Exploring Flow Fields with GPU-Based Stream Tracers in Virtual Environments

Marc Schirski,[†][1] Christian Bischof[2] and Torsten Kuhlen[1]

[1]Virtual Reality Group, RWTH Aachen University
[2]Institute for Scientific Computing, RWTH Aachen University

Abstract
In this paper we present an immersive visualization approach for the intuitive exploration of flow fields, which operates entirely within the graphics subsystem. We augment particle data with a brief history of their recent positions, thus effectively computing and displaying animated tracers facilitating the understanding of the underlying flow field. Image quality is enhanced by employing a billboard-based rendering method for the particle trajectories simulating lit tubular geometry. This leads to significantly reduced depth perception problems and depth order ambiguities. Interactivity is maintained even for large amounts of tracers by shifting the computational load to the GPU. We alleviate 3D seed point specification problems by offering interaction mechanisms with full 6 degrees-of-freedom within an immersive virtual environment.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Virtual Reality, I.3.6 [Computer Graphics]: Interaction Techniques, I.3.3 [Computer Graphics]: Display Algorithms

1. Introduction

A very intuitive means of visualizing fluid flow is the advection of matter inside a flow field and the depiction of its resulting movement. As such, it is a standard technique for flow visualization in experimental flow analysis and in the field of computational fluid dynamics (CFD). An additional approach to fluid flow analysis is the application of virtual reality (VR) techniques and methodology [BL91]. In this paper we present a system for interactively seeding and displaying stream tracers within immersive virtual environments. The extensive computational load is dealt with by employing graphics processing units (GPUs) for particle advection, data preparation for rendering, and rendering itself. For the latter we employ a high-quality, billboard-based rendering approach called Virtual Tubelets [SKH*05]. In order to utilize the advantages of virtual environments over desktop-based visualization systems, we implemented an interaction mechanism with full six degrees-of-freedom, thus alleviating problems caused by the mismatch between three-dimensional data and two-dimensional input techniques.

Due to the considerable processing power of modern programmable graphics hardware, a number of approaches for leveraging this power have been introduced recently. Applications range from very graphics-centric uses like raytracing to generalized numerical algorithms [OLG*05]. Unlike Krüger et al., who also employ particles for the exploration of a given flow field [KKKW05], we concentrate on an interactive high-quality visualization of particle traces, thus increasing visual feedback of the underlying fluid flow.

Another option for reducing response times for an interactive flow exploration is the parallelization of the computation on a dedicated computer system (e.g. [GHW*04]). Although this approach results in significant speed-ups, it is typically still lacking in terms of interactivity due to lag induced by data management and network communication. Alternatively, Kuester et al. [KBHJ01] precompute particle traces and load them at run-time depending on user input in a virtual environment, which requires considerable additional computing and storage resources.

2. Overview

The major part of our system is executed on the graphics subsystem of a given visualization computer. The main CPU is

[†] e-mail: {schirski, bischof, kuhlen}@rz.rwth-aachen.de

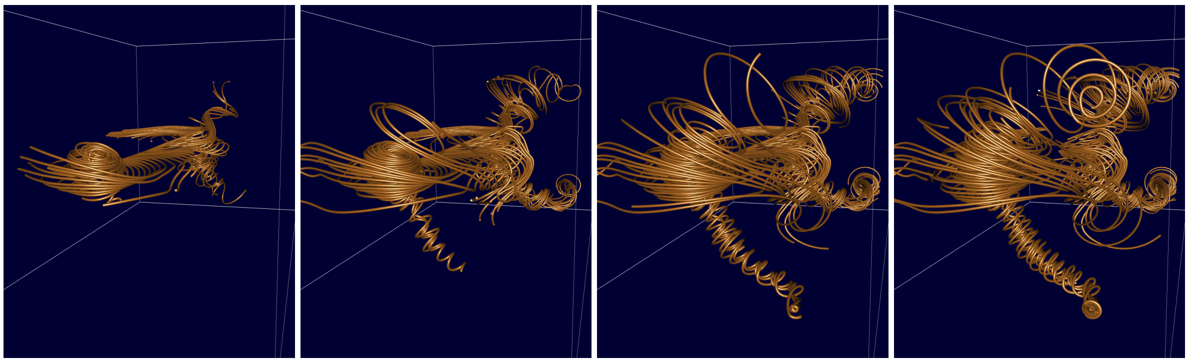

Figure 1: *Four snapshots of evolving, GPU-based particle traces on a data set containing two interacting vortices. All operations, from particle tracing to path reconstruction to rendering, are executed on the GPU.*

