INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS, VOL. 24, 1321–1340 (1997)

PARALLEL COMPUTATION OF INCOMPRESSIBLE FLOWS WITH COMPLEX GEOMETRIES

A. A. JOHNSON AND T. E. TEZDUYAR*

Aerospace Engineering and Mechanics, Army HPC Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, MN 55415, U.S.A.

SUMMARY

We present our numerical methods for the solution of large-scale incompressible flow applications with complex geometries. These methods include a stabilized finite element formulation of the Navier–Stokes equations, implementation of this formulation on parallel architectures such as the Thinking Machines CM-5 and the CRAY T3D, and automatic 3D mesh generation techniques based on Delaunay–Voronoï methods for the discretization of complex domains. All three of these methods are required for the numerical simulation of most engineering applications involving fluid flow.

We apply these methods to the simulation of airflow past an automobile and fluid–particle interactions. The simulation of airflow past an automobile is of very large scale with a high level of detail and yielded many interesting airflow patterns which help in understanding the aerodynamic characteristics of such vehicles. © 1997 by John Wiley & Sons, Ltd.

Int. J. Numer. Meth. Fluids, **24**: 1321–1340, 1997

No. of Figures: 21. No. of Tables: 1. No. of References: 38.

KEY WORDS: parallel flow simulation; complex geometries; mesh generation; automobile

1. INTRODUCTION

One of the important aspects of the finite element method is the ability to model problems with complex domains. Domains with any geometry can be handled provided that a finite element mesh can be created which accurately represents the domain. To accurately represent a domain, the mesh must have sufficient resolution on the boundary to accurately represent the solid surfaces, interfaces or free surfaces, and must have sufficient resolution in the interior to model the fluid dynamics to a required degree of accuracy. In most cases such a finite element mesh involves a large number of elements and nodes and requires a high-performance computing (HPC) platform to obtain the solution in a reasonable amount of time. Both these features of a simulation (HPC power and complex geometries) are required to solve most engineering applications.

Our implementation of the finite element method to solve fluid dynamics applications on parallel architectures has been in place for some time.[1–4] The implementation uses the data-parallel programming model on the Thinking Machines CM-5 and the message-passing programming model on the CRAY T3D. The message-passing implementation is based on the Parallel Virtual Machine

* Correspondence to: T.E. Tezduyar, Aerospace Engineering and Mechanics, Army HPC Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, MN 55415, U.S.A.

CCC 0271–2091/97/121321–20 $17.50

Received 17 January 1996
Revised 16 February 1996

(PVM) libraries[5] and is portable across architectures. We also have shared-memory implementations on Silicon Graphics multiprocessor architectures. The implementations are based on the units of parallelism of elements and nodes. Each group of elements (and each group of nodes) is assigned to a particular processor which is responsible for the computations required by this group. When data exchanges are required between the element and nodal parallel levels, a two-step gather and scatter operation[6,7] is used to facilitate the communication between the processors. To optimize this communication, we employ mesh-partitioning techniques[6,8] to reduce the amount of data that is sent through the interconnect network of the parallel architecture.

Using these parallel implementations, we have been able to solve many applications with a large number of unknowns, typically at around 10 Gflops.[7,9] One application computed on the CM-5 was for supersonic flow past a delta-wing[10] and involved over five million degrees of freedom. A recent parachute flow computation performed on the T3D involved more than 38 million equations.[11,12] We must mention here that these are coupled non-linear equations which are solved using iterative solution techniques.

As these numerical simulations become larger and are applied to complex, real-world problems, the mesh generation and management phases become more and more important. To model the complex geometries encountered in these applications, we developed automatic mesh generation software. The automatic mesh generator makes few or no assumptions on the shape of the domain; because of this, almost any geometry can be modelled. By automating this process of mesh generation, the sometimes enormous task of creating a special mesh generator for a specific application is eliminated. We also significantly decrease the time it takes between the conception of a problem and the actual numerical simulation of the fluid flow. Our package includes all the steps in the mesh generation process such as 3D modelling, surface mesh generation and 3D volumetric mesh generation.

There are several popular automatic mesh generation procedures, including the advancing front method[13,14] and the finite octree method.[15] We are using a Delaunay–Voronoï-type method.[13,16,17] Delaunay-type mesh generators are the most general and create high-quality meshes. A Delaunay-type mesh is one where each element's circumsphere contains no other nodes in the mesh. This basic property leads to the development of nodal insertion algorithms where a new node is added to an existing Delaunay-type mesh such that the new mesh is also of Delaunay type. We use an edge-swapping nodal insertion algorithm[17,18] which is then used within a general automatic mesh generation procedure. This general procedure involves many aspects such as boundary mesh generation and integrity, internal node generation, and refinement control. Along with the 3D automatic mesh generator, methods are developed to model the geometric objects. These include a computer-aided design program based on Bézier surfaces[19] and an automatic surface mesh generator which applies 2D automatic mesh generation techniques for the discretization of the geometric model. Both these programs are needed in any fully automatic mesh generation system.

This mesh generation software has been used in many of our fluid dynamics applications involving complicated geometries. These applications include supersonic flow past a fighter aircraft,[4] flow around a submarine,[12] flow through channels and spillways,[12] contaminant dispersion in a subway station and around military vehicles,[20] flow past hypersonic re-entry vehicles[21] and flow past round and ram-air parachutes.[4] In this paper we highlight the use of our automatic mesh generation software for flow past an automobile and the simulation of multiple spheres falling in a liquid-filled tube.

In Section 2 we present the semidiscrete, stabilized finite element formulation for the Navier–Stokes equations of incompressible flows. In Section 3 we highlight some of the features of our implementation of the finite element method on parallel architectures under both the data-parallel and message-passing programming models. In Section 4 we describe our automatic mesh generation procedures, including geometric modelling, surface mesh generation and 3D volumetric mesh

generation. In Section 5 we present two examples involving complex geometries. Some concluding remarks are provided in Section 6.

2. GOVERNING EQUATIONS AND NUMERICAL FORMULATION

The governing equations are the Navier–Stokes equations of viscous, incompressible flows. The fluid occupies, at time instant $t \in (0, T)$, a bounded region $\Omega \subset \mathbb{R}^{n_{\mathrm{sd}}}$ which has boundary Γ, where n_{sd} is the number of spatial dimensions. The primary degrees of freedom are velocity and pressure, denoted by $\mathbf{u}(\mathbf{x}, t)$ and $p(\mathbf{x}, t)$. The Navier–Stokes equations represent the balance of momentum and the incompressibility constraint:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{u} - \mathbf{f}\right) - \nabla \cdot \boldsymbol{\sigma} = \mathbf{0} \quad \text{on } \Omega, \quad \forall t \in (0, T), \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0 \quad \text{on } \Omega, \quad \forall t \in (0, T), \tag{2}$$

where ρ is the density of the fluid, $\boldsymbol{\sigma}$ is the stress tensor and $\mathbf{f}(\mathbf{x}, t)$ is a body force per unit mass. The stress tensor $\boldsymbol{\sigma}$ is defined as

$$\boldsymbol{\sigma}(p, \mathbf{u}) = -p\mathbf{I} + 2\mu\boldsymbol{\varepsilon}(\mathbf{u}), \qquad \boldsymbol{\varepsilon}(\mathbf{u}) = \tfrac{1}{2}[\nabla \mathbf{u} + (\nabla \mathbf{u})^{\mathrm{T}}], \tag{3}$$

where μ is the viscosity and $\mathbf{I}$ is the identity tensor.

The Dirichlet- and Neumann-type boundary conditions are given as

$$\mathbf{u} \cdot \mathbf{e}_i = g_i \quad \text{on } (\Gamma)_{g_i}, \quad i = 1, \ldots, n_{\mathrm{sd}}, \tag{4}$$

$$\mathbf{n} \cdot \boldsymbol{\sigma} \cdot \mathbf{e}_i = h_i \quad \text{on } (\Gamma)_{h_i}, \quad i = 1, \ldots, n_{\mathrm{sd}}, \tag{5}$$

where $\mathbf{e}_i$ is the Cartesian unit vector corresponding to axis i, $\mathbf{n}$ is the unit normal vector for boundary Γ, and $(\Gamma)_{g_i}$ and $(\Gamma)_{h_i}$ are complementary subsets of boundary Γ as related to the Dirichlet- and Neumann-type boundary conditions. A divergence-free velocity field is also needed as an initial condition:

$$\mathbf{u}(\mathbf{x}, t) = \mathbf{u}_0 \quad \text{on } \Omega. \tag{6}$$

The spatial domain is discretized using the finite element method, and integration in time is achieved with finite difference approximations. We partition the time interval $(0, T)$ into discrete time levels t_n, where t_n belongs to an ordered series of time steps $0 = t_0 < t_1 < \ldots < t_N = T$. For each time level n, we define the following finite element interpolation spaces for velocity and pressure:

$$(S_{\mathbf{u}}^h)_n = \left\{ \mathbf{u}^h = \left[u_i^h\right]_{i=1}^{n_{\mathrm{sd}}} \middle| u_i^h \in H^{1h}(\Omega), u_i^h \doteq g_i^h \text{ on } (\Gamma)_{g_i}, \forall i = 1, \ldots, n_{\mathrm{sd}} \right\}, \tag{7}$$

$$(V_{\mathbf{u}}^h)_n = \left\{ \mathbf{w}^h = \left[w_i^h\right]_{i=1}^{n_{\mathrm{sd}}} \middle| w_i^h \in H^{1h}(\Omega), w_i^h \doteq 0 \text{ on } (\Gamma)_{g_i}, \forall i = 1, \ldots, n_{\mathrm{sd}} \right\}, \tag{8}$$

$$(S_p^h)_n = (V_p^h)_n = \left\{ p^h | p^h \in H^{1h}(\Omega) \right\}. \tag{9}$$

Here $H^{1h}(\Omega)$ represents the finite-dimensional function space over the spatial domain Ω. Over each element domain this space is constructed using first-order polynomials.

