# Graphic workstations and supercomputers: An integrated environment for simulation of fluid dynamics problems

by F. Piccolo
V. Zecca
A. Grimaudo
C. Loiodice

**An integrated environment for simulation and visualization of physics and engineering problems of industrial interest has been set up at the IBM European Center for Scientific and Engineering Computing (ECSEC). This paper describes the environment, its components, and some experiments carried on at ECSEC to represent 3D objects displayed with the shading technique and the solution of fluid dynamics problems, all treated with the finite element method. Moreover, the paper describes the animation experiments developed to represent dynamics phenomena (fluid flows) and presents a videotape showing the time evolution of three fluid dynamics study cases.**

## Computational laboratory

Current computer technology allows researchers to simulate laboratory environments in which physical phenomena are reconstructed in order to observe the behavior of the reproduced system and measure its physical parameters. This activity generally serves as an experimental check of mathematical models theoretically defined to forecast the evolution of phenomena in different situations.

The solution of physical phenomena means knowing in advance what will happen when boundary conditions are varied. With mathematical models a physical problem can be transformed into a mathematical problem; it is possible, under suitable conditions, to assume that solution of the mathematical problem, i.e. forecasting the behavior of the model, is equivalent to solving the physical problem itself. Unfortunately, complex phenomena are described by systems of partial differential equations which have no analytical solution, at least in terms of elementary functions. Therefore another step is required to reach the solution. This step consists of employing numerical techniques to transform differential problems into numerical ones. Once again,

under suitable conditions, it is possible to assume that the solution of numerical problems is equivalent to the solution of mathematical problems and, finally, physical problems.

To solve numerical problems it is necessary to solve systems of algebraic equations, a task well suited to a computer. Realistic (i.e., complex and difficult to visualize) problems can be solved if supercomputers and powerful graphic workstations are available.

Supercomputers are required because several thousands of algebraic equations are involved: The computation time needed to solve an algebraic system depends strongly on the complexity of the problem. It is well known that numerical techniques can subdivide the domain of a problem into elements, so that physical parameter values need be determined only at the vertices (nodes) of the elements. If there are $n$ unknowns for each node and $m$ nodes, the total number of unknowns is $k = n \times m$; $k$ is a good indicator of the complexity of the problem and, for industrial problems, can be as large as 50000. The computation time is a function of $k_3$ because the solution of a dense system requires $O(k^3)$ operations. For a finite element matrix, computation time has been shown to vary as $k$ times the square of the mean bandwidth of the system, due to "skyline addressing" [1]. The skyline storage pattern stores and processes, for each row and column of the matrix, only those elements that lie between the first nonzero element and the diagonal element.

Powerful graphic workstations are required for data preparation and postprocessing, i.e., for the definition of the numerical problem and the representation of results. The definition of numerical problems consists of 1) geometrical reproduction (a structure, for example a car or a plane, must be reproduced in the correct scale); 2) definition of physical parameters (for materials constituting objects, etc.); 3) definition of boundary conditions needed for the solution of a differential problem; and 4) domain discretization, i.e., subdivision into elements according to user specifications. The set of these elements constitutes the mesh of the problem.

Simulation codes produce a great amount of data consisting of node coordinates and the values of physical parameters (e.g., velocity components, pressure, temperature, stress tensor components) that are assumed at the nodal locations and, in transient analysis, for each time step. These data, if they were presented in table form, would be unreadable. Viable solutions require some kind of graphic representation of the data so that the user can visualize the problem; this implies a need for interactive graphic tools.

In a computational laboratory, numerical simulations are possible, but these do not entirely eliminate the need for building and testing prototypes; their purpose is to reduce considerably the number of prototype tests required, and to enhance the effectiveness of testing. Thus, computational simulations and laboratory experiments are complementary, not mutually exclusive, approaches to a physics or engineering problem.

At the IBM European Center for Scientific and Engineering Computing (ECSEC) we set up an integrated supercomputing environment which constitutes our computational laboratory. We use an IBM 3090[1]/VF with six vector processors [2]; some IBM 5080 Graphics System workstations; a few IBM 6090 Graphics System workstations (the 6090 is a graphics system compatible with the 5080 but with more advanced image-rendering capabilities and a higher-resolution display); and an IBM 6180 plotter. We also use some industrial packages whose solvers have been vectorized and parallelized at ECSEC for the 3090 and whose graphic preprocessors and postprocessors have been enabled for the 5080 and the 6090, and developed in some of their features by means of programs written at ECSEC to take advantage of the IBM graPHIGS[1] software supporting the graphic capability of both workstations.

Examples of such industrial packages are the Automatic Dynamic Incremental Nonlinear Analysis (ADINA[2]) computer programs and the DAISY graphic postprocessor for PAM-CRASH.[3]

ADINA computer programs include

- ADINA for analysis of displacement and stress.
- ADINA-F [1] for analysis of 2D and 3D viscous incompressible fluid flows with heat transfer.
- ADINA-T for analysis of heat transfer in solids and structures and the solution of field problems.
- ADINA-IN (graphic preprocessor) for preparation and display of the input data.
- ADINA-PLOT (graphic postprocessor) for display of solution results.

The DAISY program provides an interactive environment for the graphic postprocessing of several structural finite element codes such as PAM-CRASH, which can be used to simulate the dynamic evolution of structures under crash conditions.

## Preprocessing and postprocessing for the finite element method

Numerical techniques have been developed to simulate many classes of problems. The best-known techniques

[1] 3090 and graPHIGS are trademarks of International Business Machines Corporation.
[2] ADINA Computer programs are registered trademarks of ADINA R&D, Inc., Watertown, MA.
[3] PAM-CRASH is a registered trademark of Engineering Systems International (ESI), S.A. Rungis, France.

are spectral methods, finite differences, and the finite element method. We focus our attention on the finite element method (FEM) to illustrate the integrated supercomputing environment developed at ECSEC. This numerical method is very useful for complex problems, as shown convincingly by both academic research and industrial practice [3].

In the finite element method, the problem domain is subdivided into a number of elements which constitute its mesh. The partial differential equations describing the phenomena are replaced by ordinary differential or algebraic equations in each element. The system of these equations is then solved simultaneously to determine values of the unknown quantities.

The primary advantages of FEM include

- Flexibility in treating arbitrarily complex domains and boundary conditions.
- The ability to design unstructured grids without the need for many grid points throughout the entire domain.
- The ability to derive comprehensive error estimates.
- The ability to determine accurate solutions within user-prescribed tolerances.

Hand-generating meshes is a very tedious task, and generating complex three-dimensional meshes is extremely difficult, if not impossible; moreover, a finite element analysis of any problem of realistic size produces a great quantity of numerical data which is difficult to examine.

A preprocessor for a finite element solver must be designed to generate and verify the input data, which may consist of a large number of nodes, elements, and parameters to specify the analysis to be carried out. The preprocessor can require any (or all) of the following:

- Mesh generation to allow users to generate meshes consisting of line, surface, or volume elements. Straight or curved lines with equidistant nodes or with gradually varying nodal distances should be available, as well as automatic connection of different parts of a mesh.
- Cartesian, cylindrical or spherical local coordinate systems which are well suited for defining nodal points. The coordinate values may then be automatically transformed into global system reference coordinates.
- A single definition of the physical parameters of materials, which should be generated only once for a group of multiple elements: An identifying number which can be referenced by several or all of the elements groups must be assigned to each material.
- A definition of element loads, which makes it possible for the user to specify the element number and the edge or surface to which the loading (pressure, heat flow, etc.) is applied. It is also necessary to define the degrees of freedom; the analysis type (steady-state or transient); the equilibrium iteration method; the time-function providing the time dependence of loads applied to the model and the amount of information to be recorded in the solver output file.
- Graphic display of the finite element model, to represent it by plotting, zooming, and scaling the mesh; changing the view direction; and displaying node and element numbers. The boundary conditions must also be available, as well as a means for hidden-line removal and outline generation.

A postprocessor must be designed for the display and analysis of output data from the solver. It must read the analysis output and organize information into a specially designed database for effective retrieval. The necessary features of a postprocessor include the following:

- Visualization of a total or partial perspective view of the model, eventually with hidden lines and surfaces removed, or with shaded representation. The visualization must also include plotting on the mesh of contour lines; color filling of stresses, strains, displacements, velocities, accelerations, temperatures, etc.; and redefinition of the range of values for contour lines or color filling in order to increase the resolution within an area around peak values.
- Zoom, translation, $x$-$y$-$z$ scaling, rotation, pivot, and symmetrization.
- Plotting of quantities to compare the results of the analysis with those obtained experimentally. For example, a variable must be plotted as a function of another variable to produce a time history graph.
- Provision for storage of each graphic display page, to create a graphic database and allow for printing.
- Results listing and scanning within the entire model, within selected regions, or along a predefined line. Scanning allows the user to search for extreme results, e.g., those exceeding a specified value.
- Animation, a new capability not generally provided by programs. Animated representations are useful in showing the evolution of simulated phenomena by displaying, in quick succession, steps of the transient analysis. Graphic workstations allow this representation, which is very important for continuous checking of dynamic phenomena; many aspects of the analysis can be lost by observing a single frame statically. In engineering problems, however, the amount of output data can be a binding limit; animation is possible only if the workstation buffer is large enough to accept the graphic structures.

