An educational two-dimensional interactive dynamic grid generator

M. DARWISH, H. DIAB and **F. MOUKALLED**, Faculty of Engineering and Architecture, American University of Beirut, PO Box 11-0236, Beirut, Lebanon

Received 2nd May 1995
Revised 10th January 1996

This paper describes IDGG, an Interactive Dynamic Grid Generator, for use as an educational tool by students studying computational fluid dynamics. The package is a Windows applications and runs on IBM PC, or compatible, computers. It is written in Pascal and built using object-oriented programming. The computer program allows the user to generate boundary-fitted curvilinear grids in any two-dimensional domain. The procedure adopted requires the user to perform the transformation step by step allowing him/her to easily grasp the concept of boundary-fitted coordinate systems. In addition, IDGG may be used by CFD researchers to display results graphically in the form of vector fields, contours, and two- and three-dimensional plots. The examples provided show the effectiveness of the package as a teaching aid.

INTRODUCTION

When adopting the control volume method [1], solutions to transfer phenomena problems over irregular domains are usually obtained by solving the governing set of conservation equations using a structured boundary-fitted grid network. The main virtue of this approach over non-structured grid methods, is the simplicity of its implementation. In such a technique, the governing transport equations are first transformed into a boundary-fitted coordinate system [2] and then discretized. An algebraic equation results at each grid point in the domain and the collection of these equations form a system which is solved to get the solution field. From the above discussion, it is clear that the first step in the solution process is the discretization of the solution domain. This task, which may be cumbersome, is the main objective of this article.

The idea emerged from observing the difficulties faced by the majority of students attending a course in computational fluid dynamics (CFD). It was noticed that students could not easily grasp the concept of boundary-fitted coordinate systems. The main complexity being in understanding the transformation process. When using boundary-fitted coordinates a point $P(x, y)$ in the physical domain is transformed into $P(\xi, \eta)$ in the transformed plane and the boundaries of the domain, no matter how complicated, are considered to be lines of constant ξ and/or η. Therefore, vertical/horizontal (or coordinate lines) of constant ξ/η in the

computational space may be curves in the physical plane. The problem becomes harder to imagine when these constant ξ or η lines are closed contours in the physical plane, i.e. in cases of a branch cut in the domain. In such situations, students experienced difficulties choosing a suitable boundary-fitted coordinate system.

Based on the above argument the intention of this paper is to present a graphical numerical tool which allows the student to visualize, in a step-by-step manner, the transformation process. The approach followed is opposite to the usual one [2] in the sense that the student starts with a rectangular domain and then transforms it into the required shape. In the intermediate steps, the student grasps easily the transformation concept.

In addition, the package allows CFD users to display results graphically in the form of contour lines, vector plots, and two- and three-dimensional plots.

In what follows, the environment specification of the program is first overviewed followed by a brief description of the numerical method employed in generating the boundary-fitted grid system. Then, a thorough discussion of the package utilities and special features is given. Finally, illustrative examples demonstrating the capabilities of the package and its virtues in the learning–teaching process are presented.

ENVIRONMENT SPECIFICATION

The program (IDGG) is built using object-oriented programming (OOP). The object-oriented paradigm [3–13] has revolutionalized the software development process. This is so because, the concepts of objects and classes are considered more natural than data and processes. Objects are collections of operations that share a state. The operations determine the messages (calls) to which the objects can respond, while the shared state is hidden from the outside world and is accessible only to the object's operations (Fig. 1). OOP reduces the complexity of simulation models through encapsulation and hierarchical structuring of components. These features are particularly relevant to simulations for several reasons: (i) it allows the use of a consistent organizing framework for the physical world, the conceptual model, and the software implementation; (ii) the availability of classes and inheritance allows a developer to focus only on the essential details of the model; and (ii) modularity localizes the effects of any necessary modification to the simulation software by allowing the developer to concentrate only on the essential details of modification.