The semidiscrete, stabilized formulation[22] can be written as follows: find $\mathbf{u}^h \in (S_{\mathbf{u}}^h)_n$ and $p^h \in (S_p^h)_n$ such that $\forall \mathbf{w}^h \in (V_{\mathbf{u}}^h)_n$ and $\forall q^h \in (V_p^h)_n$

$$\begin{aligned}\int_\Omega \mathbf{w}^h \cdot \rho\left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f}^h\right) \mathrm{d}\Omega + \int_\Omega \boldsymbol{\varepsilon}(\mathbf{w}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\, \mathrm{d}\Omega - \int_{\gamma_{\mathbf{h}}} \mathbf{w}^h \cdot \mathbf{h}^h \mathrm{d}\Gamma + \int_\Omega q^h \nabla \cdot \mathbf{u}^h \mathrm{d}\Omega \\ + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \frac{\tau}{\rho}[\rho \mathbf{u}^h \cdot \nabla \mathbf{w}^h - \nabla \cdot \boldsymbol{\sigma}(q^h, \mathbf{w}^h)] \\ \cdot \left[\rho\left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{u}^h - \mathbf{f}^h\right) - \nabla \cdot \boldsymbol{\sigma}(p^h, \mathbf{u}^h)\right] \mathrm{d}\Omega \\ + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \delta \nabla \cdot \mathbf{w}^h \rho \nabla \cdot \mathbf{u}^h \mathrm{d}\Omega = 0, \end{aligned} \tag{10}$$

where n_{el} is the number of elements in the mesh. This process is applied sequentially to all time levels.

The first four terms in equation (10) constitute the standard Galerkin formulation, while the fifth and sixth terms are the stabilization terms. The definitions of τ and δ can be found in References 7, 23 and 24. The addition of the stabilizing terms does not compromise the consistency of the formulation, since these terms are weighted with the residuals of the momentum and mass balance equations, which vanish for exact solutions.

The time integration is carried out using finite difference approximations:

$$\frac{\partial \mathbf{u}^h}{\partial t} \doteq \frac{\mathbf{u}_{n+1}^h - \mathbf{u}_n^h}{\Delta t}, \tag{11}$$

$$\mathbf{u}^h \doteq (1-\alpha)\mathbf{u}_n^h + \alpha \mathbf{u}_{n+1}^h, \tag{12}$$

$$\mathbf{f}^h \doteq (1-\alpha)\mathbf{f}_n^h + \alpha \mathbf{f}_{n+1}^h. \tag{13}$$

In proceeding with the semidiscrete solution technique, $\mathbf{u}_n^h$ is given and the only unknowns are at time level $n+1$. The computations start with

$$\mathbf{u}_n^h \doteq \mathbf{u}_0 \quad \text{for } n = 0. \tag{14}$$

With the proper choice of α, the integration in time can be based on forward ($\alpha = 0{\cdot}0$), central ($\alpha = 0{\cdot}5$) or backward ($\alpha = 1{\cdot}0$) differencing.

3. PARALLEL IMPLEMENTATION

We implemented this finite element formulation on parallel platforms based on two different programming models. We have a data-parallel implementation on the Thinking Machines CM-5[2] and a message-passing implementation on the CRAY T3D.[4] Both architectures can operate in either the data-parallel or message-passing programming models, but we use the data-parallel model for the CM-5 owing to its simplicity and the availability of advanced scientific software libraries and we use the message-passing model for the T3D owing to its greater portability between architectures.

Briefly, the data-parallel model assumes that the parallel data elements are each assigned to a different processor (if there are more data elements than processors, so-called virtual processors will be created) and each processor performs the same set of computations, synchronized with the others, on its own piece of data. In such a programming model the individual processors are somewhat transparent to the user and communications between them can be handled by simple array addresses to other parallel data elements. The message-passing model assumes that the processors are acting individually. They each run the same program but may be performing different tasks than the

others at different times. Communications are accomplished by explicit send and receive functions from one individual processor to another. In such a programming model the individual processors are not transparent and the fact that there are different processors that may be doing different things must be taken into account when programming. Although the programming models we use on the two platforms are different, they have some similarities.

On both models we have two levels of data: element and nodal levels. An equation level can be incorporated along with the nodal level. In the data-parallel model each element (and each node) is assigned to a (virtual) processor. When computations take place that involve only that parallel data unit (like an element), the computations take place without any need for communication. The assignment of elements to processors may be optimized by using Thinking Machines Scientific Software Libraries (CMSSL) routines[25] that cluster the elements into groups (or partitions) that are spatially close together and placing each element in that group on the same physical processor. In the message-passing model we explicitly assign elements and nodes to the individual processors. For proper load balancing, each processor will contain, as closely as possible, an equal number of these data elements. After all the elements and nodes are assigned, each individual processor is responsible for performing computations required by its individual data elements. Again the assignment of elements to the individual processors may be optimized by placing elements in a partition which are spatially close together on the same processor; by doing so, each processor is responsible for a particular piece of the finite element mesh.

Communication between processors in the data-parallel model may be achieved by simply addressing a data element that exists on a separate (virtual) processor. This may not be the most optimal way of performing communications though, so in places where we need to transfer data between processors many times (as in the GMRES iterative solver), we use efficient CMSSL routines to transfer data between the element and nodal levels. These are the gather and scatter routines.[6,7] These routines are optimized by using a two-step gather and scatter method where transfers between the element and nodal levels are facilitated by an intermediate, partition (processor)-level nodal vector. The CMSSL routines save the communication paths for the data elements and these paths do not need to be recomputed for repetitive data transfers. Also, the CMSSL library distributes the global nodes on the processors such that, as much as possible, the global nodes lie on the same processor as the corresponding nodes in the partition-level nodal vector. By doing this, when the partition-level nodal vector is sent to or received from the global nodal vector, the amount of data that is accessed off-processor is reduced.

In the message-passing model on the T3D no such communication libraries exist, so we created our own routines which perform these two-step gather and scatter functions. The basic send and receive routines on the T3D that perform the step of transferring data from one processor to another are accommodated by the Parallel Virtual Machine (PVM) libraries.[5] For optimal performance we use the PVM Channels routines,[26] which are CRAY-specific extensions of the standard PVM routines. The PVM Channels routines are the fastest possible means of sending data from one processor to another (while still using PVM) and we can store the communication paths for the data elements, so they need to be computed only once. We developed libraries based on the PVM Channels to perform all steps in the two-step gather and scatter method. We also developed libraries to assign global nodes to processors which minimize as much as possible the off-processor communications on the CRAY T3D. For mesh partitioning we use either those routines available in CMSSL or the Metis package.[8]

4. AUTOMATIC MESH GENERATION

An automatic mesh generator, by definition, uses some sort of boundary data alone as input and then creates the interior nodes and elements automatically based on this boundary information. Since there

are few or no assumptions made about the boundary within the mesh generator, almost any geometry can be modelled with such mesh generators.

The boundary definition we use in our package is a surface mesh composed of triangular elements. This surface mesh, whose construction is discussed later in this section, defines a surface which fully encloses the desired domain, and there could be holes within this domain with the definition of more than one enclosed surface mesh. The refinement in the interior of the domain in the final 3D mesh is dependent upon the refinement on this surface mesh; because of this, we allow other interface meshes in the input that exist just to specify a desired refinement at a location within the interior of the domain. These other interface meshes will not be a part of any boundary in the final 3D mesh.

The methods we use to perform automatic mesh generation are those based on Delaunay–Voronoï methods.[13,16,17] This means that the method is designed to create a Delaunay-type mesh. These methods are probably the most general type of automatic mesh generator and generate high-quality meshes. A Delaunay-type mesh has several features.[13] One of the most important features of a Delaunay mesh can be stated as: for each element in the mesh, its circumcircle (or circumsphere in 3D) which encompasses all nodes defining that element contains no other nodes of the mesh. This feature of a Delaunay mesh governs the way the elements construct themselves for a given set of nodes.

The Delaunay–Voronoï methods make use of node insertion algorithms. Given a Delaunay-type mesh composed of triangles in 2D or tetrahedra in 3D, we can insert a new node into this existing mesh such that the resulting mesh is of Delaunay type. When each new node is placed in the mesh, the elements around this node will rearrange themselves so as to meet this Delaunay criterion. In the mesh generation process, nodes will be inserted into the mesh one-by-one until the entire mesh satisfies a given quality criterion.

We use an edge-swapping algorithm[17,18,27] for the process of nodal insertion. The edge-swapping algorithm makes use of alternative element configurations to allow the old mesh to accommodate a new node so as to meet the Delaunay criterion. When a new node is inserted into an existing mesh, an advancing front of rearranging element configurations is swept out until the mesh again satisfies the Delaunay criterion. More information about our implementation of this algorithm can be found in Reference 7. We find this algorithm to be superior to other such nodal insertion algorithms owing to the greater amount of control we have over the process of the elements rearranging themselves.

