HARDWARE-ACCELERATED WEB VISUALIZATION OF VECTOR FIELDS

Case Study in Oceanic Currents

Mauricio Aristizabal[1], John Congote[1,2], Alvaro Segura[2], Aitor Moreno[2], Harbil Arregui[2] and Oscar Ruiz[1]

[1]*CAD/CAM/CAE Laboratory, EAFIT University, Medellin, Colombia*

[2]*Vicomtech Research Center, Donostia - San Sebastian, Spain*

Keywords: Line Integral Convolution, Hierarchical Integration, Flow Visualization, WebGL.

Abstract: Visualization of vector fields plays an important role in research activities nowadays. Increasing web applications allow a fast, multi-platform and multi-device access to data. As a result, web applications must be optimized in order to be performed heterogeneously as well as on high-performance as on low capacity devices. This paper presents a hardware-accelerated scheme for integration-based flow visualization techniques, based on a hierarchical integration procedure which reduces the computational effort of the algorithm from linear to logarithmic, compared to serial integration methodologies. The contribution relies on the fact that the optimization is only implemented using the graphics application programming interface (API), instead of requiring additional APIs or plug-ins. This is achieved by using images as data storing elements instead of graphical information matrices. A case study in oceanic currents is implemented.

1 INTRODUCTION

Vector field visualization has an important role in the automotive and aero-spatial industries, maritime transport, engineering activities and others. It allows the detection of particularities of the field such as vortexes or eddies in flow fields, but also permits exploring the entire field behavior, determining stream paths.

In what regards to WebGL, integration based flow visualization, such as Line Integral Convolution (LIC) might not be suitable, since in order to build the integrals, several computational cycles must be performed, hence, low capacity devices might have performance issues. Therefore optimization procedures must be applied.

Our contribution is a hardware-accelerated optimization scheme for LIC flow visualization implemented in WebGL. The optimization is based on a hierarchical integration procedure (Hlawatsch et al., 2011), which reuses previously calculated information in order to build the integrals and therefore avoids unnecessary calculations. Compared with other methodologies, our implementation only requires WebGL API to be performed, as it employs images as data storing elements instead of graphical information matrices. This images are rendered with the WebGL API, hence parallel computing performance is achieved without additional APIs such as CUDA or OpenCL.

This paper is organized as follows. Section 2 presents the related work. Section 3 exposes the methodology of LIC flow visualization and its optimization. Section 4 presents a case of study in oceanic currents and finally the conclusions of our work are presented in section 5.

2 LITERATURE REVIEW

A great amount of methodologies to visualize vector fields (flow fields) has been developed among the last decades. Geometric-based approaches draw icons on the screen whose characteristics represent the behavior of the flow (as velocity magnitude, vorticity, etc). Examples of these methodologies are arrow grids (Klassen and Harrington, 1991), streamlines (Kenwright and Mallinson, 1992) and streaklines (Lane, 1994). However, as these are discrete approaches, the placement of each object is critical to detect the flow's

Aristizabal M., Congote J., Segura A., Moreno A., Arregui H. and Ruiz O..
HARDWARE-ACCELERATED WEB VISUALIZATION OF VECTOR FIELDS - Case Study in Oceanic Currents.
DOI: 10.5220/0003843707590763
In *Proceedings of the International Conference on Computer Graphics Theory and Applications* (IVAPP-2012), pages 759-763
ISBN: 978-989-8565-02-0

anomalies (such as vortexes or eddies), and therefore, data preprocessing is needed to perform an illustrative flow visualization. Reference (McLoughlin et al., 2010) presents an up-to-date survey on geometric-based approaches.

On the other hand, texture-based approaches represent both a more dense and a more accurate visualization, which can easily deal with the flow's feature representation as a dense and semi-continuous flow visualization is produced. A deep survey in the topic on texture-based flow visualization techniques is presented by (Laramee et al., 2004).

Reference (Van Wijk, 2002) implements an animated flow visualization technique in which a noise image is bended out by the vector field, and then blended with a number of background images. In (Van Wijk, 2003) the images are then mapped to a curved surface, in which the transformed image visualizes the superficial flow.