For the purposes of our work (enabling of the codes for the IBM graphic workstations and their development), we classify the code sections concerning the graphic data organization, manipulation, and visualization, into three main groups:

- Graphic initialization routines. In this group we include routines that are strictly workstation-dependent, such as procedures for character font selection, and color table and window dimension setting. Particular caution is needed in the setting of parameters for routines which perform the graphic initialization, in order to correctly and effectively utilize the software and hardware capabilities provided by the graphic system.
- Basic graphic primitive routines. This group includes routines for plotting elementary graphic entities such as lines, polygons, and character strings.
- High-level graphic functions. In this group, routines performing graphic functions such as shading, transparency, and hidden-line removal can be included. The code sections which perform these functions use basic primitive graphic routines, but these are complex from an algorithmic point of view and difficult to manage because of their size.

A good example of the full potential of the IBM graPHIGS application program interface is the 3D graphic environment [4] developed at ECSEC for manipulating objects built by means of finite element definition. This environment shows the powerful capabilities of the IBM graphic systems. Performance and interactivity are certainly improved by utilizing graPHIGS primitives on the IBM 5080 or 6090 graphic system equipped with local processing capabilities. In the problem under discussion, graPHIGS software, once it has built the structural configuration, sets the window according to the object dimensions; defines the point of view; and displays the object in a predefined reference position. Then the operator can rotate the structure around the three axes, zoom it, or cut it by means of two clipping planes. These manipulations can be performed in arbitrary succession, as many times as needed. Moreover, the operator can center a point of the image and zoom it to focus on a particular zone. These operations are performed using the peripheral devices of the workstation, the eight dials and the mouse. The performance improvement obtained working in the 3D environment is easily verified by the user in terms of significant ease of use and response time. The user no longer has to type commands to perform the transformations; they are computed locally by the graphic processor, which results in a noticeable reduction in channel data-transfer requirements and host CPU load. From the programmer's point of view, a 3D graphic environment provides a substantial reduction in the complexity and quantity of graphic code.

## Visualization enhancement: Shaded objects

For 3D objects the two most important requirements to be met are realistic reproduction and reasonable computational effort. The first requirement springs from the natural desire to see an object displayed as it is in nature; the second one derives from an operative need: Visualization and analysis of results produced by a simulation code must be carried out interactively; therefore, the computation time required to produce a representation must be compatible with reasonable response time for interactive transactions.

The simplest representation of 3D objects is the so-called "wire frame." This representation is straightforward and can be developed quickly for structures analyzed with the finite element method. In fact, an object defined, for example, by shell elements can be represented by means of surface points (called nodes) for which the connections, i.e., the paths that connect nodes to form the elements, are established. In this case, by using polylines to reconstruct elements, surfaces can be represented by polygons, although these do not give a very accurate approximation of surface curvature.

It is not possible to refine the size of the elements which constitute the mesh of an object to improve the approximation, because the subdivision of structures to be analyzed by simulation codes is determined by numerical methods and not by a graphical scheme. A realistic representation must, therefore, be carried out by other techniques, which can reproduce objects without requiring variations of the mesh. A good technique is shading, which is preferred over ray-tracing to reduce the computational effort.

The appearance of polyhedral approximations to curved surfaces can be enhanced by shading them in such a way that the shading varies smoothly across each polygon. Gouraud [5] and Phong [6] have described two methods for computing shading functions. The first one is fast, but the renderings produced often show pronounced light or dark lines (Mach bands) perceived by the human eye where the spatial derivative of the shading function is discontinuous. The human eye is particularly sensitive to these discontinuities because it naturally discriminates the edges of objects. Phong's method solves this problem by ensuring mathematical continuity for the illumination function and its first-order derivatives, but it is much more expensive computationally. Many faster methods of computing Phong's shading have been proposed in the literature, however, making this approach more useful [7].

To briefly describe Gouraud's method, we assume a collection of polygons with shading values associated with each vertex. These values can be simply computed, assuming that for each vertex we know the vector normal to the surface of the object and the direction of the light rays. According to the Lambert lighting model, the intensity perceived by an observer is independent of the observer's position and varies directly with the cosine of the angle between the normal vector and the light direction. Inside a polygon we can define scan lines (horizontal or vertical) which intersect its edges. At these intersection points, the shading value is computed by linear interpolation of the shading values at the vertices. Finally, shading inside a polygon is computed along the scan lines by linear interpolation of the values which were computed where the scan lines meet the edges of the polygon.

Phong's shading is similar, except that it interpolates the normal at each vertex and then evaluates the illumination function at each point. From a computational point of view, this method, involving three components to interpolate the vectors which are to be normalized, is of course more difficult, but Phong's method avoids the Mach bands which appear in adjacent Gouraud-shaded polygons.

Both of the above techniques require that the normal vector at the polygon vertices be known. For an unknown structure defined by means of flat elements, which is typical in a FEM simulation, normal vectors at the element vertices are not known. We have developed a tool for shaded representation of objects defined by finite elements [8], starting from the consideration that each vertex is shared among more than one element. Therefore, the normal vector at this vertex can be computed by normalizing the averaged components of the normal vectors belonging to the shared elements. Knowledge of these normal vectors represents the basis for computing the shading function. To avoid an incorrect smoothing of the edge of an object, such as a cube, the tool checks the angles $\theta$ formed between the normal vectors of the pairs of elements sharing a node. If the angle $\theta$ between two elements, for example two contiguous faces of the cube, is greater than a predefined angle $\tilde{\theta}$, the algorithem does not perform the average of these two normal vectors. In this case, where there is a well-defined discontinuity of the spatial derivative of the shading function, the resulting pronounced Mach band is useful. In practice, $\tilde{\theta}$ is conveniently set to $\pi/8$.

The reconstruction of the normal vector at a vertex, averaging the components of the normal vectors of the shared elements, is a particularly ticklish procedure; errors in this process can alter the curvature of a surface so that the shape of an object is changed. To check the quality of the reconstruction, we consider a sphere defined by means of finite elements obtained by subdividing the equatorial plane and the meridian planes into 26 parts. The idea is to consider the sphere as an unknown object produced by a preprocessor for finite element analysis of structures but, at the same time, to utilize the analytical expression of this surface to check the vectors reconstructed in the vertices of the elements. If we wish to study the structural behavior of a sphere, for example a ball bearing, we define a model comprising 676 surface elements; this is a good number for structural analysis but not as effective for a graphical representation of the sphere. We must render the shaded sphere with this small number of elements and no way to refine the mesh.

Given $z = f(x, y)$, the Cartesian expression of a spherical surface, it is possible to compute the vectors

$$\frac{\partial f}{\partial x}\underline{i}, \qquad \frac{\partial f}{\partial y}\underline{j};$$

therefore, the vector normal to the surface will be

$$\underline{n} = \frac{\frac{\partial f}{\partial x}\underline{i} \wedge \frac{\partial f}{\partial y}\underline{j}}{\left|\frac{\partial f}{\partial x}\underline{i} \wedge \frac{\partial f}{\partial y}\underline{j}\right|}.$$

Let $\underline{n}^i = (n^i_x, n^i_y, n^i_z)$ be the versor in the $i$th vertex, and $\underline{n}^i_m = (n^i_{mx}, n^i_{my}, n^i_{mz})$ be the reconstructed versor in the same node. A comparison of $\underline{n}_m$ and $\underline{n}$, in the element vertices, is to be performed. We can define $dx^i = |n^i_{mx} - n^i_x|$, $dy^i = |n^i_{my} - n^i_y|$, $dz^i = |n^i_{mz} - n^i_z|$, and the average error for the three main directions as

$$\bar{E}_x = \frac{\sum_{i=1}^{N} dx^i}{N}, \qquad \bar{E}_y = \frac{\sum_{i=1}^{N} dy^i}{N}, \qquad \bar{E}_z = \frac{\sum_{i=1}^{N} dz^i}{N},$$

where $N$ is the number of nodes. In the case of the sphere, we have

$$\bar{E}_x = 0.005, \qquad \bar{E}_y = 0.003, \qquad \bar{E}_z = 0.002.$$

The tool developed at ECSEC allows the user to render objects by flat, Gouraud's, and Phong's shading techniques. The choice depends on the user's need; the more sophisticated the shading, the greater the computational cost. **Figure 1** shows three shaded spheres, each comprising 338 visible finite elements. The lower left sphere is rendered by flat shading, the upper left one by Gouraud's shading, and the sphere on the right by Phong's technique. In the sphere rendered by flat shading, the elements constituting the object are clearly visible because the color does not vary inside the elements. The curvature of the surface is not reconstructed. The sphere rendered by Gouraud's technique has the same number of elements as the flat-

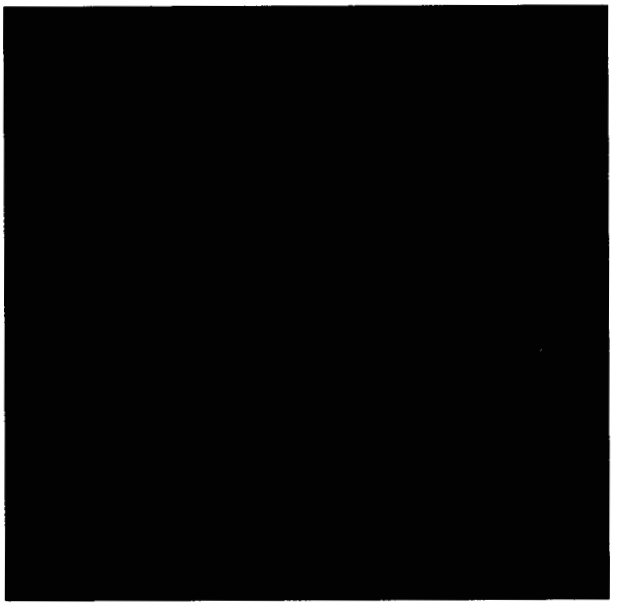


**Figure 1**

Three shaded spheres, each comprising 338 visible finite elements. The lower left sphere is rendered by flat shading, the upper left one by Gouraud's shading, and the sphere on the right by Phong's technique.

