CAMP: a Configurable Application Manager Program for Groundwater Flow and Transport Codes

CRS4
Environment Group
Technical Report

Maria Laura Clemente and Claudio Gallo

September 1998

Contents

List of Figures

1 Introduction

One of the problems of using large and complex numerical codes is the specification of input data and the visualization of the output. This problem is present particularly when irregular domains in three dimensions are treated. The computational meshes in which three-dimensional domains are discretized can easily contain up to hundreds of thousands of tetrahedral or prismatic elements whose properties, in some cases, must be specified on an element-wise basis. As can be easily understood, domain zonation, boundary condition imposition, initial condition imposition, etc. can be hugely time consuming if not impossible when made directly by the user.

In the last decade, an increasing effort has been devoted to providing numerical codes with user interfaces aimed at making codes easier to use. Generally, a user interface is built "around" the code and is considered just a non-reusable appendix to the computational core.

Many commercial programs have been developed for dealing with code interfacing. For example, some commercial software packages provide interface builders that contain several pre-built objects assembled in a flow-chart fashion. These objects are then associated to system functions and/or code execution. Some commercial GIS-based software packages are also available. These packages contain specific model-independent geographical information system (GIS) utilities for developing and deploying graphical user interfaces for numerical models. Some of these packages are provided with some specific subsurface hydrology computational codes, such has SUTRA (*Voss* 1984, *Souza* 1987) and MODFLOW (*McDonald and A.W. Harbaugh* 1988, *Harbaugh and McDonald* 1996) .

In this context, the CAMP (Configurable Application Manager Program) interface has been developed. CAMP is a GUI which has been developed for interfacing subsurface flow and transport finite element codes with some public-domain or commercial meshing/post-processing/visualization tools. The implementation of a new interface program, i.e. CAMP, has been preferred to acquiring one of the available commercial software packages. This was motivated by the need for a tool with specific features that could be rapidly interfaced with the codes available in the Environment Group of CRS4. CAMP couples the flexibility of a commercial software package with the capability of being configured to virtually any numerical code. The direct access to the source code and the possibility of modifying it makes CAMP a powerful tool for the set of numerical codes available in the Environment Group of CRS4 for describing subsurface flow and transport problems.

The present work presents CAMP's capabilities and features and can be considered as a user-manual.

2 CAMP - General description

2.1 CAMP configurability

The CAMP acronym contains the word "configurable" which describes one of its important features. CAMP can be customized to virtually any code by means of a configuration file. The following specifications are required:

- location and name of executable(s);

- dimension of the code;
- the archive directory(ies);
- the names of the available graphic tools;
- the names of the available graphic tools to visualize the mesh;
- the working directory(ies);
- I/O file names(only the exensions)[1];

An example of a configuration file is the following, which refers to a 2D-transport code (TRAN2D (*Gambolati et al.,* 1994)):

```
exedir         : [directory of the executable code]
exefile        : [name of the executable file, without directory]
dimension      : [dimension of application, 2D or 3D]
archivedir     : [directory of archive]
datafilesprefix: [the prefix name of the I/O files]
graph          : [directory of a graphic tool followed by the name of
                  it and the extension of every file to be visualize
                  with that tool followed by the name desired for that
                  file in the submenu of GRAPH]
graph          : [same as above, for every graphic tool]
grmesh         : [complete name of the graphic tool for the mesh]
inoutdir       : [working directory]
output         : [list of the extensions of the output files,
                  with also the dots]
[id-name for an input file (capital letters)]:
                 .[the extension file name]
[same as input]: .[the same if the easier interface is desired,
                   the extension name must be followed by the
                   keyword ''tcl'']
[same as above, input lists of input files]:
                  [the extension of every file must be followed by its
                   name of identification in capital letters]
```

The keyword “tcl” at the end of some lines is used for specifying different editing options: if “tcl” is present, the corresponding file is edited by means of a special interface window; otherwise, the file is read as simple text using the local editor (see later). Thus, CAMP can be adapted to virtually any particular flow and/or transport numerical code by modifying the configuration file and by slightly modifying the I/O files of the code under study. The configuration file is loaded at CAMP start up time so that the interface appears already customized for the code to be used.

