Optimizing Particle Tracing in Unsteady Vector Fields

Al Globus, Computer Sciences Corporation[1]
globus@nas.nasa.gov

Abstract

Unsteady 3-D computational fluid dynamics (CFD) results can be very large. Some recent solutions exceed 100 gigabytes. Visualization techniques that access the entire data set will, therefor, be excruciatingly slow. We show that particle tracing in vector fields calculated from disk resident solutions can be accomplished in O(number-of-particles) time, *i.e.*, visualization time is independent of solution size. This is accomplished using memory mapped files to avoid unnecessary disk IO, and lazy evaluation of calculated vector fields to avoid unnecessary CPU operations. A C++ class hierarchy implements lazy evaluation such that visualization algorithms are unaware that the vector field is not stored in memory. A numerical experiment conducted on two solutions differing in size by a factor of 100 showed that particle tracing times varied by only 10-30%. Other visualization techniques that do not access the entire solution should also benefit from memory mapping and lazy evaluation.

Introduction

CFD scientists use flow solver software to generate vector and scalar fields describing fluid flow. These fields are called the solution. The solution is computed on discretely sampled spatial grids with sufficient sample points, called nodes, to capture the complex fluid physics. Capturing ever more detailed physics requires more and more nodes, increasing storage requirements. None-the-less, a number of current visualization systems can readily visualize static 3-D numeric solutions [Bancroft90, Buning85, Legensky90, SGI92, Upson89, Wavefront91]. However, when the time dimension is added the resulting explosion in data set size severely degrades the interactive response of existing systems.

Unsteady solutions consist of a series of steady state solutions called time steps. Adding the time dimension dramatically increases solution size. For example, recent unsteady solutions at NAS[2] are 10-100 gigabytes [Globus92]. Obviously, the largest unsteady solutions are much larger than the RAM available on workstations or even supercomputers. This will be true for the foreseeable future since solvers need only hold one time step in memory at any one time, while interactive visualization software must navigate rapidly through tens, hundreds, or even thousands of time steps under user control. Thus, time dependent visualization requirements will continue to vastly exceed available memory for the foreseeable future.

Visualizing very large solutions is difficult because the time to read data from disk into memory is O(solution-size). Moreover, in many cases the vector field of interest is not output by the solver and must be calculated from solver outputs. Such fields may be called *derived* fields. For example, CFD solvers used at NAS typically output density, momentum, and energy. Scientists frequently

1. This work is supported through NASA contract NAS2-12961.
2. The NAS (Numerical Aerodynamic Simulation) Systems Division at NASA Ames Research Center is a leading supercomputer facility dedicated to the study of computational aerosciences, particularly CFD [Bailey86].

want particle traces in the velocity or vorticity fields. Calculating derived fields is O(solution-size).

Fortunately, some important visualization techniques typically access a very small portion of the solution domain. Consider particle tracing. An individual particle trace is a one dimensional path through a four dimensional solution (x,y,z and time). The data accessed to implement one step of one particle trace is O(1); *i.e.*, independent of solution size. In fact, in the common case where the vector field is precalculated and memory resident, particle tracing time is well known to be O(number-of-particles) and independent of solution size.

Combining two techniques allows O(number-of-particles) particle tracing time on disk resident solutions: 1. UNIX memory mapped files to avoid unnecessary disk IO; 2. lazy evaluation [Abelson85] to avoid unnecessary calculation of derived vector fields. Using these techniques (described below), we have demonstrated O(number-of-nodes-required) disk IO times and O(number-of-particles) CPU times when tracing particles in vector fields derived from very large, disk resident, unsteady CFD solutions. Needless to say, we achieved dramatic performance improvements over conventional techniques - one to two orders of magnitude.

Prior Work

Kelick90 mentions lazy evaluation of derived fields using C++, but gives no performance or implementation details. At our suggestion, Lane93 used lazy evaluation, but not memory mapping, to trace particles in the velocity field derived from large unsteady CFD solutions. Lane93 achieved a factor of five improvement in CPU time. Hultquist92 mentions memory mapping for steady state visualization. Eliasson89 uses lazy evaluation of computational space vector fields to compute streamlines.