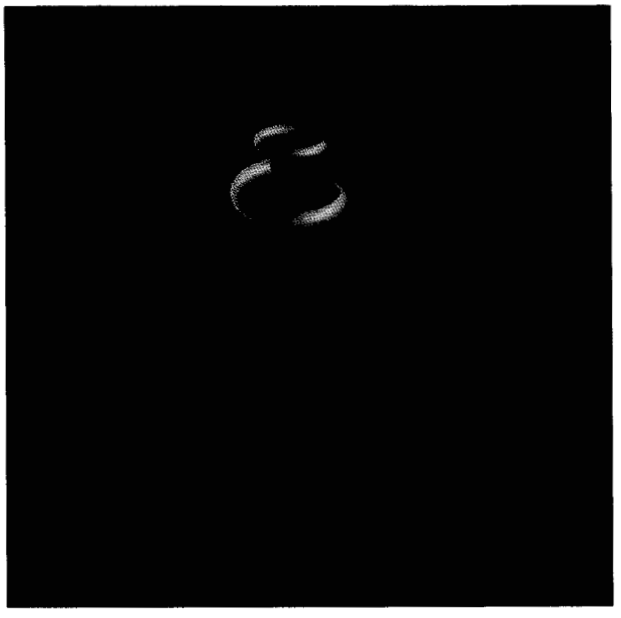


**Figure 2**

Gaussian surface, spheres, and toruses represented by Phong's shading. The objects are defined by the finite element method.

shaded sphere, but now the curvature is better represented. The best rendering is achieved by Phong's technique, which represents the spots due to reflection of radiation and the room light as well. We wish to note that the shaded representation in Figure 1 is created by using only 32 different blue tones. We chose this number to emphasize the quality of the rendering provided by the tool; if good representation is achieved with only 32 colors, the method of averaging the normal vectors must be considered satisfactory. **Figures 2** and **3** show other objects represented by Phong's shading. All of these objects are produced by the finite element method. Before the shading is computed, it is important to remove hidden surfaces to reduce the amount of virtual or disk storage and computation time required. The reduction of storage size is about 50%; the time reduction is less than 50% because a hidden-line-removal algorithm is involved.

The use of these algorithms requires a convenient way of describing a polyhedron in terms of its vertices and faces. The information needed for this description is given by three coordinates for each vertex and by a string of vertices identifying each face (the direction of travel around a face must be defined previously). This information is immediately available from the typical output data of a finite element simulation, so no reorganization of information is needed before applying these algorithms.

## Identification of internal points

In the previous section we pointed out that the shading inside elements is computed by linear interpolation along the scan lines. This means that we must know the coordinates of the points at which the interpolation is to be performed. A very similar situation occurs when we deal with another kind of representation, scalar colored fields. This representation is a useful feature of postprocessing and is made by coloring the space region occupied by an element with different colors selected from a color table, according to the values assumed by a physical parameter throughout the element. Inside an element, the values of the quantities are unknown because of the technique employed to solve these problems, i.e., a numerical technique which solves the problem at particular points (nodes). Data reconstruction within the elements, by means of interpolation, is needed to fill them with colors. For finite elements, with point values known only at the nodes, interpolation requires knowledge of the point coordinates where interpolation is to be made. This can be difficult because the finite element method adapts the element shape to the geometry of the problem. This means that 2D quadrilateral elements, for example, have nonparallel edges and are placed in the plane with arbitrary orientations.

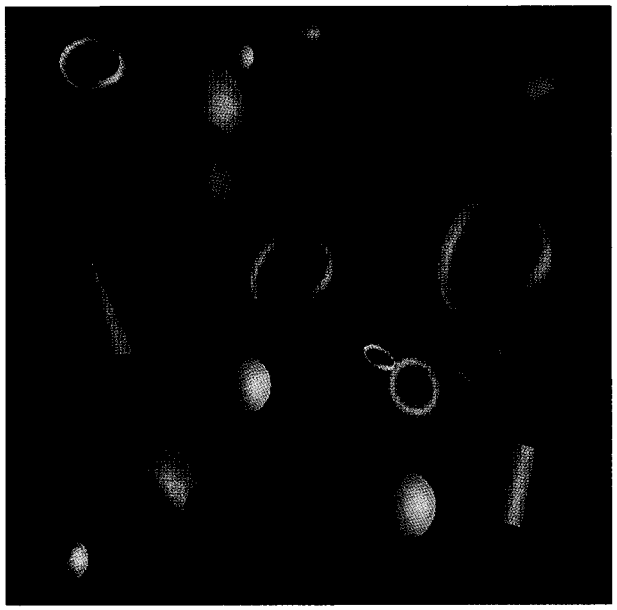


**Figure 3**

Composite of objects represented by Phong's shading. The objects are defined by the finite element method.

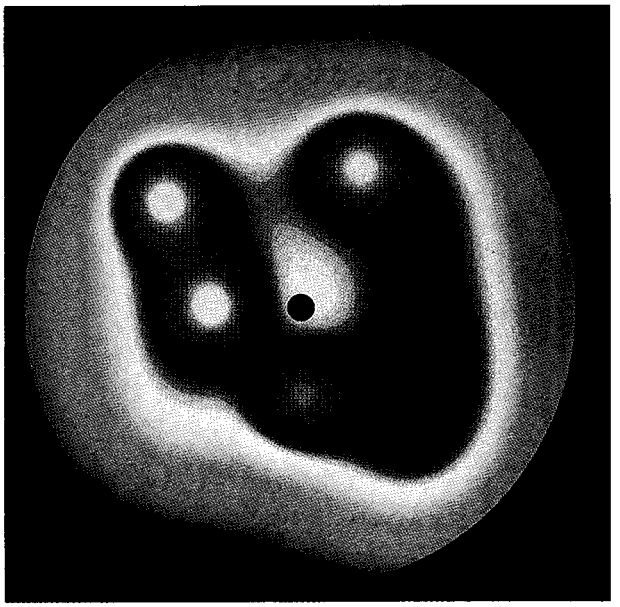


**Figure 4**

Combination of 20 Gaussians with $\sigma$ ranging between 3 and 6.

This section describes the criterion applied by the algorithm developed at the ECSEC to identify internal points.

In the case of 2D or 3D finite elements projected onto the screen, we have polygons which can be represented in a finite and discrete space $D^2$ (the display); therefore, knowledge of the coordinates of a finite number of points is needed. In a display of 1024 × 1024 pixels, each pixel is referenced by a pair of integer numbers

$$p(i, j) \begin{cases} 1 \leq i \leq 1024, \\ 1 \leq j \leq 1024. \end{cases}$$

If we consider a polygon $E$, for example a 2D element belonging to a mesh, we have to know that all the $p(i, j) \in E$. The algorithm considers the smallest rectangle $R$ circumscribing the polygon $E$ because

$$\{p(i, j) \in E\} \subset \{p(i, j) \in R\} \subset \{p(i, j) \in D^2\}.$$

Let $P_1 = (x_1, y_1)$, $P_2 = (x_2, y_2)$, $P_3 = (x_3, y_3)$, $P_4 = (x_4, y_4)$ be the vertices of the polygon $E$. It is well known that three points

$$Q_1 = (x_1, y_1), \qquad Q_2 = (x_2, y_2), \qquad Q_3 = (x_3, y_3)$$

identify a triangle $T$ whose area is

$$area(T) = \left| \frac{1}{2} (ax_1 + bx_2 + cx_3) \right|,$$

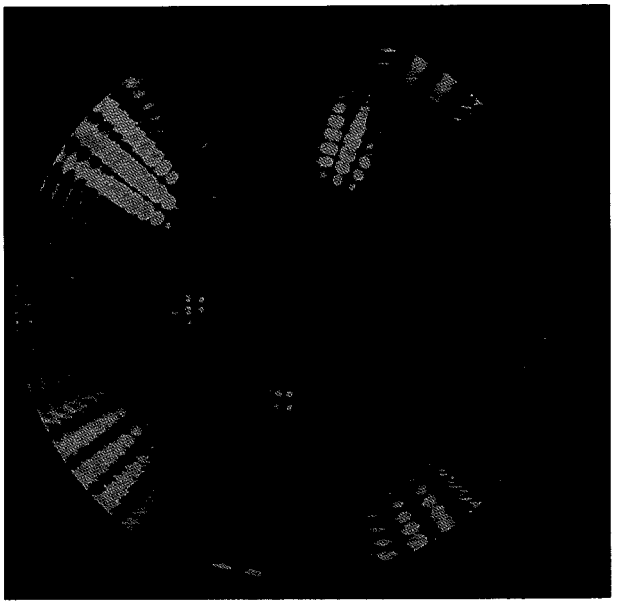


**Figure 5**

Relative error field of the interpolation of the 20 Gaussians shown in Figure 4, represented by a chromatic scale. Blue indicates an error less than 1%; cyan, 1–2%; cyan–green, 2–3%; green, 3–4%. The average error is 0.35%, and the maximum is 3.5%.

