StreamProbe: A Novel GPU-based Selection Technique for Interactive Flow Field Exploration

Mai El-Shehaly, Denis Gračanin, Mohamed Gad, Hicham G. Elmongui, and Krešimir Matković

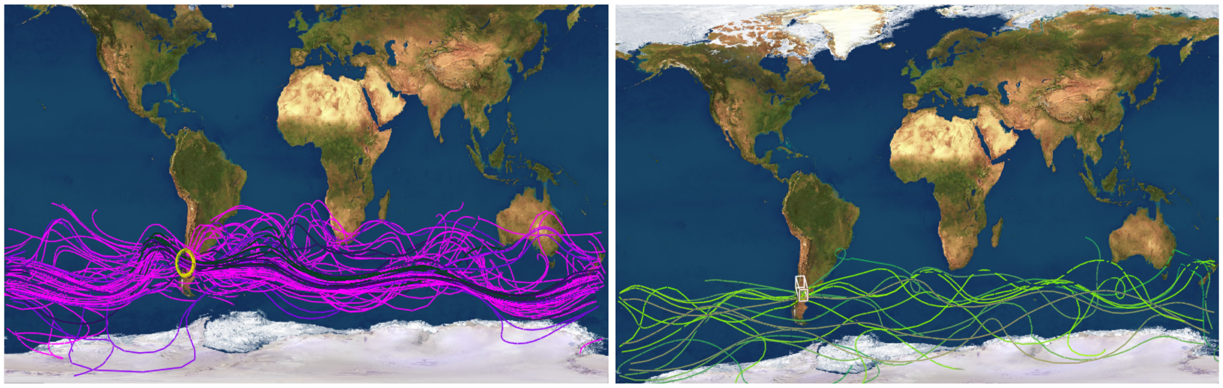

Fig. 1. Circular probe (left) and box probe (right) used to filter subsets of The Chemical Lagrangian Model of the Stratosphere (CLaMS).

Abstract—Flow visualization aims primarily at the occlusion-free and controllable visual exploration of dense flow fields. Flow fields are usually visualized using streamlines (steady-state fields) or pathlines (time-dependent flows). The challenge is how to highlight crucial features while reducing clutter to convey relevant trends to the expert user in a timely manner. We present a novel querying and selection technique for the ad-hoc exploration of dense flow fields. Our goal is to allow the user to create a geometric shape that acts as a "probe" for the field, displaying only the information that flows through a region covered by the probe. The main contribution of our technique is that it does not require the construction of expensive hierarchical data structures for pathlines representing the flow. Unlike other techniques, ours operates directly on pathlines' vertices and requires no knowledge of line features. We implemented our technique using GPU-parallelism and new features in OpenGL 4.4 shaders to achieve interactive frame rates.

Index Terms— Scientific visualization, feature selection, parallel algorithms.

1 Introduction

Dense flow field visualization has been an important research question in scientific visualization for several decades. Effective visual representation is crucial, due to the massive amounts of vector field data available through scientific simulation and measurement modalities in many application domains. Examples include velocities of wind and ocean currents, results of fluid dynamics simulations, magnetic fields, blood flow, cell migration during embryo development, and components of stress and strain in materials. However, existing visualization techniques have fallen short in addressing the full complexity of the flow field visualization problem, particularly the issue of clutter reduction both in 2- and 3-dimensional fields.

Traditionally, vector fields have been visualized using texture-based methods (spot noise [7], LIC (Line Integral Convolution) [1]) or streamlines, i.e. integral curves that are everywhere tangent to the vector field. The user places seed points to cover the domain densely with streamlines. However, using denser streamlines to show greater field strength introduced clutter. More sophisticated 2D seeding strategies were introduced to reduce this clutter [4] and extended to 3D [6].

A recent technique [2] proposed intelligent opacity optimization based on CPU-computed importance values on a per line segment basis. Despite the high quality of images in their results, the optimization process serializes heavily and the question of whether it is possible to achieve interactive rates with importance-driven opacity optimization remains an open one. The calculation of importance values constitutes an extra preprocessing step that hinders the ability to run the algorithm in real-time.