Line Integral Convolution (LIC), presented by (Cabral and Leedom, 1993), convolves the associated texture-pixels (texels) with some noise field (usually a white noise image) over the trajectory of each texel in the vector field. This methodology has been extended to represent animated (Forssell and Cohen, 1995), 3D (Liu and Moorhead II, 2004) and time varying (Liu and Moorhead, 2005; Liu and Moorhead II, 2004) flow fields.

An acceleration scheme for integration-based flow visualization techniques is presented by (Hlawatsch et al., 2011). The optimization relies on the fact that the integral curves (such as LIC) are hierarchically constructed using previously calculated data, and, therefore, avoid unnecessary calculations.

Several WebGL implementations of different applications have been done such as volume rendering, presented by (Congote et al., 2011) or visualization of biological data, presented by (Callieri et al., 2010). However, for the best of our knowledge, no other implementation that regards to LIC flow visualization on WebGL has been found in the literature or in the Web.

Conclusion of the Literature Review: WebGL implementations allow to perform applications for heterogeneous architectures in a wide range of devices from low-capacity smart phones to high-performance workstations, without any external requirement of plug-ins or applets. As a result, optimized applications must be developed. In response to that, this work optimizes the visualization of 2D steady vector fields using images as data storing elements instead of graphical information matrices. As a consequence, previously calculated data is reused, avoiding unnecessary calculations, and reducing the complexity of the algorithm to $O(\log N)$. Hence, only the graphics API is required for its implementation, preserving the multi-platform purpose of the application.

3 METHODOLOGY

This article addresses the optimization of integration-based flow visualization procedures using only the graphics API. It starts from a 2D vector field $F : X \rightarrow \mathbb{R}^2$, with $X \subseteq \mathbb{R}^2$, representing the velocity at each point inside the field. An image represented as the 2D matrix $M : [1,h_M] \times [1,w_M] \rightarrow (R,G,B,A)$, with h_M and w_M representing its height and width respectively in pixels, and a noise field $W : \mathbb{R}^2 \rightarrow \mathbb{R}$, both mapping to F.

The goal is to calculate a 2D matrix $\bar{I} : [1,h_I] \times [1,w_I] \rightarrow (R,G,B,A)$ that defines an image, with h_I and w_I representing its height and width in pixels. Each pixel is a four-component vector (R,G,B,A) of red, green, blue and alpha values $(R,G,B,A \in [0,1])$.

3.1 Line Integral Convolution

In order to visualize 2D vector field, the procedure convolves the mapping image M with its noise field W along the trajectory of each point in the vector field, as proposed by (Cabral and Leedom, 1993). For this purpose, let us define q as an arbitrary point in the field. Hence, for its associated finite trajectory c (extracted from F), with $p(s) \in c$, and s the parameter associated to c, the resulting image is obtained as follows:

$$\overline{I(q)} = \frac{\int_c M(p(s))W(p(s))ds}{\int_c W(p(s))ds} \qquad (1)$$

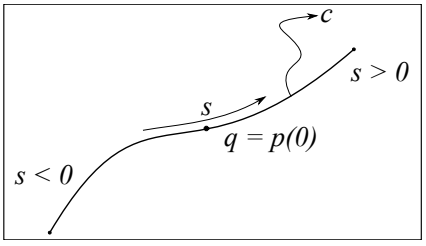

Figure 1: Trajectory of the point p inside F.

Note that q represents the starting point of the trajectory that is going to be convolved (i.e. $p(0) = q$), and the trajectory c evaluates the upstream ($s > 0$) and downstream ($s < 0$) directions. This integral is performed for all $q \in X$.

For a computer implementation, a discrete calculation of the integral is required. As a result, a PL approximation of c is performed using a first-order ap-

proximation of the point's displacement, and the integrals are calculated using an Eulerian-based method.

3.2 Hierarchical Integration

Integration over massive point trajectories in n-dimensional vector fields suffer from unnecessary step calculations since several points in the field might lie over the same trajectory and therefore share portions of the integrals (see figure 2). In response to this, hierarchical integration (Hlawatsch et al., 2011) only calculates the necessary steps and then iteratively grows the integrals reusing the data, reducing the computational complexity of the algorithm from $O(N)$, using serial integration methodology, to $O(\log N)$. The procedure is summarized as follows.

