UNSTRUCTURED ADAPTIVE MOVING MESH SOLUTION OF UNSTEADY SHEAR FLOWS AND FREE-SURFACE FLOWS

J. Blair Perot
Department of Mechanical & Industrial Engineering
University of Massachusetts, Amherst
Amherst, Massachusetts 01003, USA
perot@ecs.umass.edu

David P. Schmidt
Department of Mechanical & Industrial Engineering
University of Massachusetts, Amherst
Amherst, Massachusetts 01003, USA
schmidt@ecs.umass.edu

ABSTRACT

A method for performing anisotropic unstructured mesh adaptation is discussed that is well suited for computational fluid dynamics on parallel distributed memory computers. This algorithm which involves mesh motion, rather than refinement and coarsening, is fast, parallel, and fully conservative. The approach is effective at accurately capturing free-surface flows, boundary layers, and unsteady shear layers. Examples of some of applications are presented.

BACKGROUND

Mesh adaptation allows high quality CFD solutions to be computed by placing the mesh where it is most needed. Fluid solutions tend to have very thin but physically critical layers (shear layers and boundary layers) that must be adequately resolved in order to accurately predict the flow behaviour. In complex industrial or environmental applications, the position and extent of these layers is often difficult to predict beforehand and they often move with time. Anisotropic mesh adaptation allows these thin layers to be automatically resolved.

Classic mesh adaptation involves mesh refinement and coarsening. This strategy is not very efficient on parallel machines, particularly distributed memory machines like PC clusters. Mesh refinement and coarsening occurs primarily in a few isolated regions of the flow domain. The work of refinement and coarsening is therefore unequally distributed on the processors. In addition, when the adaptation is complete some processors now have fewer or more unknowns than their neighbours leading to a lack of load balance in the CFD part of the parallel calculation. It is possible to redistribute the cells and rebalance the simulation after the adaptation process. However this redistribution process involves extensive communication that is difficult to hide with any useful computations. Furthermore, load balancing the CFD calculation still does not address the lack of balance in the adaptation algorithm itself. Classic point insertion and removal also leads to meshes with relatively large jumps in the mesh size (up to a factor of eight for 3D Cartesian meshes) that cause strong artificial dispersion.

An alternative strategy is to move the mesh points into some regions (refinement) and away from others (coarsening). While the mesh moves, the number of mesh points on each processor remains constant, and the work to move the points is equally spread among the processors. Mesh motion is therefore well load balanced and automatically leaves the CFD portion of the code well load balanced. This approach to adaptation also leads to a very smooth mesh with very gradual shape and size changes.

There are two further benefits to mesh motion. Mesh motion does not change the number of unknowns, so the calculation time for the CFD problem remains highly predictable. On the other hand, adaptation via point insertion can lead to an explosive growth in solution times if it is not carefully controlled (typically via active human intervention, which one would like to avoid). Mesh motion can also be used to solve other CFD issues besides thin layers. For example, an excellent way to accurately resolve a moving discontinuity (like a free-surface or a shock) using very few mesh points is to move the mesh with the discontinuity.

The key elements of the mesh adaptation algorithm which will be discussed are: (1) a physically inspired equation for the mesh motion that produces a high quality mesh refined in regions of large solution gradients, (2) an anisotropic error measure on which to adapt, (3) an algorithm to maintain high quality mesh connectivity, (4) a correction to the CFD algorithm that accounts for the fact that the mesh is moving continuously during each timestep of the CFD solver. This correction conserves mass and momentum (Perot, 2000) as well as kinetic energy (Zhang *et al*., 2002), and preserves constant numerical solutions under the action of arbitrary mesh motion. The mesh motion is *not* treated as in ALE (Arbitrary Lagrangian-Eulerian) schemes where a conservative remapping from one mesh to another is performed every few timesteps (Hu *et al*., 2001).

MESH MOTION

A physical analogy is used to determine the mesh motion.