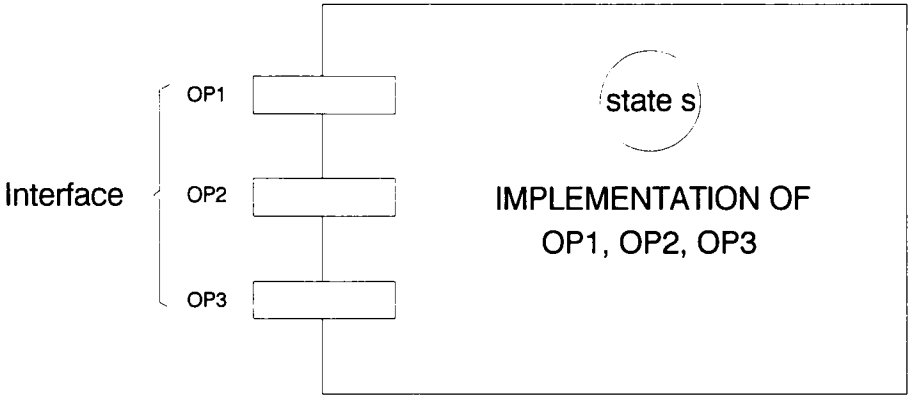

Fig. 1. Object modules.

An 80386 IBM PC, or compatible, can be used to run the package. The system should contain a minimum of 2 MB of main memory and an SVGA card. The main memory should accommodate the large heap and stack in order to store the variables that are used by the program, especially those that are employed by the three-dimensional graphs where a large dynamic array instead of a static one is used.

IDGG is developed using Turbo Pascal for Windows Version 1.5 (TPW 1.5) and hence, it exploits the Windows environment [14] to enhance programming efficiency.

MESH GENERATION

The domains of interest in this work are two-dimensional and bounded, i.e. their boundaries are well-defined. Therefore, the adopted grid generation technique should reflect and exploit this property. Several techniques can be used for generating grid systems within such spaces. The most commonly employed ones are the differential and algebraic grid generation methods [2].

IDGG is a dynamic grid generator in the sense that a new grid is automatically generated and displayed on the screen with any movement of any grid point in the domain. Thus, it should be relatively fast in order not to bore the user. This eliminates the possibility of using differential methods for grid generation which require a lot of computations since they are based on Poisson-type equations and necessitate the solution of algebraic systems of equations. As such, the most logical choice is an algebraic method.

A class of algebraic methods known as 'Transfinite mappings' or 'Transfinite interpolations' exist and many have found applications in finite element methods [15–17] for grid generation in two and three dimensions. The transformation implemented in IDGG is known as the 'bilinear' transformation. The mapping is done by using a bilinear shape function very similar to a finite element shape function. However, a special procedure is used to calculate the local coordinates of each internal grid point as a weighted function of the boundary points that are along its local coordinate, where the local coordinates are obtained from a linear interpolation procedure between the opposite boundary nodes for the east–west and north–south boundaries, respectively (Fig., 2).

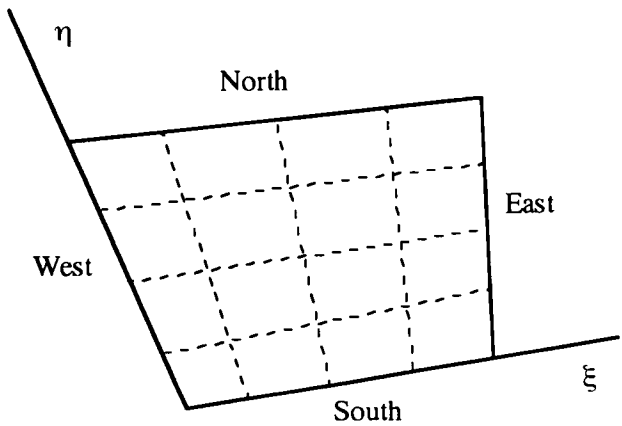

Fig. 2. An example of a boundary-fitted coordinate system.

The method requires that all the physical coordinates of the nodes on all four boundaries ($\eta = \eta_{\min}$, $\eta = \eta_{\max}$, $\xi = \xi_{\min}$, $\xi = \xi_{\max}$) of the solution domain be given. The proportionality factors are formed using the equations

$$f^{\xi}_{i,1} = \frac{x_{i,1} - x_{1,1}}{x_{ni,1} - x_{1,1}} \quad f^{\xi}_{i,nj} = \frac{x_{i,nj} - x_{1,nj}}{x_{ni,nj} - x_{1,nj}} \tag{1}$$

$$f^{\eta}_{1,j} = \frac{y_{1,j} - y_{1,1}}{y_{1,nj} - y_{1,1}} \quad f^{\eta}_{ni,j} = \frac{y_{ni,j} - y_{ni,1}}{y_{ni,nj} - y_{ni,1}} \tag{2}$$

