Visualizing Vortices in Simulated Air Flow around Bat Wings during Flight

Eduardo Hueso

Department of Computer Science
Brown University
Providence, Rhode Island 02912

CS-03-25
December 2003

Visualizing Vortices in Simulated Air Flow around Bat Wings during Flight

by

Eduardo Hueso

B.S., Universidad Simon Bolivar
Caracas, Venezuela, 2000

Project

Submitted in partial fulfillment of the requirements for the Degree of Master of Science in the Department of Computer Science at Brown University

December 2003

Visualizing Vortices in Simulated Air Flow around Bat Wings during Flight

Eduardo Hueso
Master's Thesis

Department of Computer Science
Brown University
Providence, RI, 02912

Abstract

We present a case study of our efforts towards building a set of data visualization tools to aid the understanding of airflow structures involved in the flight of bats. We use two visualization schemes that involve off-line computation of "interesting" pathlines and streamlines respectively, and an immersive virtual reality facility, CAVE, for their visualization.

Hussain's et at vortex region detection technique, λ_2 [1] is employed in our line sampling methods to emphasize vortices in the flow.

Interactive control over emphasis parameters allows users to explore a continuum between localized and contextual representations of vortices.

Our results, composed by two visualizations applied to a common dataset, lead to a subjective comparison between the effectiveness of pathlines and streamlines in the representation of vortices.

Keywords: scientific visualization, simulation, bat flight, evolutionary biology, CFD, pathlines, streamlines

1. Introduction

Dr. Sharon Swartz and her group in the Evolutionary Biology Department at Brown University have been studying, for several years now, the mechanics involved in the flight of bats. One of their goals is to understand and model the flight of the only flying mammal. Their approach involves collecting mechanical data of bat wings such as bone compositions, membrane mechanical properties, strain and stress forces and aerodynamics involved in their flight.

This paper presents a case study of our interdisciplinary collaboration efforts towards building a set of tools for visualizing simulated airflow and its application to the understanding of the aerodynamics of bat flight.

Sharon and her team captured video of over 40 individuals of 8 different species flying in a wind tunnel of the Department of Civil Engineering at The University of Queensland, Brisbane, Australia. These videos were used to extract the motion data of wing joints. A motion capture session was selected; in this case a Pteropus Policephalus was chosen for its large size and slower motion. We, the computer science collaborators, constructed an animated 3D model of the bat's wings. This model is handed to Prof. George Karniadakis's group in the applied math department who fed it into their computational fluid dynamics (CFD) simulator, Nektar. The output of the simulation, scalar and vector fields describing the airflow around the bat wings returned to us.

In our efforts to visualize significant structures, particularly vortices present in the simulated fields, we developed a visualization system based on pre-computed flow lines among which pathlines and streamlines are discussed in this paper.

Interactive control over emphasis parameters during visualization allows users to explore the continuum between a contextual representation of the flow where the whole volume is shown with an unbiased distribution of flow-lines and a localized representation where more flow-lines are displayed in vortex regions.

An immersive virtual reality facility, CAVE [2], composed of 4 back projected walls in a cubic layout, with head tracking and stereo display is used as the primary interface for our visualizations.

2. Motivation

We believe that bats employ unique, extremely energy efficient aerodynamic mechanisms, unlike those of any other animals or any human-engineered craft. Exploring these fluid dynamic phenomena presents enormous technical challenges. But, as we solve these problems, we not only gain insights into the evolution of the only flying mammals, we may discover novel ways to think about aircraft design.

Vortices, swirling motion of flow around a core, have been known and studied by fluid dynamics specialists as a means to understand important characteristics of flow. Specifically, vortices are a manifestation of energy and can be related to the generation of lift and drag. Additionally, understanding the vortical structures that exist around bat wings during flight could help categorize and compare different flying strategies of bats under different conditions and among different species.

The accurate and systematic detection of vortices in complex flow remains a hard problem. Visualization techniques allow leveraging the complexity of human's vision system to find interesting and even unexpected flow features like vortices, as well as discovering the relation between them.

3. Related Work

Vortex region and core detection

Hussain et al [1]. Introduces λ_2, a flow descriptor that takes negative values inside vortex regions. We used this measure to selectively emphasize flow-lines that pass through vortices.