The mesh is treated as a large collection of nodes. Each edge of the mesh is a line between neighboring mesh nodes and is treated as a linear spring (Habashi *et al*., 2000). The spring constant for each spring is variable. A large value of the spring constant in a local region of the mesh will cause the springs to pull strongly in that region and will result in mesh refinement in that area.

A mesh is not just defined by the node positions, but also by its connectivity. The connectivity determines which nodes are considered to be neighbors and it therefore defines the edges. This section considers a procedure for optimizing the node positions of the mesh once the edges are prescribed. In a subsequent section the issue of optimal connectivity is addressed.

In the case of a uniform spring constant for all the edges of the mesh, the equilibrium position for the nodes is one in which the edge lengths are all approximately the same size. The result of this algorithm on a very simple 2D mesh with only one node that is free to move (the others are on a fixed boundary) is shown in figure 1. If the boundary nodes move (because they are on a moving free-surface or a moving slid wall) the interior mesh will adjust to make the interior cells as smoothly distributed as possible.

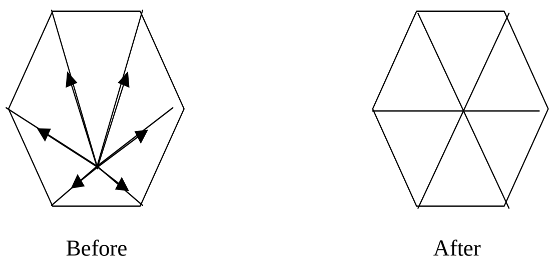

Figure 1. Schematic representation of mesh smoothing in two-dimensions. Arrows in the first picture represent spring forces acting on the central node. Boundary nodes are assumed fixed in this simple example.

The mathematical equation for the smoothing operation is,

$$\mathbf{x}_{node}^{n+1} - \mathbf{x}_{node}^{n} = -\mathbf{G}_{n2e}^{T} k_e \mathbf{G}_{n2e} \mathbf{x}_{node}^{n+1} \qquad (1)$$

where $\mathbf{G}_{n2e}\mathbf{x}_{node}^{n+1} = \mathbf{x}_{node2}^{n+1} - \mathbf{x}_{node1}^{n+1}$ is the difference operator between two nodes on an edge, and its transpose is a summation of all the edges touching a single node. This equation is an exact implementation of the *unsteady* linear spring analogy. Boundary nodes do not obey this equation. Their motion is either given (such as on a moving wall), or their motion is specified to be Lagrangian (such as on a free-surface interface).

The unsteady form of the equation allows the springs to relax in a finite time rather than obtaining their equilibrium distribution at every time step. This is useful in the nonlinear case when the spring constant is a function of the solution error and therefore a function of position. In this formulation, the spring constant is a dimensionless number. If $k \gg 1$ the relaxation to equilibrium happens in less than one time step. If $k \approx 1$ the mesh will not move more than the local mesh spacing in one time step. In the results presented in this paper, the spring constant is always normalized such that its maximum value is one.

The solution of the mesh relaxation equation (Eqn 1) is carried out using a diagonally preconditioned Conjugate Gradient solver. Smoothing an initial mesh can take some time depending on the quality of the initial mesh. However, once the simulation is evolving the mesh solver takes 2-10 iterations to converge and is extremely fast. With the spring constant less than or equal to 1 the system is highly diagonally dominant and almost any iterative method will converge quickly. This is far more cost efficient than the common practice of regenerating an entirely new unstructured mesh (after a few timesteps) and it maintains a very high quality mesh at every timestep. This algorithm does not change the mesh connectivity so all control volumes retain the same neighbors as they move and distort. This allows the code to avoid the remapping stage of ALE methods and incorporate the mesh motion directly into the discrete control volume equations (presented in a later section).