which describe the distribution of the boundary grid points. The coordinates of the interior grid points are then calculated from the following interpolation formulae

$$x_{i,j} = x_{1,j} + f^{\xi}_{i,j}(x_{ni,j} - x_{1,j}) \tag{3}$$

$$y_{i,j} = y_{i,1} + f^{\xi}_{i,j}(y_{i,nj} - y_{i,1}) \tag{4}$$

where the values of functions f^{ξ} and f^{η} for the interior points are calculated by interpolating between the two boundary functions defined in the previous equations. The interpolation practice employed is as follows

$$f^{\xi}_{i,j} = \frac{(nj - j)f^{\xi}_{i,1} + (j-1)f^{\xi}_{i,nj}}{nj - 1} \tag{5}$$

$$f^{\eta}_{i,j} = \frac{(ni - i)f^{\eta}_{1,i} + (i-1)f^{\eta}_{ni,j}}{ni - 1} \tag{6}$$

This method of grid generation is very fast since it does not require the solution of algebraic systems of equations and is well-suited for the current application.

WINDOW RESOURCES

The Windows resources used include accelerator keys, bitmaps, cursors, dialog boxes, icons, menus and strings. Most resources can be defined by a resource script, which is an ASCII text file that defines the attributes of a resource. The easiest way to create a resource is to use a resource workshop which is separated into editors that allow the creation and modification of the resource in a program.

The key method used by most Windows programs to receive commands from the user are menus. Using menus in programs is a two-step process. First, the menu is created by a menu editor. Second, the source code is modified to respond to menu commands. Dialog boxes provide the user with a convenient way to enter data in an application. A dialog is itself a window that contains child windows called dialog box controls, i.e. pushbuttons, radio buttons, checkboxes, combination boxes, static text fields, group boxes, list boxes, scroll bars, icons, and edit boxes. Furthermore, the interface classes provided by Turbo Pascal were used to accelerate the development process. These classes encapsulate the basic behaviour of the Windows system such as windows, dialog boxes, documents, applications, etc.

OBJECT-ORIENTED DESIGN DESCRIPTION

Object declaration

The objects used (Fig. 3) in the package include:

(a) TMAINWINDOW: this object initializes the setting of the program by calling the resource files and defining the attributes of the main window. It receives all the menu commands and passes them to the subwindow.
(b) TSUBWINDOW: this object is responsible for defining the boundaries of the client area. It receives all the mouse commands from the defined window and takes the proper actions by initializing the necessary objects and calling the proper methods of those objects.
(c) TDRAWBOUNDARY: this object mainly performs four main functions: (i) defining and controlling the boundary of the generated mesh; (ii) drawing the corresponding mesh; (iii) consolidating and calculating the internal points of the mesh; and (iv) reading and drawing a mesh from a file.
(d) TFIELD: this object performs three main functions [18]: (i) it draws the contour plots of the corresponding file; (ii) it draws three-dimensional plots—using the z-depth algorithm—after performing the necessary rotations and point sorting in a file; and (iii) it draws two-dimensional plots.
(e) TVECTOR: this object performs a single function that draws vector plots of a previously selected file.

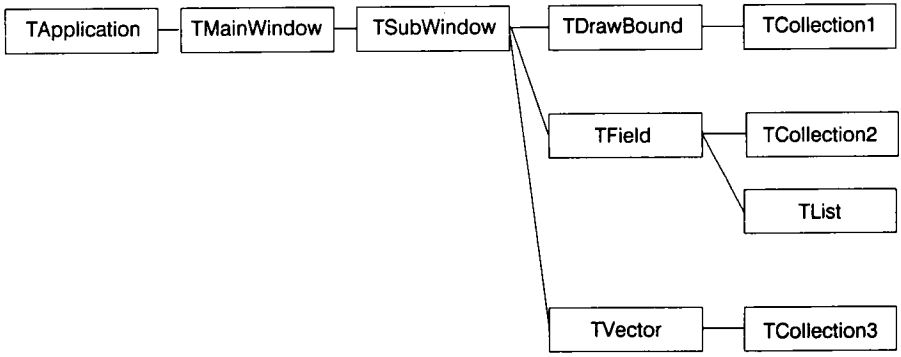

Fig. 3. Objects used in IDGG.

