Eurographics Conference on Visualization (EuroVis) (2014)
N. Elmqvist, M. Hlawitschka, and J. Kennedy (Editors)

Per-Fragment Image-based Flow Visualization using Interactive Surface Extraction from Unstructured Grids

F. Niebling[1] and M. Becker[2] and T. Schlegel[1]

[1] Technische Universität Dresden, Germany
[2] Stellba Hydro GmbH, Germany

Abstract

We present a method for image-based visualization of flow fields on unstructured grids. For interactive exploration of flow data from CFD simulations, we combine GPU-accelerated surface extraction methods and line integral convolution (LIC) in a multi-pass algorithm. In contrast to other highly efficient methods, computation of stream lines is performed directly on the unstructured grid, where in previous methods the flow field had to be confined to the extracted surface. The introduced algorithm is based on three passes: Surface extraction, rendering, and computation of stream lines to perform line integral convolution. Point location, a major performance factor in stream line computation on unstructured grids, can be greatly accelerated by reusing data from the surface extraction pass. This allows us to achieve interactive frame rates on current generation graphics hardware for the post-processing of unstructured CFD datasets.

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

1. Introduction

Direct visualization of a complete, large scale simulation dataset is often impractical due to the occlusion of interesting flow features by other data. Providing tools for interactive reduction of datasets to extract meaningful data via filtering helps users to analyze large-scale flow fields. Interactive isosurface and cut-surface extraction is an effective way to visualize various characteristics of fluid dynamic simulation results. Scalar fields such as pressure, or scalar data derived from fields of higher dimensionality, e.g. the magnitude of a velocity field, can be efficiently displayed on these surfaces via color coding. To enable dynamic probing of data fields, low latency surface extraction and mapping mechanism have to be developed that provide immediate visual feedback to the user's control motions.

Simulations performed on hybrid unstructured grids (USGs), i.e. USGs that contain more than a single cell element type, are widely used in various engineering domains. Contrary to structured grids, each vertex position has to be stored separately in unstructured grids. The topology, i.e. the connectivity between these vertices must be explicitly defined. In contrast to simulation data on structured grids, proximity of data in world coordinates generally does not have to correlate with proximity in memory in any dimension, which makes optimization of memory accesses difficult for the general case. Since there is no implicit structure inherent in the topology, neighbor relationship between cells has to be explicitly stored. USGs typically consist of an array of elements, an array specifying connectivity, and an array of vertex positions. Using the information stored in the connectivity array, cells are built from points as 2D or 3D elements such as lines, triangles, tetrahedra or hexahedra. Indirect addressing can be used to allow for the sharing of vertices between neighboring cells.

The presented algorithm provides integrated iso- and cut-surface extraction from USGs and computation of line integral convolution on these surfaces using multiple rendering and computation phases.

2. Related Work

2.1. LIC

In the visualization of CFD simulation data, lines and animated particles are used to provide insight into the local flow field. Hedgehog plots have been used as the standard method to visualize the global flow directions in dense flow

DOI: 10.2312/eurovisshort.20141160

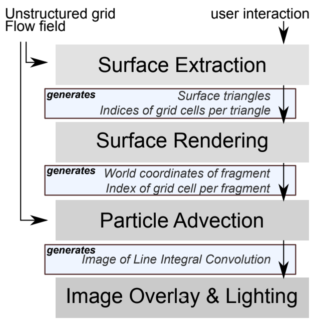
Unstructured grid
Flow field
user interaction
Surface Extraction
generates Surface triangles
Indices of grid cells per triangle
Surface Rendering
generates World coordinates of fragment
Index of grid cell per fragment
Particle Advection
generates Image of Line Integral Convolution
Image Overlay & Lighting

Figure 1: Overview of the proposed algorithm consisting of multiple computation and rendering passes.

fields, although it is hard for the user to reconstruct the flow from discrete samples. To provide a global visualization of dense flow fields, texture-based methods such as Digital Differential Analyzer (*DDA*) [CL93], *Spot Noise* [vW91] and *LIC* [CL93] have been developed. The basic idea of these flow visualization techniques is to blur textures containing noise along the local vectors in a flow fields. As opposed to DDA convolution where complete streamlines are approximated, *LIC* computes locally exact streamline advection.