used only for the parametrization of the computation via user interaction and for initiating the computation and rendering process, resulting in an animated depiction of stream tracers moving through the flow field. Instead of dealing with instantaneous particle data, we concentrate on computing and visualizing animated particle traces. This results in a more coherent picture of the flow field due to prolonged display not only of the current position of any given particle, but also of its recent history. As a conventional, line-based depiction of particle trajectories suffers from depth perception problems and depth order ambiguities, we use a billboard-based approach for rendering, which creates the illusion of tubular geometry. Orienting triangle strips towards the viewer, capping these strips to avoid visual artifacts, and faking illumination via normal mapping lead to a convincing simulation of complex geometry at little computational cost.

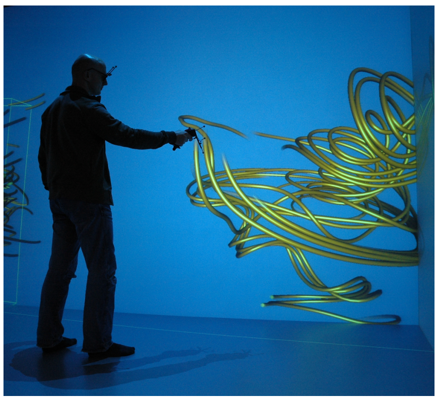

Figure 2: *A user interactively seeds particle traces in an immersive virtual environment.*

3. Interactive Seed Point Specification

Seed points are specified directly in the virtual environment by means of a 6-DOF input device. The user just points at the desired location and starts seeding particle traces by pushing the device's button. Similar to dye injectors from classical, experimental flow visualization particle seeding continues until the button is released. In contrast to classical flow visualization, the user is able to move the bubbler around freely, thus effectively "painting" particle tracers into the flow domain (see figure 2). Alternatively, automatic seeders can be defined, which continuously release particles into the fluid flow.

4. GPU-based Particle Advection

In order to perform interactive particle advection on a GPU, a number of subtasks have to be accomplished. These include storing flow field and particle data in graphics memory, integrating the particles through the flow domain using the programmable graphics pipeline, and inserting new, user-defined particles into the particle population.

4.1. Data Storage

For now, we concentrate on the exploration of flow fields defined on a cartesian grid and stored in a 3D floating point texture, resulting in a direct mapping of texels to grid points. Texture data consists of 4-tuples encoding flow velocity along the three major axes and an additional scalar. Instead of deriving the scalar attribute from the flow field on the fly, we pre-calculate it and download it to the graphics system along with the flow data itself. On the graphics system, texture data is stored in 4-component 3D textures with 16 bits per channel. This allows for making use of the interpolation capabilities of current graphics hardware.

Particle data is stored in 2D textures with a focus on preserving the particles' histories. A 4-component texture with 32bit floating point values is used for storing particle positions and the corresponding scalar attributes. In its simplest form, every column contains a single particle path and every row contains information about the whole particle population for the corresponding instant in time (see figure 3).

4.2. Integration

The movement of a particle is computed incrementally through numerical integration with the fourth-order Runge-Kutta method (RK_4), which is independently executed for every particle. As GPUs exhibit maximum performance for uniform computations on a wide data stream, we rely on a fixed step size.

Advancing the particle traces is a two step process. First, the current particle positions are integrated through the flow field using RK_4, then the particle histories are preserved by copying lines 0 to $m-2$ of the source texture to lines 1 to $m-1$ of the destination texture. For the next integration step, source and destination textures are swapped.

4.3. Seeding

For every new particle to be injected into the flow domain, its complete history has to be reset, i.e. a whole column in the particle position texture has to be written to. The particle positions in the column are set to the seed point for the new particle by drawing a line primitive into the off-screen render buffer associated with the particle texture. For seeding multiple new particles along a line, adjacent columns are written to by rendering a single quad instead of multiple lines, which results in a linear interpolation of the seeding positions for the start and end points of the line.

