Visualization of the New Orleans Windfield Dataset

Randall E. Hand*

U.S. Army Corps of Engineers, Engineering Research & Development Center

ABSTRACT

The theme of the IEEE Visualization 2005 Contest is "Rendering Revolution", and as such a revolutionary approach is required. Development of an application to effectively visualize the New Orleans Windfield dataset using all open-source tools that is capable of enabling interactive manipulation of the data on a typical medium-grade personal computer requires implementation of both new data management algorithms and and new data visualization and computation techniques.

Developing a system that was capable of competing with the supercomputers and visualization clusters available today was both challenging and rewarding, resulting in the application submitted.

1 INTRODUCTION

The theme of the IEEE Visualization 2005 Contest is "Rendering Revolution". As today's datasets grow larger and larger, visualizations requiring supercomputers or large clusters are becoming more commonplace. This takes visualization out of the hands of the common user and places it in the realm of high-dollar corporations or wealthy researchers.

By implementing better techniques for data management and smarter algorithms for rendering, visualization of large-data can be put back in the hands of the common man or scientist.

2 ALGORITHMS

2.1 Data Management

The data for the contest was initially provided as a pair of ASCII FieldFlow files of approximately 682Megabytes in size. The first step of the project was to convert this into something more manageable and more feature-filled.

The obvious first step was to convert the data into a binary format. This would reduce the 8 or 10 byte numbers into 4 byte floats, resulting in approximately a 50% size savings, and and several orders of magnitude of time savings during loading. This, however did not fully address the problems of the organization of the data.

Initially, the grid was specified separate from the actual data. Also, the grid was specified both as a collection of triangulated surfaces and volume tetrahedra. The simplest portion of the data to extract and visualize was the surfaces, so a routine was written to extract each of the 4 surfaces (floor, city, walls, & ceiling) into 4 separate files of a custom format. This custom format would contain only the vertices necessary for the surface, along with their data, and the triangles necessary to construct the surface.

The volume tetrahedra presented a further problem. With just over 12 million tetrahedra specified in the data, a smarter algorithm was needed to better facilitate sorting and searching in the data. A simply binary space partition could sort the data into significantly smaller and more manageable portions, while a bounding box surrounding each portion could be used to speed up searches.

An algorithm was developed that first computed a bounding box of the data. This bounding box was then split along it's longest axis into 2 equal-sized boxes. The number of tetrahedra included in each box was then computed, and if the number exceeded a specified threshold the box would be split again along it's longest axis. This recursive algorithm could easily subdivide the dataset into "blocks" of a user-specified size. The one drawback would be the duplication of tetrahedra along the block boundaries, but the effects of this were negligible compared to the gains.

2.2 Data Visualization

2.2.1 Surface Visualization

The first and simplest part of the visualization was rendering of the context bounding surfaces of the simulation. By rendering these colormapped to a user specified variable, both context and data could be shown simultaneously.

One problem most visualizations suffer from is improper interpolation in colors. By improperly interpolating colors between adjacent grid points, visualizations can be misleading to the researcher and lead to incorrect conclusions. An example is shown below [Figure 1].

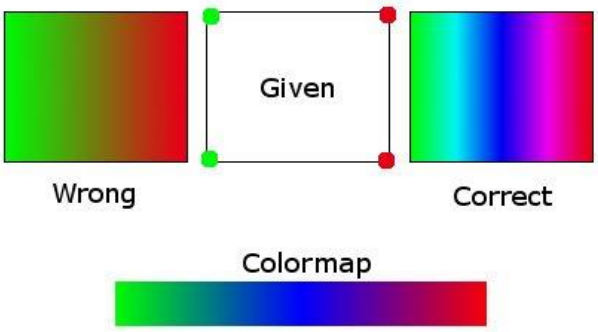


Figure 1: An example of Correct vs Incorrect color interpolation

The box on the left shows the interpolation only between the colors Green and Red, as shown in the center box. A proper interpolation would recognize that there must exist somewhere between the low and high values, the center values that comprise the blue area of the colorbar, and insert them accordingly. In OpenGL this can be accomplished by not using per-vertex color information, but per-vertex texture information and a 1D texture containing the colormap. OpenGL will correctly interpolate between two points in a 1D texture, containing all the values in

* Email: randall.e.hand@erdc.usace.army.mil

between, thereby containing the blue area between the green and red.