Ming Jiang [3] et al. presents a geometric method for verifying the existence of vortices by seeding streamlines in the vortex core and analyzing their swirl. Our streamlines method for showing vortices is inspired in this idea.

Visualization of time varying flow data

David Lane [4][5], presents NASA's approach to their unsteady flow analysis toolkit (UFAT), he describes the pre-computation of pathlines, streamlines and streaklines and recommends streaklines for the better understanding of unsteady flow. He also describes the parallelization of a particle tracer for flow visualization of large datasets [6]. Inspired in his results we chose to represent flow data with pre-computed flow-lines.

Jason Sobel et at. [7] Presents a method for synoptic visualization of pulsatile flow based of the pre-computation of pathlines in a Poisson disk time/space distribution, he applied his technique to the visualization of blood flow in a coronary artery.

Rachel Weinstein [8] presents a case study of unsteady airflow visualization around still bat wings during flight. She modeled a still bat and visualized simulated airflow around it, Rachel's work was the first prove of concept in simulation and visualization of air around bat wings and exposed the main difficulties of simulating time varying flow on deforming geometry.

4. Methods

Simulation Data

The data acquisition process that yields the data used in our visualizations involved several steps that are worth mentioning. First, a pair of synchronized, high frequency videos of life bats flying in a wind tunnel with reflective markers stuck on to their joints is taken. The Peak [9] motion capture system is used to extract 3D motion data from the videos. A full wing beat of data is extracted and made perfectly cyclic. In order to make the motion cyclic, the difference between the last and first frames of the motion is divided by the number of frames in the cycle and added accumulatively to every frame of the cycle. Some considerations were taken to make the seam second order continuous. After some cleanup and smoothing, the motion data is used to animate a simple 3D polygonal model of the surface of the bat's wings. This model is constructed by connecting the motion markers with triangles. The animated polygonal model is the base for generating a sequence of tetrahedral meshes of the volume around the wings. A single tetrahedral mesh can be deformed to fit a number of frames, typically between 2 and 10 depending of the rate of deformation. When the deformation of the mesh becomes too extreme, elements degenerate and a new tetrahedral mesh needs to be created. Mesh generator Gridgen [10] with scripting capabilities was used to generate up to 60 tetrahedral meshes for one wing beat.

These meshes are the input to Nektar [11], Brown's proprietary numerical simulator that solves Navier Stokes equations to obtain unsteady vector and scalar fields that describe the motion of air around the animated bat wings. The simulation we used in our experiments produced 40 time steps of periodic data that describe a whole wing beat cycle.

Velocity and pressure fields are obtained from the simulation. Additional data fields, which we call derived fields, can be computed from them and stored in Nektar's mesh structure to take advantage of its spatial interpolation by polynomial spectral elements. Some of the derived fields we compute and use for visualization are vorticity, shear stress, and λ_2 [1].

Datalines

The size and complexity of the raw simulation data makes it impractical for real-time inspection. Our approach implements an offline pre-visualization step where interesting structures get computed from the flow data.

One of the most common and basic ways of visually representing fluid flow is through curves. We developed a general abstraction of a line as a sequence of points in space or space-time where each point can hold a number of flow measurements (e.g., velocity, vorticity, pressure, lamnda2) of the flow at its location. Sets of datalines can be computed and stored in files and then loaded by the visualization program for real-time display and interaction.

By varying the time relationship between points on the lines and the way they are integrated over the different vector fields we were able to compute some of the most commonly used flow lines.

For line integration we used the Adams method with an adaptable *dt* and a fairly low tolerance for high accuracy. However, for storage purposes, a significantly larger *dt* was used to sample more coarsely the finely integrated lines.

The computation of datalines is spatially limited to a user-defined box around the bat. Lines are culled outside of this box, i.e. no seeds are considered outside of the box and the integration of a line stops when it falls out of the box.

For practical reasons, the integration of a line is also stopped when a user-defined maximum number of sample points is reached or when the magnitudes of the vector field drops bellow a given threshold.

Computing a pathline

A pathline describes the path of a weightless particle in the flow.

