VISUALIZATION OF FLOW FIELDS IN THE WEB PLATFORM

Mauricio Aristizabal
Universidad EAFIT
[1]CAD/CAM/CAE Laboratory
Carrera 49 # 7 Sur - 50
Colombia (050022), Medellin, Antioquia
maristi7@eafit.edu.co

John Congote[1]
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
jcongote@vicomtech.org

Alvaro Segura
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
asegura@vicomtech.org

Aior Moreno
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
amoreno@vicomtech.org

Harbil Arregui
Vicomtech Research Center
Mikeletegi Pasealekua, 57
Parque tecnológico
Spain (20009), Donostia - San Sebastian, Guipuzcoa
harregui@vicomtech.org

Oscar E. Ruiz
Universidad EAFIT
CAD/CAM/CAE Laboratory
Carrera 49 # 7 Sur - 50
Colombia (050022), Medellin, Antioquia
oruiz@eafit.edu.co

ABSTRACT

Visualization of vector fields plays an important role in research activities nowadays. Web applications allow a fast, multi-platform and multi-device access to data, which results in the need of optimized applications to be implemented in both high and low-performance devices. The computation of trajectories usually repeats calculations due to the fact that several points might lie over the same trajectory. This paper presents a new methodology to calculate point trajectories over a highly-dense and uniformly-distributed grid of points in which the trajectories are forced to lie over the points in the grid. Its advantages rely on a highly parallel computing implementation and in the reduction of the computational effort to calculate the stream paths since unnecessary calculations are avoided by reusing data through iterations. As case study, the visualization of oceanic streams in the web platform is presented and analyzed, using WebGL as the parallel computing architecture and the rendering engine.

Keywords

Streamlines, Trajectory, Hierarchical Integration, Flow Visualization, WebGL.

1 INTRODUCTION

Vector field visualization plays 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 flow paths.

In particular, ocean flow visualization is important in maritime navigation and climate prediction, since the movement of sea water masses produces variations in air temperature and wind currents. Therefore, flow visualization becomes determinant to represent the ocean's behavior.

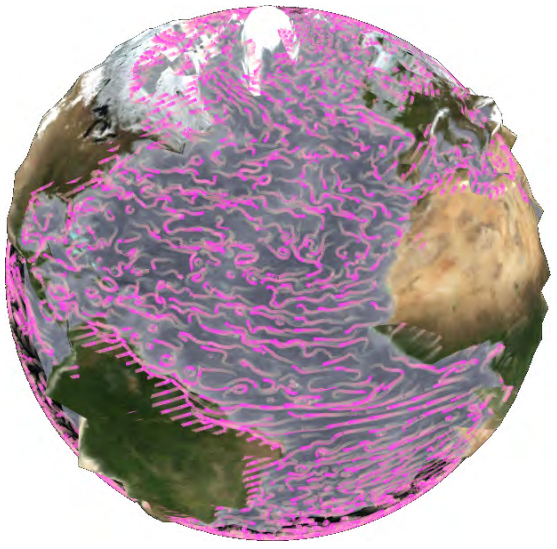


Figure 1: flow visualization of Atlantic ocean currents in WebGL. Hierarchical integration was used to reduce the total number of iterations required to calculate the paths.

With the growth of a great diversity of devices, development of multi-platform applications has become a common goal for developers. The Web is de-facto a universal platform to unify the development and execution of applications. However, challenges arise since applications must be optimized in order to be useful as well as on high as on low-performance devices.

The development of parallel computing hardware for all the different devices is increasing and the development of applications and computer-based procedures are taking in advance this capability. The contribution of this paper is a new methodology to calculate point trajectories of a highly dense grid of points over n-dimensional vector fields, in which the trajectories are forced to pass over the grid points (Figure 1). This allows to implement a hierarchical integration procedure ([HSW11]), which takes advance of previously calculated data in order to avoid repetitive and unnecessary calculations, and reduces the complexity of the algorithm from linear to logarithmic. The procedure is suitable to be implemented over highly parallel computing architectures due to independent calculations and the number of computations to be performed. We employ WebGL as the parallel computing engine to calculate the iterations, using the inherently parallel rendering procedure, and images are used to store the data through the iterations.