Description of Technique

Memory Mapping

The first problem is to read from disk into memory only those solution nodes necessary to interpolate the vector field at particle positions. To do this precisely requires an inordinate number of slow disk seeks and some very complex software. Fortunately, the virtual memory systems available on most workstations is designed to manage disk to memory transfers of data as needed. What we need is a way to tell the virtual memory system to treat a file as part of a process' virtual memory space. The UNIX function mmap() does exactly that [UNIX]. Memory then caches the disk file as transparently as the CPU cache in a RISC processor caches main memory. Using mmap() makes disk IO time O(number-of-nodes-in-cells-containing-particles), modified by buffering effects.

Using memory mapped files can lead to management problems when some files are immediately read into memory and other files are memory mapped. Globus93 shows how to transparently access data allocated in a number of different ways, including memory mapping, by using a C++ class hierarchy to encapsulate blocks of data.

Lazy Evaluation

Avoiding unnecessary disk IO is not enough. If derived fields are calculated for the entire domain, then visualization time will be O(solution-size). Lazy evaluation involves calculating the vector field only at nodes actually accessed by the tracing algorithm. We accomplish this using a C++ class hierarchy representing fields with virtual member functions for accessing the field data. Visualization algorithms using these classes are unaware that the vector field accessed may not be stored in memory.

To hide a field's implementation (stored or lazy evaluation) from tracing algorithms, we note that such algorithms need the value of the vector field at any point; and need nothing else from a field. Unless this point happens to lie on a node, the value of the vector field must be interpolated from nearby nodes. The interpolation code is identical for stored and lazy fields and can be placed in a common base class, but the mechanism for determining the vector at a node differs and requires two derived classes. Thus, returning the vector value of the field at a node must be a *virtual* member function. When a field is stored, the member function returns the value stored in memory. When the field is lazy evaluated, the member function gets the values stored in the solution, performs the appropriate computation, and returns the result. The C++ virtual member function mechanism chooses the right implementation at run time.

A sketch of code to trace a particle in a regular 3-D field using stored or lazy fields follows. Many details are left out. In particular, the fields we used in our experiment were multiple-zone, iblanked, curvilinear grids, not the regular grid in this code; and we used a modified RK2 rather than Euler integration. This does not change the fundamentals.

Assume that `class vector`[1] is appropriately implemented and that the solution is stored in a `field` object. Particle trace code might look something like this:

```
// one Euler integration step
vector oneStep(const vector& p, const field& f, float deltaT)
  { return p +  (deltaT * f.interpolate(p)); }
```

The field classes might look like:

```
class field {
public:
  virtual vector nodeValue( int i, int j, int k ) = 0;
  vector interpolate( const vector& p )
  {
       int i,j,k = integer parts of p
       vector v = // interpolate using nodeValue(i,j,k),
                  // nodeValue(i+1,j,k),
                  // nodeValue(i,j+1,k), etc.
                  // and fractional parts of p
       return v;
  }
};

class precomputedField : public field {
  float data[isize][jsize][ksize][vectorSize];
  public:
       vector nodeValue( int i, int j, int k )
           { return vector( data[i][j][k] ); }
};
```

1. Code fragments are in `Courier` font.

```
class lazyField : public field {
   const field* Solution;
public:
   lazyField( const solution* s ) { Solution = s; }
   vector nodeValue( int i, int j, int k )
   {
         // lazy evaluation
         return desiredOperation( Solution.nodeValue(i,j,k) );
   }
};

vector desiredOperations( const vector& solutionVector )
   { // calculate and return the vector desired }
```

Experimental Results

Several experiments were conducted by tracing a number of particles through two solutions with different sizes. The large solution [Atwood92] contained about 3.3 million nodes per time step. The smaller solution contained about 32 thousand nodes per time step. Both solutions were defined on multiple 3-D curvilinear, overlapping, iblanked grids [Walatka92]. The velocity field was calculated from density and momentum values in the solutions. In addition, the tracing method transformed the velocity field from physical to computational space [see Eliasson89]. Solutions were memory mapped and calculations used the lazy evaluation technique. Each solution contained 15 time steps. Each solution node contained 20 bytes per time step. For each experiment the number of particles traced was constant and the number of cells spanned by initial particle locations was the same.