With a uniform spring constant, mesh motion is extremely effective at maintaining high quality meshes even while the boundary points of the mesh move. Figure 2 shows a simulation of a fluid ligament collapsing under the influence of surface tension. Surface tension tends to bring the two ends together but it also causes a pinching instability at the center of the ligament that can separate the ligament into two parts. Using classic fixed mesh adaptive methods to solve this problem is difficult. With classic adaptation methods only a small part of the total domain is occupied by the ligament at any particular time. The large density ratio of air and water (1000) causes numerical difficulties in the momentum equation and the small mesh sizes necessary to resolve the interface require very small timesteps to be used.

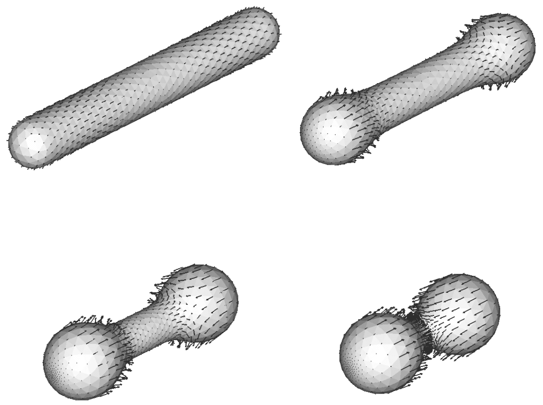

Figure 2. Collapse of a fluid ligament under the action of surface tension using a moving mesh algorithm with uniform spring constants. Surface motion is Lagrangian, interior motion is via Eqn. 1. Re = 6, We = 3.

The discrete mesh motion equation (Eqn. 1) can be viewed as a discretization of the following partial differential equation.

$$\frac{\partial \mathbf{x}}{\partial t} = \alpha \nabla \cdot k \nabla \mathbf{x} \tag{2}$$

Elliptic equations of this character are often used to defined the mesh characteristics (Harten *et al*., 1983 and Huang & Russel, 1997). In certain limits the discrete mesh motion equation is also equivalent to purely heuristic mesh distribution algorithms. For example when $k = 1/N$ where N is the number of node neighbors Eqn. 1 is equivalent to placing each node at the average position of its neighbors.

Note that in three-dimensions, the mesh motion equation (Eqn 1) does not remove sliver cells. However, it can be modified so do so. In this work, this is not necessary since the sliver cells are removed by the algorithm which optimizes the mesh connectivity (presented in a later section).

ANISOTROPIC MESH ADAPTION

With the addition of variable spring constants it is also possible to use mesh motion to adapt the mesh to important *internal* solution features. In this work this is achieved by setting the spring constant to be proportional to the second derivative of the solution in the direction of the spring. This causes the mesh to be pulled into regions where the solution gradients are changing rapidly and stretched where the solution is linear (or constant).

This approach automatically performs anisotropic mesh adaptation which is so important for thin structures such as boundary layers and shear layers. Springs aligned along the thin structure will have small solution second derivatives along the spring direction. These springs will have relatively small spring constants and relatively long lengths. On the other hand, springs aligned across the layer will have large solution second derivatives along the spring direction. These springs will have very large spring constants and short lengths. The result is a mesh (even an unstructured one) that is highly refined across the layer, but uses a course resolution along the layer.

Consider first, the adaptation of the solution to a single solution variable, ϕ. The current results use the kinetic energy as the adaptation variable. The preliminary spring constant on each edge is then given by,

$$\hat{k}_e = \frac{(\mathbf{x}_{n2} - \mathbf{x}_{n1}) \cdot (\nabla \phi|_{n2} - \nabla \phi|_{n1})}{(\mathbf{x}_{n2} - \mathbf{x}_{n1}) \cdot (\mathbf{x}_{n2} - \mathbf{x}_{n1})} \tag{3}$$

This spring constant is then normalized so that the maximum value on the domain is equal to 1, and for stability reasons we frequently limit the extreme values of the spring constant to be at least 1/100 of the mean value of the spring constant, and less than 100 times the mean value. The final spring constant is then given by,

$$k_e = \frac{MIN(MAX(\hat{k}_e, \overline{\hat{k}_e}/A), \overline{\hat{k}_e}A)}{MAX(k_e)} \tag{4}$$

where the value A=100 is used in this work.