The edge-swapping nodal insertion algorithm is simply a method to insert one node into an existing mesh and forms the basis for our general mesh generation package. The steps we use to construct a Delaunay-type mesh using this nodal insertion algorithm are listed below.

1. Generate a mesh of a parallelepiped box with eight nodes and five tetrahedral elements which encloses all the nodes within the input surface mesh. This mesh will be the starting mesh of the algorithm and any nodal insertion later will be modification of this mesh.
2. Using the nodal insertion algorithm, insert all the boundary nodes one-by-one into the existing mesh.
3. Remove all elements, and the eight nodes from Step 1, which are on the exterior of the desired domain. The exterior portions of the mesh are those not within the enclosed geometry defined by the input surface mesh.
4. Insert new nodes into the remaining interior portions of the mesh until the mesh satisfies some quality criterion.

We now provide some details for Steps 2–4.

Step 2

The final mesh must reconstruct the desired geometry of the objects being modelled. This means that the surface of the resulting 3D mesh must match the input surface mesh. This property may not

exist in a mesh if the nodes of the surface mesh are inserted in any order without taking into account the elements of the surface mesh. In our implementation we insert the boundary nodes into the mesh in such a way that the elements of the surface mesh do exist within the 3D mesh (a triangular surface element exists within the 3D mesh if it matches a face of a tetrahedral element).

Our boundary node insertion algorithm goes as follows. We first reorder the input surface elements in such a way that by going through the list from the first element to the last, an advancing front of elements is swept out across the model surface. We then begin a loop through the list of surface elements, and if any node of a surface element is not inserted into the 3D mesh, we insert that node. Once all this surface element's nodes are in the 3D mesh, we check to see whether the surface element exists within the 3D mesh, and if so, we tag the face within the 3D mesh that represents this surface element. Through a modification of the edge-swapping nodal insertion algorithm we do not allow these tagged faces to be altered in any way during the remainder of the boundary node insertion process. By altering the nodal insertion algorithm in this way, the triangulation (at least of just the boundary nodes) is called a constrained Delaunay triangulation.

Once all boundary nodes are inserted into the 3D mesh, we determine whether any surface elements are not represented within the 3D mesh. If so, there will be holes in the representation of the surface mesh within the 3D mesh. In practice we have seen that only a very small number of the surface elements are not represented within the 3D mesh at this step in the algorithm. We assumed that the discretization of the surface mesh was fine enough to represent the geometry accurately, so we look for other faces in the 3D mesh which will close these holes. We then mark these other faces as being a representation of our surface mesh.

There have been several other proposed schemes to force the automatic mesh generator to reconstruct the desired boundary within the 3D mesh. George *et al.*[16] advocate just inserting the boundary nodes in the usual manner and then later rearranging the tetrahedral elements so as to recreate the surface mesh within the 3D mesh. We find this implementation complicated. Our method seems to be fairly robust and fails to represent the surface elements in only a few cases. In such cases an increase in the refinement in the problem areas of the model usually fixes the problem. A future implementation may incorporate some ideas from Reference 16 to reconstruct the elements in these problem areas.

Step 3

Since we assured that the surface mesh has been reconstructed within the 3D mesh composed of the boundary points in Step 2, we can remove all the elements on the exterior sides of this representative surface mesh. We now have a valid 3D tetrahedral element mesh with the proper boundary representation.

Step 4

The mesh composed of just boundary nodes needs to be completed by insertion of interior nodes. Before explaining this process, we need to define what we call the height function. The height function specifies the desired level of refinement in the interior of the mesh and in our implementation the height function specifies the desired edge length of the elements. The desired edge length at the boundary nodes is taken from the refinement of the surface mesh, and since the mesh composed of just boundary nodes is a valid finite element mesh, we use the function space defined by the existing elements to specify the height function throughout the domain. Since we use linear basis functions, the refinement in the interior is linearly dependent on the refinement of the surface mesh.

With the definition of the height function we can now compare the ratio between the actual refinement and the desired refinement for each element. We find the element with the largest ratio between the actual and specified refinements, and if this ratio is larger than some threshold value, we insert a new node at the centre of this element's circumsphere. The insertion of the new node at the centre of the circumsphere works well, since the location of the new node is equidistant from each of the nodes making up this element and the distance to any other node in the mesh is larger than this length owing to the Delaunay criterion. This location of a new node is also recommended in Reference 17. We continue this process until the ratio between the actual and specified refinements is below the threshold value for each element within the mesh.

In our mesh generation package we have the option of generating layers of thin, semistructured elements around boundaries so as to better model boundary layer features of the flow. We create these boundary layer elements and nodes between Steps 3 and 4 of the mesh generation procedure, when the 3D mesh containing just the boundary nodes has been created but no internal nodes have been generated.

At each node of the input surface mesh there exists a unit normal vector pointing into the desired domain. We generate a series of nodes along each normal vector. The spacing and number of these nodes are defined by the desired thickness and number of layers. Once all boundary layer nodes are defined, we check each one to see whether it is too close to an existing boundary node, in which case it is eliminated, or to another boundary layer node, in which case the two close boundary layer nodes are merged. We then insert each remaining boundary layer node into the 3D mesh using the nodal insertion algorithm. Once all boundary layer nodes are within the 3D mesh, thin boundary layer elements will have been constructed with these new nodes owing to the Delaunay property of the mesh. We then tag these thin boundary layer elements and do not let these tagged elements be altered in any way during the process of generating the internal nodes. If we did not do this, the structure of the boundary layer elements might be lost owing to the creation of a close internal node.

As an example of our 3D mesh generation program, a view of a cross-sectional slice of a tetrahedral element mesh can be seen in Figure 1. Notice in Figure 1 that we created a rectangular interface in the interior of the mesh just to specify a refinement level at this location. By doing this, we can concentrate the smaller elements in a region around the object. A view of the height function at this same cross-section can be seen in Figure 2, where the contours of the desired edge length are plotted. The effect of this refinement interface can clearly be seen in this figure. A different cross-sectional view of this mesh is shown in Figure 3, where the thin boundary layer elements can more clearly be seen.

Along with the program to generate the actual 3D tetrahedral element mesh, we have several other programs which are used in the mesh generation process. The most important of these programs are our interactive geometric modelling program and the automatic surface mesh generator.

Geometric modeller

The geometric model definition for our 3D objects is composed of quadrilateral and triangular Bézier surfaces.[19] This method of geometric modelling is very flexible and most shapes can be represented as a patchwork of these types of surfaces. Also, since these surfaces are parametric ones, $\mathbf{X} = \mathbf{X}(u, v)$, where u and v are the parameters, the discretization of these surfaces will be much easier to perform. We also include in the geometric model the desired refinement of the 3D mesh at each corner of the Bézier surfaces.

We developed an interactive graphic modelling program to help the user to create the sometimes very complex models. The program also has many features which do a lot of the work

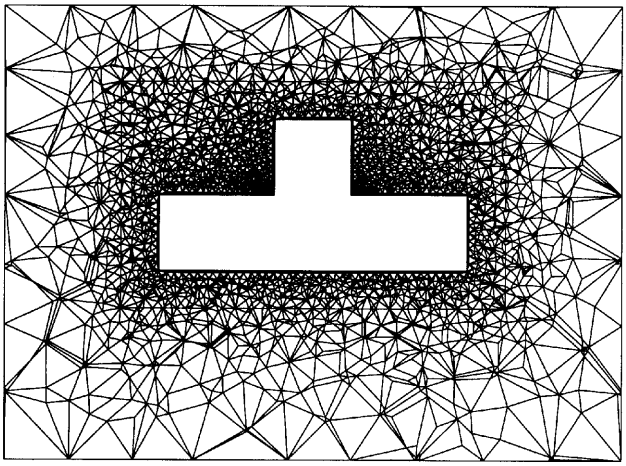


Figure 1. Cross-sectional view of example 3D tetrahedral element mesh

for the user while building models. A view of the control points and Bézier edges from the modelling program for the earlier example model can be seen in Figure 4, and a view of the Bézier surfaces can be seen in Figure 5. The structured mesh representing the surfaces in Figure 5 are just for display purposes and have nothing to do with the final surface mesh of this model.