File structure

As mentioned before, the main feature of the package is the dynamic grid generator. However, other features are included in IDGG and depending on its usage, several types of files (mesh, contour, vector, or fluid files) may be created or read by the program. Therefore, to be able to properly employ it, the user should know the structure of the various types of files. These are as follows:

(1) For mesh files (name.msh), the file contains the data in the following sequence:
   (a) ni and nj (number of grid points in the ξ and η directions, respectively).
   (b) the x-coordinate of all grid points.
   (c) the y-coordinate of all grid points.

(2) For contour files (name.con), the file contains the data in the following sequence:
   (a) ni and nj.
   (b) the x-coordinate of all grid points.
   (c) the y-coordinate of all grid points.
   (d) the z values at all grid points.

(3) For vector files (name.vec), the file contains the data in the following sequence:
   (a) ni and nj.
   (b) the x-coordinate of all grid points.
   (c) the y-coordinate of all grid points.
   (d) the u values at all grid points, where u is the x-component of the velocity vector.
   (e) the v values at all grid points, where v is the y-component of the velocity vector.

(4) For fluid files (name.fld), the file contains the data in the following sequence:
   (a) ni and nj.
   (b) the x-coordinate of all grid points.
   (c) the y-coordinate of all grid points.
   (d) the u values at all grid points.
   (e) the v values at all grid points.
   (f) the z values at all grid points.

DESCRIPTION OF PROGRAM OPERATION

The hierarchical menu structure is shown in Fig. 4. IDGG is executed by double-clicking on its Windows icon. The main menu appears on the screen (Fig. 4) offering the following six entries: File, Edit, MeshPlots, Contours, 3D-Graphs, and VectorPlots. Any of these entries offers several options and may be selected by pointing at it with the mouse cursor and clicking the mouse button.

By selecting the first entry (File), the user may open a new file, load a previously saved one, save and print data, or exit the program. The Edit entry is used to clear the screen and to draw additional contour lines over already-drawn ones. By selecting MeshPlots, the user is offered three sub-entries which allow him/her to start a new mesh, generate a mesh, or consolidate the mesh before saving it. To draw contours, the Contours option should be chosen offering five sub-entries. By selecting sub-entry Selection, a window appears on the screen allowing the user to choose between uniform or selected contours. The number and values of contours to be drawn is entered through the Input option. The user may assign a specific colour to a contour in sub-entry Palette. Contours are drawn by choosing the DrawCont option. If the student wish to plot the variation of a variable along a line of constant x or y, then he/she has to choose sub-entry 2D-Graphs. Vector plots are plotted by selecting DrawVec under VectorPlots. Three-dimensional plots are performed in entry 3D-Graphs which gives the user flexibility in rotating the graph to see it from different angles with or without shading. The next section will show how easy it is to generate a grid and to use the various options offered by IDGG.

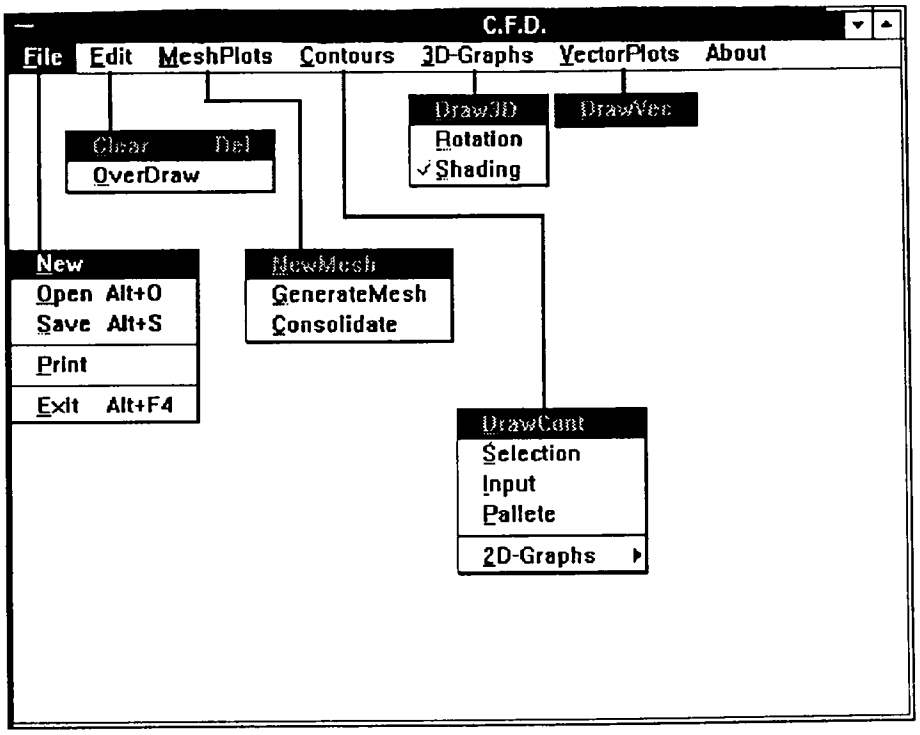

Fig. 4. Hierarchical menu structure of IDGG.