When adaptation on multiple flow variables is desired, the spring constant is taken to be the maximum of all the individual spring constants. This causes the mesh to adapt to the least resolved flow variable at any location. However, adaptation on multiple variables is not addressed further in this work. For incompressible flows, the kinetic energy has proven to be sufficient in all the flows tested.

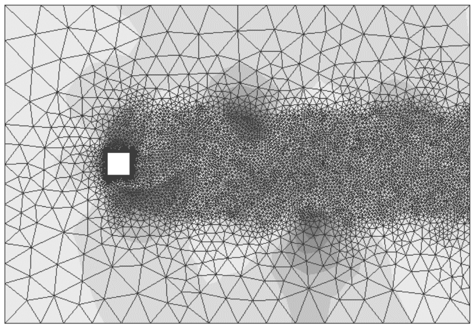

Figure 3. Mesh and solution for the unsteady RANS flow around a square cylinder at Re=21,400. The mesh in the object boundary layer and wake has been refined by human interaction. It is still not sufficient for an accurate (and mesh independent) answer.

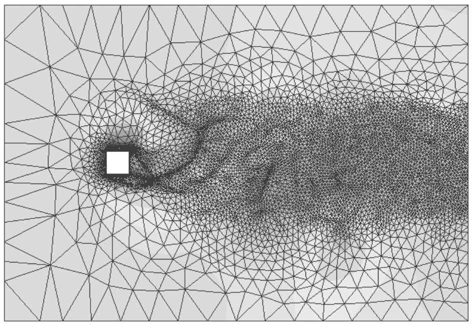

Figure 4. Adaptive mesh refinement applied to the unsteady vortex shedding behind a square cylinder. This uses exactly the same number of mesh cells as in figure 3 (24,234) but now an accurate (mesh resolution independent) solution is achieved.

An example of anisotropic mesh adaptation via mesh motion is shown in Figures 3 and 4. Figure 3 shows a mesh that has been adapted by a human to try to resolve the important areas of interest. The adapted mesh in figure 4 contains the same number of mesh points as the mesh in figure 3 but has over 20 times the mesh resolution across the critical shear layers. The vortex shedding from a square cylinder is unsteady, and the resolution adapts in time, maintaining high resolution on the moving shear layers at all times. However, it is important to note that the mesh motion is not Lagrangian, and the elliptic nature of the mesh motion equation (Eqn 1) means the mesh never becomes twisted or entangled.

OPTIMAL CONNECTIVITY

As discussed earlier, a mesh consists of two important properties, the placement of the nodes and how those nodes are connected together. The mesh motion algorithm optimizes the node positions but not necessarily the mesh connectivity.

Optimal mesh connectivity is maintained via a mesh flipping algorithm (Joe, 1989). Unlike the mesh motion algorithm, flipping does not take place during the timestep. It occurs more like the ALE remap step at the end of the timestep calculation. The interpolation that is necessary during the flipping process is highly local and very infrequent. The cost and numerical impact of the flipping process is very small since only approximately 1 in every 5,000 cells gets flipped per timestep.

At this time the flipping process can only be applied to triangular meshes in 2D and tetrahedral cells in 3D. In two dimensions, flipping is a local process where two triangles are replaced by two different triangles with better properties. The process is illustrated in figure 5. The usual criterion for flipping in 2D is to remove small angles. From the figure it can be seen that this also has the additional benefit of removing large angles as well. It is not possible with flipping alone to make all angles acute. However it is possible to with flipping to make the mesh Delaunay. In 2D the Delaunay mesh is the optimal (in many senses) connectivity for any given node distribution. It maximizes the minimum angle both locally and globally, it ensures that no node lies within the circumcircle of any other node, it means a locally orthogonal dual mesh (the Voronoi dual) is well formed, and it is the most logical mesh for defining nearest neighbor connectivity.