Given a 4D seed point in space-time (*x,y,z,t*) a line is integrated backward and forward in time over the unsteady velocity field. At every step of the integration the updated velocity field is used. The forward and backward lines are sampled at a constant *dt* and concatenated to make a single one. The time of the last point computed on the backward line is stored as the start time of the line.

Computing a streamline

A streamline is a line tangent to the velocity vector field at a fixed instant in time.

Given a seed point in space-time, a streamline is computed by integrating forwards over the instantaneous velocity vector field at the seed's time.

Because time remains still we will refer to *s* as the parameter along the curve used for integration.

In our algorithm the *ds* used for sampling streamlines adapts to make the spatial steps, *dx*, be close to a user defined feature size. This way the spatial sample frequency of streamlines is not sensitive to magnitude variations on the vector field.

Visualization based on particle paths

At a high-level, we pre-compute a set of pathlines that will be used to represent the flow during visualization. We find this set by choosing a finite set of 4D seeds, i.e. points in space-time. For some of these seeds, we'll discuss which later; pathlines are computed using the method described in section 4.2. At run-time, we draw particles at the start of pathlines (not the seed position) and animate them until they reach the end of their pathlines.

A key issue is determining *which* particle paths should be pre-computed. Ideally we would want to find a set such that all flow features are represented by particles flowing downstream.

The goal that all features be represented requires that some particle pass through each feature that *could* exist. Due to the discretization of the flow required by computer simulation, a distance, *D*, exists below which no further significant features can exist. In simulations computed using Nektar [11], *D* is a function of the polynomial order of Nektar's spectral elements and the local density of mesh elements. In the case of our bat flight flow dataset the density of elements is higher near the bat wings and tapers down towards the edges of the volume. Our ideal goal of capturing all flow features will be met if a set of pathlines can be computed such that every point *p* in space is no further than *D (p)* from a point on some pathline. This condition can always be met by adding additional pathlines to fill gaps, often at the cost of redundancy in particle path coverage in other regions.

In time varying flow we need to consider the time extension of the features we'd like to capture, i.e. for every point in space-time we want to have at least one particle closer than *D* in space at a time closer than *T*. The notion of the distance *D* can be extended to include time as a fourth dimension.

For simplicity we assume *D* to be constant within the whole simulation volume and period.

In order to generate pathline sets of manageable size for visualization, i.e. pathline sets that fit in main memory of the visualization system, a limit needs to be imposed on the number of pathlines in the set. This means that either the resolution or the coverage of pathlines might need to be compromised. Not all regions in the flow contain the same amount and size of features, giving priority to certain "interesting" regions allows for a more efficient use of the pathline budget within the volume.

In our system, a list of potential seed points is collected and a subset containing the ones in "interesting" regions is used to compute pathlines. We identify vortical regions in the flow by measuring λ_2 [1], a flow descriptor that takes negative values inside vortex regions.

Generating seeds for pathlines

A descriptor of the flow, *s*, is sampled at fixed intervals of *x, y, z* and time at a relatively high resolution. A list with the sampled 5D points (*x, y, z, t, s*) is held in memory and sorted by *s*.

Computing particle paths from a list of seeds

A 4D grid is used to segment space-time. The resolution for each dimension of the grid is specified independently by the user and responds to the desired value of *D*. Grid cells can be marked as visited. Initially all cells are unmarked. When a pathline is computed all the grid cells it passes through get marked as visited.

The list of sorted seeds is traversed and for each seed the grid cell it falls in is determined. If the grid cell is not already marked as visited a pathline is computed from that seed, otherwise, the seed is skipped. The algorithm finishes when all grid cells are marked as visited, when the seeds are all consumed or when the maximum number of pathlines allowed is reached.

Data interface

Note that our algorithm for computing pathline sets uses the previously computed pathlines to decide if a seed is used or skipped. Because pathlines typically traverse the whole simulation period, generating them in sequential order requires random access in the temporal dimension of the data, i.e. all time-steps of data need to be available. Our data interface handles this even in cases where the data is larger than the 2GB addressing capability of a 32-bit process. The way this is done is by loading each time step in a separate process. These processes can potentially be in different machines. A main process receives the data requests and uses a socket to communicate with the appropriate child process to obtain the data. Even when distributed through different machines, this scheme for data access is faster than having to load and unload full time-steps of data from disk repeated times.