Different from other procedures, in which the calculation of the trajectories is performed for each point in particular, our methodology allows to merge its calculation for all the points in which the field is discretized. Therefore, the number of unnecessary computations is critically reduced.

This paper is organized as follows: Section 2 presents the related work. Section 3 exposes the methodology in which the contribution of this work is explained. Section 4 presents a case of study in oceanic currents and finally section 5 concludes the article.

2 LITERATURE REVIEW

2.1 Flow Visualization

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 ([KH91]), streamlines ([KM92]) and streaklines ([Lan94]). However, as these are discrete approaches, the placement of each object is critical to detect the flow's anomalies (such as vortexes or eddies), and therefore, data preprocessing is needed to perform an illustrative flow visualization. An up-to-date survey on geometric-based approaches is presented by [MLP10].

However, in terms of calculating those trajectories for determined points in the field, the procedures usually compute for each point the integrals, and, as a result, the procedures are computationally expensive for highly dense data sets.

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 (instead of sparse and discrete) flow visualization is produced. A deep survey in the topic on texture-based flow visualization techniques is presented by [LHD04].

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 is presented by [VW02]. Then, in [VW03] the images are mapped to a curved surface, in which the transformed image visualizes the superficial flow.

Line Integral Convolution (LIC), presented by [CL93], is a widely implemented texture-based flow visualization procedure. It convolves the associated texture-pixels (texels) with some noise field (usually a white noise image) over the trajectory of each texel in some vector field. This methodology has been extended to represent animated ([FC95]), 3D ([LMI04]) and time varying ([LM05, LMI04]) flow fields.

An acceleration scheme for integration-based flow visualization techniques is presented by [HSW11]. 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. As a result, the computational effort is reduced, compared to serial integration techniques, from $O(N)$ to $O(\log N)$, where N refers to the number of steps to calculate the integrals. Its implementation is performed on Compute Unified Device Architecture (CUDA), which allows a parallel computing scheme performed in the Graphics Processing Unit (GPU), and therefore the computation time is critically reduced. However, it requires, additionally to the graphic Application Programming Interface (API), the CUDA API in order to reuse data, and hence, execute the procedure.

2.2 WebGL literature review

The Khronos Group released the WebGL 1.0 Specification in 2011. It is a JavaScript binding of OpenGL ES 2.0 API and allows a direct access to GPU graphical parallel computation from a web-page. Calls to the API are relatively simple and its implementation does not require the installation of external plug-ins, allowing an easy deployment of multi-platform and multi-device applications. However, only images can be transfered between rendering procedures using framebuffer objects (FBOs).

Several WebGL implementations of different applications have been done such as volume rendering, presented by [CSK11] or visualization of biological data,

presented by [CADB10]. A methodology to implement LIC flow visualization with hierarchical integration, using only WebGL, was presented by [ACS12], in which FBOs are used to transfer data between different rendering procedures, and therefore allowing to take in advance the parallel computing capabilities of the rendering hardware, in order to perform the different calculations. However, for the best of our knowledge, no other implementation that regards to streamline flow visualization on WebGL has been found in the literature or in the Web.

2.3 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 calculation of point trajectories in n-dimensional vector fields over highly dense set of points, forcing the trajectories to lie over the points in the set. As a consequence, previously calculated data can be reused using hierarchical integration, avoiding unnecessary calculations and reducing the complexity of the algorithm.

3 METHODOLOGY

The problem that we address is stated as: given a set of points and a vector field that exists for all of these points, the goal is to find the finite trajectory that each point will reproduce for a certain period of time.

Normal calculation of point trajectories in n-dimensional vector fields, requires to perform numerical integration for each particular point in order to reproduce the paths. In the case of a dense set of points, the procedures suffer from unnecessary step calculations of the integrals, since several points in the field might lie over the same trajectory of others. Hence, some portions of the paths might be shared. Figure 2 illustrates this situation.