Given streamlines σ parametrized by arclength s that are defined by

$$\frac{d}{ds}\sigma(s) = v(\sigma(s),t) \qquad (1)$$

and the initial condition

$$\sigma(0) = \sigma_0 \qquad (2)$$

the *LIC* algorithm computes intensities for each texel located at $x_0 = \sigma(s_0)$ by evaluating the convolution integral

$$I(x_0) = \int_{s_0-L}^{s_0+L} k(s-s_0)T(\sigma(s))ds \qquad (3)$$

where σ is the streamline through the center of the texel, k is a given filter kernel with length L, and T is a texture filled with white noise [SH95].

2.2. IBFVS

Image Based Flow Visualization (*IBFV*) [vW02] and Image Based Flow Visualization for Curved Surfaces (*IBFVS*) [vW03] are alternatives to traditional *LIC*-based approaches for unsteady flow. Each image is the result of warping the previous image along the flow direction, blended with background images containing white noise. Van Wijk uses CPU-based projection of the velocity vectors at mesh vertices to the image space. The texture is then advected over the mesh according to the velocity vectors stored at the projected mesh vertices. The distorted texture coordinates computed by the mesh advection are then mapped to the original 3D surface mesh vertices.

The main difference between the IBFVS algorithm and our proposed approach is the projection of the vector field to image space that is performed during IBFVS that is an inheritance of the original Lagrangian-Eulerian Advection (*LEA*) algorithm [JEH02]. This makes particle advection easier, but leads to incorrect results when the flow is not confined to the extracted surface.

2.3. ISA

Image Space Advection (*ISA*) [LJH03] distorts a regular, rectilinear mesh defined in 2D image space. The vectors in a flow field at the vertices are encoded in the RGB channels of a texture and rendered using Gouraud shading to interpolate the vectors resulting in a velocity image. The velocity vectors projected onto the image plane are then used for image-space advection of the local flow field.

The difference between *ISA* and the proposed algorithm is that in *ISA* the flow field is rendered to an interpolated *velocity image*. The *velocity image* is then interpreted as the vector field and is used for the texture advection in image space [LWJH04]. The proposed algorithm is able to compute particle advection using the original unstructured grid from the CFD simulation.

3. Algorithm

We present a multi-pass algorithm that combines interactive surface extraction and per-fragment computation of line integrals for flow visualization (see also figure 1). By rendering data from the extracted surface into an off-screen floating point buffer, the generated fragments can be used as starting points for the computation of line integrals. This allows us to re-use data from the surface extraction pass, to avoid expensive operations such as point location in USG cells for particle advection. The *LIC* pass then uses the generated streamline starting positions to compute line integrals in the original unstructured grid.

3.1. Surface Extraction from Hybrid Unstructured Grids on GPUs

Our method for efficient manycore extraction of surfaces from *USGs* is based on NVIDIA's surface extraction algorithm for Cartesian grids on GPUs [NVI08]. The algorithm treats unstructured grids as a stream of individual cells that can be processed independently in parallel. Given an input stream of cells, output streams of triangles, normals and mapped data elements are provided for immediate rendering or further post-processing on the parallel GPU.

The algorithm is split into multiple steps:

- *Classification:* All cells in the USG are classified according to how many vertices are generated by isosurface or cut-surface extraction in the individual cell. Two output streams are generated as input for the following stages. The first data stream is a stream containing the number of vertices generated in the cell. The second stream is a classification of boolean values (0/1) specifying if any vertices in the cell are generated at all. The number of generated vertices can be determined by calculating an index into a triangle lookup table such as the ones used in extensions of the marching cubes algorithm for hybrid unstructured grids [NSNGH12].