All software was implemented in C++ running inside SuperGlue [Hultquist92], a visualization environment based on Scheme (a dialect of LISP). The software ran on an SGI 320 VGX with 64 Mbytes of main memory. Because the SGI did not have sufficient disk space for the large solution, all solution data resided on a CONVEX accessed via NFS over Ethernet.

Disk access via NFS was very slow, slightly less than 250 thousand bytes per second. This could have been improved by executing the code on the CONVEX. Unfortunately, the CONVEX C++ compiler is rather primitive. Our multi-dimensional array indexing technique was also very slow. As this was not the focus of the research, the code was not optimized. Note that both disk access and the tracing code were much slower than achievable; so the absolute results described here are of no importance, only the relative times between the two different sized solutions.

Each trial involved tracing a certain number of particles three times through each solution. Between each run within an experiment, all of memory was filled with 1's to insure that no data remained in memory between test runs. Average, minimum, and maximum wall clock times were computed. Only minimum times are presented here because system load influenced average and maximum times. Average times were about 20% greater than minimum times. Table 1 contains the results.

Table 1:

Measure	Large Solution (sec.)	Small Solution (sec.)	Large/Small
984 particles	23.00	21.00	1.10
98 particles	4.00	3.00	1.33
98 particles/CPU only	2.72	2.44	1.11
98 particles/ non-CPU only	1.28	0.55	2.33
Time to read one solution time step from disk	270.89	2.71	99.96
Disk IO time saved using memory mapping	269.21	2.16	NA

Tracing particles in the large solution took 10-30% longer than in the small solution. The processing necessary for each particle should be identical, so where is the difference coming from? Notice that non-CPU time is 2.33 times longer for the large solution. This suggests that the particles in the large solution pass through more cells; i.e., the grid is denser. Still unexplained is the 11% difference in CPU time between the large and small solutions. Remember that the solutions are on multiple overlapping curvilinear grids. The unexplained 11% might reflect more particles in the larger solution passing from one curvilinear grid to the next, an operation that takes some CPU time.

It should be noted that workstation memory was full when the read tests were conducted, so the times reflect paging data out as well as reading solution data in. This is realistic for unsteady visualization. A full disk read took 100 times longer for a large solution, as one would expect given the difference in sizes.

Discussion

Calculating a set of 1-D paths through a 4-D solution is clearly well suited to the optimizations discussed in this paper. Cutting planes and other techniques that access only a portion of the solution should also benefit. However, many visualization techniques, such as volume rendering, marching cubes [Lorenson87] isosurface calculations, and critical point finders [Helman90] must access every data point in a solution and will not benefit from the optimization approach presented here.

In this work there is no attempt to remember the values calculated by lazy evaluation. In many cases this would improve performance. This area could benefit from additional research.

Summary

Using memory mapped files and lazy evaluation of derived fields, particle tracing from disk resident solutions can be O(number-of-particles) rather than O(number-of-solution-nodes). In most cases, these optimization techniques provide much faster particle tracing than reading the entire solution into memory and calculating derived fields over the entire solution domain before particle tracing begins.

Acknowledgments

J. Hultquist and K. Miceli reviewed drafts and made many helpful comments. I would like to thank the members of the Applied Research Branch of the NAS Division of NASA Ames Research Center for the superb working environment to which I have become addicted. This work is supported through NASA contract NAS2-12961.

References

Note: some of these documents are available on the World Wide Web. For these, the URL is provided.

[Abelson85] H Abelson, G. J. Sussman, and J. Sussman, *Structure and Interpretation of Computer Programs*, MIT Press, Cambridge, MA, 1985.