---

[1] It is assumed that all the I/O files are composed as follows: <prefix-name>.<extension>

2.2 Tcl/Tk as the programming language of CAMP

CAMP is written using the public-domain programming library **Tcl/Tk** (*Ousterhout* 1994, *Welch* 1995) . Several reasons have led us to choose this instead of other programming libraries. Tcl/Tk is rather simple to learn and is now widely used; it is available on the World Wide Web for several platforms: UNIX (in all its versions), DOS, Windows, and Macintosh (see later for references); it can be directly interfaced with other libraries (like C, Vtk, Java, PERL, etc.) and allows an easy management of both graphical and system functions (**exec** command); there also several packages which have been developed to increasing its capabilities and its ability to interface with other software/languages.

It is to be remarked, however, that Tcl/Tk is upgraded and improved continuously, thus some very specific functions cannot be perfectly compatible between two major releases. However, this is a common deficiency regarding every widely used software, both public-domain and licensed/commercial.

2.2.1 Tcl

Tcl (*Ousterhout* 1994, *Welch* 1995) is a string-oriented language which provides high-level generic programming facilities for handling variables, procedures, conditional and repetitive loops, etc. When programming in Tcl no variable/subroutine declaration is necessary.

All variables are local by default, but they can be globalized when explicitly declared so (keyword **global**). Complex structures such as arrays of arrays and arrays of lists can be easily built and handled. Some high-level global operators are available (e.g. **foreach** statements for scanning all elements of an array). The globality of a variable (array) is checked by all its elements by defaults. There are also a lot of commands for handling strings, lists and files.

2.2.2 Tk

Tk (*Ousterhout* 1994, *Welch* 1995) is a toolkit for the X Window System which contains some preselected objects for handling windows, such as: buttons linked to commands (there can be a few kinds of buttons: simple, radio and check), entries (usually for paths), listboxes (for example, to list a certain number of files), scrollbars (to see a hidden region of a listbox), text widget (to show and edit a file or a text), canvases (to show a drawing which can be editable by mouse) and other specific graphics widgets (*Ousterhout* 1994, *Welch* 1995) . After creating a toplevel widget and the necessary objects, the **pack** geometry manager allows to choose the relative positions and distances of every element from the others.

3 How does CAMP work?

The CAMP interface, shown in Figure 1, presents itself with a main window in which some of the functionalities are already present: ARCHIVE, MESH, EDIT, RUN, GRAPH, INFO, and EXIT. Each of these label displays, when clicked, a menu in which a list of functions are made available (each of these will be described below). A brief description of the aforementioned widget is given here.

- ARCHIVE: activates the window for handling the archive of a previous set of simulations and allows for storing test cases in a compressed format.
- MESH: contains all the functionalities related to mesh generation, visualization and preparation in a format suitable for the numerical code.
- EDIT: this window allows file editing and parameter setup for the simulations.
- RUN: this menu is devoted to run, stop, and change the priority status of the loaded code.
- GRAPHS: post-processing and visualization of results in two and three dimensions.

3.1 The ARCHIVE Manager window

The archive window, when activated by the main window, appears as shown in Figure 2, while the flowchart is shown in Figure 17. The functions STORE, RETRIEVE, EDIT, and INFO are sufficiently self-explanatory. The entry on the top of the window shows the default archive directory which has been pre-selected in the configuration file. This directory can be changed by using the canvas on the bottom, which displays the directory tree. The listbox on the right shows the list of the files of the selected subdirectory. The entry on the right hand column allows to set a new path in which the current testcase can be stored. If the selected directory does not exists, it is automatically created and data are stored within it. Archive directories have been thought to include in their name the substring “arch” followed by a directory with every name. The left button of the mouse allows the user to move within the archive directory.

After choosing the target directory, *STORE* activates the copying of the input data file from the working directory to that previously selected.

The command *RETRIEVE* allows the user to retrieve I/O files from the archive directories; the archive window contains a useful feature that allows for visiting the stored data. If the file has already been created, the *EDIT INFO* button allows to edit it. This file is intended to contain a sufficiently detailed description of the selected testcase.