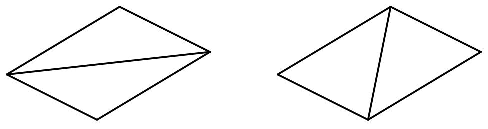

Figure 5. Example of flipping two triangles to optimize the mesh connectivity.

Perhaps most importantly for the flipping process, it can be shown that flipping only the local triangle pair sets that violate the Delaunay criteria will lead to the globally Delaunay (and optimal) mesh connectivity. There are many ways to determine if the local Delaunay criterion is violated by two neighboring triangles. In this work we use a comparison of the triangle circumcenter and centroid positions.

$$\lambda = (\mathbf{x}_2^{cent} - \mathbf{x}_1^{cent}) \cdot (\mathbf{x}_2^{circum} - \mathbf{x}_1^{circum}) \qquad (5)$$

If $\lambda < 0$ the two triangles are flipped.

Before flipping the solution is approximated on the surrounding local nodes. After flipping the solution is re-interpolated from the surrounding nodes onto the new triangles. The interpolation is performed in such a way that mass and momentum are conserved. However, the process is slightly diffusive and does lead to very small amounts of artificial diffusion. The amount of artificial diffusion is proportional to the number of flips, which tends to be small.

In three dimensions the flipping process is still possible. However it is now necessary to flip two tetrahedra into three, or vice versa. It has been demonstrated (Perot & Nallapati, 2003) that all other degenerate or more complex flipping situations can be decomposed into these two types of flips (2 to 3 tetrahedra and 3 to 2 tetrahedra)., if the 2 to 3 flips are all performed before the 3-2 flips. An illustration of a 3D tetrahedral flip is shown in figure 6.

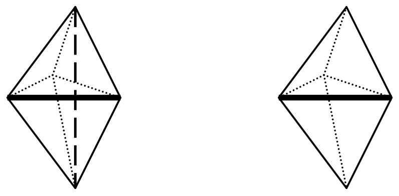

Figure 6. Example of flipping in three dimensions from 3 tetrahedra to 2. The central edge is removed along with the three faces connected to it, and replaced with a single face.

The situation in 3D is not as attractive mathematically as in 2D. The Delaunay mesh no longer maximizes the minimum angle. Flipping only locally non-Delaunay combinations does not always lead to a globally Delaunay mesh. Knots are possible in 3D which must be untangled by temporarily flipping a locally Delaunay (optimal) tetrahedra set into a non-Delaunay configuration. Heuristics (Nallapati and Perot, 2000) for choosing which pairs to flip in order to untie knots have always proven successful to date. Finally, the Delaunay criteria is not always optimal in 3D. It does not remove sliver cells.

In 3D the criterion of locally maximizing the minimum angle proves more successful. This criterion is no longer equivalent to the Delaunay criteria as it was in 2D. In addition, no rigorous statements of optimal global connectivity can be made. Nevertheless, this procedure removes sliver cells and has proven successful to date when implemented on a variety of 3D moving meshes.

Mesh flipping is a local process, so if domain decomposition is used to distribute the problem on a parallel processor it is possible for each processor to flip the cells in its particular domain independently of the other processors. This part of the algorithm is therefore trivially parallel. However it is possible that the flipping may not be well load balance. More cells on one processor might require flipping. In addition, it is more difficult to flip those cells that have a neighboring cell on a different CPU. This is achieved in the current implementation by transferring one (or two) of the cells between the CPUs so that all the cells required in a particular flipping operation reside on a single CPU. Since this transfer operation is relatively slow, only one (collective) CPU transfer per timestep is made. This may not be optimal, further flipping of cells on the CPU boundaries might make for a better connected mesh. However, for unsteady problems this suboptimal procedure is sufficient. This procedure is also used to load balance the number of cells on each CPU since they do not remain constant during the 3D flipping procedure.