REPRESENTATIVE GRAPHICAL PLOTS

In this section, several case studies are presented which demonstrate mesh generation, contour plots, vector plots, and two-dimensional plots.

Grid generation is illustrated through two examples: grids around a car (Fig. 5) and around an airfoil (Fig. 6). However, discussion on how to generate a grid is given in general terms and clarified by reference to the example problems.

To generate a new grid, the File option should be chosen. Since the grid is new, the New Sub-entry should be selected. By so doing, a bordered area appears on the screen within which the grid is to be generated. On the top of this area, appear the Screen and World coordinates which automatically give the location of the cursor. By using the mouse, a rectangle can be drawn in that region with a preassigned number of grid points in both x and y directions (Figs 5(a) and 6(a)). The number of grid points to be used may be changed from its default value from sub-entry NewMesh under entry MeshPlots. After generating the initial rectangular mesh (Figs 5(b) and 6(b)), the transformation of this rectangular shape into the required shape may be started. This is done by simply moving highlighted points in the domain to the desired position by monitoring the 'world coordinates' displayed on the top of the screen. After any movement, a new grid is automatically and dynamically generated and displayed (Figs 5(c) and 6(c)). The process is instantaneous and the user will not notice the time taken to perform it. The student distorts the original rectangular region step by step until

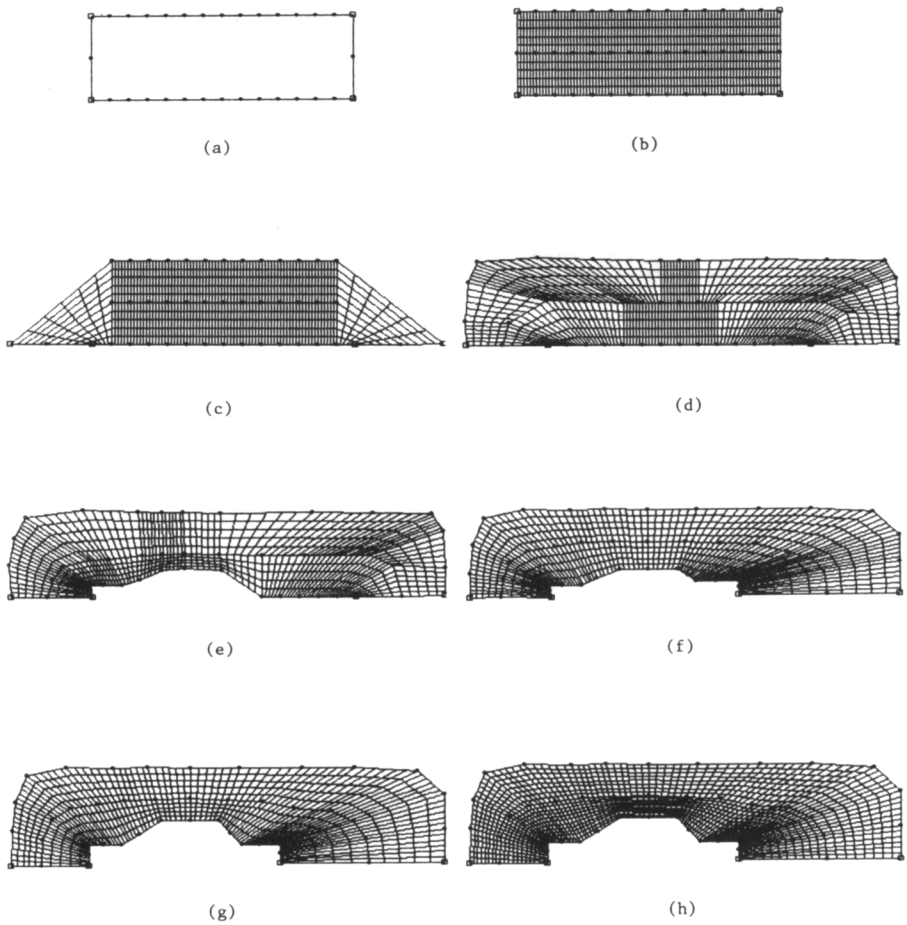

Fig. 5. Grid generated around a car.