- *Index computation for stream compaction:* Since it is likely that there are cells in the USG that are not cut by the surface, the output streams have to be compacted. To be able to leave out empty cells, exclusive prefix sums for the aforementioned output streams are computed to be used in the next stage.
- *Stream compaction:* Using the stream containing the number of generated vertices per-cell, along with the stream containing the exclusive prefix sum of the boolean classification, starting indices for vertices in a compacted output stream can be computed forming an element index stream. We use CUDPP [HSO07] for computation of the exclusive parallel prefix sum of the classification stream.
- *Triangle and normal generation:* With the element index stream designating the position of triangle vertices in the surface output stream, and the stream containing the exclusive prefix sum of generated vertices, the generated triangles can be independently scattered to the output stream. This can be achieved by using triangle lookup tables as in the marching cubes algorithm. During generation of the triangle output stream, further compacted streams containing per-face normals, interpolated data on vertices and the cell index in the USG per generated triangle are generated.

The actual extraction of the surface is independent per-cell, allowing for a scalable pleasantly parallel execution on parallel GPGPU clusters. Stream compaction to allow for a compact representation of the resulting triangle mesh is based on building parallel prefix sums, which introduces an additional computational complexity of $O(log_2(n))$, where n designates the number of cells in the USG.

3.2. Rendering Pass

We use the rendering pipeline to extract fragments from the extracted surface where particle advection should be performed given a specific viewpoint, thereby gaining view-frustum culling, backface culling and omission of sub-pixel triangles. In the first render pass, the USG cell indices are provided as vertex attributes to a GLSL shader program along with the triangle vertices of the extracted surface. The triangles are then rendered to a floating point RGBA buffer allowing for the storage of 32 bit of data per channel. In the vertex shader, the world coordinates of the triangle vertices along with the USG element index are passed on to the fragment shader. The fragment shader stores the interpolated world coordinates of the fragment in the RGB channel of the render target. The alpha channel is used to store the element index. The resulting texture in the render target consists of the world coordinates for the center of each fragment and the index of the cell in the original USG grid.

Since lighting is omitted in this pass for lack of space in the RGBA buffer, a lightmap to provide depth cues to the user needs to be generated in an additional rendering pass. This is especially helpful for the visualization of flow on iso-surfaces (see Figure 2).

3.3. Stream Line Computation and Line Integral Convolution

Following the first render pass is an implementation of *LIC* as a post-processing of the generated floating point RGBA image using CUDA kernels. For every fragment in the texture containing an alpha value that is a valid cell in the USG, computation of the convolution integral through the center of this fragment is performed (see Formula 3). The starting point of the streamline computation is known to be in the USG cell referenced in the alpha buffer. Therefore, additional computationally expensive point location operations can be avoided.

The method for decomposition of complex USG elements into tetrahedra described by Kenwright et al. [KL96] is used to do on-the fly decomposition of complex cells during particle advection. The advantage of using runtime decomposition compared to offline decomposition is that the memory requirements for an USG containing complex cells is much lower than for a tetrahedral mesh. Particles leaving individual cells during advection can be detected as part of the tetrahedral decomposition method. Neighbor search [HNF07] can then be applied to locate the new cell that the particle traveled to.

The output of the post-processing kernels is an image where the fragments containing the rendered surface have been replaced by the intensity values resulting from the computation of the convolution integral, augmented with colors designating scalar properties of the flow, e.g. the magnitude of the local flow field.

4. Results

4.1. Surface Extraction

Surface extraction on GPUs is able to achieve much higher performance than sophisticated CPU implementations even for hybrid unstructured grids. Although highly-irregular memory accesses that are needed in the processing of USGs still are a bottleneck for parallel manycore algorithms, the memory intensive classification of cells, where every cell and every vertex in each cell have to be accessed, is much faster on newer generation GPUs. For large computational grids, distribution of the USG to a GPU cluster using domain decomposition is a feasible approach.