GOVERNING EQUATIONS

The mass and momentum equations for a moving distorting control volume are:

$$\tfrac{d}{dt}\int \rho dV + \int \rho(\mathbf{u}-\mathbf{v})\cdot \mathbf{n} dA = 0 \quad (6)$$

$$\tfrac{d}{dt}\int \rho \mathbf{u} dV + \int \rho \mathbf{u}(\mathbf{u}-\mathbf{v})\cdot \mathbf{n} dA = \int \tau \cdot \mathbf{n} dA \quad (7)$$

where $\mathbf{v}$ is the velocity of the control volume surface. In addition there is an additional equation

$$\tfrac{d}{dt}\int dV = \int \mathbf{v}\cdot \mathbf{n} dA \quad (8)$$

which states that the rate of change of the volume of the control volume is directly proportional to the normal velocity of its surface. Any definition of the mesh velocity (and more importantly the face integral of its normal component) must be consistent with this last equation. In two dimensions,

$$\int \mathbf{v}\cdot \mathbf{n} dA = \mathbf{v}_f^{centr} \cdot \tfrac{1}{2}(\mathbf{n}_f^{n+1} A_f^{n+1} + \mathbf{n}_f^{n} A_f^{n}) \quad (9)$$

is consistent. Note that both the face normal and area at the beginning and end of the time step must be used, but that only the velocity of the mesh face center of gravity is necessary.

In 3D the time variation of the geometry is more complex and The velocity of the mesh nodes must also be accounted for.

$$\int \mathbf{v}\cdot \mathbf{n} dA = \mathbf{v}_f^{centr} \cdot \tfrac{1}{2}(\mathbf{n}_f^{n+1} A_f^{n+1} + \mathbf{n}_f^{n} A_f^{n}) - \tfrac{\Delta t^2}{12}\mathbf{v}_f^{centr} \cdot \sum^{edges} (\mathbf{v}_{n1} \times \mathbf{v}_{n2}) \quad (10)$$

Although the last term is small, and frequently of the same order as the timestepping error, it can not be neglected. Equation (8) can not be approximately satisfied.

Having obtained a consistent mesh velocity the inclusion of mesh motion into a finite volume scheme is straight forward. The convective velocity is altered by the mesh velocity and the changing volume is included inside the time derivative. With these changes no remeshing at the end of the timestep is necessary. The inclusion of mesh motion into finite element schemes is discussed in Tezduyar et al. (1992) and Tourigny & Hulsemann (1998).

With this approach it is possible to develop numerical methods that conserve mass, momentum, kinetic energy, and even vorticity (Perot, 2000). It should be noted that moving mesh methods are not restricted to simple topologies. Figure 7 shows the break up of a fluid ligament into a binary drop distribution. The fluid is in a vacuum, and in the final frame there is no mesh in the region between the droplets.

Mesh motion is also advantageous for multiscale problems. It allows a very fine mesh to be placed about a small moving object that is embedded in a much larger simulation domain. Figure 8 shows a zoom in on the mesh and the velocity field surrounding a small liquid droplet. The droplet is only a small portion of the simulation and is embedded in a turbulent gas flow with scales up to 50 times larger than the droplet diameter. The mesh transitions smoothly as it moves away from the object. There are none of the issues of mesh interpolation that occur with methods that use two overlapping meshes. The single mesh makes this method easy to decompose and parallelize.