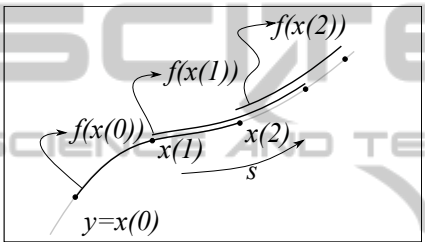

Figure 2: Trajectory overlapping in line integral evaluation.

Let us define some arbitrary line integral $f : Y \rightarrow \mathbb{R}^m$, with $Y \subseteq \mathbb{R}^n$, bounded by its trajectory c, and its discrete approximation like described in equation 2.

$$f(y) = \int_c w(x(s))ds \approx \sum_{i=1}^{t} w(x(i))\Delta s \qquad (2)$$

where t is is the maximum number of steps, representing the curve length, $x(0) = y$ is the starting point of the trajectory path. The integration procedure is performed for all points $y \in Y$ in parallel.

For the sake of simplicity assume that $\Delta s = 1$, $\forall y \in Y$ and therefore $f(y) \approx \sum_{i=1}^{t} w(x(i))$. The algorithm starts with the calculation of the first integration step for all the points in the field, this is:

$$f_0(y) = w(x(1)) \qquad (3)$$

It is required to store the last evaluated point $x(1)$ over the growing trajectory and the value of $f_1(y)$ for every point in order to reuse them in the following steps to build the integral. With this, the following is to update the value of the integral, using the sum of the previously calculated step at y and the step evaluated at its end point ($x(1)$), namely,

$$f_1(y) = f_0(x(0)) + f_0(x(1)) \qquad (4)$$

In this case, the end point of $f_1(x(0))$ is $x(2)$ as the calculation evaluates $f_0(x(1))$, and therefore the next iteration must evaluate f at $x(0)$ and $x(2)$ in order to grow the integral. In general, the k'th iteration of the procedure is calculated as

$$f_k(y) = f_{k-1}(x(0)) + f_{k-1}(x(end)) \qquad (5)$$

where '*end*' represents the parameter associated to the tail point of the trajectory.

3.3 Data Storing

Despite graphics APIs does not allow to transfer data calculated on shaders (such as matrices, vectors or single values), between rendering procedures, they can transfer images. An image is a 2D matrix that stores at each position a four-component vector that defines its color, i.e. $[1,h] \times [1,w] \rightarrow (R,G,B,A)$, with h and w its height and width in pixels respectively. A pixel color is defined by red, green blue and alpha values ($R,G,B,A \in [0,1]$).

As a result, for any field $X \in \mathbb{R}^2$ it is possible to store four values associated to any point $p_j \in X$ in an image that maps to X. Note that pixels are discrete values and p_j might be continuous. In this case, average and interpolation of data are required to both read and write the values on the image that maps to the field, and therefore avoid high error between calculations. If more than four values are required to be stored, it is needed to render (calculate) more than one image in order to store those values.

4 CASE STUDY

Many data services regards to the measurement of earth's information such as wind flow, superficial temperature, ocean salinity and others. For this application, the vector field that defines the velocity is required. The Global Ocean Data Assimilation System (GODAS)[1] provides this information in a netCDF format (Hankin et al., 2010), which allows, throughout servers, to visualize the data as an image information. The ocean's longitudinal and latitudinal components of the velocities are gray-scale images. They represent for a full white color ($R = G = B = 1.0$) a velocity component of 3.0 and for a full black color ($R = G = B = 0.0$) a velocity component of -3.0.

Taking advantage of massive parallel computing capabilities and data interpolation of SIMD programming architectures, it is possible to optimize the calculation of the integrals using images. Although

[1] http://www.esrl.noaa.gov/psd/

graphic libraries can not return or store previously calculated data as arbitrary matrices, they are capable of performing render-to-texture operations. This allows to store a rendered image in GPU's memory instead of displaying it, to then access it from another rendering procedure and access its information. As a result, images might be used as data storing elements instead of graphical information matrices.