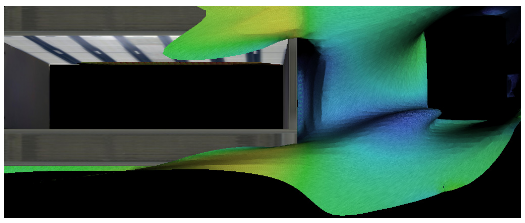

Figure 2: Airflow in a datacenter on an isosurface of temperature (velocity color coded)

4.2. Integrated Surface Extraction and LIC

The performance of the presented algorithm mainly depends on the number of fragments used for post-processing of the image. Sort-first parallelization by partitioning the image space, allowing multiple GPUs to process parts of the dataset in parallel, thus leads to big improvements in the performance of the algorithm.

Although no reuse of calculated streamlines is performed, the combined surface extraction and particle advection benefits from current GPGPU architectures. Especially at close viewing distances, partitioning of the image space into 2D blocks for processing using CUDA kernels increases the probability that fragments processed inside the same block are part of the same USG cell. The individual threads participating in the computation of streamlines in a single block are then able to exploit cached access to USG data structures on the GPU, considerably increasing performance.

Transient flow data is supported for static USGs, where the upload of time-dependent flow fields from host to device memory can be performed during classification in the surface extraction step. Unfortunately, this technique is infeasible for time-dependent grids since there is not enough bandwidth available between host and device on current hardware.

Visual differences between LIC performed on the 2D surface grid and on the 3D computational grid are minimal where the flow field is closely confined to the extracted surface. When the flow field is closer to being perpendicular to the extracted surface, the impression of surface flow is weakened (see Figure 3).

5. Conclusion

We have presented an algorithm for integrated surface extraction and computation of line integral convolution for visualization of flow fields on unstructured grids. Using a GPU-based multi-pass algorithm we can achieve interactive exploration of flow simulation data.

Our algorithm is able to calculate convolution integrals on surfaces extracted from USGs using parallel manycore devices such as programmable GPUs. It avoids several shortcomings of previous developments:

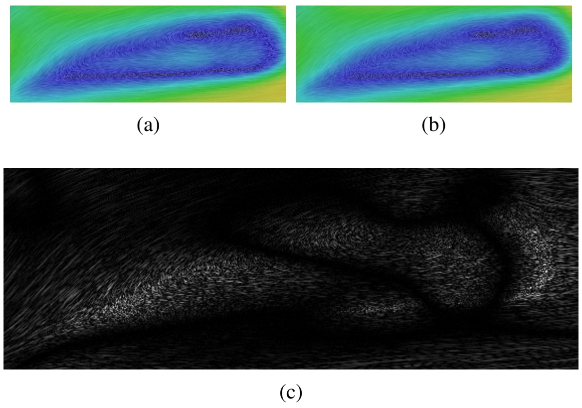

Figure 3: (a) LIC on flow field projected to surface, (b) LIC on 3D CFD flow field, (c) normalized difference image

- The unstructured grid does not need to be resampled to a structured grid at any time of the algorithm. This avoids artifacts introduced by interpolation. Complex tree structures for cell location (e.g. Cell-Tree [GJ10]) - which are still unable to perform cell location for millions of particles in realtime - can be omitted, since starting cells for particle tracing are already computed during the surface extraction step.
- The flow field does not have to be projected to the surface where *LIC* is applied. Instead, particle advection is performed on the original CFD grid. Projection of the vector field is only appropriate for surfaces that are close approximations of the local flow field, i.e. stream surfaces. Otherwise, aliasing artifacts might be added to the visualization, possibly leading to erroneous conclusions about the underlying flow.
- Streamline computation is performed per-fragment of the generated image, providing a decoupling of image resolution and object space. Therefore, both coarse visualizations stemming from large triangles, and duplicate computations for small objects are eliminated.
- Readback of rendered images from the GPU to CPU-accessible memory is not required, unless parallel or remote rendering is performed and copying of image data to remote hosts is essential.