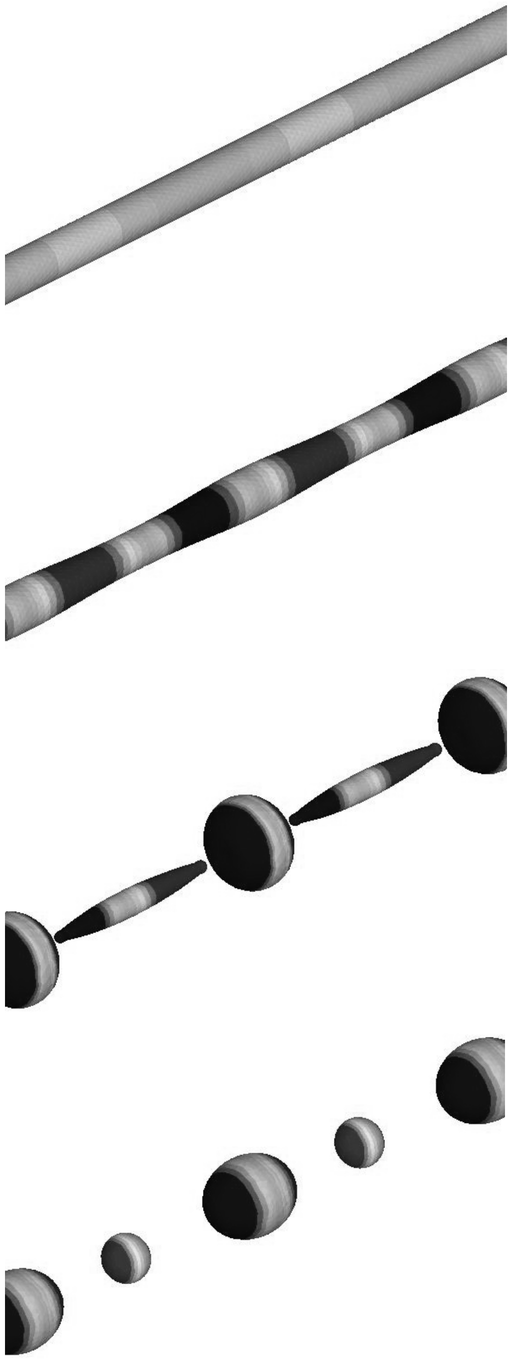

Figure 7. Breakup of a fluid ligament into a binary droplet distribution.

CONCLUSIONS

Mesh adaptation has the potential to significantly enhance CFD solution accuracy. Fluids frequently exhibit moving surfaces and thin internal shear layers. Capturing these discontinuities and near discontinuities is often very important for the overall solution behavior. *A priori* meshing of these structures is frequently impossible, and refining the entire mesh can be extremely expensive.

Moving mesh adaptation provides many advantages over

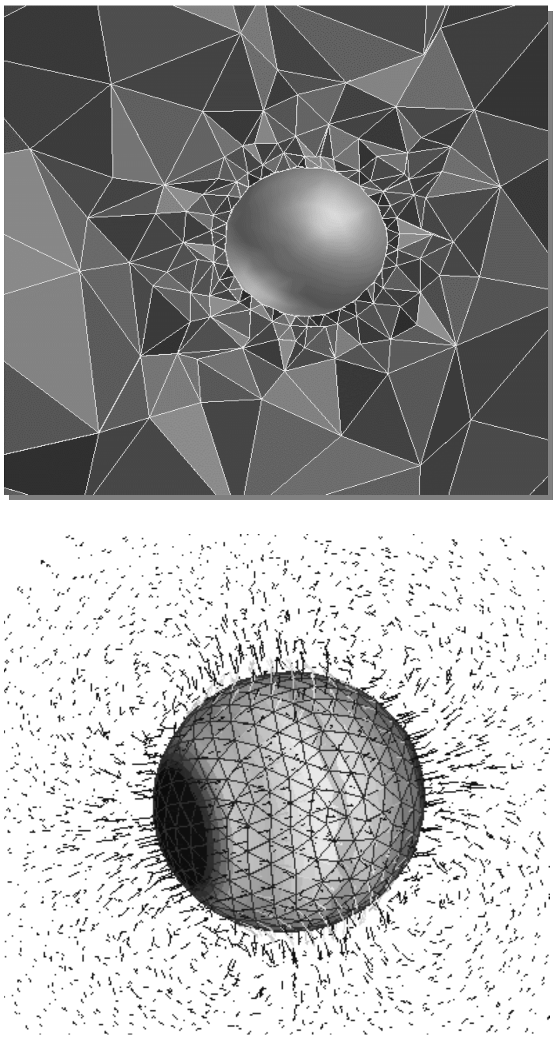

Figure 8. The 3D mesh and velocity field near a small liquid droplet embedded in a turbulent gas flow.