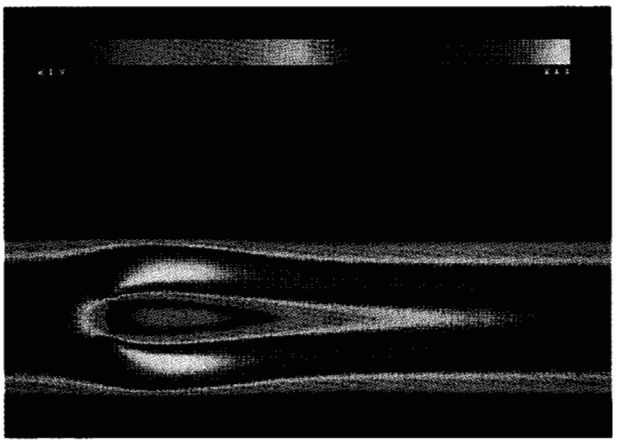


**Figure 6**

A time step related to the analysis of fluid flow past a flat plate in a pipe.

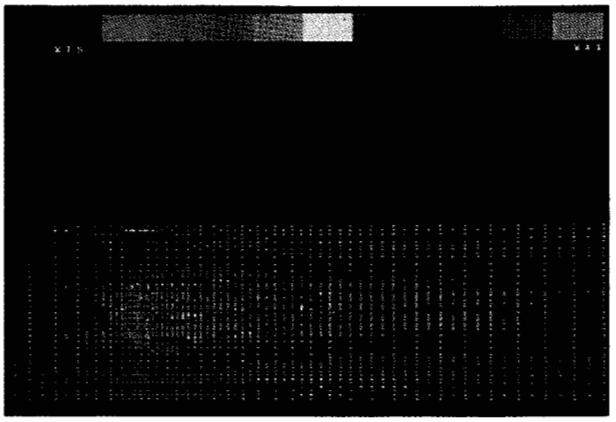


**Figure 7**

Relative error registered on the five nodes of the 2D finite elements of Figure 6, where the interpolation is made. The average error between the value computed by ADINA-F and the interpolated value is 5.9%. In the color table the first ten colors are associated with errors between 1% and 10%, respectively; the last color denotes errors greater than 10%. Such an error occurs only in 10% of the nodes.

where $a = (y_2 - y_3)$, $b = (y_3 - y_1)$, $c = (y_1 - y_2)$. For each point $Q = (x_q, y_q) \in R$, four triangles are defined: $T_1 = P_1QP_2$, $T_2 = P_2QP_3$, $T_3 = P_3QP_4$, $T_4 = P_4QP_1$. For each of them the areas $area(T_j)$, $j = 1, \cdots, 4$ are computed. A point $Q \in E$ if

$$\sum_{j=1}^{4} area(T_j) = area(E),$$

where $area(E)$ is the area of the polygon $E$.

## Visualization enhancement: Scalar and vector colored fields

For the representation of scalar fields, our tool performs a linear interpolation in terms of scan lines. With reference to the element $E$ introduced in the previous section, let $f_1, f_2, f_3, f_4$ be the values of the function $f(x, y)$ in the nodes $P_1, P_2, P_3, P_4$, and let $\{y = y_{max} - k; k = 0, 1, \cdots, n_y\}$ be the family of scan lines where $n_y$ is the number of integers between $y_{min}$ and $y_{max}$. The lines $y_{min}$ and $y_{max}$ are respectively the minimum and maximum ordinates of the rectangle $R$ circumscribing $E$. If we choose a scan line $s$, it intersects two edges of $E$ at the points $F = (x_F, y_F)$ and $G = (x_G, y_G)$.

Our algorithm is divided into two steps. The value of $f(x, y)$ at $F$ is computed with a linear interpolation between the values $f_2$ and $f_3$. If we indicate with $d(F, P_3)$ and $d(P_2, P_3)$ the distance between $F$ and $P_3$ and the distance between $P_2$ and $P_3$, we have

$$\frac{f_F - f_3}{f_2 - f_3} = \frac{d(F, P_3)}{d(P_2, P_3)}.$$

The value of $f$ at $F$ is therefore

$$f_F = \left[1 - \frac{d(F, P_3)}{d(P_2, P_3)}\right] f_3 + \frac{d(F, P_3)}{d(P_2, P_3)} f_2.$$

For the point $G$ we have

$$f_G = \left[1 - \frac{d(G, P_4)}{d(P_1, P_4)}\right] f_4 + \frac{d(G, P_4)}{d(P_1, P_4)} f_1.$$

Successively, the values of $f$ on the scan line $s$ between $F$ and $G$ are computed. For a point $P$ on $s$ we have

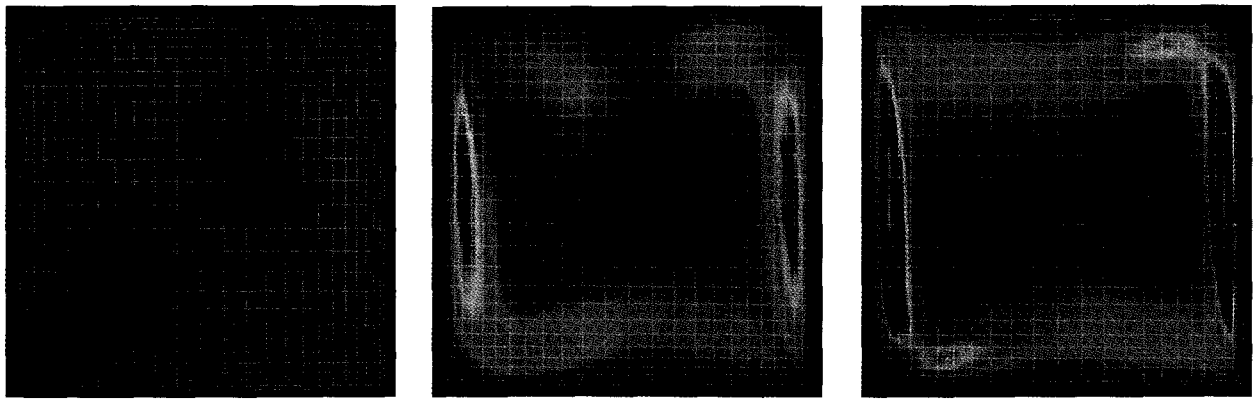
$$\frac{f_P - f_F}{f_G - f_F} = \frac{d(P, F)}{d(G, F)}.$$

and then

$$f_P = \left[1 - \frac{d(P, F)}{d(G, F)}\right] f_F + \frac{d(P, F)}{d(G, F)} f_G.$$

Once we have identified the values of the scalar field at the pixels, we access the color table.

We chose linear interpolation for performance reasons. We tested its accuracy in two ways. First we interpolated on a field made up of a combination of 20 Gaussians, as shown in **Figure 4**, with $\sigma$ ranging between 3 and 6. The average error is 0.35% and the maximum is 3.5%. **Figure 5** shows the relative error field of the interpolation. We then ran a test on a real ADINA-F output. The 2D ADINA-F elements have nine nodes, with four of them located on the vertices. We interpolate on these four nodes, again for performance reasons. The other five nodes are used to evaluate our interpolation algorithm on a real problem. **Figure 6** shows a time step related to the analysis of the fluid flow past a flat plate in a pipe. This simulation has been used as a study case for the fluid



(a) (b) (c)

**Figure 8**

Speed field for the fluid flow in a thermal cavity. The time-steps shown are (a) the 10th, (b) the 186th, and (c) the 372nd, and the mesh is superimposed.

dynamics computational laboratory at ECSEC, and is treated more extensively in the next section. The average error between the value computed by ADINA-F and our interpolated value is 5.9%. **Figure 7** shows the relative error registered on the five nodes mentioned above. We believe it is very significant to know the number of the points where the approximation is affected by a relative error greater than 10%. For this study case, 10% of the nodes exhibited such an error. Considering the low degree of the interpolation polynomials and the fast computation provided by the tool, the tests show satisfactory behavior.

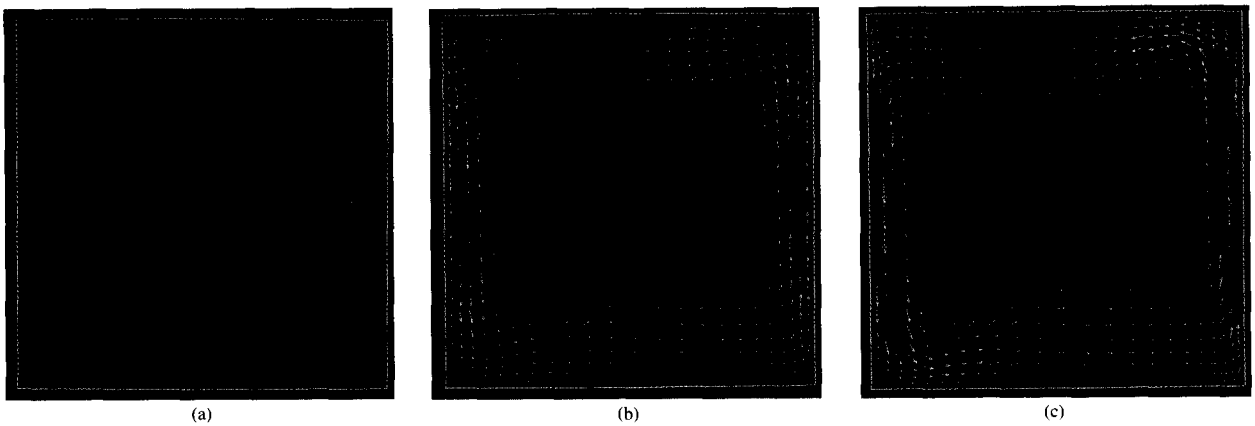
Our tool also represents vector fields. In particular, velocity fields can be represented by arrows originating from the nodes. The length of the arrows is a function of the speed, and their color can be associated with the speed or with any other scalar field, such as temperature.