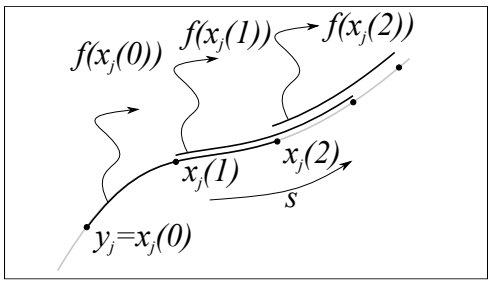


Figure 2: Trajectory overlapping in several point paths.

In order to avoid repeated computations, we propose a methodology to calculate trajectories of highly dense grid of points, in which the paths are forced to lie over the points in the grid, i.e., the paths are generated as a Piecewise Linear (PL) topological connection between a set of points that approximates the trajectory. With this, hierarchical integration [HSW11] is employed to iteratively compute the paths and reuse data through the iterations.

3.1 Hierarchical Integration

Since line integration over n-dimensional vector fields suffers from repeated calculations, hierarchical integration [HSW11] only calculates the necessary steps and then iteratively grows the integrals reusing the data. This reduces the computational complexity of the algorithm from $O(N)$, using serial integration, to $O(\log N)$. The procedure is summarized as follows.

For an arbitrary point in the field $y \in Y$ with $Y \subseteq \mathbb{R}^n$, let us define $f : Y \rightarrow \mathbb{R}^m$, as an arbitrary line integral bounded by its trajectory c_y. Consider its discrete approximation as described in equation 1.

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

where t is the maximum number of steps required to reproduce c_y with Δs the step size. $x(0) = y$ is the starting point of the trajectory to be evaluated and w is the function to be integrated. The integration procedure is performed for all points $y \in Y$ in parallel.

We 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. Namely,

$$f_0(y) = w(x(1)) \qquad (2)$$

It is required to store the last evaluated point $x(1)$ over the growing trajectory and the partial value of the integral for all the points y in order to reuse them in the following steps to build the integral. With this, the next action 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,

$$f_1(y) = f_0(x(0)) + f_0(x(1)) \qquad (3)$$

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

$$f_k(y) = f_{k-1}(x(0)) + f_{k-1}(x(end)) \qquad (4)$$

It is important to remark that each iteration of this procedure evaluates two times the integration steps evaluated in the previous iteration. As a result, the total number of integration steps t is a power of two,

and the hierarchical iterations required to achieve this evaluations is reduced by a logarithmic scale, i.e., $k = \log_2 t$. Also notice that the evaluation of the vector field is performed only once, in the calculation of the first step, which avoids unnecessary evaluations of the vector field, which are computationally demanding for complex vector fields. Figure 3 illustrates the procedure up to four hierarchical steps.

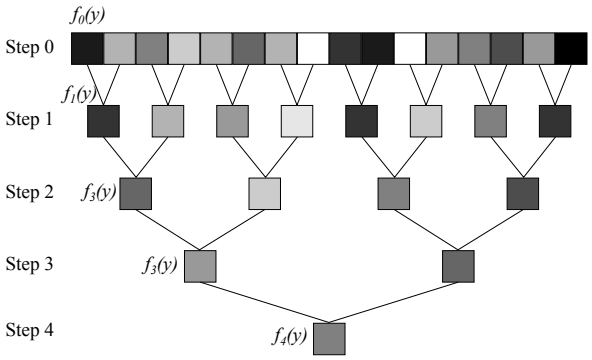


Figure 3: Exponential growth of hierarchical integration methodology. At step 3, the procedure evaluates 8 serial integration steps, meanwhile at step 4 it evaluates 16 serial integration steps.