Ma et al. [5] proposed the use of information visualization tools to aid the visualization of scientific 3D flow fields. Their graph-based approach provides the user with a high level view of relationships among pathline clusters and spatio-temporal regions, both represented as graph nodes. The main disadvantage is the poor scalability of their approach both in terms of computation and storage space due to the fact that their selection process relies on the construction of two hierarchical data structures in which field data is organized.

Our work is inspired by the work proposed by Ma et al. [5] which led us to realize that interactive navigation and filtering through a pathline set can be of great benefit to the exploration and ROI identification process. The main challenge was to do so interactively without much preprocessing of the data, while keeping memory usage manageable for a GPU to operate on. We describe "StreamProbe", our proposed GPU-parallel algorithm that makes use of advances in recent GPU hardware and Graphics libraries to avoid the need for data preprocessing and expensive data structure construction.

2 Proposed Approach

We propose a novel streamline/pathline visualization and interaction technique that makes use of the geometry shader stage of OpenGL's shading language, in order to address the problem of filtering and clutter reduction as one of collision detection. The user can select from a variety of geometric shapes (e.g. a circle, a cylinder, or a bounding box) to define a probe for the flow field. As the user drags the probe

• *Mai El-Shehaly is with Virginia Tech, Department of Computer Science. E-mail: maya70@vt.edu*

Manuscript received 31 Mar. 2014; accepted 1 Aug. 2014; date of publication xx xxx 2014; date of current version xx xxx 2014. For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org.

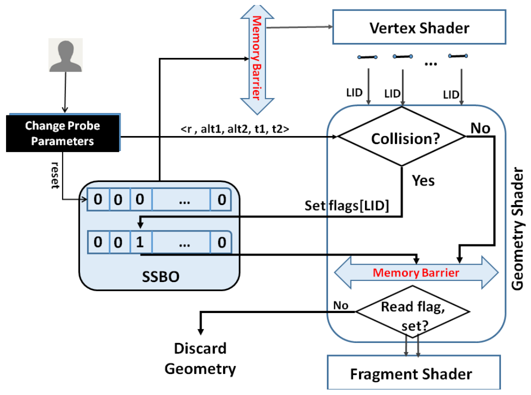

Fig. 2. Workflow of StreamProbe algorithm.

around, every trajectory the probe intersects is visible while all others are culled out.

Figure 1 shows sample pathline selections given two different probes centered at the location of the 2011 Puyehue-Cordón Caulle volcanic eruption [3]. The circular probe on the left has a radius of three degrees in geographic coordinates. The box probe on the right allows the user to specify an altitude range for the polylines of interest, ultimately providing a 3D selection tool.

The intersection algorithm can be implemented by a simple intersection test in the geometry shader, like tests used for collision detection in game development. However, the geometry shader only has access to a single geometric primitive (a line segment in the case of line strips) or a maximum of two primitives (in the case of line adjacency), so the above approach will only make visible line segments under the probe. We use new features in OpenGL 4.x that allow us to create a Shader Storage Buffer Object (SSBO) in GPU memory that can be used for general read and write access in shaders, which in turn set visibility flags for the complete line strip geometry.

Figure 2 shows how the created SSBO is accessed by both the application and geometry shader sides. Synchronization points implemented as memory barriers are inserted on both sides to ensure that correct data is being accessed by the different shader invocations. The flags array is sized according to the number of pathlines in a given dataset and contains a set of boolean switches, one for every polyline. The algorithm is designed to perform intersection tests in parallel by every geometry shader invocation on each line segment independently and display complete polylines for which any line segment passes the test. Once a line segment intersects the probe, it will set the flag for the corresponding polyline in the flags array to "ON" which means that this pathline should be visible in the final render. A synchronization point is executed to ensure that all threads have finished their intersection tests and the switches for the visible pathlines have been set.