When the mesh is thick, the density of nodes is such that the human eye cannot resolve the arrows referring to adjacent nodes. For this reason we select from four to nine nodes for each element, depending on their density.

## Visualization enhancement: Animation of scalar and vector fields

The solution of fluid dynamics problems is generally carried on in time steps that follow the temporal behavior of the phenomenon to be studied. Our tool is able to represent the output data of ADINA-F for both static and transient analysis. For the transient case, a sequence of images corresponding to the time steps is generated. Such a sequence can be shown at a fast rate on an IBM 5080 or 6090 Graphics System to provide an animation of the problem evolution. To obtain good animation, however, all the images must reside in the local buffer of the workstation, and the most binding constraint is the buffer size. The buffer size of the 5080 is 4.5 MB, and to represent scalar fields it is possible to allocate only 13 byte maps of 320000 pixels, displayable with a refresh rate of four pictures per second. The 6090 allows a more realistic animation because of its buffer size of 32 MB, within which 100 byte maps can be allocated. The 6090 has a higher refresh rate, thus increasing the quality of the animation because more frames are available and screen updating is faster. For vector fields, with reference to a buffer size of 4.5 MB, it is possible to animate sequences consisting of 100 frames, each of them dealing with 1000 arrows. More frames are available in this case because an arrow can be drawn by specifying five points by means of a polyline. In this case, each frame requires 1000 polylines plus the graphic attributes. On the 6090 it is possible to store seven hundred frames of the same complexity.

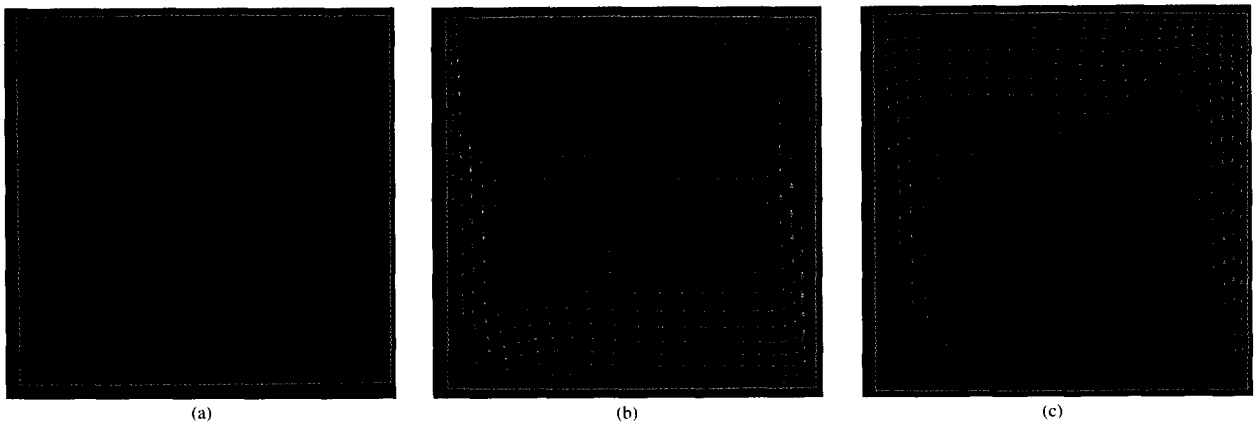
Current graphic workstations can generate longer and more accurate animations, but the representation of a prolonged dynamic phenomenon still requires videotape recording. We have analyzed three fluid dynamics problems extending over several hundreds of time steps, and their animated representations are thus only possible on videotape. In the videotape sequence that supports this paper, we show animations of the behavior of a fluid flow in a thermal cavity, past a plate in a pipe, and past a nozzle. For the cavity, the velocity field is represented twice by means of arrows located at the nodal points. The color of the arrows is associated with the speed and then with the temperature value computed at the nodes. Scalar fields representing speed and temperature complete the first part of the sequence. The velocity field and the velocity module are represented for flow past a



(a) (b) (c)

**Figure 9**

Velocity field for the fluid flow in a thermal cavity. The time-steps shown are (a) the 10th, (b) the 186th, and (c) the 372nd. The color is associated with the speed and is selected from a palette of 256 colors, as shown in the related videotape.



(a) (b) (c)

**Figure 10**

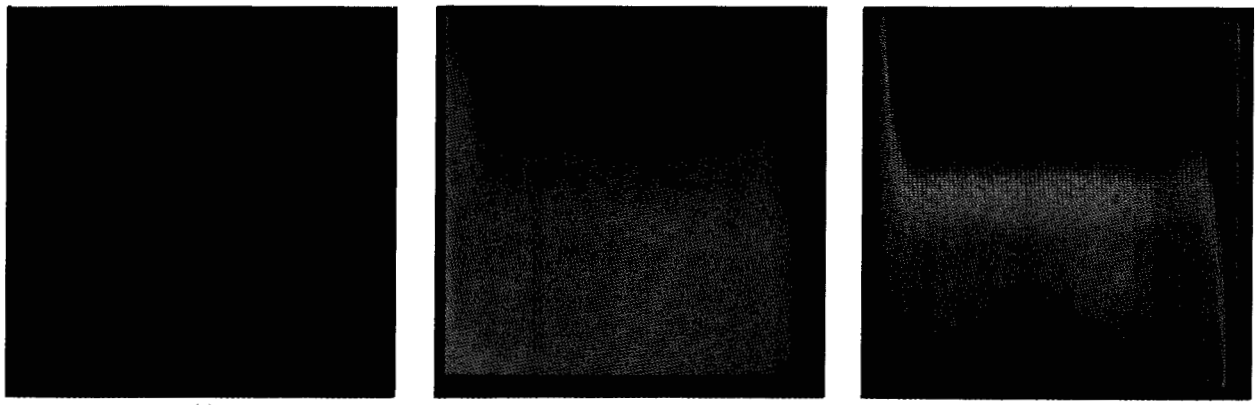
Velocity field for the fluid flow in a thermal cavity. The time-steps shown are (a) the 10th, (b) the 186th, and (c) the 372nd. The color is associated with the temperature.

plate as well as for flow past a nozzle with the addition of the pressure field. The videotape contains about 3500 24-bit images, amounting to five gigabytes. The images were recorded on videotape at the IBM UK Scientific Center at Winchester.

## The fluid dynamics computational laboratory at ECSEC

The solution of fluid dynamics problems of industrial interest by means of simulation codes often requires huge computation times even on powerful computers. Better performance can be obtained with supercomputers such as the IBM 3090 multiprocessor with Vector Facility if the simulation codes are optimized for these machines, i.e., if the I/O handling and the algorithm structure are organized to achieve concurrent execution on the available processors (parallelization) and, for each processor, to exploit pipelined hardware (vectorization). In an early phase of our research project, we proved that a code such as ADINA-F, not originally written for a supercomputer, could easily be adapted to a supercomputer if efficient vectorization and parallelization software tools were available [9]. In the case under consideration, a great increase in performance has been achieved, although the code remains written entirely in FORTRAN and the basic algorithms have not been modified.

Recent developments in computer technology such as vector processing, multitasking, powerful channels, and high-performance memory, which are peculiar to modern



(a) (b) (c)

**Figure 11**

Temperature field for the fluid flow in a thermal cavity. The time-steps shown are (a) the 10th, (b) the 186th, and (c) the 372nd.

supercomputers such as the IBM 3090 vector multiprocessor, contribute to the dramatic decrease of computation time for industrial problems. The optimization of ADINA-F exploited the supercomputing features of the 3090. Original I/O activity has been optimized by moving files into central storage; the finite element assembly process and the solver have been modified to take advantage of the Vector Facility and the multiprocessing feature of the 3090. Performance results confirm that tools such as the IBM Parallel FORTRAN vectorizing and parallelizing compiler are very effective in supercomputer exploitation. Satisfactory performance was achieved without modifying algorithms or changing the paths of the original source.

The use of Parallel FORTRAN made possible a doubling of global speed for vectorization, an increase of up to three times for parallelization and vectorization on two processors, and an increase of up to four times for four processors. Further increases have been derived from I/O optimization.

ADINA-F is supported by the graphic preprocessor ADINA-IN and the graphic postprocessor ADINA-PLOT, as well as the previously described graphic tools developed at ECSEC; these, together with the 3090 vector multiprocessor and the graphic systems, make up our fluid dynamics computational laboratory.

## Simulated fluid dynamics problems

The analysis of three 2D fluid dynamics problems has been carried on at ECSEC in our computational laboratory. The first case simulates 300 seconds of natural convection in a square cavity with vertical walls heated so as to create a free convective flow. The number of nodal points in the mesh is 2401, and there are 372 time steps amounting to a total of 11 hours of CPU time. The second problem studies 300 seconds of flow past a flat plate placed in a channel; 3858 nodal points constitute the mesh, and 372 time steps are simulated for a total of three hours of CPU time. The last case reproduces 200 seconds of flow around a nozzle. The mesh consists of 3761 nodal points, and 250 time steps are simulated for a total of six hours of CPU time. In all these cases the fluid is incompressible and viscous; results are shown in the videotape sequence and in the figures cited in the following discussion.