5. High-Quality Rendering

The final rendering as billboarded geometry requires additional data to be generated. For the body of a single tubelet, two vertices have to be sent to the graphics pipeline for every particle position. In addition, two vertices have to be added to the start and the end of the tubelet body for a believable simulation of rounded ends. Furthermore, for every tubelet four vertices have to be provided for its start and end caps each (see figure 4).

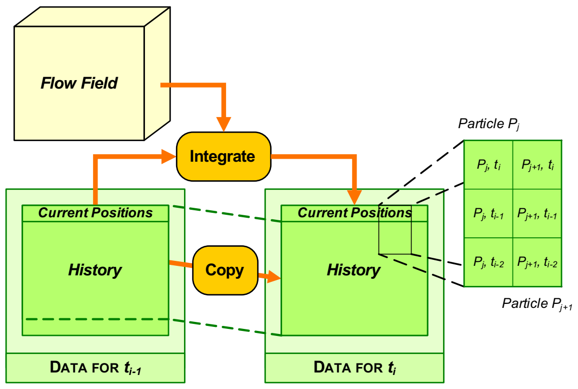

Figure 3: *Updating particle information stored in a 2D texture.*

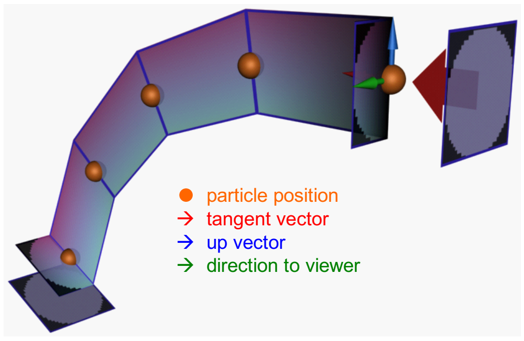

Figure 4: *Anatomy of a Virtual Tubelet.*

While current graphics hardware is capable of accessing texture data in the vertex shader, its efficiency is rather limited in comparison to the direct rendering of vertex arrays, especially when dealing with very uniform access patterns. Thus, we expand the data into additional render targets before copying it into vertex arrays. The latter step is necessary, as direct render-to-vertex-array functionality is only supported through vendor-specific OpenGL extensions.

For all tubelet vertices, the positions of the corresponding particles and the tubelet tangents are provided as vertex position and texture coordinate arrays. This information is augmented by offset information in order to distinguish between multiple vertices belonging to a single particle position. For the tubelet bodies, vertex positions are generated by "smearing" the particle positions over the corresponding entries in the render target. Tangent vectors are computed in a similar fashion, but instead of just copying particle information, the difference between two adjacent positions is computed. While the positional data for the tubelet caps is computed separately, directional and color information is copied from the tubelet data into the caps' respective vertex attribute arrays. As the structure of the tubelet geometry does not change, an index buffer is created and downloaded to the graphics system only once. It defines triangle strips forming the tubelets' bodies from the aforementioned vertex arrays and is reused for drawing every frame.

During rendering, the billboarded tubelet bodies are oriented to face the viewer and a normal map is applied, which emulates the rounded surface of a tube. Providing the fragment shader with user and light positions allows for computing per-pixel diffuse and specular illumination, which results in a convincing simulation of tubular geometry (see figure 1). This helps especially with reliably conceiving orientation and depth order of the particle trajectories. In addition, even when stopping the animation, the flow structure can easily be understood.

Desktop	Particle Count / History Length			
Data set	256/256	512/256	512/512	1024/1024
DVortex	212	120	66	18
Engine	240	120	64	17
Engine$_s$	212	106	56	15

CAVE	Particle Count / History Length			
Data set	256/256	512/256	512/512	1024/1024
DVortex	146	85	48	12
Engine	150	89	49	12
Engine$_s$	133	80	43	10

Table 1: *Timing results in frames-per-second on a desktop-based system and in a cluster-driven virtual environment.*

6. Results

On our test systems, we are able to provide a high-quality depiction of interactively seeded stream tracers by shifting the major workload onto the GPU. By displaying tracers instead of instantaneous particle positions, an intuitive understanding of the underlying flow field is considerably facilitated, which is enhanced even further by stereoscopic, user-centered projection. The interactive seeding of the particle traces allows for an explorative analysis of the CFD simulation results.