Displaying Particle Paths

Particles are released at the upstream side of the simulated volume and advected along the pre-computed pathlines.

Particles are released at a self-adaptable rate that keeps the total number of particles in the flow constant. Particles die when they get to the end of their associated pathlines and that triggers the creation of new ones. A pathline longer than the simulation time period can carry more than one particle separated by the simulation period. When rendering a frame at time *t* during the visualization, the number of new particles that need to be released is computed as the max number of particles allowed in the flow minus the number of existent ones. For every new particle, a pathline with start time near the current time needs to be selected from the pathline set. By choosing pathlines with start time near the current time we are forcing new particles to start flowing at the upstream end of the flow.

Selection of pathlines

Pathlines in a set get sorted by their min, max or average value of some flow measure along them. For example, we sampled λ_2 at every point on every pathline and stored it with it. During visualization we find the point with min λ_2 on each line and sort the list of lines in increasing order according to this min value. A randomness factor is used to determine how many of the pathlines used to drive particles during visualization are taken from the head of the list and how many are taken at random from anywhere in the list.

A randomness of zero means that particles are released on pathlines in the order in which they appear in the sorted list. This technique allows prioritizing the use of pathlines according to flow properties. A randomized selection of pathlines gives a more synoptic view of the pathline set.

Particle eels

In our system particles can be rendered in two different ways, which we call eels and snow.

Eels are displayed as motion blurred semitransparent lines. The GL_LINE primitive is used to draw them. The center of an eel, which represents the position of the particle at the current time, is drawn with maximum opacity. The eel fades to transparent further from the center as it represents past or future times. The user can control the extent of the motion blur interactively to make longer or shorter eels.

Eel color and transparency can be mapped to flow measures previously sampled and stored with the pathline set.

Snow

The GL_POINT primitive is used to draw simple white dots that represent particles as snowflakes. The graphical simplicity of this representation allows drawing a lot more snowflakes than eels.

Color mapping

Both color and transparency can be mapped to a descriptor of the flow along the pathline.

Two extreme colors, min_color and max_color, are defined in a configuration file. Two data values, min_map_value and max_map_value are controlled interactively by the user. The color of a particle is computed as a linear interpolation between min_color and max_color that depends on the data value.

Transparency is mapped independently and can be mapped to a different flow descriptor. The mapping equation is the same as in color mapping. In this case, the values of min and max transparency are fixed to 0.0 and 1.0 respectively.

The user has interactive control over the size of the mapping window, max_map_val – min_map_val, and the center of the window, min_map_val + (max_map_val – min_map_val / 2.0)

Visualization based on streamline sequences

A streamline in unsteady flow exists for an instant in time only. When used in a time varying visualization, a coherent sequence of lines should be displayed such that the user gets the impression of a single line that is deforming with the flow. We'll refer to such a sequence as a *streamline sequence.*

To prevent lines from appearing and disappearing in an erratic way during visualization, for a given seed in space we compute a *streamline sequence* that covers the whole simulation period. In pro of temporal coherence the number of streamlines computed to produce the sequence is normally greater than the number of time-steps in the data. Linear interpolation is used to compute intermediate vector fields. In general, we try to display a minimum of 15 lines per second. Based on the assumption that the speed of the visualization is never set to less than a quarter of the original simulation speed we pre-compute up to 60 streamlines per second of simulation.

A 4D seed in space-time is used for each streamline in a *streamline sequence*, we'll refer to this set of seeds as a *seed sequence*.

Generating seeds for streamlines

A descriptor of the flow, *s*, is sampled at fixed intervals of *x, y, z* and time at a relatively high resolution. The best value in time is recorded for every point in space creating a 4D spatial seed of the form (*x,y,z,s*) where *s* is the best value found in time at point [*x,y,z*]. After covering the whole volume the seeds are sorted by *s*.

Computing streamlines from a list of seeds

Analogous to pathlines, each streamline marks volume cells in a 4D grid. A whole sequence of streamlines is computed if at least one cell in the time dimension of the spatial seed is not marked as visited.