2.2.2 Vector Visualization

The most effective form of vector visualization is pathlines. As our data is time-invariant and steady-state, pathlines would show the propagation of an agent through the data from any user-specified point. A 2nd order Runge Kutta algorithm was implemented for an accurate interpolation, but localizing a point within a tetrahedra was an intensive task given the 20million tetrahedra.

The binary space partition implemented in the data converter proved useful here, as the problem was suddenly reduced from a brute-force search of all 20million tetrahedra to a search of a user-defined number of tetrahedra after finding which block contained the desired point. Localizing the point within a block was trivial, as the blocks were all axis aligned. This change reduced the time required to compute a streamline from hours to minutes.

Furthermore, the spatial coherence of the algorithm could also be exploited. Once the block containing the point was found, the same block would probably also be used for future points, so future searches would start with the previous search's result. The same could also be done for the tetrahedra, as the Runge Kutta operation required multiple query's of the data at nearby points, often within the same tetrahedra. This enabled computation of streamlines in seconds.

2.2.3 Volume Visualization

The only remaining information to be displayed is the scalar data within the tetrahedra. This can easily be displayed as an isosurface computed using a marching tetrahedra algorithm. The user is allowed to select both the variable and value of the isosurface, then it is rendered amongst the contextual and pathline data.

An isosurface of a low value of Momentum Magnitude could be used to see what portions of the city are shielded from high winds by buildings or natural phenomenon.

3 Results

The resulting applications were run on a personal computer with Windows XP running on Athlon 3Ghz processor, 1 Gig of RAM, and an Nvidia 5900 graphics card.

3.1 Data Conversion

The data conversion was run first. Initial trials attempted to break the volume tetrahedra into blocks of 5,000 tetrahedra, but were terminated after several hours of runtime. After several trials, 100,000 tetrahedra per block was found to be an acceptable number, completing in approximately 30 minutes. This created the following 5 files:

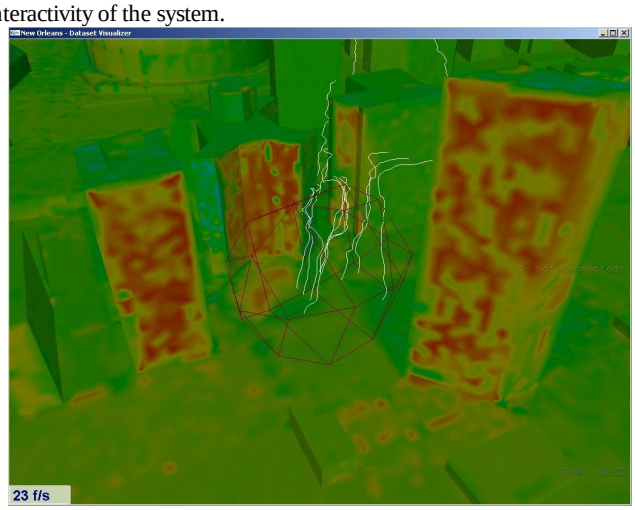
Surface-1.reh	100KB
Surface-2.reh	1,202KB
Surface-3.reh	21,080KB
Surface-4.reh	19,496KB
Volume.reh	368,721KB

The four surface files contain the 4 bounding surfaces of the simulation, the ceiling, walls, city, and floor. The volume file contains the entire tetrahedral volume, broken up into 268 blocks of no more than 100,000 tetrahedra each.

3.2 Data Visualization

The results from the Data Conversion step were then copied into the visualization directory. The visualization program loaded them into memory in approximately 10 seconds, and started with the four bounding surfaces rendered at 23 frames per second.

The first process attempted was to generate 10 pathlines in the data. The 10 pathlines were computed in approximately 15 seconds, and had no visible impact on the framerate or interactivity of the system.



Finally, an isosurface was generated at a low value of momentum magnitude to find the areas of the city shielded from the high winds. An isosurface at 0.060324 took approximately 5 seconds to generate, and lowered the framerate to 7 frames per second when rendered, but was still interactive.

4 Conclusion

The resulting visualization is both interactive and smooth, allowing for interactive exploration of the data through multiple effective means on virtually any modern computer.