Real-Time Interactive Time Correction on the GPU

Mai Elshehaly* Virginia Tech
Denis Gračanin † Virginia Tech
Mohamed Gad Ain Shams University
Junpeng Wang Virginia Tech
Hicham G. Elmongui ‡ Alexandria University

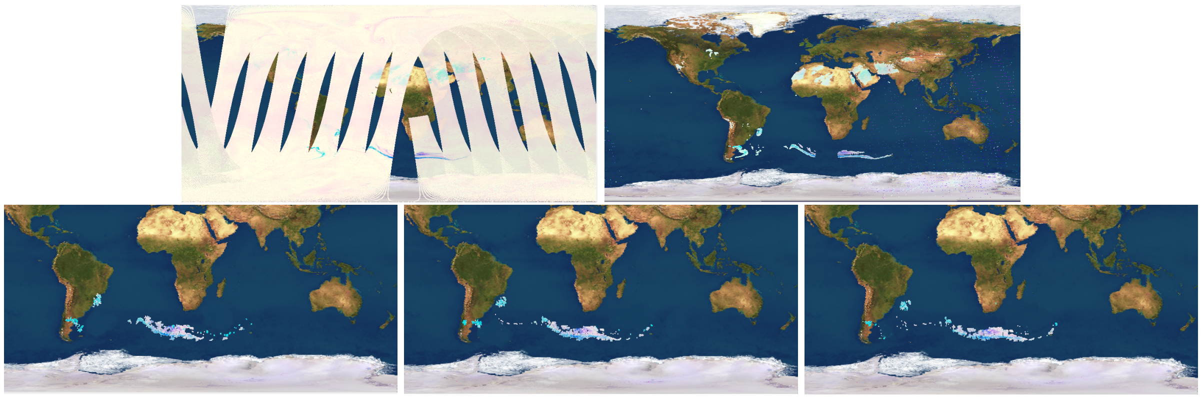

Figure 1: **Top:** Data collected by NASA's Aqua satellite (left) and extracted ash plume (right) during the first 12 hours of June 8th 2011; **Bottom:** results of interactive time correction for the plume at 3:00 AM (left), 6:00 AM (middle) and 9:00 AM (right) of the same day.

Abstract

The study of physical phenomena and their dynamic evolution is supported by the analysis and visualization of time-enabled data. In many applications, available data are sparsely distributed in the space-time domain, which leads to incomprehensible visualizations. We present an interactive approach for the dynamic tracking and visualization of measured data particles through advection in a simulated flow. We introduce a fully GPU-based technique for efficient spatio-temporal interpolation, using a *kd*-tree forest for acceleration. As the user interacts with the system using a time slider, particle positions are reconstructed for the time selected by the user. Our results show that the proposed technique achieves highly accurate parallel tracking for thousands of particles. The rendering performance is mainly affected by the size of the query set.

Index Terms: I.3.8 [COMPUTER GRAPHICS]: Applications

1 Introduction

Visualization of time-variant data offers insights about the dynamic behavior of physical phenomena. In order to gain such insight, domain scientists strive to obtain a sequence of instantaneous images with the highest possible information content at consecutive time instances. The availability of such instantaneous images at high enough temporal resolution can create an animation for the evolution of the phenomenon under study. Unfortunately, measurements obtained by scanning modalities can contain large spatio-temporal gaps, which make such animation impossible.

An example from the atmospheric science domain, where data collected over the course of 12 hours on June 8th, 2011 by the Atmospheric Infrared Sounder (AIRS) [2] is depicted in Figure 1 (top left). The orbits of the satellite are shown; each consisting of a large number of measurements at different (longitude, latitude) positions and different timestamps. An extracted ash plume which resulted from the "Puyehue Cordón Caulle" volcanic eruption (that took place on June 4th 2011 in Chile) is shown in the top right part of Figure 1. The plume is disconnected due to the spatio-temporal gaps between the satellite orbits. Namely, the patterns to the left of the gap were captured between 1:00 AM and 5:00 AM on June 8th; whereas the patterns to the right were captured between 9:00 AM and noon of the same day. The analysis of plume behavior is complicated by the fact that these sparsely detected patterns make the only information available about the plume during this 12 hours period. Plume shape and motion within this time frame are not available from raw measurements. We map this dataset to a dynamic simulated flow of air parcels [3], in order to fill the gaps and correct data through time. Both datasets where made available through the IEEE Scientific Visualization contest of 2014.