Automatic surface mesh generation

Once the geometric model is built, we perform automatic mesh generation on each Bézier surface so as to create a surface mesh composed of triangular elements. The process of automatic mesh generation on a Bézier parametric surface is very similar to a 2D implementation of our 3D mesh generation procedure. We still use an edge-swapping nodal insertion algorithm, but in 2D. The mesh is stored in the parametric space (u, v) while it is being constructed, but the decisions made about when to swap elements in the edge-swapping algorithm are made in 3D space (the mesh in the

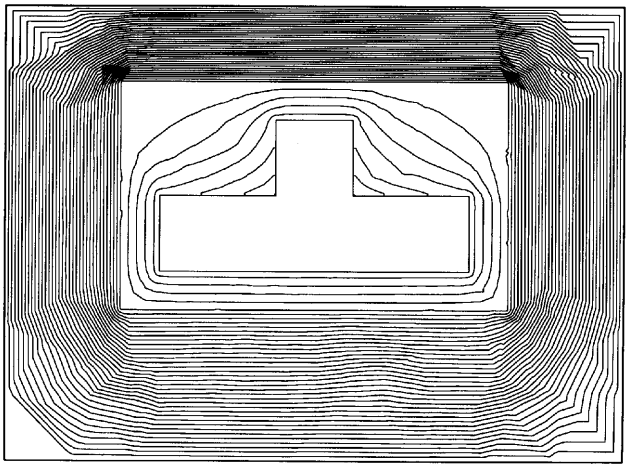


Figure 2. Cross-sectional view of height function which represents desired edge length in interior of domain

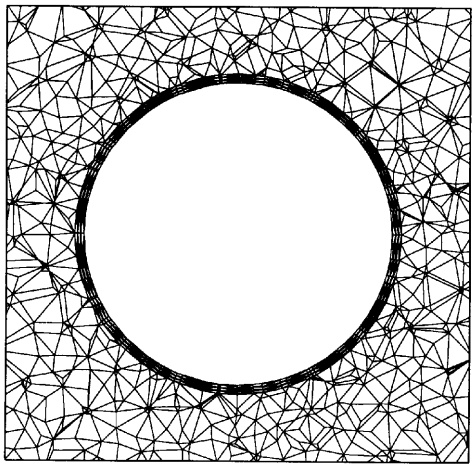


Figure 3. Closer cross-sectional view of example 3D tetrahedral element mesh

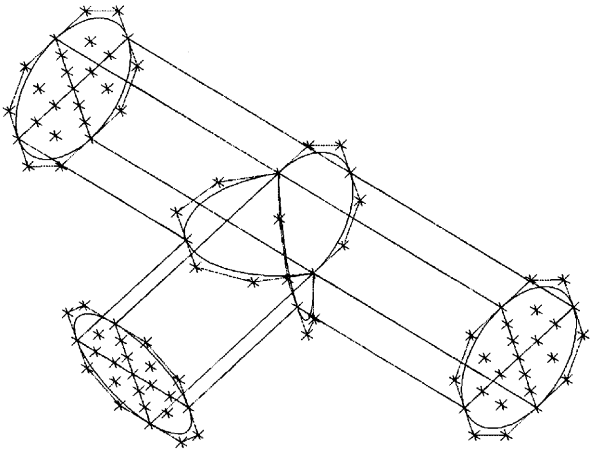


Figure 4. Control points and Bézier edges as viewed from modelling program

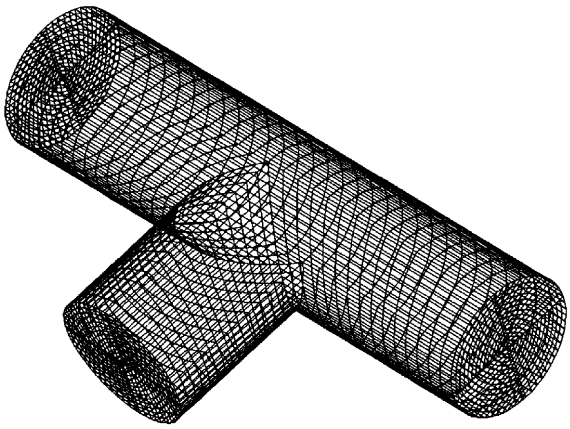


Figure 5. Bézier surfaces as viewed from modelling program

parametric space mapped into 3D). The generation of new nodes in the interior of the surface mesh is the same as in the 3D mesh generation procedure, but since a circumcircle is not clearly defined on a parametric surface, we use an iterative algorithm to find a point on the Bézier surface that is equidistant from each node comprising an element. As in the 3D mesh generation procedure, the refinement on this surface mesh is based on the refinement at the corners of the Bézier surfaces that was specified in the geometric model. More details about this surface mesh generation procedure can be found in Reference 7.

This method for generating a surface mesh seems to be fairly robust and fails for only some special cases. In our experience the method only fails for highly distorted Bézier surfaces. In such cases the geometric model will have to be slightly modified to create better-shaped Bézier surfaces. We believe that with some improvement to the algorithm, the method will be fully robust. The surface mesh generated for the example model in this section can be seen in Figure 6. Notice in the surface mesh of Figure 6 that we specified a higher refinement level at the intersection of the two components than at the ends of the model sections.

5. EXAMPLES

Airflow past an automobile

In this problem, airflow past an automobile traveling at 55 mph is simulated. The automobile is modelled after a Saturn SL2 and the model is quite detailed. The model generated with our interactive modelling program has 292 Bézier surface and contains wheels, rear-view mirrors, recessed headlights and a spoiler (see Figure 7). Since the geometry is symmetric, only half of the automobile is modelled. The surface mesh for our model contains 35,307 nodes and 70,937 triangular elements. The automatic mesh generator created a 3D mesh with 227,135 nodes and 1,407,579 tetrahedral elements. We then reflected the mesh to carry out the flow simulation for the full automobile. This final mesh contains 448,695 nodes and 2,815,158 tetrahedral elements.

A view of the surface of the 3D mesh can be seen in Figure 8. A 2D slice of the 3D mesh at a section cutting both wheels can be seen in Figure 9. Notice in Figure 9 that we created a more refined region within the 3D mesh to concentrate the smaller elements around the automobile and in the wake region. It is in these regions where the main flow features are located. Outside this refinement region,

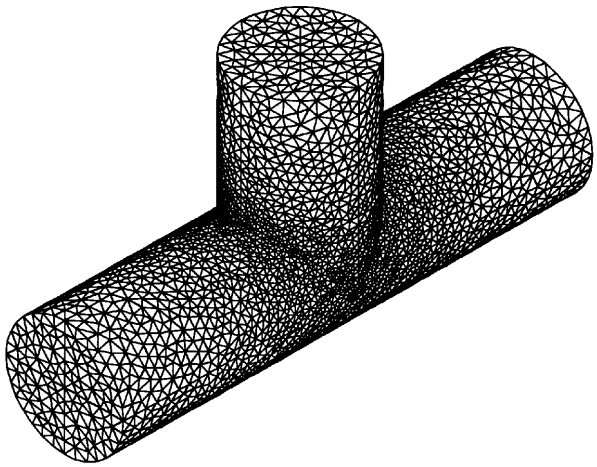


Figure 6. Surface mesh for example model

Figure 7. Front and rear views of automobile

larger elements are used to fill up the rest of the domain. Also notice in Figure 9 that we created three very thin layers of elements around the automobile and wheels. This is done so as to capture the boundary layer features of the flow more accurately. A closer view of these boundary layer elements can be seen in Figure 10.

At 55 mph ($Re = 6{\cdot}9 \times 10^6$ based on the total automobile length) the flow around the automobile is assumed to be incompressible and turbulent, so the time-averaged, incompressible Navier–Stokes equations are solved. We use a Smagorinsky turbulence model in this simulation.[28–30] This is a zero-equation model where the viscosity coefficient is augmented with an eddy viscosity coefficient ν_t so as to model the unresolved (subgrid) scales in the flow. The eddy viscosity is given by

$$\nu_t = C_s h_e^2 (\varepsilon_{ij}\varepsilon_{ij})^{1/2}, \tag{15}$$

where C_s is a model constant (a value of 0·1 was used) and h_e is an approximate measure of element length. For the efficient use of the computing resources a matrix-free version of our flow solver is

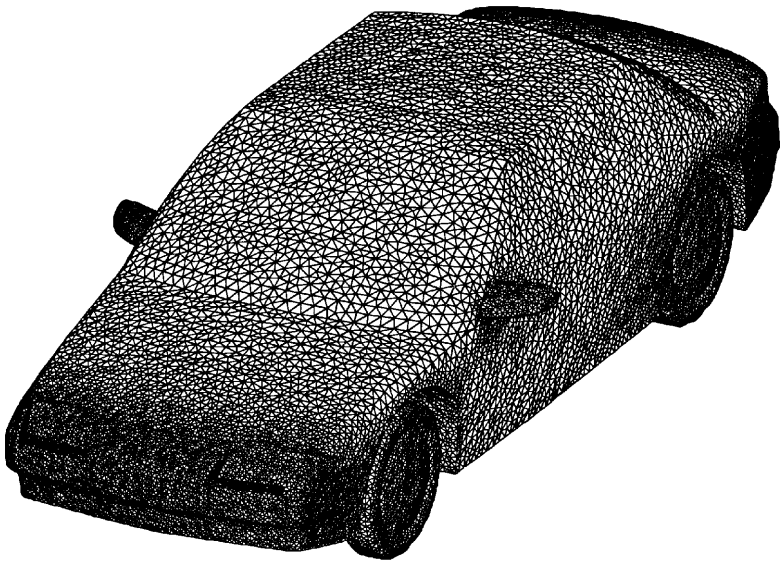


Figure 8. Surface of 3D tetrahedral element mesh for automobile (448,695 nodes and 2,815,158 elements)

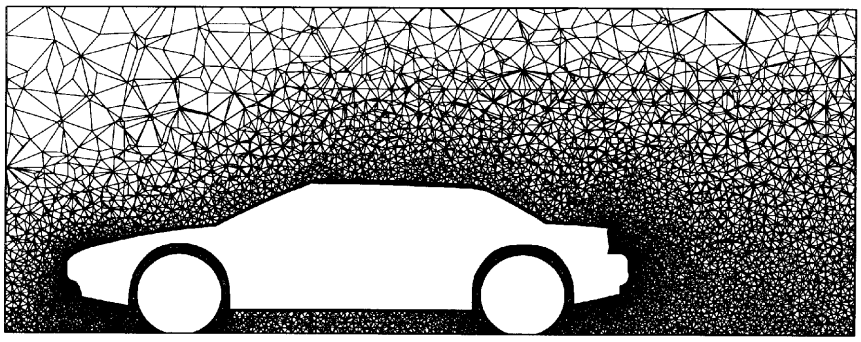


Figure 9. Cross-section of 3D tetrahedral element mesh for automobile at section cutting both wheels

used to obtain the solution. In the matrix-free implementation the left-hand-side matrix is never stored, but its influence within the iterative solver, through a matrix multiplication, is recomputed directly as needed.