3.2 The MESH window

The MESH window, which is shown in Figure 3, contains all the functions for creating the computational mesh, starting by a set of boundary points (“Polygon”) edited (“Edit_File”, under Polygon menu) or drawn by mouse in here as well (“Window_Canvas”, under Polygon menu); this is schematized in Figure 20. The polygon window tool is quite interesting because it allows the user to draw a polygon (Figure 5), to move or to remove points just drawn and, finally, to save it into a file <file>.poly which is the starting step for the triangulation. In case of three-dimensional codes, the same widget allows also to choose the number of layers and the relative thicknesses, as shown in figure 5 (see below “How to edit the polygon” for more information on this regard). As presented in Figure 3, several functions for triangulation are available: Triangle, Showme, Accept, Prt_Stat, New_Stat, Options, Mesh and Random. The entry “file name” allows the user to write the name of a file containing the polygon for the mesh which is to be created from the interface or can be already present in the working directory activated. The square button on the right of filename tells the program to activate the current boundary file for starting the mesh. *ACCEPT* allows the user to compose the command to be executed when pressing *TRIANGLE* (refer to Triangle user guide for a detailed description of the mesh-generator program (*Shewchuk* 1996)). TRIANGLE runs the mesh-generator using the "file.poly" as input. Triangle produces two files: the <file>.node and the <file>.ele which contain node coordinates and elements connectivity, respectively. It is also possible to modify the mesh (quality and max area) by changing the <file>.node and the <file>.ele.

Triangle runs the mesh-generator code; "triangle" is a two-dimensional quality mesh generator and Delaunay triangulator.

By clicking "Save_Mesh" the mesh is saved in <file>.grid; now it is also possible to view the new mesh (as shown in Figure 4 for a 2D mesh, and in Figure 6 in case of a 3D mesh). Once the triangulation is performed, the button "Refiner" triggers another widget off (see Figure 7 and the flowchart in Figure 21); this widget allows to see the mesh obtained so far. The identification of different zones in the domain and the possibility of performing some mesh refinements are described below.

3.2.1 How to edit the polygon

As shown in Figure 5, the nodes defining the boundary polygon (clicking the left button of the mouse), move them (dragging with the central button of the mouse), remove them (moving the mouse onto the node you want to delete and typing 3); by the time a node is created, its coordinates appear in the listbox "coordinates". Each change in the drawing area causes the same change in this listbox. For example, if a node is moved, its coordinates are update in real time.

After creating the points that will compose the boundary of the mesh, the connectivity is to be setup. Edges between two nodes are drawn typing 1 on the first node and typing 2 on the second one. Immediately the edge is drawn, and shown in the listbox "connectivity" (by means of the numbers of the two nodes which are joined). The edges can be removed in the same way as the nodes. Some useful options are available in the command panel under the drawing area. In order to facilitate the editing of the polygon, it is possible to display a grid clicking the checkbutton near "grid off"; in the entry there is a default step, but it is possible to set another step for the grid. Near "cursor" there are the two dynamical coordinates (x and y) of the cursor. Near "geometry" there are the two pixel values of the geometry of the drawing area. It is possible to change these values typing another two and clicking APPLY. The geometry of xmin, ymin, xmax, ymax is in absolute coordinates, which are the same of the cursor and the same of the nodes displayed in the "coordinates" listbox. RESET allow to delete everything edited in the drawing area. For a better triangulation it is possible to keep some area out the triangulation by editing internal polygons with a hole into everyone of them. The total number of the holes must be selected into the appropriate cell (near "holes") and then they can be typed in the drawing area. In case of 3D mesh, it is also possible to select the number of layers (the default is 1) and, clicking the button near "layer", the widget will enlarge in order to display the layers. Furthermore, the user can choose the thicknesses, by dragging their boundary lines. After setting-up the polygon belonging to the mesh boundary, SAVE registers it onto a file <file>.poly.

3.2.2 The mesh refiner

The mesh refiner window, which is shown in Figure 7, displays the current <file>.grid created in the calling window (i.e., the MESH window). Some features are similar to the previous session, like grid, cursor, geometry, xmin, ymin, xmax, ymax, and APPLY. The flowchart is shown in Figure 21.