Table 1 shows the results of performance measurements on two reference systems, a desktop system and a visualization cluster driving a cuboid five-sided CAVE-like virtual environment. All systems are equipped with NVIDIA GeForce 6 graphics cards. Test data consists of a data set with two interacting vortices, resampled into a cartesian grid of size 256×128^2 and denoted as *DVortex*, and a data set with a time step of an interior combustion engine simulation, resampled to 128^3. The latter includes the velocity magnitude as scalar data. Measurements were taken with and without sampling and visualizing the scalar data, denoted as *Engine$_s$* and *Engine*, respectively. The results of the performance measurements are given in frames-per-second, as this is the crucial factor, which determines usability for an interactive exploration in an immersive environment.

As shown by the measurements, interactive frame rates are maintained for quite large particle populations with a long history. Even 1024 tracers with a history of 1024 steps are handled quite well. However, for this extreme case the frame rates are slightly too low for a comfortable exploration within a virtual environment. Besides larger output resolutions and overhead for communication and synchronization, the main reason for reduced frame rates on the cluster nodes are slightly less powerful graphics cards, i.e. GeForce 6800 GT compared to a GeForce 6800 Ultra in the desktop system. Separate measurements indicate that a rather large part of the computational load is taken up by tubelet reconstruction and rendering.

7. Conclusions and Future Work

In this paper, we presented a GPU-based approach for an interactive specification and high-quality visualization of particle traces. Bringing it into an immersive virtual environment with 6-DOF user interfaces allows for an intuitive exploration of static flow fields. An improved impression of the structure of the underlying flow field is conveyed especially by the display of the particles' histories. In combination with a high-quality depiction as tubular geometry comprehension problems and visual ambiguities are minimized.

An issue, that came up during the use of our method for the exploration of non-enclosed flows, was that many particles left the flow domain quite quickly, leading to quite a volatile visualization. The use of automated, user-placed seeders remedies this problem. However, some kind of annotation system for marking promising seeding positions or distinctive flow features would be useful.

Acknowledgements

The authors would like to thank the Institute of Aerodynamics (AIA), RWTH Aachen University, for the engine and dual vortex data sets kindly made available.

References

[BL91] BRYSON S., LEVIT C.: The virtual windtunnel: an environment for the exploration of three-dimensional unsteady flows. In *VIS '91: Proceedings of the 2nd conference on Visualization '91* (1991), IEEE Computer Society Press, pp. 17–24.

[GHW*04] GERNDT A., HENTSCHEL B., WOLTER M., KUHLEN T., BISCHOF C.: VIRACOCHA: An Efficient Parallelization Framework for Large-Scale CFD Post-Processing in Virtual Environments. In *Proceedings of the IEEE SuperComputing (SC2004)* (November 2004).

[KBHJ01] KUESTER F., BRUCKSCHEN R., HAMANN B., JOY K. I.: Visualization of particle traces in virtual environments. In *VRST '01: Proceedings of the ACM symposium on Virtual reality software and technology* (New York, NY, USA, 2001), ACM Press, pp. 151–157.

[KKKW05] KRÜGER J., KIPFER P., KONDRATIEVA P., WESTERMANN R.: A Particle System for Interactive Visualization of 3D Flows. *IEEE Transactions on Visualization and Computer Graphics 11*, 6 (2005).

[OLG*05] OWENS J. D., LUEBKE D., GOVINDARAJU N., HARRIS M., KRÜGER J., LEFOHN A. E., PURCELL T. J.: A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports* (August 2005), pp. 21–51.

[SKH*05] SCHIRSKI M., KUHLEN T., HOPP M., ADOMEIT P., PISCHINGER S., BISCHOF C.: Virtual Tubelets – efficiently visualizing large amounts of particle trajectories. *Computers & Graphics 29*, 1 (2005), 17–27.