the required shape is obtained (Figs 5(c)–(g) and 6(c)–(g)). In the intermediate steps the user learns how the transformation is taking place. With repetition, the process will be familiar to him/her and he/she will be able to plan in his/her mind the intermediate steps needed. Without noticing, the user will become familiar with the concept of boundary-fitted co-ordinates and he/she will be able to generate curvilinear grids and easily solve problems in complex geometry. The program also allows one to increase the number of grid points in any region (Figs 5(h) and 6(h)) and at any time during the transformation process. Since it is easier to manipulate fewer grid points, users should start with a small number of grid points and obtain the required shape. Then, the number of points in specific regions may be increased by double-clicking the mouse in these regions. Since, as mentioned earlier, the user may change the location of interior points, he/she can make sure by so doing, that the grid is not too skew. Moreover, it should be stated here that if the shape of the physical domain is complicated, then perfect orthogonality of the grid cannot be achieved even with the best available methods that are much more expensive than the one proposed here. The

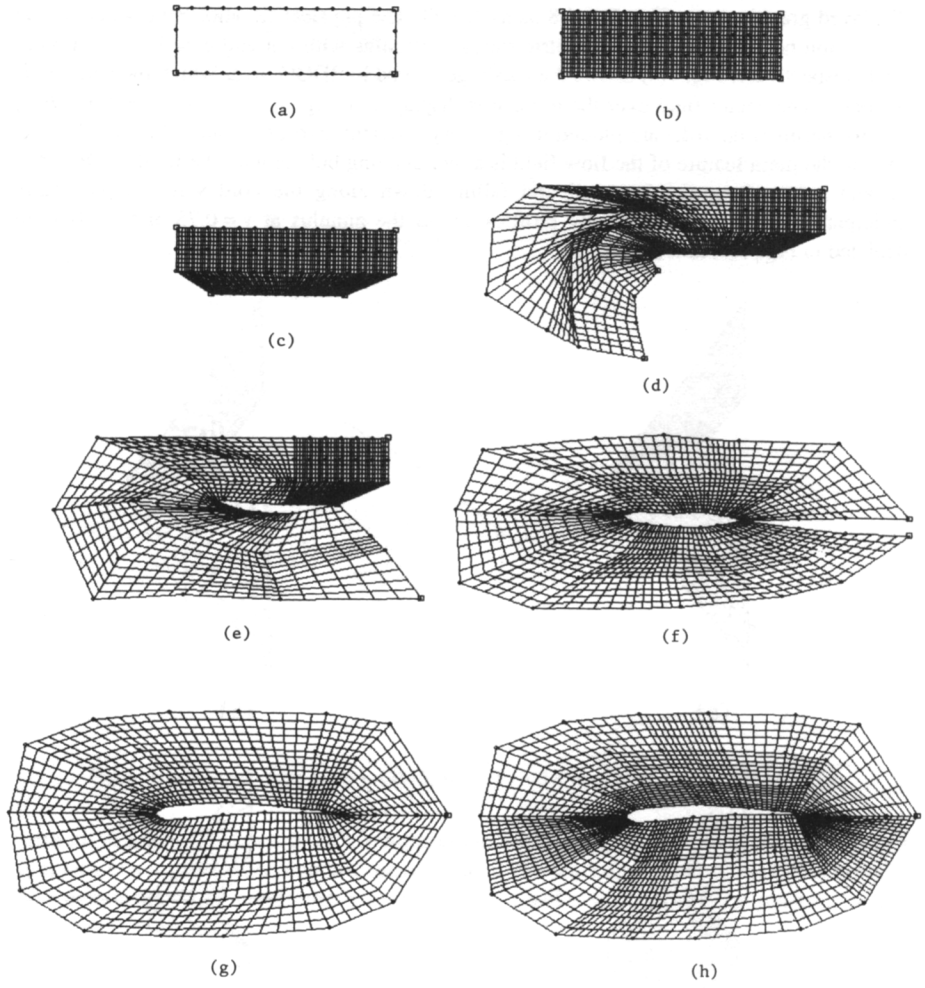

Fig. 6. Grid generated around an airfoil.

technique presented in this paper offers the user complete manual control (by selecting a grid point and moving it by the mouse) of orthogonality and smoothness of the grid. Therefore, even though the program is menu-driven, the deformation and transformation of the initial simple rectangular grid into a boundary-fitted grid describing complex geometries is done manually. For the user, this is equivalent to deforming a piece of plasticine of rectangular shape into a desired complicated shape with the hands. This is what the package offers the user. Furthermore, it should be clarified that the program is not intended to be used as a sophisticated grid generator. Rather, it is developed to help new CFD workers to understand the concept of boundary-fitted coordinate systems.