After setting up the suitable geometry, near "zone" it is possible to select the number of zones to be refined and, clicking the button, the widget enlarges itself allowing the editing of the areas. Before editing an area (in the same way of the polygon in the previous section) the user must select the number of the polygon he is going to draw and click the button near "edit zone n.".

It is possible to change zone configurations and their number and so clicking RESET all of them can be deleted. In the zone panel (on the right top of the window), the user can check how many points each zone has got, which characteristic is assigned to this zone (identified by its number), and select the areas that are to be refined.

The button REFINE triggers off the refinement which appears as a news mesh in the window canvas after performing some background computations (node creation, new topology, element reordering) activated so far (as seen in Figure 8). Also, it is possible to refine more than once, by resetting after every refinement of the mesh and by setting up a new zonation configuration. Once completed the mesh setup, the button SAVE dumps the new mesh in <file>.grid.

3.3 The EDIT window

The *EDIT* window (as shown in Figure 9 and schematized by flowchart in Figure 18) allows the set up of soil parameters, boundary conditions, initial conditions, etc. A list of checkbuttons linked to the input configured data files in the following order is displayed: MODEL PARAMETERS, BOUNDARY CONDITIONS, INITIAL CONDITIONS, SOIL PROPERTIES, GRID and VELOCITY FIELD. Each of these checkbuttons allows modifications of the corresponding file. Each file can be edited by means of a text editor (a Tcl/Tk built-in editor) or by means of a cell-fashion widget, where a number is specified for each parameter. The choice of the editor for each file is specified in the configuration file in a file-by-file fashion. Generally, the "Tcl" editor is recommended for short files (MODEL PARAMETERS, SOIL PROPERTIES), while the grid file is generally edited using a standard text editor such as "vi". Below an example of "tcl" and "text" editor specifications are shown (see configuration file section):

```
BOUNDARY_CONDITIONS: .bc
SOIL PROPERTIES: .matsol tcl
```

In Figure 10 "Tcl" editor layout is shown. We remark that the layout of this window is not built *a-priori*: the input file is "interpreted" depending on its structure; the Tcl editor window is built at run time, according to a restricted set of rules that input data file layout must observe. Another Tcl-interpreted file example is shown in Figure 11 (see in Appendix A the list of rules).

For each input file the following buttons are displayed: OK, RESET, OLD FILES and CLOSE. After changing a file, *OK* confirm the change (and the data are saved in an indexed file); *RESET* leave the file unchanged; *OLD* reload a previous file (indexed names distinguish old files) by means of a listbox which scan the current working directory; *STORE* creates an archive directory (see section ARCHIVE above). A particular layout is reserved for the boundary condition editor (see Figure 12) in case of 3D mesh. In this widget there is the possibility of choosing some boundary nodes (which are visible clicking "Mesh view") both for a single layer ("not replicated nodes", on the left side of the widget) and for all the layers ("replicated nodes", on the right side of the widget). First of all, the user has to edit the total number of non-replicated nodes; by typing "return" all the nodes are assigned with an "unset" string. Then the user has to edit the number of every node in 2D representation (which is shown after clicking "Mesh view"), its layer, its initial value. With "return" these data are stored and displayed in the listboxes below, where the absolute number of the node is also stored (that is, in 3D). The way to storing replicated node is very similar and it is on the right side of the same window. After setting up the values for the start up time, clicking the button "go to final time" brings you to the widget of the final time which has the same values of initial time, by default. Now it is possible to change these values and finally, clicking SAVE to store all in the file. The other functionalities are the same as the other input file widgets.

3.4 The RUN menu

The RUN menu (Figure 22) enables the user to run the code, to stop it, and to modify its priority (referred to UNIX system operator) as well. When clicking on the "run" widget, the executable is run with the input specified in the EDIT window.

3.5 The INFO window

CAMP provides information of three different levels:

- interface-general
- code dependent (variables, etc.)
- testcase dependent

The interface-general information level is aimed at providing explanations on the general features of CAMP as a sort of user manual. The code information level is related to features of the numerical code specified in the configuration file; it is currently in a text format, but it could be easily transformed in a hypertextual format. The testcase info-level is related to testcase simulations which are stored in archives directories. This info is editable by the user through the "archive manager" widget (see above).