[Atwood92] C. A. Atwood and W. R. Van Dalsem, “Flowfield Simulation about the SOFIA Airborne Observatory,” *30th AIAA Aerospace Sciences Meeting and Exhibit*, January 1992, Reno, Nevada, AIAA 92-0656.

[Buning85] P. G. Buning and J. L. Steger, “Graphics and Flow Visualization in Computational Fluid Dynamics,” AIAA-85-1507-CP, AIAA 7th Computational Fluid Dynamics Conference, 15-17 July, 1985, Cincinnati, OH.

[Bailey86] F. R. Bailey, “Status and Projections of the NAS Program,” *Symposium on Future Directions of Computational Mechanics*, ASME, Anaheim, CA, 7-12 December, 1986.

[Bancroft90] G. Bancroft, F. Merritt, T. Plessel, P. Kelaita, R. McCabe, and A. Globus, “FAST: A Multi-Processing Environment for Visualization of CFD,” *Proceedings Visualization ‘90*, IEEE Computer Society, San Francisco, 1990.

[Eliasson89] Eliasson, "Stream 3d: Computer Graphics Program for Streamline Visualization," *Advances in Engineering Software*, 1989, Vol. 11, No. 4, pp. 162-168.

[Globus92] A. Globus, “A Software Model for Visualization of Time Dependent 3-D Computational Fluid Dynamics Results,” NASA Ames Research Center, NAS Systems Division, Applied Research Branch technical report RNR-92-031, November 1992. URL http://www.nas.nasa.gov/RNR/Visualization/AlGlobus/Unsteady/alUnsteady.html.

[Globus93] A. Globus, “C++ Class Library Data Management for Scientific Visualization,” OON-SKI ’93, Sunriver, Oregon, April 25-27, 1993. URL http://www.nas.nasa.gov/RNR/Visualization/AlGlobus/tData/tData.html.

[Helman90] J. L. Helman and L. Hesselink, “Surface Representation of Two- and Three-Dimensional Fluid Flow Topology,” *Proceedings Visualization ‘90*, IEEE Computer Society, San Francisco 1990.

[Hultquist92] J. P. Hultquist and E. L. Raible, “SuperGlue: A Programming Environment for Scientific Visualization,” *Proceedings IEEE Visualization ‘92*, October, 1992, Boston, MA.

[Kerlick90] G. D. Kerlick, “Moving Iconic Objects in Scientific Visualization,” *Proceedings Visualization ‘90*, IEEE Computer Society, San Francisco 1990.

[Lane93] D. Lane, “Visualization of Time-Dependent Flow Fields,” *Proceedings IEEE Visualization ‘93*, October, 1993, San Jose, CA., pp. 32-38.

[Legensky90] S. M. Legensky, “Interactive Investigation of Fluid Mechanics Data Sets,” *Proceedings Visualization ‘90*, IEEE Computer Society, San Francisco, 1990.

[Lorenson87] W. E. Lorenson and H. E. Cline, “Marching Cubes: a High Resolution 3d Surface Construction Algorithm,” *Computer Graphics*, Vol. 21 No. 4, July 1987, pp. 163-169.

[SGI92] *IRIS Explorer User’s Guide*, Silicon Graphics Computer Systems, Document 007-1369-010, 1992.

[UNIX] *UNIX* mmap *Man Page*.

[Upson89] C. Upson, T. Faulhaber, D. Kamins, D. Laidlaw, D. Schlegel, J. Vroom, R. Gurwitz, A. van Dam, “The Application Visualization System: A Computational Environment for Scientific Visualization,” *IEEE Computer Graphics and Applications*, July 1989, pp. 30-41.

[Walatka92] P. P. Walatka, P. G. Buning, *Plot3d User’s Manual*, NASA Technical Memorandum 101067, NASA Ames Research Center.

[Wavefront91] *The Data Visualizer User’s Guide*. Wavefront Technologies, Inc.