3.2 Stream Path Calculation

In order to perform the visualization of a vector field using trajectory paths, lets assume a homogeneously distributed set of points

$$Y = \{y,\ z \in \mathbb{R}^n | y - z = \Delta y, \\ \Delta y \text{ is constant } \forall z \text{ adjacent to } y\} \quad (5)$$

and a n-dimensional vector field $F : \mathbb{R}^n \to \mathbb{R}^n$. The goal is to calculate for each point $y \in Y$, the PL approximation of the trajectory that the point will describe according to F, defined by the topological connection of a particular set of points $A_y \subset Y$. Figure 4 illustrates the approximation.

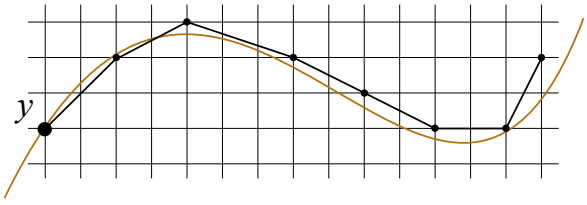


Figure 4: PL approximation of the trajectory by the procedure.

The trajectory c_y of an arbitrary point y in the field is defined as

$$c_y = x_y(s) = \int_l F(x_y(s))ds \quad (6)$$

where l represents a determined length of integration.

Using hierarchical integration, for each point in the field the first step of the PL trajectory is calculated, this is, the corresponding end point of the first step of the integral is computed using a local approximation of the point in the set of points.

$$x_y(0) = y \quad (7)$$

$$y' = x_y(0) + \gamma F(x_y(0)) \quad (8)$$

$$x_y(1) = \arg\min_{x_y}(Y - y') \quad (9)$$

where y' is the first iteration result of the Euler integration procedure, γ is a transformation parameter to adjust the step given by the vector field to the local separation of the set of points and $x_y(1)$ is defined as the closest point in Y that approximates y'. The value of $x_y(1)$ is then associated (and stored) to y. The set A_y contains the reference to the points of the trajectory that describes y, and therefore for equation 8, A_y is defined as:

$$A_y^0 = \{x_y(1)\} \quad (10)$$

Similarly to the hierarchical integration procedure, the next steps are performed to join the calculated steps in order to grow the trajectories. Therefore, for each point y, its computed trajectory is joint with its last point's trajectory, this is, for the step in equation 8.

$$A_y^1 = A_y^0 \cup A_{x_y(1)}^0 = \{x_y(1), x_y(2)\} \quad (11)$$

Note that each iteration of the procedure will increase the number of points in the trajectory by a power of two. Therefore, the growth of the paths is exponential. In general, the k'th iteration is calculated as

$$A_y^k = A_y^{k-1} \cup A_{x_y(2^k)}^{k-1} = \{x_y(1), x_y(2), \ldots, x_y(2^{(k+1)})\} \quad (12)$$

The accuracy of the procedure is strongly determined by the discretization (density) of Y, since it is directly related to the step size in the integration procedure, i.e., the approximation of the first step end-point is determinant. In order to increase the accuracy of the procedure, the computation of the first step can be calculated with e.g., a 4th order Runge Kutta numerical integration, however, it might significantly increase the computation time of the procedure if the computation time of the vector field function is relevantly high.

3.3 Time-Varying Data

In terms of unsteady flow fields, i.e., time-varying vector fields, the generation of the trajectories might seem difficult. In that case, as proposed in [HSW11], time is considered another dimension of the vector field.

Therefore, the set of points is formed with the position coordinates of the points and discretized time steps, producing an $n+1$-dimensional grid of points.

It is also determinant for the accuracy of the procedure that the density of the discretization set is high, in order to increase the precision of the approximated trajectories.

3.4 Animation

Dynamic scenes are demanding in most of the visualization procedures. We consider in this section two kinds of dynamic scenes. A first kind of procedures refers to when the vector field is steady, i.e., it remains constant through the time. In this case, the goal is to visualize the motion of the particle all across the field.

Since the paths for all the points in the field are calculated, the representation of the particle's trajectory through the frames is simple. Consider a point y and its approximated trajectory given by the set of points A_y. Notice, as described in section 3.2, that the first point of the set $A_y[1]$, i.e., $x_y(1)$, represents the next point in which y will lie in a determined period of time. As a result, at a posterior frame, the displayed trajectory should be $A_{x_y(1)}$.