Note that there is no race condition here as geometric primitives that fail the test will not write anything to the shared flags array. After synchronization, each primitive will go ahead and read the flag value for its corresponding polyline, if the value is ON then geometry for this thread's vertices are emitted by the geometry shader, otherwise the primitive is culled. The use of a memory barrier in our geometry shader is well justified. Due to the highly parallel nature of OpenGL, several rendering iterations can overlap in time so some geometry shader invocations can be working on iteration i for example while others would be processing primitives from iteration $i-1$. This resulted in a flicker effect in our final render. To overcome this problem, we let the application reset all flags whenever probe parameters are changed by the user, after which a second synchronization point was inserted on the application side.

2.1 Results

We tested our approach using Intel Core i7 CPU with 8GB memory and GeForce GT 740M/PCIe/SSE2 GPU with 1792MB total graphics memory, OpenGL version 4.4.0 and GLSL 4.40 NVIDIA via Cg compiler. We use a simple vertex array that contains all the vertices in a dataset in a vertex buffer on the GPU and a CPU-side vector of structures that holds indices of the first and last vertices of each streamline. This vector is constructed by the CPU once before the render. Table 1 shows the time needed for the CPU to construct this vector of structures and its size in memory for different dataset sizes. When compared to the amount of time it takes to construct streamline hierarchies [5], our approach is much more scalable and efficient in both space and time. The average frame rates are invariant to the size of the data with up to five thousand streamlines (Figure 3). For datasets with larger size, the frame rate drops but remains interactive.

Table 1. Pathline array size and construction time

No. Pathlines	Size in Memory (KB)	Time (sec)
600	4.68	3.88
800	6.25	9.25
1800	14.06	9.29
2400	18.75	10.7
3600	28.125	39.8
4800	37.5	66.5
5400	42.18	36.7
5817	45.44	39.5

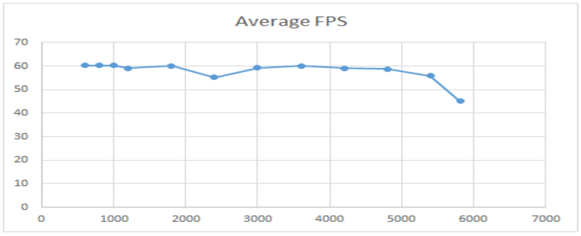

Fig. 3. Average frames per second for different numbers of pathlines.

3 Conclusion

We proposed StreamProbe, a parallel scalable approach for streamline/pathline exploration and clutter reduction that makes use of recent advances in general purpose GPU shading. Our technique is very efficient in terms of memory usage and performance and does not rely on complex data structures. Interactivity and scalability to larger datasets make StreamProbe a promising tool for interactive exploration of vector fields. Our future work will focus on applying StreamProbe to datasets that cannot fit in GPU memory and must be streamed between CPU and GPU.

References

[1] B. Cabral and L. C. Leedom. Imaging vector fields using line integral convolution. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 263–270. ACM, 1993.

[2] T. Günther, C. Rössl, and H. Theisel. Opacity optimization for 3D line fields. *ACM Transactions on Graphics (TOG)*, 32(4):120, 2013.

[3] IEEE. http://www.viscontest.rwth-aachen.de/index.html, 2014.

[4] B. Jobard and W. Lefer. Multiresolution flow visualization. In *Proceedings of the 9th International Conference in Central Europe on Computer Graphics, Visualization and Digital Interactive*, pages 34–35, 2001.

[5] J. Ma, C. Wang, C. Shene, and J. Jiang. A graph-based interface for visual analytics of 3D streamlines and pathlines. *IEEE Transactions on Visualization and Computer Graphics*, 2013.

[6] O. Mattausch, T. Theußl, H. Hauser, and E. Gröller. Strategies for interactive exploration of 3D flow using evenly-spaced illuminated streamlines. In *Proceedings of the 19th Spring Conference on Computer Graphics*, pages 213–222. ACM, 2003.

[7] J. J. Van Wijk. Spot noise texture synthesis for data visualization. *ACM SIGGRAPH Computer Graphics Newsletter*, 25(4):309–318, 1991.