As mentioned earlier, IDGG may also be used as a post-processor by CFD workers. This is not the main objective of the program but can be considered as an additional feature. To demonstrate its capabilities, results of computations using a CFD code are analysed and

displayed graphically in Figs 7 and 8 using IDGG. The physical situation represents natural convection heat transfer in an eccentric rhombic annulus with hot and cold inner and outer walls respectively. Fig. 7(a) shows the mesh, generated by IDGG, over which the solution is obtained. The vector field over the domain is displayed in Fig. 7(b). Streamlines (Fig. 7(c)) and isotherms (Fig. 7(d)) are plotted using a very efficient algorithm due to Stezler [19]. As shown, the main feature of the flow field is a recirculating bubble with the fluid (assumed to be air) rising along the hot wall and falling down along the cold wall, as physically anticipated. Finally, the v-velocity profiles across the annulus at $y = 0.75$ and $y = 0.5$ are depicted in Figs 8(a) and (b).

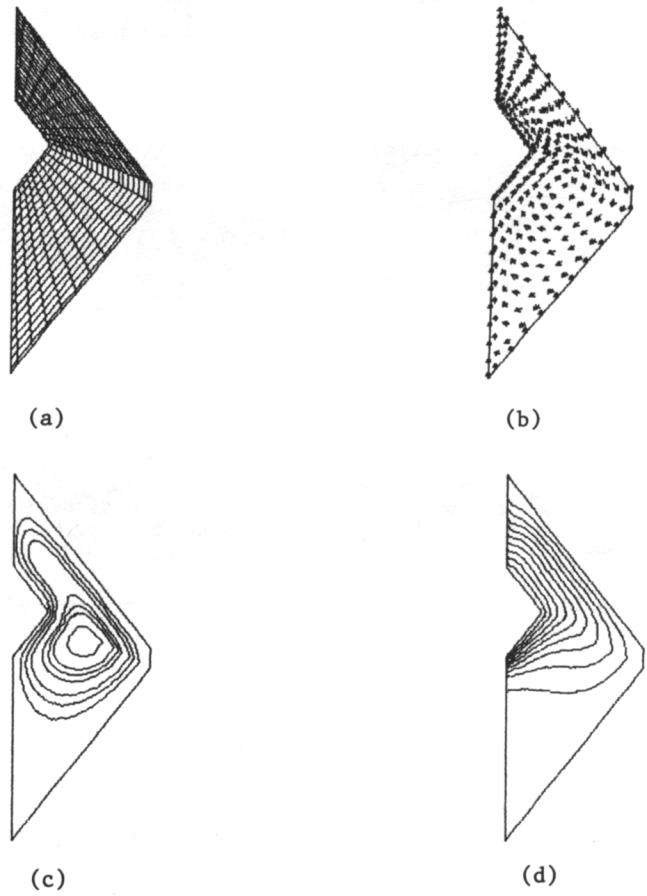

Fig. 7. (a) Coordinate lines, (b) vector plots, (c) streamlines, and (d) isotherms in an eccentric rhombic annulus.

CONCLUSION

An educational graphical tool for grid generation and analyses of transfer phenomena results was presented. The package is an application of OOP to CFD modelling. It provides the user