Regarding that LIC integrates along both the upstream and the downstream directions hierarchical integration must be implemented separately for each direction and then merged to obtain the final image $\bar{I}$.

The mapping image (M) is a gray-scale noise image. Therefore the values of each color components are the same for each pixel, i.e. $R = G = B$. The weight field used is $W(q) = 1.0 \ \forall q \in X$. In order to perform LIC with hierarchical integration, for both directions, one value is needed to store the partial integral (since M is single-valued), and two values are required to store the two coordinates that define the final point of the partially evaluated trajectory. As a consequence, two images are required to store the values for each point q in the field. For this purpose let us define the following convention of data storing in the next two images in order to store the data:

Image $Q : [1, h_Q] \times [1, w_Q] \rightarrow (R, G, B, A)$, is used to store the data in the upstream direction for each point q_j in the field. The first component (where the red component is stored) is used to store the partially evaluated convolution and the second and third components (green and blue components) store the coordinates of the tail point of the trajectory. The fourth component is always used as 1.0. Note that since the weight field is 1.0 for all points in the field, the convolution becomes the average of all values of M along the partially evaluated trajectory. As a result, the update throughout the iterations of the image is calculated as follows

$$Q_k(q_j) = \begin{bmatrix} \frac{Q_{k-1}(p(0)).R + Q_{k-1}(p(end)).R}{2} \\ Q_{k-1}(p(end)).G \\ Q_{k-1}(p(end)).B \end{bmatrix} \quad (6)$$

As a consequence, the final visualization of the flow is calculated as the average between the integral components of images Q and S.

4.1 Data Storage in WebGL

WebGL permits to set up vertex and index buffers, to change rendering engine state such as active texture units or transform matrices, and to invoke drawing primitives. It also allows to perform render-to-texture operations using FBOs, which allow to render into images instead than displaying the information. WebGL also incorporates floating-point texture management as color attachments in FBOs, which enables to store the rendered images with floating point precision values, instead of 8-bit integers.

In order to perform render-to-texture operations in WebGL it is needed to define the necessary FBOs. Each FBO has an associated texture (image) to which the rendering code is performed. Additionally, for the matter of this implementation, it is important to consider that when the rendering procedure of an FBO is being performed, it does not allow to access the same memory space (in GPU) to which the rendering operation will write the information.

As a result, the flow visualization, regarding the hierarchical integration, requires the twice as FBOs than the number of rendered images, this is, one FBO is used to calculate and update the image and the other is used to copy the resulting image in another place of memory in order to allow data to be reused for the next iteration. Hence 4 FBOs are required to perform LIC flow visualization for this case study.

4.2 Flow Visualization in WebGL

Final results are shown in figure 3. The final LIC image $\bar{I}$ is used to determine the brightness of each point in the field. Color-mapping is determined by the speed at each point in a warm-cold color-scale.

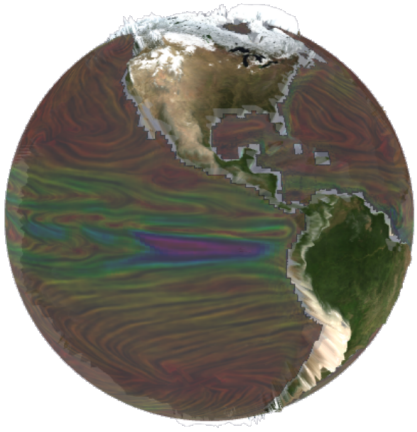

Figure 3: Visualization of the Pacific and Atlantic oceans currents using LIC with hierarchical line integration in WebGL. Four iterations achieve 32 integration steps and shows distinguishable streams.

Final results were obtained after 4 hierarchical iterations (see figure 3), this is, 16 total integration steps for both the upstream and downstream directions, making a detailed visualization with long and distinguishable trajectories.

5 CONCLUSIONS AND FUTURE WORK

This article presents a hardware accelerated LIC flow visualization in WebGL. The procedure is suitable to be performed on multi-platform and in low-capacity devices such as smart phones and tablet computers since no other API or plug-in is required to perform the procedure, and the number of iterations that is usually required is reduced. The procedure is not only suitable for WebGL implementation but for any render-to-texture capable parallel graphics implementation.