The second type of procedure refers when vector field is varying with the time. Complementary to the animation stated before, a second kind of dynamic scene is comprised since it is also important to visualize the changes that a trajectory suffers in the time. In the case of time varying data, as in the steady case, all the points have an associated trajectory. In order to animate the change of one trajectory, from one frame to another, the trajectory that will be represented refers to the one of the point with the same point coordinate, but the next time coordinate. i.e., $A_{y,t+\Delta t}$.

4 CASE STUDY

In this section the visualization of 2D oceanic currents using the proposed methodology is performed. The implementation has been done in WebGL, so the methodology's parallel computing capabilities are fully used. WebGL offers the possibility to use the rendering procedure to calculate images (textures) through Framebuffer Objects, and then use those rendered textures as input images for other rendering procedures. As a consequence, for this implementation we associate the pixels of an image to the points in the field, and therefore, the rendering procedure is used to compute the different hierarchical iterations, which are stored in the color values of the pixels. Finally, the trajectories are hierarchically constructed. The implementation was performed on an Intel Core2Quad 2.33 GHz with 4 GB of RAM and with a nVidia GeForce 480.

4.1 Implementation

For a $w \times h$ grid of points (w and h being its width and height respectively in number of elements), images of size $w \times h$ in pixels are used, in which a particular pixel (i, j) is associated with the point (i, j) in the grid. Since for each particular pixel, a four component vector is associated, i.e., a vector of red, green, blue and alpha values, each value can be associated as a particular position of another pixel. This is, if the value of a pixel is r, then its associated pixel coordinates are given by

$$i = r \bmod w \tag{13}$$

$$j = \frac{r-i}{w} \tag{14}$$

where mod represents the remainder of the division of r by w. As a result, if for each hierarchical integration, only the last point of the trajectory is to be stored, then one image can store four hierarchical iterations.

For the zero'th hierarchical iteration, and the image I to store its calculation, the value of a pixel (i, j) is given by

$$i_0 = i + kF_i(i, j) \tag{15}$$

$$j_0 = j + kF_j(i, j) \tag{16}$$

$$\tag{17}$$

where the parameter '0' refers to the hierarchical step 0, k represents the scaling factor of the vector field, and $F_i(i, j)$ represents the component of the vector field over the direction of i, evaluated at the point (i, j). The vector field used in this case study is shown in figures 5(a) for the direction of i and 5(b) for the direction of j.

In general, the k'th step is calculated as follows,

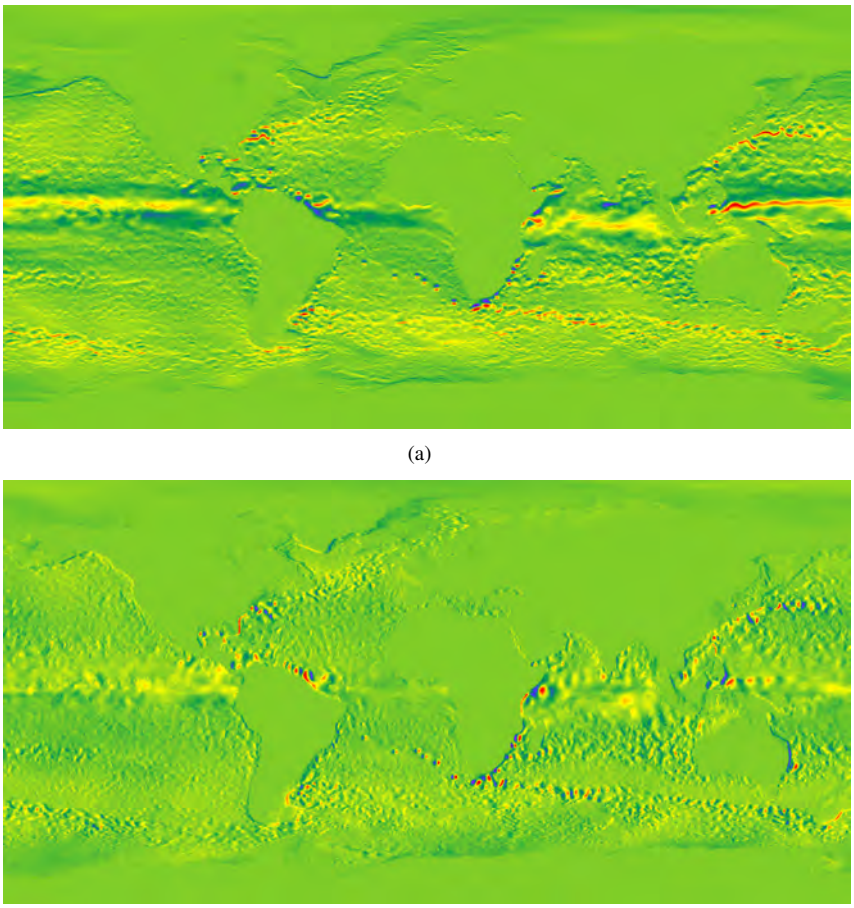
$$i_{next} = I(i, j, k-1) \bmod w \tag{18}$$