*e-mail: maya70@vt.edu
†e-mail:gracanin@vt.edu
‡elmongui@alexu.edu.eg

2 Spatio-Temporal Interpolation

We propose a novel GPU-parallel spatio-temporal interpolation technique that aims to reconstruct motion information at any ungauged location. Satellite detections are fed to the vertex shader, and are advected in parallel by interpolating flow trajectories. All particles are tracked in parallel and rendered at their updated positions at a destination time; one that is interactively specified by the user through a time slider. This results in a smooth animation of the plume, while filling the spatio-temporal gaps. The bottom sequence in Figure 1 shows the reconstructed plume at three different times within the first 12 hours of June 8th, 2011. Algorithm 1 summarizes the tracking process as executed by the vertex shader.

The vertex shader accepts an array containing, for each particle to be tracked, a vector **v** for the initial particle position (v_x, v_y, v_z) and the timestamp T_s of this incoming particle. The target destination time T_d as selected by the user is passed as a shader uniform value. The output per vertex is an updated position that tracks the given particle to time T_d. The newly calculated position is stored in the interpolated position vector (**ipv**) that is then passed to the fragment shader for rendering. In previous work [1], a spatio-temporal interpolation technique was proposed to perform this advection on

Algorithm 1 Vertex shader algorithm for point tracking.

```
INPUT:   $\mathbf{v} = (v_x, v_y, v_z, T_s)$   query vertex
INPUT:   $T_d$   user selected destination time
OUTPUT:   **ipv**   interpolated position vector
$T_1$ = round_to_nearest_simulation_step($T_s$);
$T_2$ = round_to_nearest_simulation_step($T_d$);
TIME_FRAME = define_tracking_time_frame($T_d$);
if $T_s \in TIME\_FRAME$ then
    $NN_{source}$ = traverse_kd_tree($T_1$);
    $NN_{dest}$ = fetch($NN_{source}$, $T_2$);
    for $\mathbf{p}_s \in NN_{source} \mid \exists_{=1} \mathbf{p}_d \in NN_{dest}$ do
        $\mathbf{p\prime}_s$ = ADJUSTT($\mathbf{p}_s, T_1, T_s$);
        $\mathbf{p\prime}_d$ = ADJUSTT($\mathbf{p}_d, T_2, T_d$);
        $L^2 = \|(v_x, v_y, v_z) - \mathbf{p\prime}_s\|$ ;
        $\mathbf{w} = \mathbf{w} + \|\mathbf{p\prime}_d - \mathbf{p\prime}_s\| / L^2$;
        $\mathbf{h} = \mathbf{h} + 1/L^2$;
    end for
    $\mathbf{D} = \mathbf{w}/\mathbf{h}$;
    $\mathbf{ipv} = (v_x, v_y, v_z) + \mathbf{D}$;
end if
```

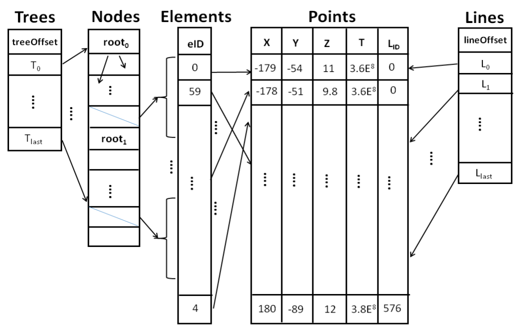

Figure 2: Data structures kept in SSBOs for interactive tracking.

the CPU. In this work, we present an acceleration data structure that enables the entire calculation to be performed on the GPU shader.