It is proposed the development of a human user interaction application for the visualization of oceanic currents as a subsequent activity to this work. Apart from this, other topics might be suitable as future work such as the implementation of this methodology to visualize 3D vector fields and to other integration-based visual applications.

ACKNOWLEDGEMENTS

This work was partially supported by the Basque Government's ETORTEK Project (ITSASEUSII) research program and CAD/CAM/CAE Laboratory at EAFIT University and the Colombian Council for Science and Technology COLCIENCIAS. GODAS data (ocean velocity information) was provided by the NOAA/OAR/ESRL PSD, Boulder, Colorado, USA, from their Web site at *http://www.esrl.noaa.gov/psd/*, and ohter information such as earth's bathymetric, topologic and satellite images, was provided by the NASA's Earth Observatory from their web site *http://earthobservatory.nasa.gov/*.

REFERENCES

Cabral, B. and Leedom, L. (1993). Imaging vector fields using line integral convolution. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 263–270. ACM.

Callieri, M., Andrei, R., Di Benedetto, M., Zoppè, M., and Scopigno, R. (2010). Visualization methods for molecular studies on the web platform. In *Proceedings of the 15th International Conference on Web 3D Technology*, pages 117–126. ACM.

Congote, J., Segura, A., Kabongo, L., Moreno, A., Posada, J., and Ruiz, O. (2011). Interactive visualization of volumetric data with webgl in real-time. In *Proceedings of the 16th International Conference on 3D Web Technology*, pages 137–146. ACM.

Forssell, L. and Cohen, S. (1995). Using line integral convolution for flow visualization: Curvilinear grids, variable-speed animation, and unsteady flows. *Visualization and Computer Graphics, IEEE Transactions on*, 1(2):133–141.

Hankin, S., Blower, J., Carval, T., Casey, K., Donlon, C., Lauret, O., Loubrieu, T., Srinivasan, A., Trinanes, J., Godoy, O., et al. (2010). Netcdf-cf-opendap: Standards for ocean data interoperability and object lessons for community data standards processes. In *Oceanobs 2009, Venice Convention Centre, 21-25 septembre 2009, Venise*.

Hlawatsch, M., Sadlo, F., and Weiskopf, D. (2011). Hierarchical line integration. *Visualization and Computer Graphics, IEEE Transactions on*, 99:1–1.

Kenwright, D. and Mallinson, G. (1992). A 3-d streamline tracking algorithm using dual stream functions. In *Proceedings of the 3rd conference on Visualization'92*, pages 62–68. IEEE Computer Society Press.

Klassen, R. and Harrington, S. (1991). Shadowed hedgehogs: A technique for visualizing 2d slices of 3d vector fields. In *Proceedings of the 2nd conference on Visualization'91*, pages 148–153. IEEE Computer Society Press.

Lane, D. (1994). Ufat: a particle tracer for time-dependent flow fields. In *Proceedings of the conference on Visualization'94*, pages 257–264. IEEE Computer Society Press.

Laramee, R. S., Hauser, H., Doleisch, H., Vrolijk, B., Post, F. H., and Weiskopf, D. (2004). The state of the art in flow visualization: Dense and texture-based techniques. In *Computer Graphics Forum*, volume 23, pages 203–221. Wiley Online Library.

Liu, Z. and Moorhead, R. (2005). Accelerated unsteady flow line integral convolution. *IEEE Transactions on Visualization and Computer Graphics*, pages 113–125.

Liu, Z. and Moorhead II, R. (2004). Visualizing time-varying three-dimensional flow fields using accelerated uflic. In *The 11th International Symposium on Flow Visualization*, pages 9–12. Citeseer.

McLoughlin, T., Laramee, R. S., Peikert, R., Post, F. H., and Chen, M. (2010). Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum*, volume 29, pages 1807–1829. Wiley Online Library.

Van Wijk, J. (2003). Image based flow visualization for curved surfaces. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 17. IEEE Computer Society.

Van Wijk, J. J. (2002). Image based flow visualization. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 745–754. ACM.