$$j_{next} = \frac{I(i, j, k-1) - i_{next}}{w} \tag{19}$$

$$I(i, j, k) = I(i_{next}, j_{next}, k-1) \tag{20}$$

In the case that k is greater than four, then more images are used to store the values of the hierarchical iterations.

With that, all the desired steps are stored in the necessary images. In order to build the trajectories from those images, a generic line, formed by k^2 points, is required. Each point in the trajectory needs to have an associated value, that refers to its order in the trajectory, i.e., the first point in the trajectory has an index 0, the second point the value 1 and so on. With this index associated to each point of the trajectory of the point y, the position of each point is calculated as described in Algorithm 1, where HL is the hierarchical level that needs to be evaluated and the function $evalHL()$ returns the new position of the point y, for a particular hierarchical level.



(a)

(b)

Figure 5: Oceanic currents information. Color-scale of the oceanic currents magnitude in the (a) longitudinal and (b) latitudinal directions, of the 1^{st} January of 2010. Images generated using data from the NASA's ECCO2 project.

Require: $index$ Index referring to the element position into the line.
y Position of the initial point of the trajectory.

Ensure: y_{end} Position of the i'th element of the line in the space.

Finished = $false$
while not Finished **do**
  $HL = \text{floor}(\log_2(index))$
  $index = index - 2^{HL}$
  $y = \text{evalHL}(HL, y)$
  if $index == 0$ **then**
    $y_{end} = y$
    Finished = $true$