In the first simulation the fluid is contained in a square of side $L = 1$. With reference to a coordinate system $(xOy)$ with the origin in the lower left corner of the cavity, the boundary conditions describing the physical situation are as follows:

- Horizontal walls insulated.
- Heating of the vertical walls given by $T = T_0$ at $x = 0$ and $T = f(t)$ at $x = L$, where $t$ is the time and $f(t)$ a linearly increasing function.
- No-slip conditions imposed on the walls.

The initial conditions are $T = T_0$ and $v_x = v_y = 0$ all over the fluid domain. If $f(t) > T_0$, it can be expected that fluid near $x = L$ will become hotter due to conduction from the heated wall, and thus less and less dense. It will therefore rise because of a buoyancy force acting upward. On the contrary, the fluid near $x = 0$ is always colder than the surrounding fluid, and hence more dense, so it will move downward. In this way a counterclockwise circulation pattern will result. Characteristic parameters of these phenomena are the Rayleigh number,

$$Ra = \frac{\rho g \beta (T - T_0) L^3}{\mu \alpha},$$

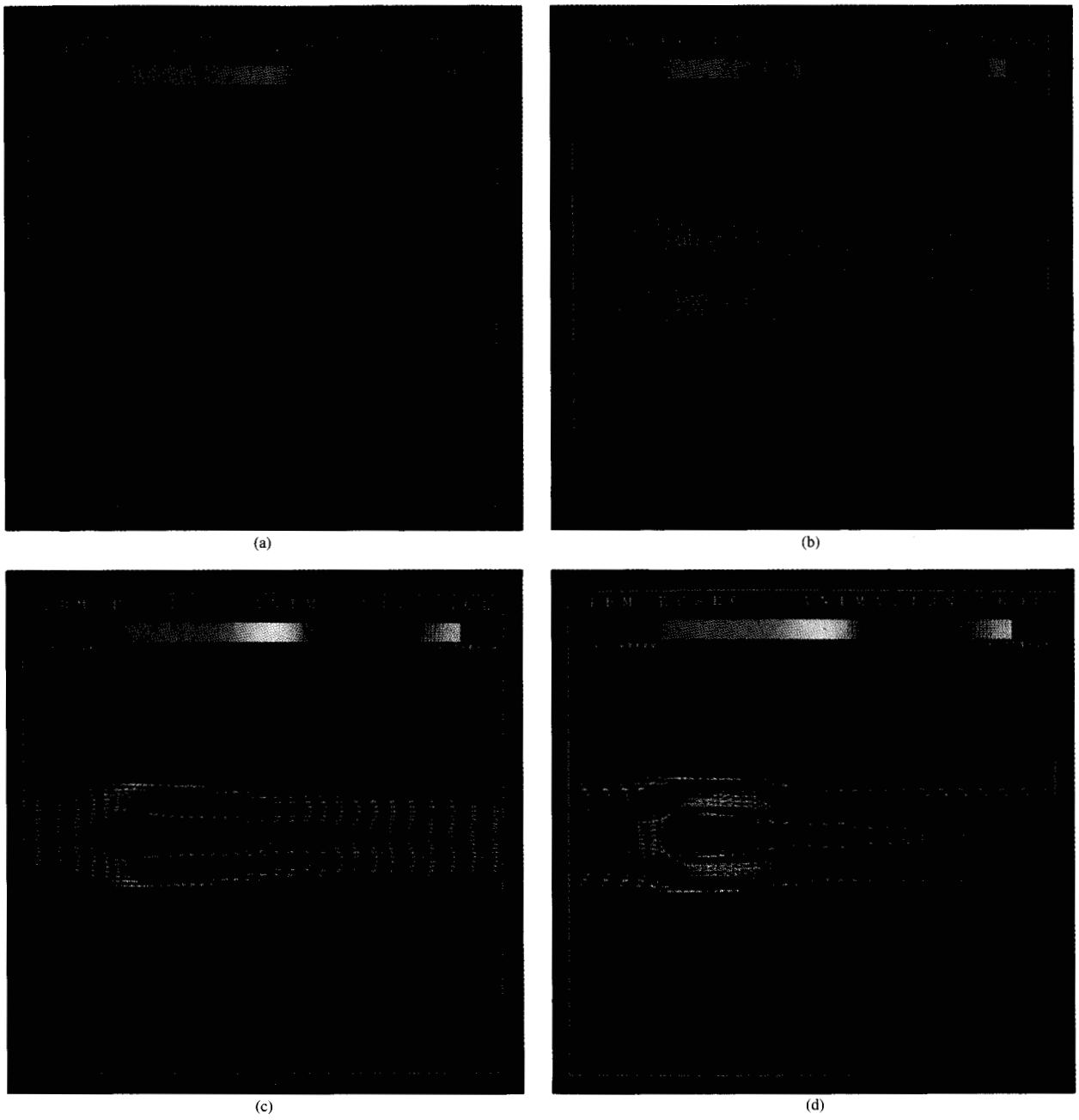



**Figure 12**

Velocity field for the fluid flow past a flat plate in a pipe. The time-steps shown are (a) the 10th, (b) the 120th, (c) the 240th, and (d) the 372nd. The color is associated with the speed and is selected from a palette of 256 colors.

and the Prandtl number, $Pr = \nu/\alpha$, where

$\rho$ = density,
$g$ = gravity acceleration,
$\beta$ = volume expansion coefficient,
$T$ = temperature,
$\mu$ = viscosity,
$\alpha$ = diffusivity, and
$\nu$ = $\mu/\rho$.

In this experiment their values are $T_0 = 0$, $\rho = 37.5$, $\beta = 1$, $\mu = 1$, $\alpha = 26.625$, and

$$f(t) = \frac{10}{3} \times t \quad \text{for} \quad 0 \le t \le 300.$$

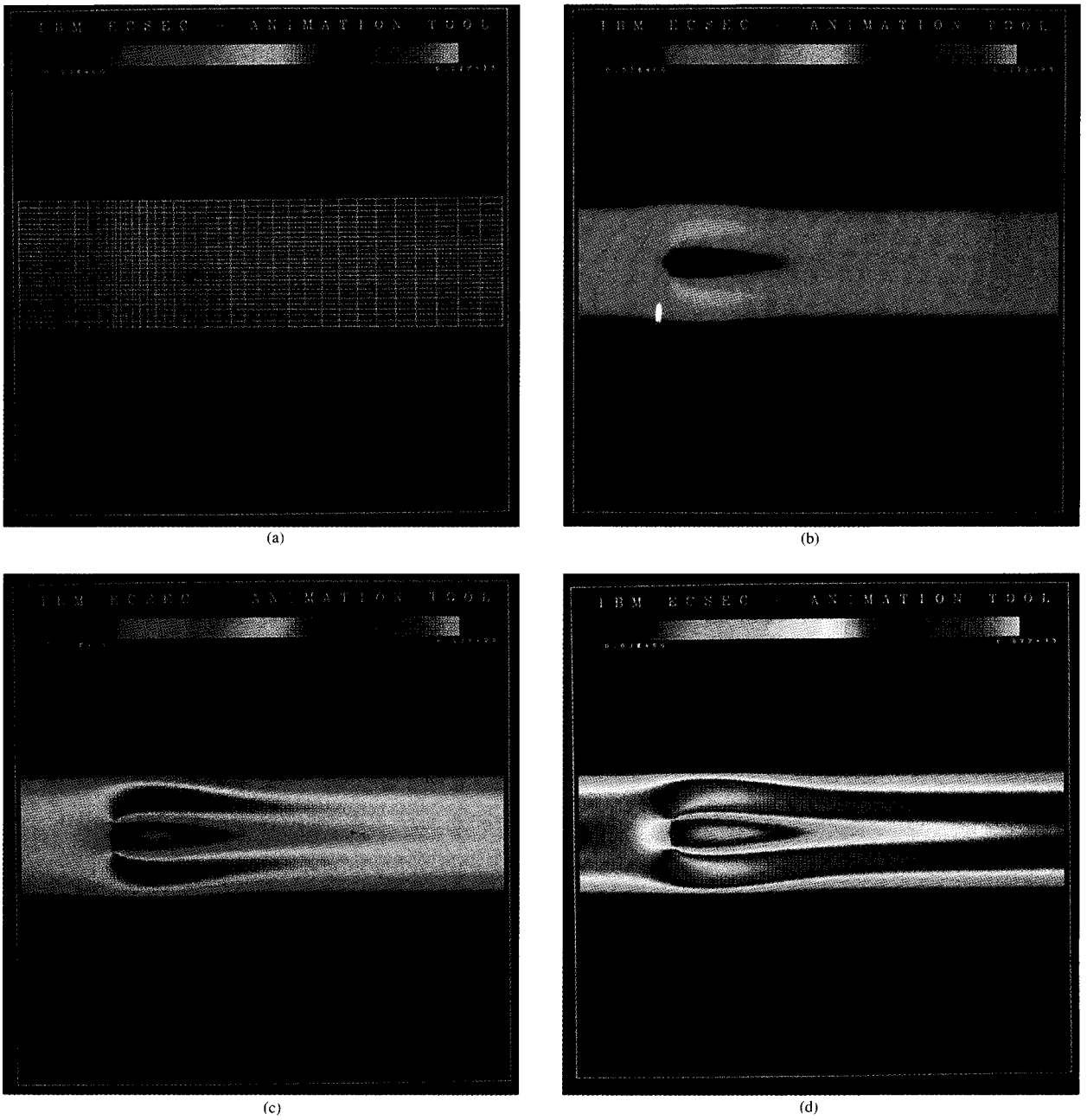



**Figure 13**

Speed field for the fluid flow past a flat plate in a pipe. The time-steps shown are (a) the 10th, (b) the 120th, (c) the 240th, and (d) the 372nd. The mesh is superimposed on the first frame.