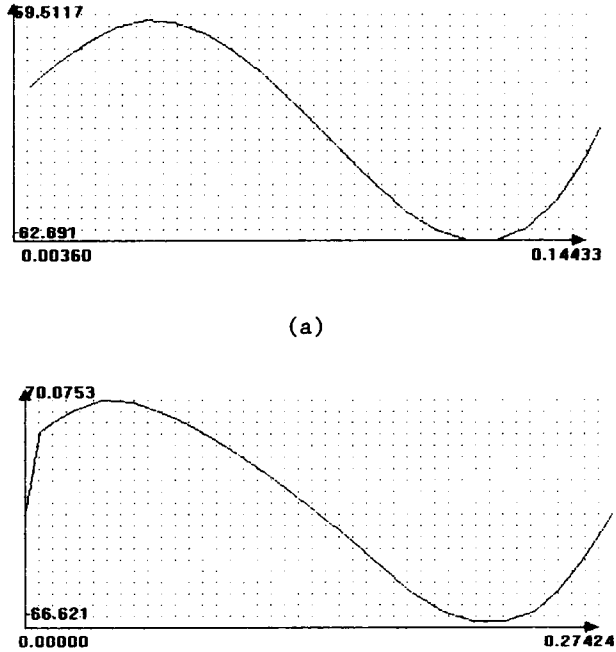

Fig. 8. v-velocity profiles at (a) $y = 0.75$, and (b) $y = 0.5$ in an eccentric rhombic annulus.

with an efficient tool on a low-cost workstation and facilitates two- and three-dimensional graphics in a user-friendly Windows environment. The teaching effectiveness of the program was demonstrated by presenting several example problems. Copies of the program will be provided to users on request to the authors.

ACKNOWLEDGEMENT

The authors would like to thank Hassan Shaar, Issam Bazzi, Abdel-Raouf Kreidly and Yousef Nizam for the computations they have carried out.

REFERENCES

[1] Patankar, S. V., *Numerical Heat Transfer and Fluid Flow*, New York, Hemisphere Publishing Corporation, 1980.
[2] Thompson, J. F., *Numerical Grid Generation*, North Holland, New York, 1982, pp. 171–192.
[3] Wegner, P., 'The object-oriented classification paradigm', *Research Directions in Object-Oriented Programming*, MIT Press, Cambridge, MA, 1987.
[4] Wegner, P., 'Concepts of paradigms of object-oriented programming', *OOPS Messenger*, **1**(1), 211–219 (1990).
[5] Cox, B., 'Message/object programming: an evolutionary change in programming technology', *IEEE Software*, **1** (1), 50–61 (1984).

[6] Wegner, P., 'Dimensions of object-oriented modeling', *Computer*, **25**(10), 12–21 (1992).
[7] Booch, G., 'Object-oriented development', *IEEE Trans. Software Eng.*, **SE-12**(2), (1986).
[8] Sharble, R., and Cohen, S., 'The object-oriented brewery: a comparison of two object-oriented development methods', *Software Engineering Notes*, **18**(2), 60–73 (1993).
[9] Shlaer, S., and Mellor, S., 'Understanding object-oriented analysis', *Design Center Magazine*, Hewlett-Packard Company, January 1989.
[10] Champeaux, D., Lea, D., and Faure, P., 'The process of object-oriented design', Conf. Proc. on Object-Oriented Programming Systems, Languages and Applications, Vancouver, Canada, 18–22 October 1992.
[11] Rine, D., and Bhargava, B., 'Object-oriented computing', *Computer*, **25**(10), 6–11 (1992).
[12] Meyer, B., 'The case of object-oriented design', *IEEE Software*, **4**(2), 50–64 (1987).
[13] Ozden, M., 'Graphical programming of simulation models in an object-oriented environment', *Simulation*, **56**(2), 104–118 (1991).
[14] Microsoft Windows, *User's Guide*, 1992 edn.
[15] Gordon, W. J., and Hall, C. A., 'Construction of curvilinear systems and applications to mesh generation', *Int. J. Num. Methods Eng.*, **7**, 461–477 (1973).
[16] Haber, R., Shephard, M. S., Abel, J. F., Gallagher, R. H., and Greenberg, D. P., 'A general two-dimensional, graphical finite element pre-processor utilizing discrete transfinite mappings', *Int. J. Num. Methods Eng.*, **17**, 1015–1044 (1981).
[17] Gordon, W. J., and Thiel, L. C., 'Transinite mappings and their applications to grid generation', in Thompson, J. F. (Ed.), *Numerical Grid Generation*, North Holland, New York, 1982, pp. 171–192.
[18] Foley, J. D., Van Dam, A., Feiner, S., and Hugher, J., *Computer Graphics: Principles and Practice*, 2nd edn, Addison-Wesley, New York, 1990.
[19] Stezler, J. F., 'Plotting of contours in a natural way', *Int. J. Num. Methods Eng.*, **24**, 131–138 (1987).