The simulation is carried out under two flow conditions. One simulation models the actual road conditions of the automobile. This means that the wheels are spinning and the freestream velocity is imposed on the road as well as the inflow boundary. The second simulation models wind tunnel conditions. This means that the tires (as well as the autobody) have zero velocity and there are slip conditions imposed on the road. This second simulation is performed so as to better compare the drag coefficient with that measured in a wind tunnel. We also created a second mesh without rear-view mirrors so as to see the effect of this component on the overall drag on the automobile. Both meshes are similar in refinement and have almost the same number of nodes and elements. All images of the flow field are for the mesh with rear-view mirrors.

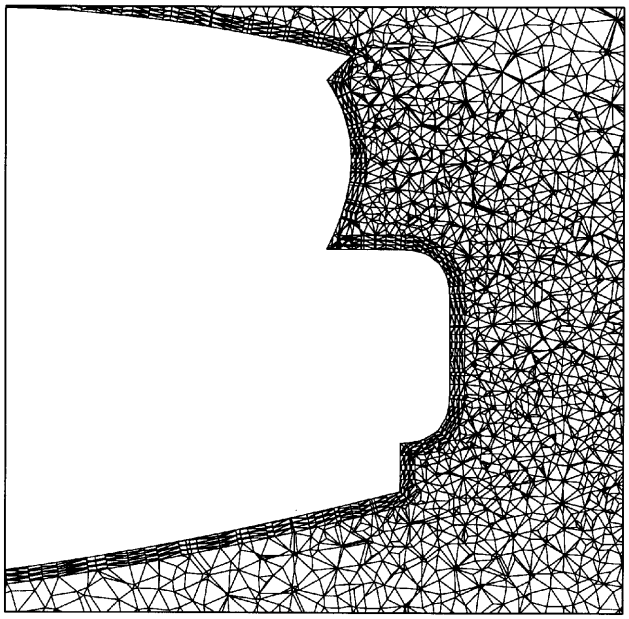


Figure 10. A close-up view of boundary layer elements in automobile mesh

Table I: Computed drag and lift coefficients

	Road conditions		Wind tunnel conditions	
	C_D	C_L	C_D	C_L
With mirror	0·455	0·290	0·354	0·162
Without mirror	0·448	0·306	0·327	0·171

The computed lift and drag coefficients for the automobile are given in Table I. For reference, the drag coefficient stated for a Saturn SL2 automobile is 0·343,[31] but it is important to remember that our computational model is only an approximation to a Saturn SL2. An interesting thing to note from Table I is the rather large increase in the lift and drag coefficients of the automobile under road conditions over those under wind tunnel conditions.

In Plates 1 and 2 can be seen the streamlines past the automobile under both flow conditions. Plate 1 is a top view and Plate 2 is a side view. In Plate 3 can be seen the rendering of pressure distribution on the surface of the automobile under road conditions.

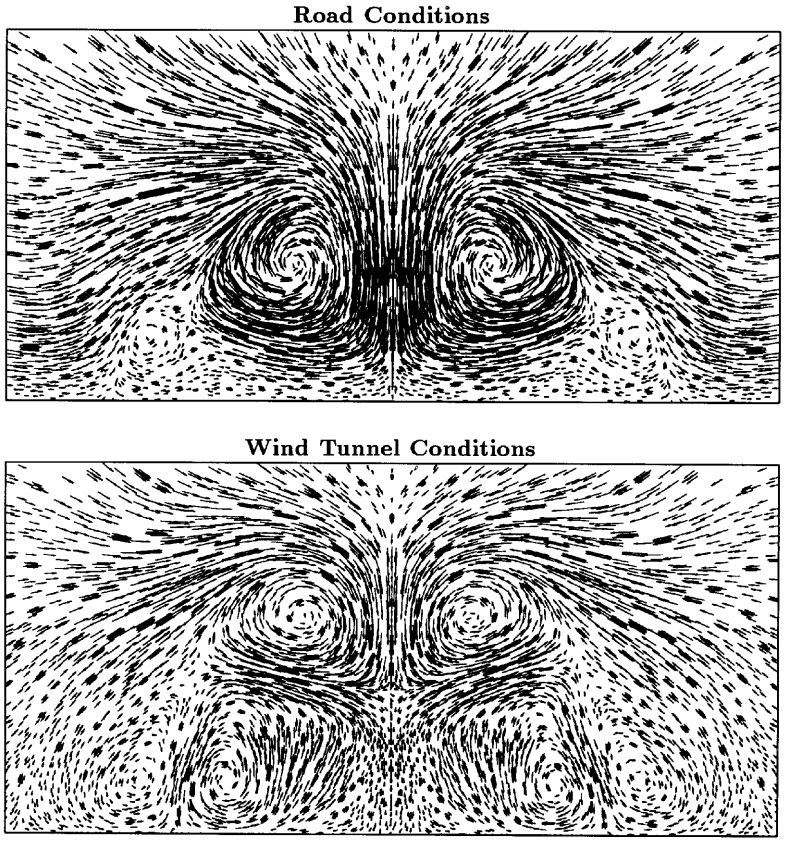


Figure 11. Velocity vectors at cross-section normal to incoming velocity at quarter car length behind automobile

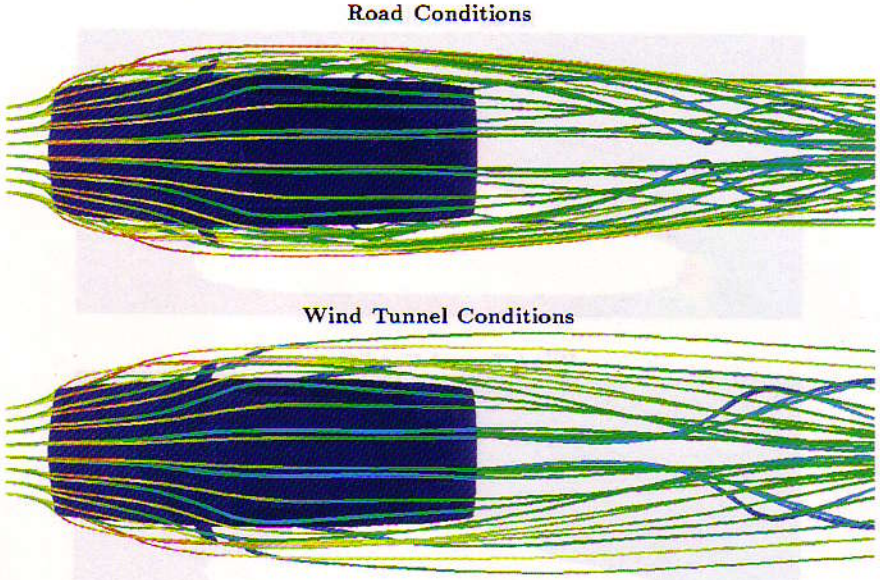


Plate 1. Top view of streamlines

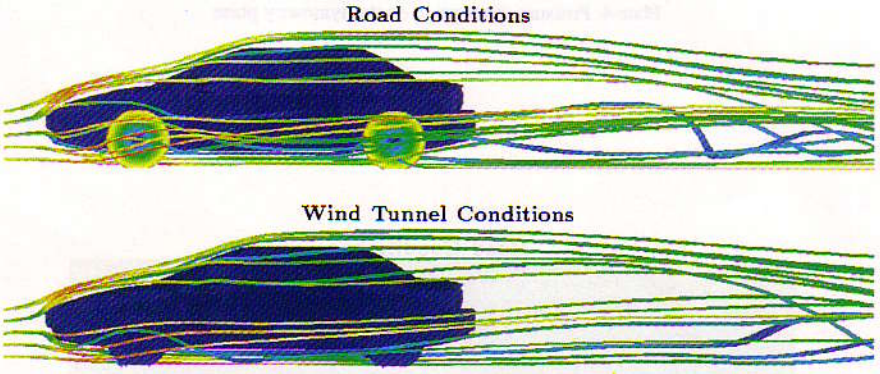


Plate 2. Side view of streamlines

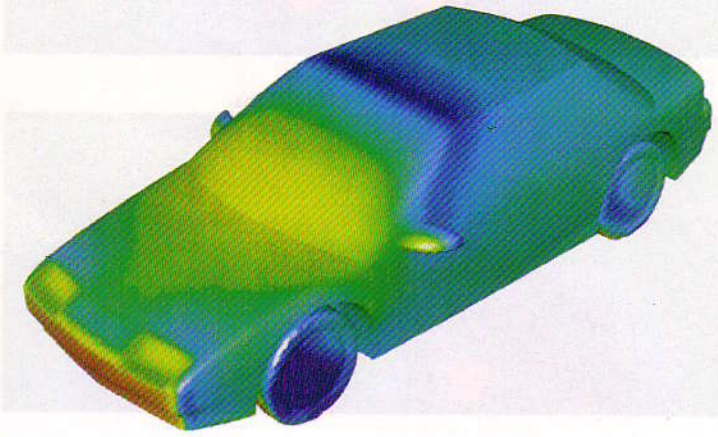
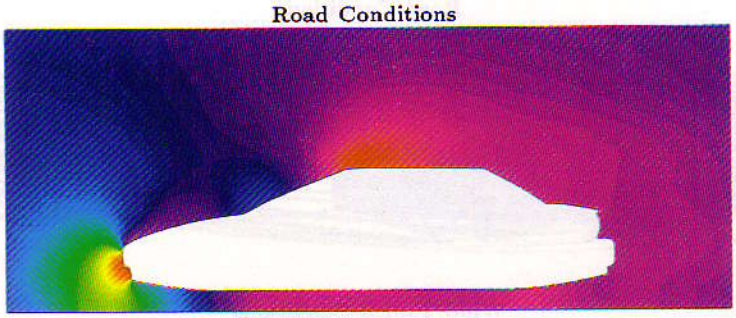