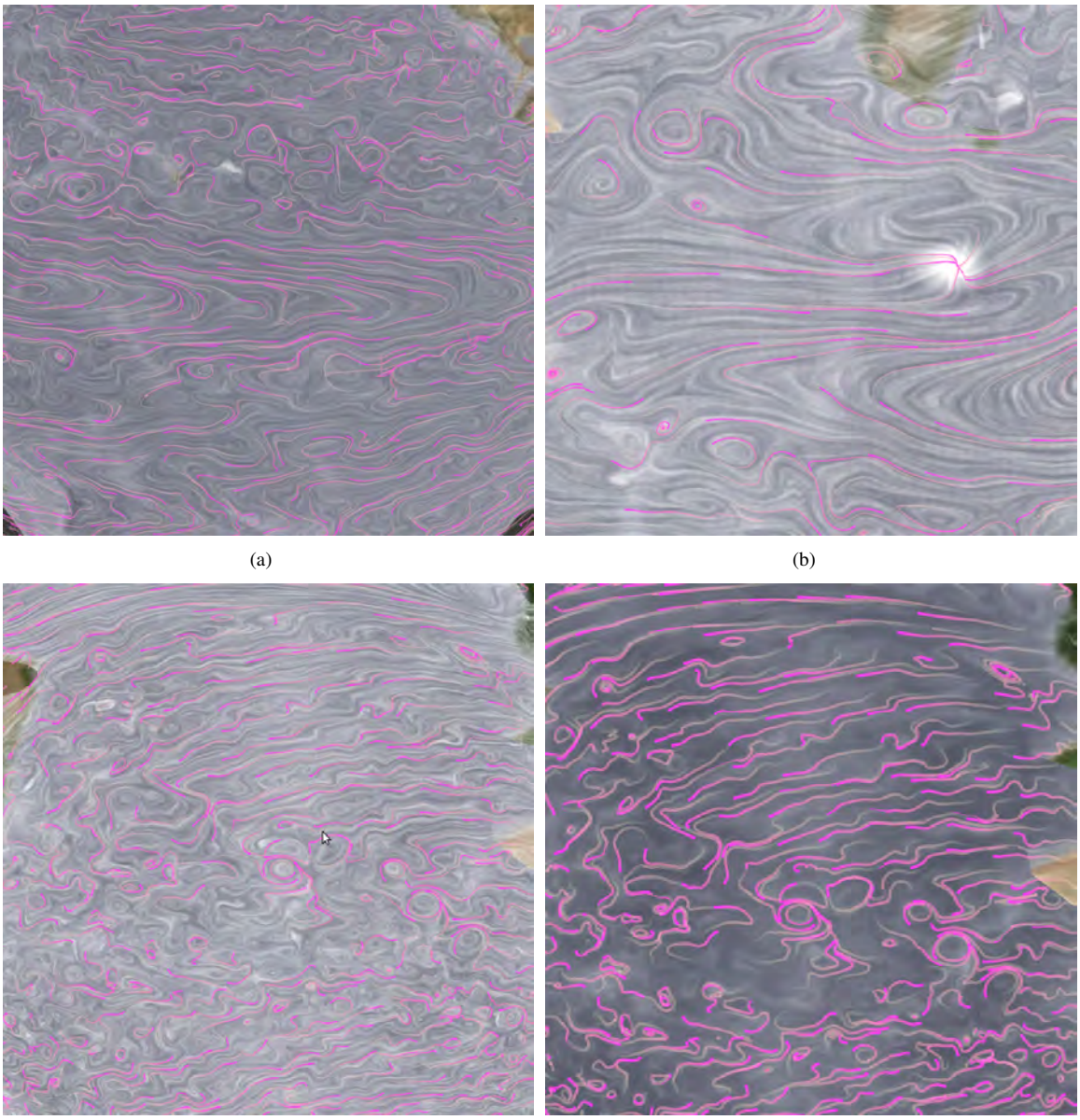
Algorithm 1: Procedure to reconstruct the streamlines using the already calculated hierarchical levels.

4.2 Results

A general grid of 2048×2048 points is used as the world's discretization. The vector field information was acquired from the NASA's ECCO2 project (see figure 5), in which high-resolution data is available. Only six hierarchical levels, i.e., $2^6 = 64$ points in the trajectory are used for each point in the field, as a result only 2 images are required to calculate the trajectories.

The time needed to compute all the hierarchical levels (from 0 to 6) was 3 ms. The trajectory computation was performed to 10000 equally-distributed points all over the field, which means that 64000 points need to be transformed by the trajectory computation. The computation time of those trajectories was 670 ms (Final results are shown in Figure 6).

In order to compare the visualization using the proposed methodology, the LIC visualization of the vector field using the methodology proposed in [ACS12] was inserted below the stream line visualization. It shows that for this level of discretization (2048×2048), the visualization is correct and accurate. However, sparse data might produce inaccurate results.