The particle tracking mechanism, as summarized in Algorithm 1, has two major bottlenecks which constitute the bulk of processing time: *(i)* identifying the k-nearest neighbors of a given query point in simulation step T_1, to retrieve a position vector $\mathbf{p}_s$ for each neighbor, and *(ii)* tracking these k neighbors through time to determine whether they exist or not at the destination time step T_2, and to fetch their corresponding positions $\mathbf{p}_d$ at that step. These two steps define two different access patterns to the flow field data residing in the GPU memory. Given the huge number of points typical to flow simulation results, linearly searching the dataset is unacceptable and will cause the GPU to timeout. Also keeping multiple copies of flow data in shader memory is not an option.

Figure 2 depicts data structures kept in Shader Storage Buffer Objects (SSBOs) to optimize memory access, and resolve these bottlenecks. **Points:** is a sorted array of structures that stores position, time, and trajectory information for all simulation points. Particles in this array are sorted by trajectory number to simplify time advection. This, however, makes neighborhood extraction unintuitive for the shader. **Lines:** contains the offset at which data corresponding to each trajectory begins in the **Points** array. **Elements:** is an indexing vector to support spatio-temporal neighborhood fetching. Constructing this vector starts with a hash function that maps indices of points to different buckets based on their timestamps. Next, the **Elements** array is passed as an initial vector to the kd-tree forest construction algorithm. As the kd-tree forest construction proceeds, element indices are reordered within the memory segment of each time step (bucket) separate from all others. **Nodes:** contains a linearization of tree nodes for all time steps. **Trees:** a tree offsets array that keeps track of where the root node for each kd-tree resides in the **Nodes** array.

2.1 Results

Figure 3 displays measured parameters every 12 hours sampled across a time span from 06/04/2011 to 06/16/2011 including number of particles to be tracked in each time frame (black), maximum heap size needed per tracking point (grey), and average frames per second (red curve). The major effect on performance is inflicted

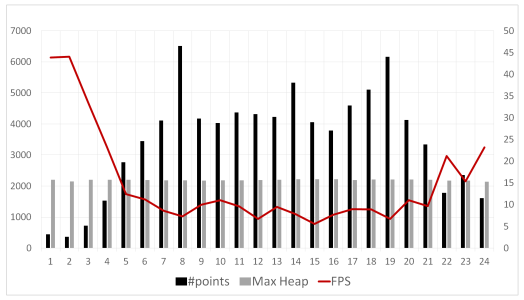

Figure 3: Twelve days of tracking ash detections: number of tracked points (black), maximum heap size (grey), and **fps** (red).

by the number of points in the tracking set that fall within a given time frame. This can be attributed to the fact that as the number of points increases, the GPU tries to accommodate their individual memory requirements to maintain a priority queue for a Best Bin First traversal for each.

The accuracy of the corrected plume, and its ability to emulate the exact shape of the volcanic plume are evaluated by calculating the cross correlation coefficient with a delineated plume on weather satellite images. A correlation range of 0.67 to 0.82 was obtained, and varies across time as the tracking error gets reset approximately every 12 hours, once satellite data becomes available.

3 Conclusion

We presented a real-time GPU-based particle tracking technique for the interactive visualization of dynamic physical phenomena. The proposed tracking algorithm is executed by OpenGL's vertex shader and requires no communication with the CPU. A novel combination of acceleration data structures was developed and optimized in order to achieve interactive rates. Our results have shown that real-time advection of particles and tracking them through the flow is feasibly achievable at interactive rates through intelligent use of the GPU's Shader Storage Buffer Objects (SSBOs). However, the memory requirements for kd-tree traversal pose some limitations on the number of particles that can be tracked and rendered in parallel. Our future work will focus on optimizing memory usage for Best Bin First traversal on the GPU and the analysis of tracking accuracy in the long run.

References

[1] M. Elshehaly, D. Gračanin, M. Gad, H. G. Elmongui, and K. Matkovic. Interactive Fusion and Tracking For Multi-Modal Spatial Data Visualization. *Computer Graphics Forum*, 2015.

[2] L. Hoffmann, S. Griessbach, and C. I. Meyer. Volcanic emissions from airs observations: detection methods, case study, and statistical analysis, 2014.

[3] D. S. McKenna, P. Konopka, J. Grooß, G. Gunther, and R. Müller. A chemical lagrangian model of the stratosphere (CLaMS). In *12th Conference on the Middle Atmosphere*, 2002.