Plate 3. Rendering of the pressure distribution on the surface

Road Conditions



Wind Tunnel Conditions

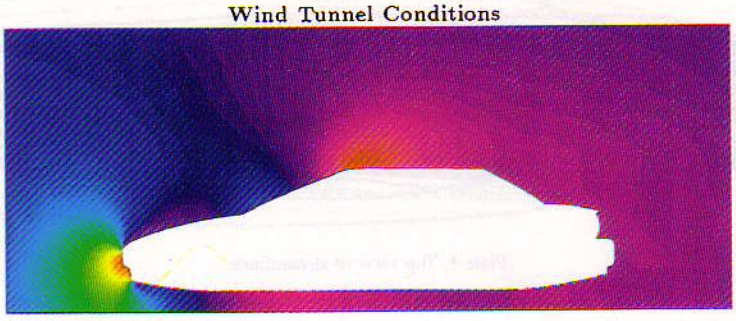
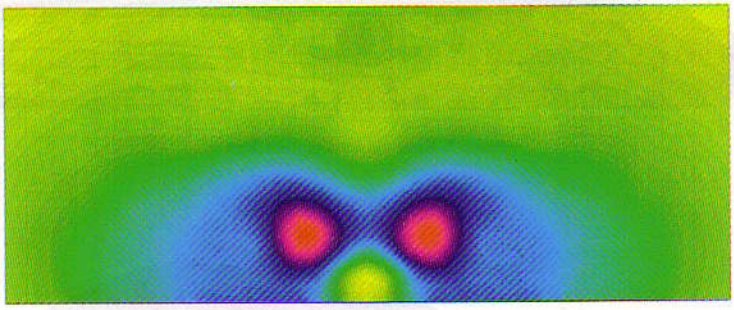


Plate 4. Pressure distribution on the symmetry plane

Road Conditions



Wind Tunnel Conditions

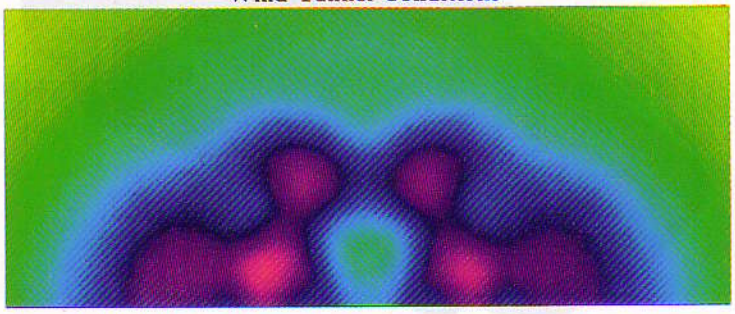


Plate 5. Pressure distribution at a cross section normal to the incoming velocity at $^1/_4$ car lengths behind the automobile

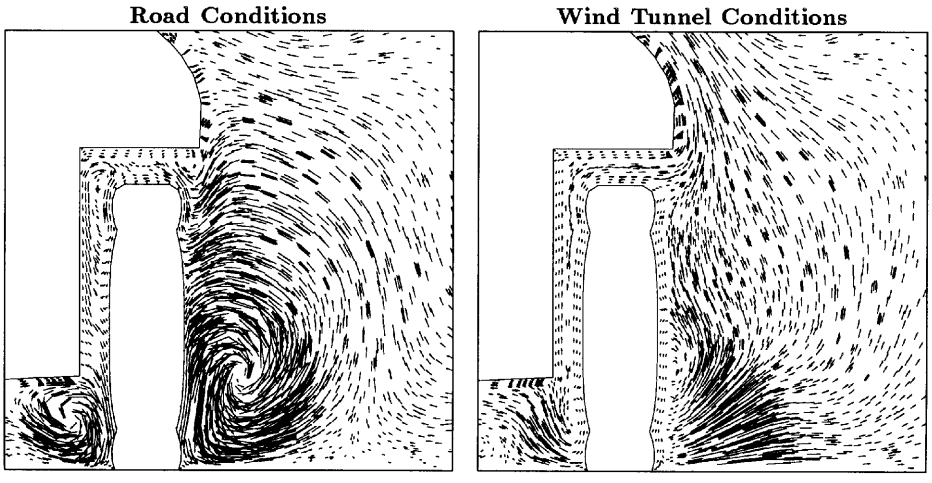


Figure 12. Close-up view of velocity vectors at cross-section normal to incoming velocity at location of rear wheels

The velocity vectors at a cross-section normal to the incoming velocity at a section a quarter of a car length behind the automobile under both flow conditions can be seen in Figure 11 and at a section at the location of the rear wheels under both flow conditions in Figure 12. The velocity vectors at a cross-section at a rear part of the automobile (behind the wheels) can be seen in Figure 13. The pressure distribution on the symmetry plane is shown in Plate 4. The pressure distribution at a cross-section perpendicular to the inflow velocity a quarter of a length behind the automobile is shown in Plate 5.

There are four main features of the flow field which can be observed within these figures and plates. These include the trailing vortices in the wake of the automobile, the flow patterns induced by the rotating wheels, the recirculation regions and the overall pressure field. All these features in the computed flow field compare qualitatively well with those observed by other researchers both experimentally and computationally.[30,32–35] A more detailed analysis of these automobile flow simulations can be found in Reference 7.

Multiple spheres falling in a liquid-filled tube

In this case we simulate multiple spheres falling in a liquid-filled tube. The spheres interact with the fluid forces and each other as they are allowed to rotate and translate governed by Newton's laws of motion. A space–time version[23,24] of the formulation given by equation (10) is used in this simulation. This problem has previously been described in References 7 and 36 and the use of the automatic mesh generator for this application will be highlighted here.

In this application there is the requirement that there may be any number of spheres at any location within the tube. Because of this requirement, the only option in discretizing the domain is with an automatic mesh generator. Throughout the simulation the spheres move and the motion of the mesh is accommodated using the automatic mesh moving scheme described in References 36 and 37. When the mesh gets too distorted, we remesh by generating an entirely new mesh with the automatic mesh generator and then project the solution from the old mesh on to the new one. By using the automatic mesh-moving scheme together with remeshing, we minimized the number of remeshes and there is no restriction on the type of motion each sphere is allowed.

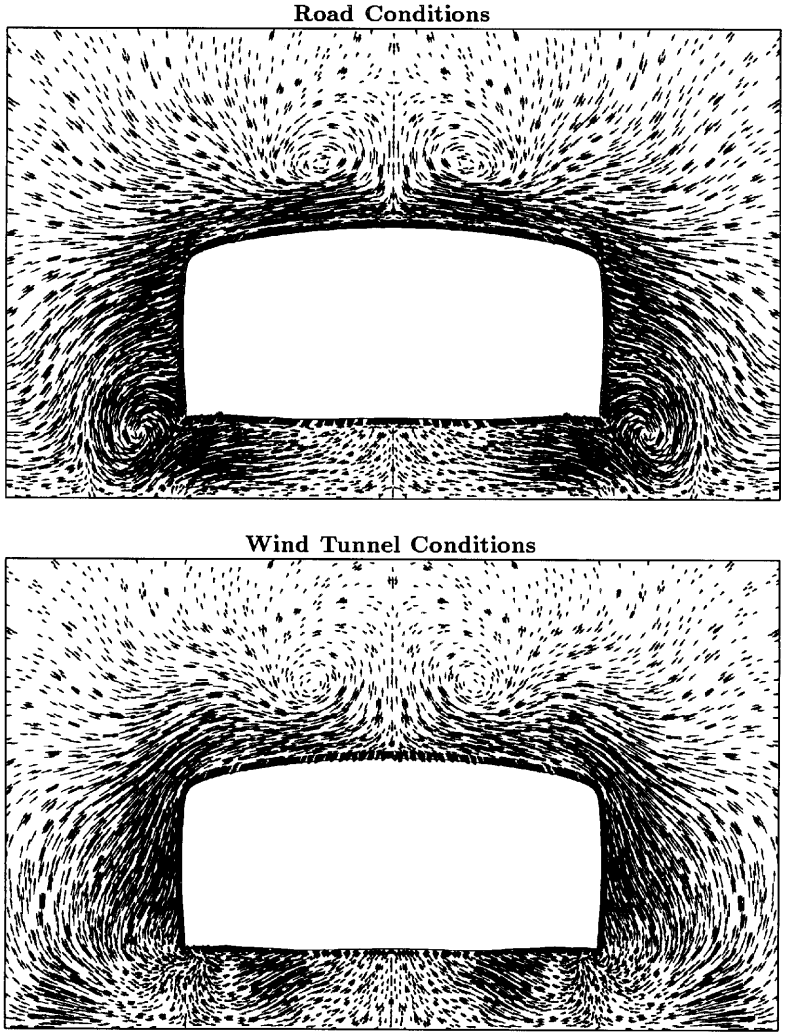


Figure 13. Velocity vectors at section normal to incoming velocity at section at rear part of automobile (behind wheels)

For this application where we need to run the automatic mesh generator many times (owing to repetitive remeshing), we created a special version of the mesh generator for this specific geometry. This program has a very simple input file format and concentrates the nodes in regions where mesh refinement is important, such as those close to the wake of each sphere. As with the general automatic mesh generator, this special version has the ability to create structured layers of elements around each sphere to better model the boundary layer features of the flow.