Displaying Streamlines

Streamlines are represented with GL_Lines connecting the sample points. Their color and alpha can be mapped to any pre-sampled flow measure that was stored with the line. Line sequences are displayed throughout the whole simulation period.

Selection of streamline sequences

Sequences of streamlines are sorted by their min, max or average value of *s* at their seed. The same selection method used for pathlines is used to select the streamlines that will be displayed during visualization.

Visual properties of streamlines (kelp)

Streamlines are drawn using anti-aliased GLLines. Their color and transparency can be mapped to any flow data value.

5. Results and discussion

We used our system to generate two different visualizations of the same simulation, one based on pathlines and the other based of streamlines.

Simulation

The simulation data used came from solving Navier Stokes equations on uncompressible fluid using an unstructured tetrahedral mesh. The flow's main velocity is approximately the one used in the wind tunnel during live capture.

A single periodic wing beat was simulated at Reynolds number = 10 producing 40 time-steps of unsteady flow data.

Particles in vortex regions

The number of pre-computed pathlines is such that the size of the dataset is roughly 400 MB. This is the RAM capacity of our lower end visualization machines. Pathlines are sorted by λ_2.

Seed sampling resolution: (30 x 10 x 30 x 40)
Pathline grid resolution: (30 x 10 x 30 x 40)
Line sampling dt: 0.01875
Number of lines: (30 x 10 x 30 x 40)

A reasonable resolution for the temporal dimension of the grid is one that matches the time discretization of the data.

In order to create a pathline set that covers the whole volume, we made the max number of lines match the number of grid cells used by the algorithm. This guarantees that no grid cells are left unvisited.

Both color and alpha are mapped to λ_2. Lower λ_2 values are represented with red particles whereas higher λ_2 is represented with blue particles.

figure 1

figure 2

The visual simplicity of snowflakes makes the time varying visualization particularly easy to read and pleasant to watch. Small points are fast to render and allow a higher density of them at smooth frame rates giving a better coverage of the volume.

Particle paths seeded at low λ_2 regions are clustered around vortices in space and time. These recognizable clusters flow downstream and become the signatures of vortices in the wake. This wake structure prevails even after the vortices die. This effect is noticeable when displaying only those particles that pass through low λ_2 regions.

Random selection of pathlines, on the other hand, can be used to avoid patterns in the distribution of particles introduced by periodic selection of pathlines.

Both visual cluttering and performance of the graphics subsystem limit the number of particles that can be displayed simultaneously. Using random selection of pathlines allows a user to trade space and frame rate for time. By displaying fewer particles but waiting longer the user eventually gets to see all existent paths in an uncluttered and smooth visualization.

By adjusting the randomness and transparency interactively and changing the sort criterion for pathlines the user can shift the focus of the visualization and find the right balance between emphatic and synoptic representation of the flow.

Streamlines in vortex regions

We used a localized seeding of streamlines in low λ_2 regions to show vortices.

Streamlines are integrated downstream only assuming that their seeds are at the beginning of interesting flow structures that extend downstream. The main goal in this approach is to capture swirling motion around a vortex core.

In visualization, both color and transparency get mapped to λ_2 values. Lines are drawn yellow and opaque in low λ_2 regions an turn red and transparent in higher λ_2 regions. The seed points of streamlines are represented with white dots.

figure 3

Discussion

Particles, when seeded at low λ_2 regions, work well at capturing the signature of vortices in the wake. Streamlines, on the other hand, show the instantaneous structure of the vortices but have no information about their history. In this sense, the two visualizations complement each other.

Our visualizations combine velocity information, intrinsic in pathlines and streamlines, with λ_2 to emphasize vortices. We believe that, by combining these two data attributes in an intuitive way, our visualizations may be able to tell more about vortices than iso-surfaces of λ_2. Additionally, being velocity the quantity scientists are more comfortable with, it provides a link to understanding λ_2, which is a less intuitive measure of flow.