traditional adaptation via point insertion and removal (or remeshing). The solution cost remains fixed, with mesh resources moving smoothly (in time and space) to where they are needed most. In addition mesh motion is highly parallel and leaves the CFD calculation well load balanced. Mesh flipping is used to maintain optimal connectivity. The mesh flipping algorithm is highly parallel but not perfectly load balanced. Fortunately, the small number of cells that are flipped per timestep means this has little impact on the overall performance.

Anisotropic adaptation is straightforward in the moving mesh framework. Anisotropic adaptation is critical for internal features that are very thin or flat, such as shear layers or boundary layers.

Finite volume methods are very common in CFD, and it is clear that the inclusion of moving meshes into these methods is quite simple. The significant advantages of these moving mesh modifications were demonstrated in a number of test cases.

REFERENCES

Habashi, W. G., Dompierre, J., Bourgault Y., Ait-Ali-Yahia, D., Fortin, M. and Vallet, M-G., 2000, "Anisotropic Mesh Adaptation: Towards User-Independent, Mesh-Independent and Solver-Independent CFD Solutions: Part I: General Principles," *International Journal for Numerical Methods in Fluids*, **32** (6), p. 725-744.

Harten, A., Hyman J. M., 1983, “A self-adjusting grid for the computation of weak solutions of hyperbolic conservation laws”, *J. Comput. Phys.* **50, p.** 235.

Hu, H. H., Patankar, N. A. & Zhu, M. Y., 2001, “Direct numerical simulations of fluid-solid systems using the arbitrary Lagrangian-Eulerian Technique”, *J. of Comput. Phys.*, **169**, p. 427-462.

Huang W. and Russel R. D., 1997, “Analysis of Moving Mesh Partial Differential Equations with Spatial Smoothing”, *SIAM J. Numer. Anal*. **34**, 1106-1126.

B. Joe, B., 1989, “Three-dimensional triangulations from local transformations”, *SIAM Journal of Scientific and Statistical Computing*, pp.718-741.

Nallapati R. and Perot, J. B., 2000, “Numerical simulation of free surface flows using a moving mesh”, *2000 American Society of Mechanical Engineers, Fluids Engineering Summer Conference*.

Perot, J. B., 2000, “Conservation Properties of Unstructured Staggered Mesh Schemes”, *J. Comput. Phys*., **159**, p. 58-89.

Perot, J. B. and Nallapati, R., 2003, “A moving unstructured staggered mesh method for the simulation of incompressible free-surface flows” , *J. Comput. Phys*., **184**, p. 192-214.

Tezduyar, T. E., Behr, M. and Liou, J., 1992, “A New Strategy for Finite Element Computations Involving Moving Boundaries and Interfaces--The DSD/ST Procedure: I. The Concept and the Preliminary Numerical Tests”, *Comput. Meth. Appl. Mech. Engrg.*, **94,** p. 339-351.

Tourigny, Y. & Hülsemann, F**.,** 1998, “A New Moving Mesh Algorithm for the Finite Element Solution of Variational Problems”, *SIAM J. Numer. Anal*. **35,** p. 1416-1438.

Zhang, X., Schmidt, D. and Perot, J. B., 2002, “Accuracy and Conservation Properties of a Three-Dimensional Unstructured Staggered Mesh Scheme for Fluid Dynamics”, *J. Comput. Phys*., **175**, p. 764-791.