The first example involves two spheres falling together. The spheres are initially in a staggered configuration and throughout the simulation they exhibit the behaviour of drafting, kissing and tumbling reported in Reference 38. The sphere configurations along with a cross-section of the mesh at three instants during the simulation are shown in Figure 14. Notice in Figure 14 that the stretching of the mesh due to the automatic mesh-moving scheme can clearly be seen in the mesh cross-sections.

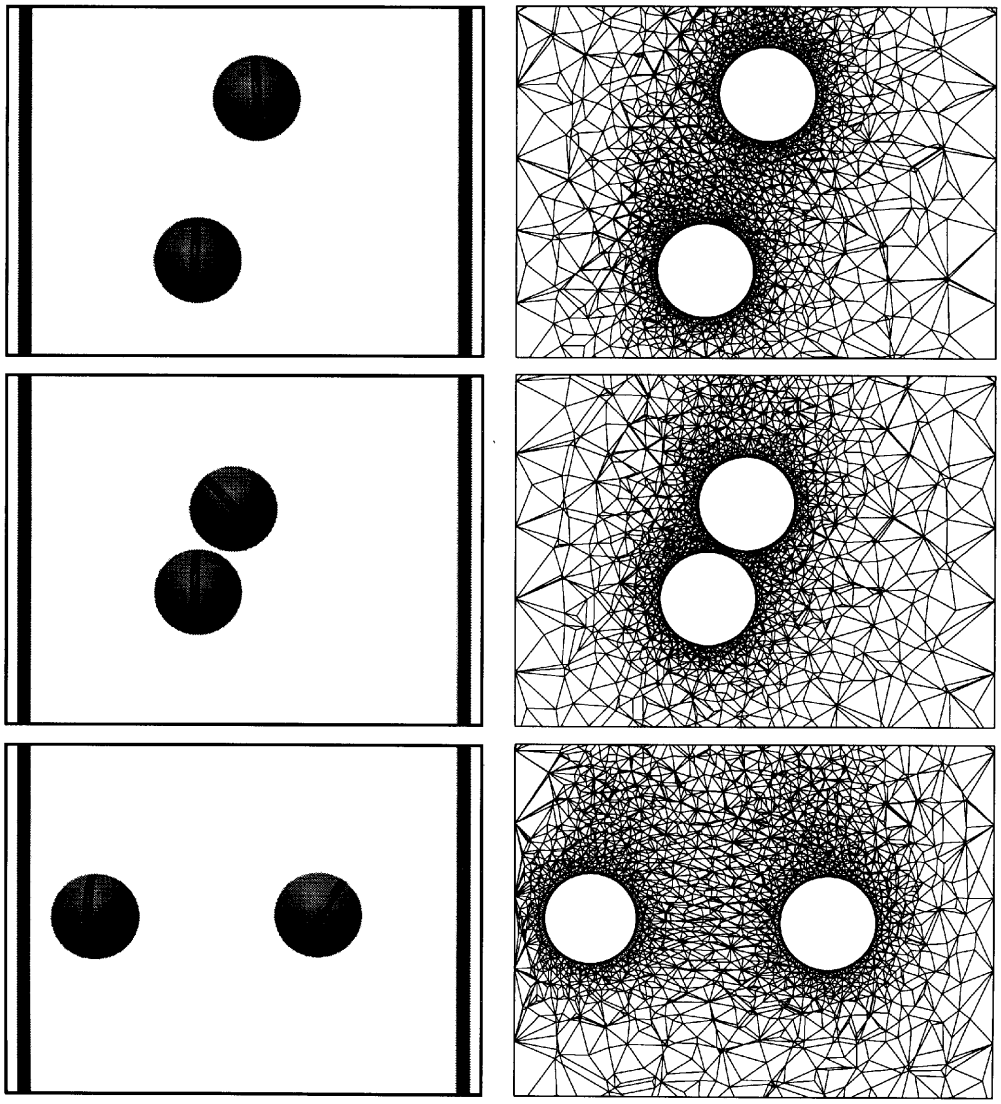


Figure 14. Configuration and mesh cross-section at three instants during simulation of two spheres falling in liquid-filled tube

A close-up view of one of the mesh cross-sections can be seen in Figure 15, with the boundary layer elements around each sphere.

The second example involves five spheres initially in a pyramid configuration. There are four spheres arranged in a square with one other sphere in the centre and above all the others. Throughout the simulation the centre sphere moves through the square and eventually settles to a level even with the other four to form a $\otimes$ shape. The sphere configurations along with a cross-section of the mesh at three instants are shown in Figure 16. The mesh stretching can also be seen in this figure.

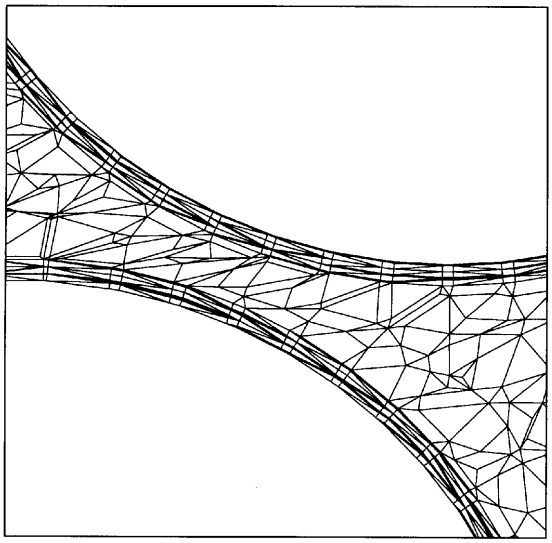


Figure 15. Close-up of cross-section of mesh, highlighting structured layers of elements around each sphere

6. CONCLUDING REMARKS

In this paper we have demonstrated our methods for the solution of flow problems involving complex geometries. These methods include a stabilized finite element formulation implemented efficiently on parallel architectures. This parallel implementation is very general and can be applied under two programming models and on several architectures. We also have an automatic mesh generation tool which allows us to discretize domains of very complex shape in a reasonable amount of time. This mesh generator is based on Delaunay–Voronoï methods with edge-swapping techniques. Also, we have the tools which allow us to model and discretize objects with complex geometries, and this is an essential component in simulation of most engineering applications.

With these computational tools we can apply numerical methods to the solution of very challenging engineering applications. In the past, numerical methods have mostly been applied in basic research and an idealised applications, but today, with the use of these methods and others like them, numerical analysis can be useful in the actual design process in engineering. We demonstrated such an application here with the simulation of airflow past an automobile performed at a scale and with such detail that would have been impossible only a few years ago.

ACKNOWLEDGEMENTS

This research was sponsored by ARPA under NIST contract 60NANB2D1272 and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory co-operative agreement number DAAH04-95-2-0003/contract number DAAH04-95-C-0008, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. CRAY C90 time was provided in part by the University of Minnesota Supercomputer Institute.