By using transparency and randomness to smoothly fade out contextual information of the interesting features we produce soft representations of the features that can also be related smoothly to their spatial and temporal context. These characteristics of our visualizations together with interactive control over the emphasis parameters, and immersive VR displays, provide an appropriate and comfortable environment for free exploration of the data. We believe that our fuzzy representations of flow data present certain advantages over sharper feature representations like isosurfaces and isocurves. Particularly, they represent more contextual information and don't force the user to limit their understanding to isolated features.

It's important to keep in mind that the simulation used to produce the data shown in this paper is of incompressible flow at Reynolds number of 10 (Re = 10). We hypothesize that many of the effective properties of our visualizations will extend to new datasets at higher Re, however, we do expect some difficulties in representing fast moving vortices with sequences of streamlines.

6. Conclusions

We used a volume filling technique that guaranties distribution of flow lines that covers the whole simulated volume at a given resolution with low redundancy.

By randomly selecting sets of flow lines over time we eventually show all possible features in the flow without clutter or distracting patterns.

Flow descriptors like λ_2 can be used to emphasize interesting regions. By sorting flow lines and providing control over the amount of randomness used in their selection during visualization, we give the user continuous control over the degree of emphasis.

Interactive control over emphasis parameters, like transparency and randomness allow the user to balance the ratio of contextual to localized information. Using λ_2 in combination with velocity, implicit in streamlines and pathlines, conveys more information than iso-surfaces of λ_2 on their own.

We believe that immersiveness, stereoscopic vision, peripheral vision, body centered navigation and large-scale displays are attributes of our immersive virtual reality facility, CAVE, which help significantly in the exploration of time varying flow data.

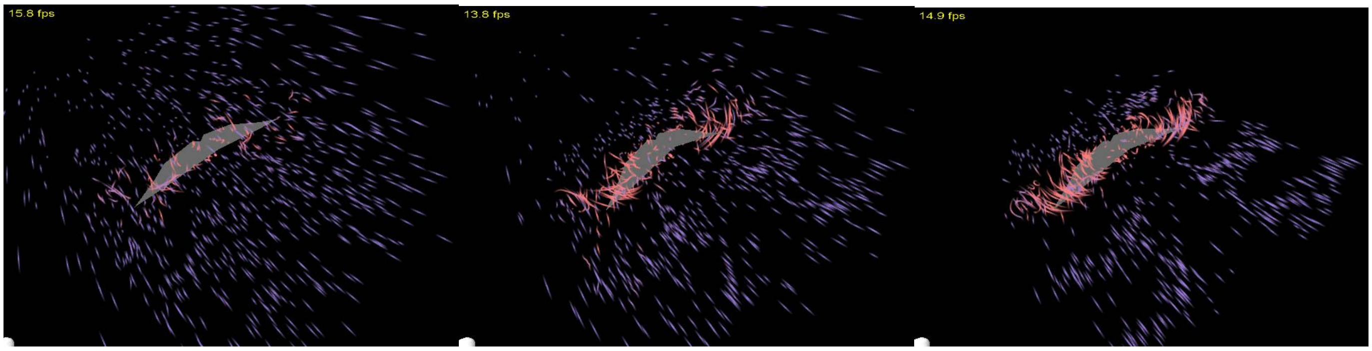

Figure 1: Particle eels are used to display pathlines. a, b and c show variations of randomness in the selection of pathlines, 1.0, 0.15 and 0.0 respectively.