6. Acknowledgments

This work was supported in part by a grant from German Federal Ministry of Education and Research (BMBF).

References

[CL93] Cabral B., Leedom L. C.: Imaging Vector Fields Using Line Integral Convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1993), SIGGRAPH '93, ACM, pp. 263–270. doi:10.1145/166117.166151. 2

[GJ10] Garth C., Joy K. I.: Fast, memory-efficient cell location in unstructured grids for visualization. *IEEE Transactions on Visualization and Computer Graphics 16*, 6 (2010), 1541–1550. doi:http://doi.ieeecomputersociety.org/10.1109/TVCG.2010.156. 4

[HNF07] Haselbacher A., Najjar F. M., Ferry J. P.: An efficient and robust particle-localization algorithm for unstructured grids. *Journal of Computational Physics 225* (August 2007), 2198–2213. URL: http://portal.acm.org/citation.cfm?id=1280287.1280451, doi:10.1016/j.jcp.2007.03.018. 3

[HSO07] Harris M., Sengupta S., Owens J. D.: Parallel prefix sum (scan) with cuda. In *GPU Gems 3*, Nguyen H., (Ed.). Addison Wesley, Aug. 2007. 3

[JEH02] Jobard B., Erlebacher G., Hussaini M.: Lagrangian-Eulerian advection of noise and dye textures for unsteady flow visualization. *Visualization and Computer Graphics, IEEE Transactions on 8*, 3 (2002), 211–222. doi:10.1109/TVCG.2002.1021575. 2

[KL96] Kenwright D., Lane D.: Interactive time-dependent particle tracing using tetrahedral decomposition. *IEEE Transactions on Visualization and Computer Graphics 2*, 2 (jun 1996), 120–129. doi:10.1109/2945.506224. 3

[LJH03] Laramee R., Jobard B., Hauser H.: Image space based visualization of unsteady flow on surfaces. In *Visualization, 2003. VIS 2003. IEEE* (oct. 2003), pp. 131 –138. doi:10.1109/VISUAL.2003.1250364. 2

[LWJH04] Laramee R. S., Wijk J. J. V., Jobard B., Hauser H.: ISA and IBFVS: Image space based visualization of flow on surfaces. *IEEE Transactions on Visualization and Computer Graphics 10* (2004), 637–648. 2

[NSNGH12] Narayan A., Sreevalsan-Nair J., Gaither K., Hamann B.: Isosurface extraction from hybrid unstructured grids containing pentahedral elements. In *GRAPP/IVAPP* (2012), Richard P., Kraus M., Laramee R. S., Braz J., (Eds.), SciTePress, pp. 660–669. URL: http://dblp.uni-trier.de/db/conf/grapp/grapp2012.html#NarayanSGH12. 3

[NVI08] NVIDIA: CUDA C/C++ SDK Code Samples, 2008. URL: http://developer.nvidia.com/cuda-cc-sdk-code-samples. 2

[SH95] Stalling D., Hege H.-C.: Fast and resolution independent line integral convolution. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1995), SIGGRAPH '95, ACM, pp. 249–256. doi:10.1145/218380.218448. 2

[vW91] van Wijk J. J.: Spot noise texture synthesis for data visualization. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1991), SIGGRAPH '91, ACM, pp. 309–318. doi:10.1145/122718.122751. 2

[vW02] van Wijk J. J.: Image based flow visualization. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2002), SIGGRAPH '02, ACM, pp. 745–754. doi:10.1145/566570.566646. 2

[vW03] van Wijk J.: Image based flow visualization for curved surfaces. In *Visualization, 2003. VIS 2003. IEEE* (oct. 2003), pp. 123 –130. doi:10.1109/VISUAL.2003.1250363. 2