These values give $Pr = 0.71$, $Ra = (T - T_0) \times 10^3$, i.e., $Ra_{max} = 10^6$.

To follow the boundary layer developing on the walls, it was necessary to increase the number of nodes near walls and corners relative to the number near the center of the cavity. The mesh utilized is shown in **Figure 8**.

The results are shown in **Figures 9–11**. It can be noted that after an initial transient, a central core with secondary recirculation eddies begins to develop. Moreover, velocity fields in the corners begin to be more relevant, together with a significant horizontal gradient of temperature.

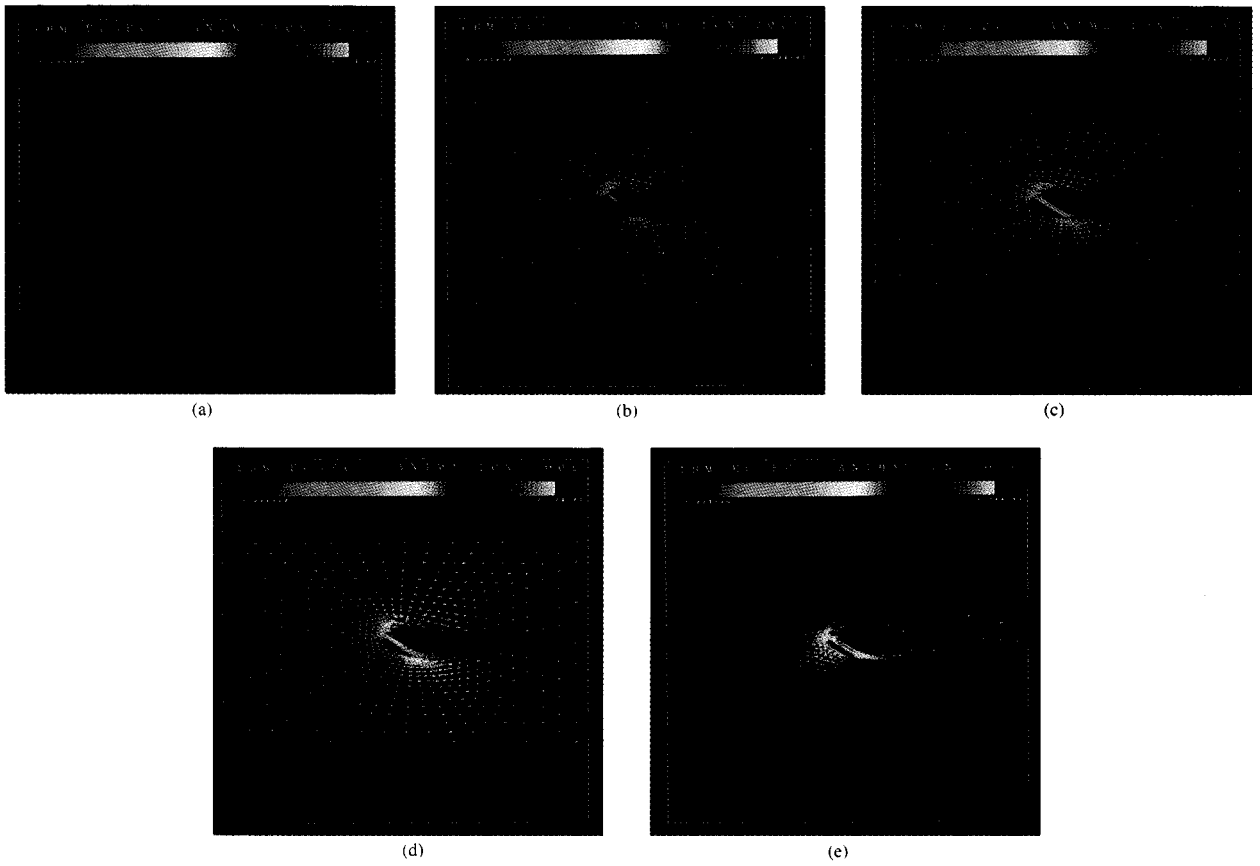



**Figure 14**

Velocity field for the fluid flow past a nozzle. The time-steps shown are (a) the 10th, (b) the 80th, (c) the 125th, (d) the 160th, and (e) the 250th. The color is associated with the speed and is selected from a palette of 256 colors.

In the second study case the fluid flows in a pipe whose geometrical dimensions are $L_x = 7$ and $L_y = 1.8$; for the plate the dimensions are $l_x = 0.04$ and $l_y = 0.36$ ($= L_y/5$). Because of the entry length of the phenomenon, the plate is placed far enough from the channel entrance to permit the flow to develop a parabolic velocity profile. The rest state has been taken as initial condition. The transient analysis has been carried out with the following boundary conditions:

- Inlet pressure field $p = f(t)$ and $v_y = 0$.
- No-slip conditions on the plate surface and on the lower and upper boundaries of the pipe.

Values for the physical parameters and for the imposed loads are $\rho = 1$, $\mu = 0.9$, $f(t) = 50t$ for $0 \leq t \leq 300$. The characteristic parameter in this case is the Reynolds number

$$Re = \frac{\rho}{\mu} lU,$$

where $l$ and $U$ are the characteristic length and velocity, respectively. In this case $l = L_y/2 = 0.9$ and $Re = U$, so at most $Re$ will be $Re_{max} = v_{max} = 267$. The results are shown in **Figures 12** and **13**; we can see that the parabolic front of the flow in the channel is broken by the plate, and two new fronts will result downstream, together with a vorticity region that grows with the velocity. Asymptotically, the parabolic front is reestablished.

In the third case we have two plates $d = 0.5$ apart and tilted at $\alpha = 30°$ from the horizontal line, thus forming a nozzle. Each plate has a length $L = 2d = 1$. The field dimensions are $L_x = 26$ and $L_y = 14$. The dynamic analysis was carried out with rest as the initial condition and with the following boundary conditions:

- Input field $v_x = f(t)$ and $v_y = 0$ on $x = 0$.
- $v_x = f(t)$, $v_y = 0$ on the problem domain frontier $y = 0$ and $y = L$.
- No-slip conditions on the plate surface.

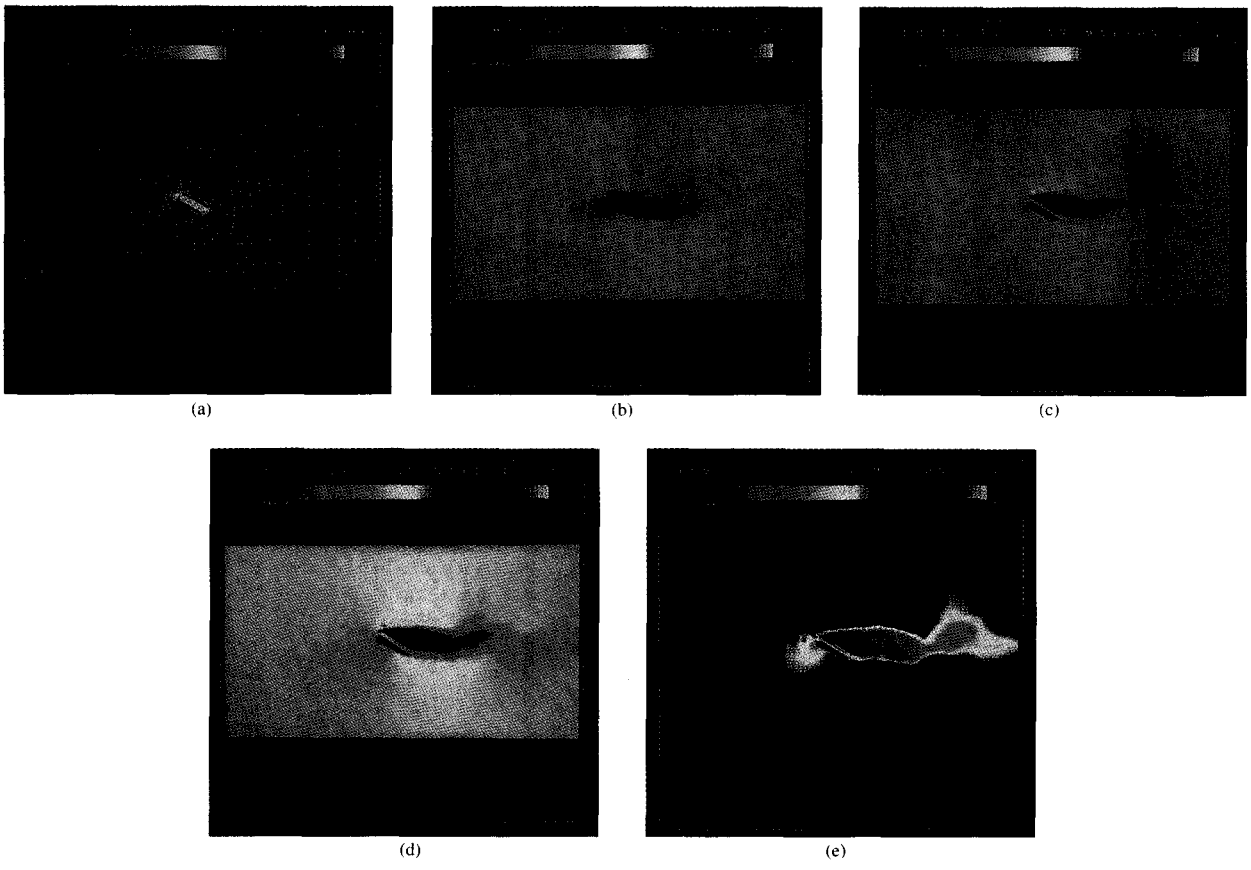



**Figure 15**

Speed field for the fluid flow past a nozzle. The time-steps shown are (a) the 10th, (b) the 80th, (c) the 125th, (d) the 160th, and (e) the 250th. The mesh is superimposed on the first frame.