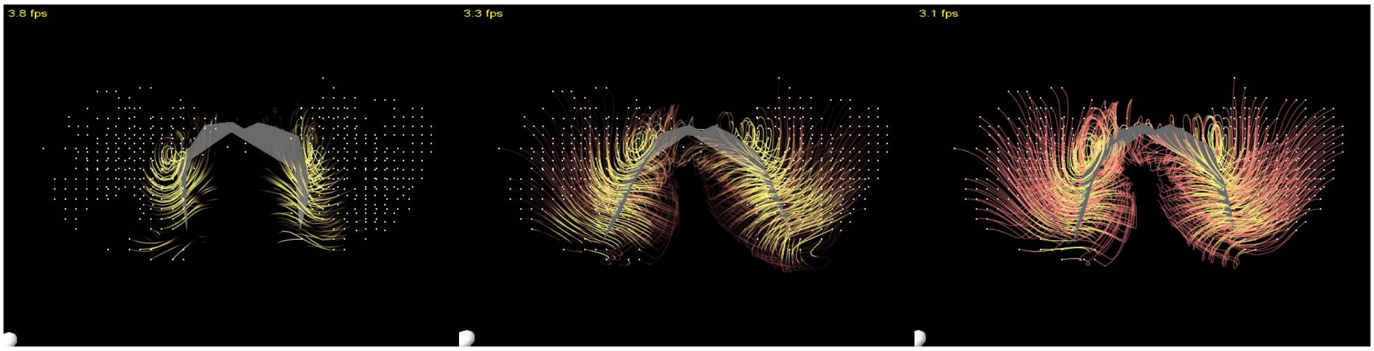

Figure 2: Vortices are shown by streamlines seeded at low λ_2 regions. a, b, c show variations of the transparency mapping window.

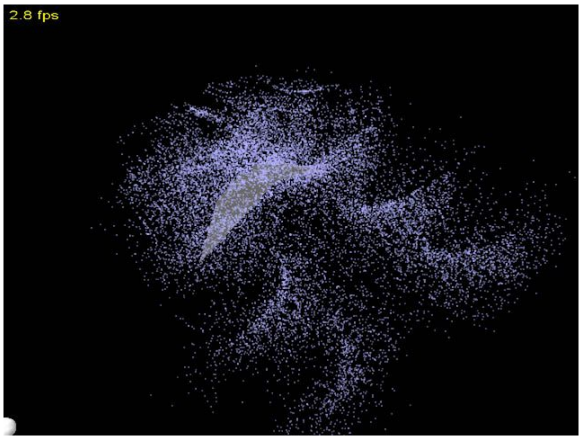

Figure 3: White dots, “snow flakes”, are used to represent particles at low λ_2, regions. Zero randomness emphasizes pathlines that pass through vortices.

References

[1] J. Jeong and F. Hussain. "On the Identification of a Vortex", J. Fluid Mechanics, Vol. 285, pp. 69--94, 1995.

[2] Cruz-Neira C., D. J. Sandin and T. A. DeFanti, "Surround Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE", Computer Graphics, SIGGRAPH Annual Conference Proceedings, 1993.

[3] M. Jiang, R. Machiraju, and D. Thompson, "Geometric Verification of Swirling Features in Flow Fields," Proceedings of IEEE Visualization 2002, Boston, MA, October 2002

[4] M. Nielson, Hans Hagan, and Heinrich Muller, editors, Los Alamitos, CA, 1997, **Scientific Visualization** - **Overviews**, **Methodologies** and **Techniques**, Chapter 5, Scientific Visualization of Large-Scale Unsteady Fluid Flows by David A. Lane, pages 111-133.

[5] D. Lane. Scientific visualization of large scale unsteady fluid flow. In Scientific Visualization Surveys, Methodologies and Techniques, pages 125--145. IEEE Computer Society Press, 1996.

[6] David A. Lane. Parallelizing a particle tracer for flow visualization. Technical Report RND-94-011, Numerical Aerospace Simulation Facility (NAS), NASA Ames Research Center, Moffet Field, CA, 1994.

[7] Jason Sobel, Andrew Forsberg, David H. Laidlaw, Robert Zeleznik, Daniel Keefe, Igor Pivkin, George Karniadakis, and Peter Richardson. Particle flurries: a case study of synoptic 3d pulsatile flow visualization. unpublished (pdf) (bibtex: Sobel-2002-PFC), November 2002

[8] Rachel Weinstein, Igor Pivkin, Sharon Swartz, David H. Laidlaw, George Karniadakis, and K. Breuer. Simulation and visualization of air flow around bat wings during flight. unpublished (pdf) (bibtex: Weinstein-2002-SAV), August 2002.

[9] Peak Performance, http://www.peakperform.com/

[10] Gridgen, http://www.pointwise.com/gridgen/

[11] Nektar, http://www.cfm.brown.edu/crunch/projects.html

This project by Eduardo Hueso

is accepted in its present form by the Department of

Computer Science as satisfying the

requirement for the degree of Master of Science

Date________________

____
Prof. David Laidlaw