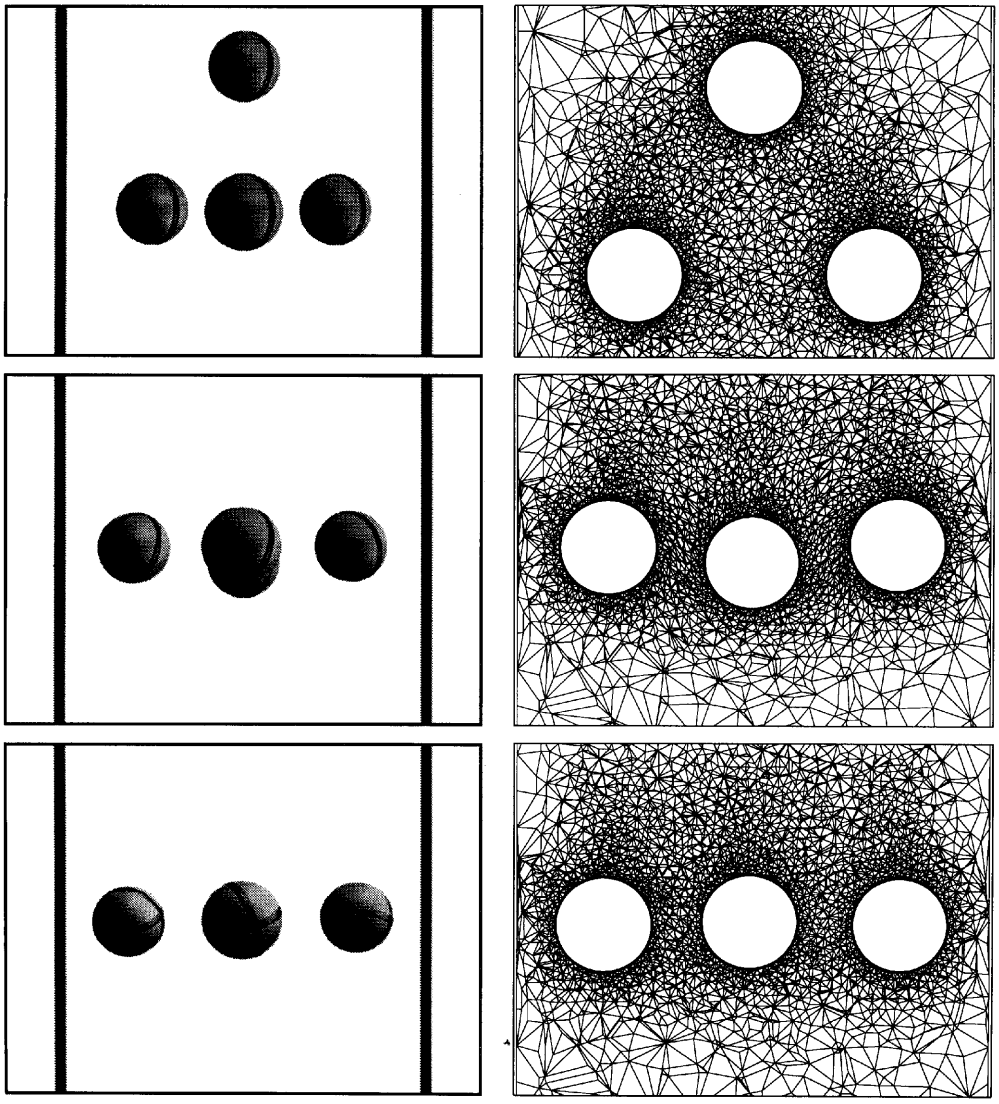


Figure 16. Configuration and mesh cross-section at three instants during simulation of five spheres falling in liquid-filled tube

REFERENCES

1. T. E. Tezduyar, M. Behr, S. Mittal and A. A. Johnson, 'Computation of unsteady incompressible flows with the finite element methods—space-time formulations, iterative strategies and massively parallel implementations', in P. Smolinkski, W. K. Liu, G. Hulbert and K. Tamma (eds), *New Methods in Transient Analysis*, AMD Vol. 143, ASME, New York, 1992, pp. 7–24.
2. M. Behr, A. Johnson, J. Kennedy, S. Mittal and T. E. Tezduyar, 'Computation of incompressible flows with implicit finite element implementations on the Connection Machine', *Comput. Methods Appl. Mech. Eng.*, **108**, 99–118 (1993).
3. T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson and S. Mittal, 'Parallel finite-element computation of 3D flows', *IEEE Comput.*, **26**(10), 27–36 (1993).
4. T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro and C. Waters, '3D simulations of flow problems with parallel finite element computations on the Cray T3D', in *Computational Mechanics '95, Proc. Int. Conf. on Computational Engineering Science*, Mauna Lani, HI, 1995.

5. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam, *PVM: Parallel Virtual Machine. A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, 1994.
6. Z. Johan, K. K. Mathur, S. L. Johnson and T. J. R. Hughes, 'An efficient communications strategy for finite element methods on the Connection Machine CM-5 system', *Comput. Methods Appl. Mech. Eng.*, **113**, 363–387 (1994).
7. A. A. Johnson, 'Mesh generation and update strategies for parallel computation of flow problems with moving boundaries and interfaces', *Ph.D. Thesis*, University of Minnesota, 1995.
8. G. Karypis and V. Kumar, 'METIS: unstructured graph partitioning and sparse matrix ordering systems', *Tech. Rep.*, Department of Computer Sciencee, University of Minnesota, 1995; METIS is available on the WWW.
9. S. Aliabadi and T. E. Tezduyar, 'Parallel fluid dynamics computations in aerospace applications', *Int. j. numer. methods fluids*, **21**, 783–805 (1995).
10. T. E. Tezduyar, S. K. Aliabadi, M. Behr and S. Mittal, 'Massively parallel finite element simulation of compressible and incompressible flows', *Comput. Methods Appl. Mech. Eng.*, **119**, 157–177 (1994).
11. S. Aliabadi, 'Large scale simulation capability for ram air parafoils is achieved on the Cray T3D', *AHPCRC Bull.*, **5**(3) (1995).
12. T. E. Tezduyar and A. A. Johnson, 'The world of flow simulation', in *Lecture Notes on Finite Element Simulation of Flow Problems*, Japan Society of Computational Fluid Dynamics, Tokyo, 1995.
13. P. L. George, *Automatic Mesh Generation, Application to Finite Element Methods*, Wiley, Chichester, 1991.
14. T. J. Baker, 'Developments and trends in three-dimensional mesh generation', *Appl. Numer. Math.*, **5**, 275–304 (1989).
15. M. S. Shephard and M. K. Georges, 'Automatic three-dimensional mesh generation by the finite octree technique', *Int. j. numer. methods eng.*, **32**, 709–749 (1991).
16. P. L. George, F. Hecht and E. Saltel, 'Automatic mesh generator with specified boundary', *Comput. Methods Appl. Mech. Eng.*, **92**, 269–288 (1991).
17. T. J. Barth, 'Aspects of unstructured grids and finite-volume solvers for the Euler and Navier–Stokes equations', in *Special Course on Unstructured Grid Methods for Advection Dominated Flows*, Advisory Group for Aerospace Research and Development, 1992, pp. 6-1–6–61.
18. B. Joe, 'Construction of three-dimensional Delaunay triangulation using local transformations', *Comput. Aided Geom. Design*, **8**, 123–142 (1991).
19. G. Farin, *Curves and Surfaces for Computer Aided Geometric Design, A Practical Guide*, Academic, New York, 1993.
20. T. E. Tezduyar and A. A. Johnson, 'High performance computing', in *Lecture Notes on Finite Element Simulation of Flow Problems*, Japan Society of Computational Fluid Dynamics, Tokyo, 1995.
21. M. Litke, 'Reusable launch vehicle II', *University of Minnesota Aerospace Design Project*, 1995.
22. V. Kalro and T. E. Tezduyar, 'Parallel finite element computation of 3D incompressible flows on MPPs', in W. G. Habashi (ed.), *Solution Techniques for Large-Scale CFD Problems*, Wiley, New York, 1995, pp. 59–81.
23. T. E. Tezduyar, M. Behr and J. Liou, 'A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space–time procedure: I. The concept and the preliminary tests', *Comput. Methods Appl. Mech. Eng.*, **94**, 339–351 (1992).
24. T. E. Tezduyar, M. Behr, S. Mittal and J. Liou, 'A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space–time procedure: II. Computation of free-surface flows, two-liquid flows, and flows with drifting cylinders', *Comput. Methods Appl. Mech. Eng.*, **94**, 353–371 (1992).
25. *CMSSL for CM Fortran: CM-5 Edition, Version 3.1*, Thinking Machine Corporation, 1993.
26. PVM and HeNCE Programmer's Manual, SR-2501 3.0 edition, Cray Research, 1993.
27. C. L. Lawson, 'Properties of *n*-dimensional triangulations', *Comput. Aided Geom. Design*, **3**, 231–246 (1986).
28. Y. Morinishi and T. Kobayashi, 'Large eddy simulation of complex flow fields', *Comput. Fluids*, **19**, 335–346 (1991).
29. C. Kato, A. Iida, Y. Takano, H. Fujita and M. Ikegawa, 'Numerical prediction of aerodynamic noise radiated from low Mach number turbulent wake', *Conf. Proc. 31st AIAA Aerospace Sciences Meet. Exhit.*, New York 1993.
30. T. Kobayashi, 'A review of CFD methods and their application to automobile aerodynamics', in *SAE Special Publication SP-908, Vehicle Aerodynamics: Wake Flows, CFD, and Aerodynamic Testing*, Society of Automotive Engineers, 1992, pp. 53–64.
31. D. J. Holt, 'Saturn: the vehicles', *Automot. Eng.*, **98**, 34–43 (1990).
32. A. Cogotti, 'Aerodynamic characteristics of car wheels', in *Technological Advances in Vehicle Design Series, SP3; Impact of Aerodynamics on Vehicle Design*, International Journal of Vehicle Design, 1983, pp. 173–196.
33. M. Takagi, K. Hayashi, Y. Shimpo and S. Uemura, 'Flow visualization techniques in automotive engineering', in *Technological Advances in Vehicle Design Series, SP3; Impact of Aerodynamics on Vehicle Design*, International Journal of Vehicle Design, 1983, pp. 500–511.
34. A. J. Scibor-Rylski, *Road Vehicle Aerodynamics: Second Edition*, Wiley, Chichester, 1984.
35. Society of Automotive Engineers, 'Aerodynamic flow visualization techniques and procedures', *SAE Infor. Rep. HS J1566*, 1986.
36. A. A. Johnson and T. E. Tezduyar, 'Simulation of multiple spheres falling in a liquid-filled tube', *Comput. Methods Appl. Mech. Eng.*, **134** 351–373 (1996).
37. A. A. Johnson and T. E. Tezduyar, 'Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces', *Comput. Methods Appl. Mech. Eng.*, **119**, 73–94 (1994).
38. A. F. Fortes, D. D. Joseph and T. S. Lundgren, 'Nonlinear mechanics of fluidization of beds of spherical particles', *J. Fluid Mech.*, **177**, 467–483 (1987).