For this simulation $\rho = 1$, $\mu = 0.01$, $f(t) = t/20$ for $0 \leq t \leq 200$, so that

$$Re = \frac{\rho}{\mu} lU = 50U,$$

having taken $l = d = 0.5$ as the characteristic length. The results for this case are shown in **Figures 14–16** and on the videotape. An unsteady vorticity downstream from the nozzle increases in intensity and extension with an increase in velocity load.

## Conclusions

A computational fluid dynamics laboratory such as the one established at ECSEC allows researchers and engineers to solve large problems and to analyze results by taking advantage of the animation technique. Animation is a very powerful tool for the visualization of transient phenomena because with the advent of the IBM 6090 Graphics System it is now possible to employ workstations with local buffers large enough to accommodate the byte maps necessary to represent several hundred time steps. Animated visualization is of great importance for the comprehension of complex problems because of the large volumes of data generally involved. As the problems to be solved become more complex, the need for computational power increases, and only powerful supercomputers can satisfy that requirement. Vector processors can be usefully exploited as well as parallel processors, thanks to the vector and parallel nature of the finite element method. An example of the implementation of FEM to solve Navier–Stokes equations is the ADINA-F industrial program for fluid dynamics simulation. This code benefits from the supercomputing environment set up at ECSEC because it was optimized for the IBM 3090 Vector Facility multiprocessors with Extended Architecture.

## Acknowledgments

We wish to thank Marina Russo and Fabio Matta for their support. The collaboration of William Ricketts

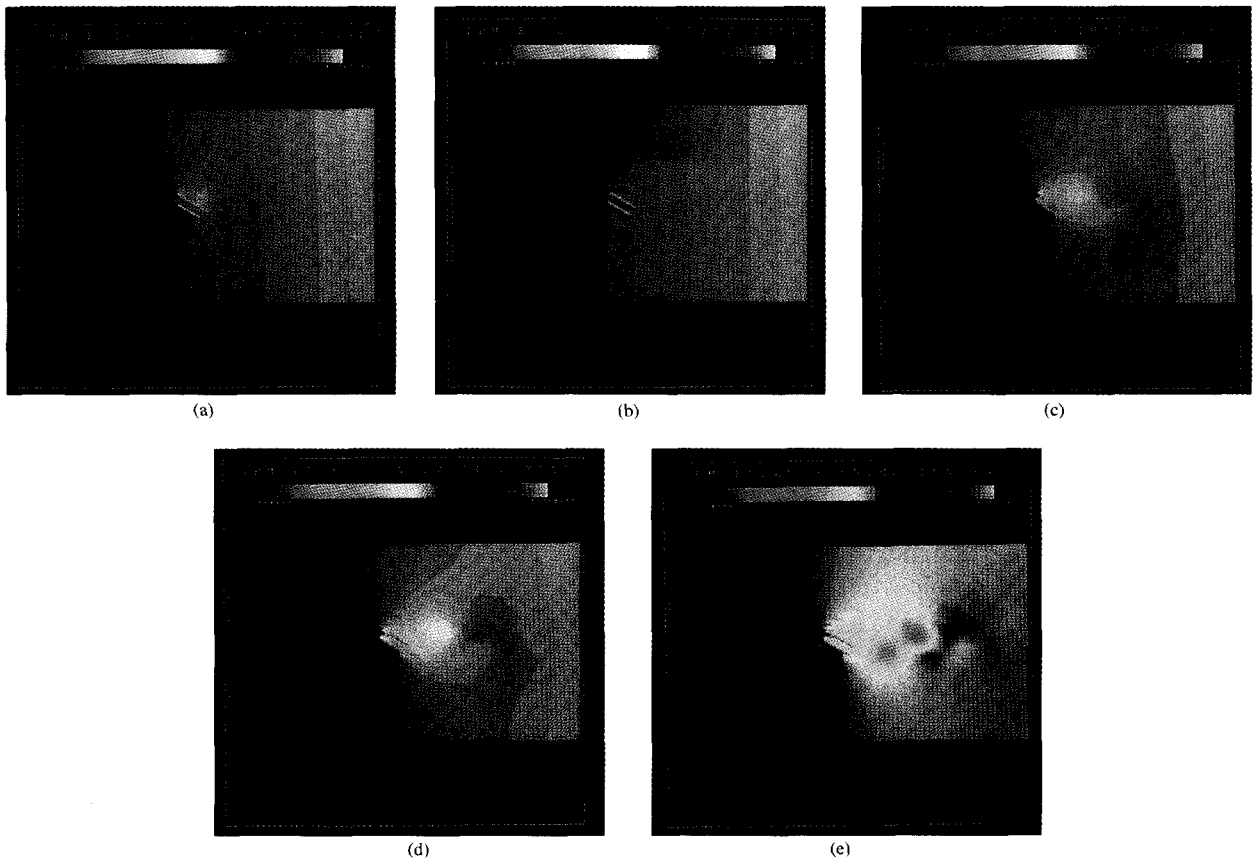



**Figure 16**

Pressure field for the fluid flow past a nozzle. The time-steps shown are (a) the 10th, (b) the 80th, (c) the 125th, (d) the 160th, and (e) the 250th.

from the IBM United Kingdom Scientific Center is also gratefully acknowledged.

## References

1. K. J. Bathe, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall Inc., Englewood Cliffs, NJ, 1961.
2. S. G. Tucker, "The IBM 3090 System: An Overview," *IBM Syst. J.* **25,** 4–19 (1986).
3. K. J. Bathe and J. Dong, "Solution of Incompressible Viscous Fluid Flow with Heat Transfer," *Comput. & Struct.* **26,** 17–31 (1987).
4. P. Angeleri, D. F. Lozupone, F. Piccolo, and J. Clinckemaillie, "PAM-CRASH on the IBM 3090/VF: An Integrated Environment for Crash Analysis," *IBM Syst. J.* **27,** 541–560 (1988).
5. H. Gouraud, "Continuous Shading of Curved Surfaces," *IEEE Trans. Computers* **20,** 623–628 (1971).
6. B. T. Phong, "Illumination for Computer Generated Images," *Commun. ACM* **18,** 311–317 (1975).
7. G. Bishop and D. M. Weimer, "Fast Phong Shading," *Computer Graph.* **20,** 103–106 (1986).
8. F. Piccolo and D. F. Lozupone, "Evoluzione della Tecnica delle Ombre per la Sintesi di Oggetti Tridimensionali," *Note di Informatica* **22,** 38–45 (1990).
9. F. Piccolo and V. Zecca, "Vectorization and Parallelization of an Industrial Fluid-Dynamical Finite Element Code on the IBM 3090/VF," *Applications of Supercomputers in Engineering: Fluid Flow and Stress Analysis Applications*, Elsevier Publishing Co., New York, 1989, pp. 121–131.

*Received November 15, 1989; accepted for publication October 19, 1990*

**Fabrizio Piccolo** *IBM Italy, European Center for Scientific and Engineering Computing (ECSEC), via Giorgione 159, 00147 Rome, Italy.* Dr. Piccolo obtained his degree in physics from the University of Rome in 1986. From 1986 to 1987 he worked as an aerospace engineer on the ESA-Remote Sensing Satellite-1. In 1987 he joined IBM at ECSEC, where he has been working on finite element method applications and on visualization of engineering code results. For his contribution to this area, Dr. Piccolo has received an Outstanding Technical Achievement Award.

**Vittorio Zecca** *IBM Italy, European Center for Scientific and Engineering Computing (ECSEC), via Giorgione 159, 00147 Rome, Italy.* Dr. Zecca obtained his degree in electronic engineering from Rome University in 1980. From 1982 to 1985 he worked in the aerospace field, with responsibility for data management for the San Marco project. In 1985 he joined the Rome Scientific Center and was one of the initiators of ECSEC; for his contribution to the area of parallel processing Dr. Zecca has received an Outstanding Technical Achievement Award. His current interest is in exploiting features of supercomputers for scientific/industrial applications, particularly vectorization, parallelization, and data in memory.

**Annamaria Grimaudo** *School for Advanced Studies in Industrial and Applied Mathematics (SASIAM), 70010 Valenzano-Bari, Italy.* Dr. Grimaudo obtained her degree in physics from the University of Bari in 1985. From 1986 to 1988 she attended the SASIAM, where she received the Certificate of Expertise in Applied Mathematics. During 1989, she was at ECSEC as a visiting scientist, working on the visualization of engineering code results. Dr. Grimaudo subsequently obtained the Certificate of Expertise in Industrial Mathematics from the SASIAM.

**Claudia Loiodice** *University of Rome "La Sapienza," P. le Aldo Moro 5, 00185 Rome, Italy.* Dr. Loiodice obtained her degree in mathematics from the University of Rome in 1989; her thesis, developed at ECSEC, was on the visualization of fluid dynamics field quantities. She is currently working in the field of numerically intensive computing applications.