), pp. 22–26. 2

[Hal01] HALLER G.: Distinguished material surfaces and coherent structures in three-dimensional flows. *Physica D 149* (2001), 248–277. 7

[KSSW09] KINDLMANN G., SAN JOSE ESTEPAR R., SMITH S. M., WESTIN C.-F.: Sampling and visualizing creases with scale-space particles. *IEEE Transactions on Visualization and Computer Graphics 15*, 6 (2009), 1415–1424. 2, 3, 9

[KTW06] KINDLMANN G., TRICOCHE X., WESTIN C.-F.: Anisotropy creases delineate white matter structure in diffusion tensor mri. In *Proceedings of Medical Imaging Computing and Computer-Assisted Intervention, MICCAI '06* (2006). 2

[KTW07] KINDLMANN G., TRICOCHE X., WESTIN C.-F.: Delineating white matter structure in diffusion tensor MRI with anisotropy creases. *Medical Image Analysis 11*, 5 (October 2007), 492–502. 2, 3, 9

[Lin98] LINDEBERG T.: Edge detection and ridge detection with automatic scale selection. *Int. J. Computer Vision 30*, 2 (1998), 117–154. 3

[LLP*10] LI R., LIU L., PHAN L., ABEYSINGHE S., GRIMM C., JU T.: Polygonizing extremal surfaces with manifold guarantees. In *Proceedings of the 14th ACM Symposium on Solid and Physical Modeling* (2010), SPM '10, pp. 189–194. 2

[PEMF98] PIZER S., EBERLY D., MORSE B., FRITSCH D.: Zoom-invariant vision of figural shapes: The mathematics of cores. *Comp. Vision Image Understanding 69*, 1 (1998), 55–71. 2

[PS08] PEIKERT R., SADLO F.: Height Ridge Computation and Filtering for Visualization. In *Proceedings of Pacific Vis 2008* (March 2008), Fujishiro I., Li H., Ma K.-L., (Eds.), pp. 119–126. 2

[SFS05] SCHEIDEGGER C. E., FLEISHMAN S., SILVA C. T.: Triangulating point set surfaces with bounded error. In *SGP '05: Proceedings of the third Eurographics symposium on Geometry processing* (Aire-la-Ville, Switzerland, Switzerland, 2005), Eurographics Association, p. 63. 2, 6

[SP99] STETTEN G. D., PIZER S. M.: Medial-node models to identify and measure objects in real-time 3-d echocardiography. *IEEE Transactions on Medical Imaging 18*, 10 (1999), 1025–1034. 2

[SP07] SADLO F., PEIKERT R.: Efficient visualization of lagrangian coherent structures by filtered AMR ridge extraction. *IEEE Trans. Vis. Comput. Graph. 13*, 6 (2007), 1456–1463. 2

[SSFS06] SCHREINER J., SCHEIDEGGER C. E., FLEISHMAN S., SILVA C. T.: Direct (re)meshing for efficient surface processing. *Computer Graphics Forum 25*, 3 (Sept. 2006), 527–536. 2, 3

[SSS06] SCHREINER J., SCHEIDEGGER C., SILVA C.: High-quality extraction of isosurfaces from regular and irregular grids. *IEEE Transactions on Visualization and Computer Graphics 12*, 5 (2006), 1205–1212. 2, 6

[STS10] SCHULTZ T., THEISEL H., SEIDEL H.-P.: Crease surfaces: From theory to extraction and application to diffusion tensor MRI. *IEEE Transactions on Visualization and Computer Graphics 16*, 1 (2010), 109–119. 2, 7, 8, 9

[SWH05] SAHNER J., WEINKAUF T., HEGE H.-C.: Galilean invariant extraction and iconic representation of vortex core lines. In *Proc. Eurographics / IEEE VGTC Symposium on Visualization (EuroVis '05)* (Leeds, UK, June 2005), K. Brodlie D. Duke K. J., (Ed.), pp. 151–160. 2

[TG92] THIRION J.-P., GOURDON A.: *The 3D Marching Lines Algorithm and Its Application to Crest Lines Extraction*. Tech. Rep. 1672, Institut National de Recherche en Informatique et en Automatique, Unité de Recherche INRIA Rocquencourt, May 1992. 2

[TKW08] TRICOCHE X., KINDLMANN G., WESTIN C.-F.: Invariant crease lines for topological and structural analysis of tensor fields. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1627–1634. 2

[TSW*05] THEISEL H., SAHNER J., WEINKAUF T., HEGE H.-C., SEIDEL H.-P.: Extraction of parallel vector surfaces in 3D time-dependent fields and application to vortex core line tracking. In *Proc. IEEE Visualization 2005* (Minneapolis, U.S.A., October 2005), pp. 631–638. 2