(a) (b)

(c) (d)

Figure 6: Different view points of the final visualization of the oceanic currents using hierarchically calculated streamlines. Six hierarchical steps were used to achieve this visualization. The LIC visualization of the flow is used below the streamlines in order to visually compare the different methods and to enhance the visualization

5 CONCLUSIONS AND FUTURE WORK

This article presents a novel methodology to calculate points trajectories in highly dense point sets, in which the trajectories are formed as piecewise-linear connections of the points in the set. This allows to merge the calculation of the different trajectories and use iteratively the data to avoid repeated and unnecessary calculations, hence, accelerating the process. The procedure is suitable to be implemented in parallel architectures such as WebGl, OpenCL or CUDA, since the calculations of the integrals for one point are independent from the calculation of the other points.

As a result, thanks to the use of hierarchical integration, the procedure reduces the computational complexity of the calculation of the trajectories from linear to logarithmic. The methodology deals with n-dimensional and time-varying data and animated visualization can be easily achieved due to the fact that the trajectories are calculated for all the points in the set.

Since the procedure performs an approximation of the trajectories using a piecewise-linear connection of the points in the set, the accuracy of the algorithm is strongly influenced by the discretization distance between the points, because this distance determines a lower bound in the integration step to be used.

Ongoing research focuses on the adaptation of this methodology to require less computational effort (processing and memory effort), so that extremely low-performance devices such as smart-phones and tablets might be able to perform an accurate and complete flow visualization using streamlines. Related future work includes the adjustment of the grid point positions along the iterations and the increase in the accuracy of the calculated trajectories.

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. The vector field information was acquired form NASA's ECCO2 project in *http://ecco2.jpl.nasa.gov/*.

6 REFERENCES

[ACS12] Mauricio Aristizabal, John Congote, Alvaro Segura, Aitor Moreno, Harbil Arriegui, and O. Ruiz. Hardware-accelerated web visualization of vector fields. case study in oceanic currents. In Robert S. Laramee Paul Richard, Martin Kraus and José Braz, editors, *IVAPP-2012. International Conference on Computer Vision Theory and Applications*, pages 759–763, Rome, Italy, February 2012. INSTICC, SciTePress.

[CADB10] M. Callieri, R.M. Andrei, M. Di Benedetto, M. Zoppè, and R. Scopigno. Visualization methods for molecular studies on the web platform. In *Proceedings of the 15th International Conference on Web 3D Technology*, pages 117–126. ACM, 2010.

[CL93] 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.

[CSK11] J. Congote, A. Segura, L. Kabongo, A. Moreno, J. Posada, and O. Ruiz. 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, 2011.

[FC95] L.K. Forssell and S.D. Cohen. 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, 1995.

[HSW11] M. Hlawatsch, F. Sadlo, and D. Weiskopf. Hierarchical line integration. *Visualization and Computer Graphics, IEEE Transactions on*, (99):1–1, 2011.

[KH91] R.V. Klassen and S.J. Harrington. 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, 1991.

[KM92] D.N. Kenwright and G.D. Mallinson. 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, 1992.

[Lan94] D.A. Lane. Ufat: a particle tracer for time-dependent flow fields. In *Proceedings of the conference on Visualization'94*, pages 257–264. IEEE Computer Society Press, 1994.

[LHD04] R.S. Laramee, H. Hauser, H. Doleisch, B. Vrolijk, F.H. Post, and D. Weiskopf. 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, 2004.

[LM05] Z. Liu and R.J. Moorhead. Accelerated unsteady flow line integral convolution. *IEEE Transactions on Visualization and Computer Graphics*, pages 113–125, 2005.

[LMI04] Z. Liu and R.J. Moorhead II. Visualizing time-varying three-dimensional flow fields using accelerated uflic. In *The 11th International Symposium on Flow Visualization*, pages 9–12. Citeseer, 2004.

[MLP10] T. McLoughlin, R.S. Laramee, R. Peikert, F.H. Post, and M. Chen. Over two decades of integration-based, geometric flow visualization. In *Computer Graphics Forum*, volume 29, pages 1807–1829. Wiley Online Library, 2010.

[VW02] J.J. Van Wijk. Image based flow visualization. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 745–754. ACM, 2002.

[VW03] J.J. Van Wijk. Image based flow visualization for curved surfaces. In *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, page 17. IEEE Computer Society, 2003.