3.6 The GRAPH menu

The window *GRAPH* (see the flowchart in Figure 19) gives the user the possibility to use some graphical tools for visualizing results generated by the simulation which has been run. After selecting the visualizer and the variable to be visualized (this is made in the configuration file), the user can select the frame to be displayed (i.e., at what time the solution is to be plotted). By clicking on OK, the plot of the requested output is displayed. In Figure 13, it is possible to see three different frames of a concentration output. Currently, everything is setup for the PLOTMTV visualizer, but not much effort is required for preparing similar routines for other visualizers.

4 An example

In this section CAMP is used for running simulations with the CODESA-3D code (*Gambolati et al.,* 1998) , a numerical simulator for modeling density-dependent flow and mass transport in porous media. A simple test case is considered: the fate of a plume of dense contaminant in a saturated aquifer (*M. L. Clemente et al.,* 1998). A three-dimensional domain 100 m long, 100 m wide and 30 m high is discretized into 758 triangular elements in the x-y plane by means of an unstructured mesh. The 2D grid has been replicated along the z direction to obtain the 2940-nodes/13644-tetrahedral element 3D-mesh. An impermeable block of clay was assumed to be present in the aquifer. This block was considered to span over the entire depth of the aquifer and it was removed from the global mesh on each layer: no-flow boundary conditions were assumed in the corresponding internal boundary. The domain was assumed to be isotropic and heterogeneous: two horizontal zones of different hydraulic conductivity were identified, the first one, from 30 to 15 m, with a value of 0.1 m/day, and the second one, from 15 m to the domain bottom, with a value of 1.0 m/day. An initial condition of uniform potential head distribution was assumed over the whole domain. A Darcy flux on the x-z plane was obtained by imposing a prescribed potential head of $h = 50$ m (Dirichlet flow boundary condition) on the left side of domain; moreover, a constant infiltration rate of 4.32×10^{-5} m/day was imposed on the top surface. The domain was assumed to be initially free of contaminant, i.e., $c(x, t = 0) = 0$, and then the contaminant entered the aquifer from a salt deposit located in a central zone at the top of the domain (Dirichlet transport boundary condition, $c(x^*, t) = 1$, with x^* belonging to the selected zone). The density ratio was $\epsilon = 0.07$, the viscosity ratio and the diffusion coefficient were set to zero, and the dispersivity coefficients were $\alpha_L = \alpha_T = 10$ m.

The boundary of the computational domain is drawn clicking the points (points can be added, cancelled, moved, etc.) with the mouse, as explained in the previous section; then the file *[filename].poly* is created and represents the starting step for the triangulation. The number of layers and relative thicknesses for the generation of the 3D computational domain can be easily imposed.

The triangulation is obtained using the Triangle command panel, and is visualized by clicking the *Show_mesh* button , as illustrated in Figure 14. By clicking the *Save_Mesh* button, the 2D grid is replicated the desired number of times and saved in file *[filename].grid.*

After generating the grid file, the CODESA-3D code was run by clicking the *RUN* button on the main window (see Figure 1). The window GRAPH enables the user to select some graphical tools (such as PLOTMTV, XMGR) for visualizing results generated by the simulation which has been run. Concentration, potential head, pressure, and velocity profiles can easily be shown at any desired time step and on each layer of the computational domain, using the frame selector panel (see Figure 15).

For example, Figure 16 represents the concentration profiles at time $T=$ 1000 days on the first, and third layer.

5 Final remarks and future developments

The CAMP user interface has been presented. CAMP capabilities have been described and the software package has been applied to the density-dependent CODESA-3D finite element simulator. Configurability is one of the most important features of CAMP and its application to a wide variety of numerical codes is relatively straightforward. CAMP customization is obtained by means of a configuration file in which paths to executables, graphic tools, archive directories, and working directories are to be specified.

Currently CAMP can handle files deriving from previous simulations or from scratch. Many improvements of the current version of CAMP could be implemented. A list of possible upgrades of CAMP includes

- Code-dependent modules loaded at run time, particularly with regard to the code-dependent info. These modules would handle also some *ad-hoc* modules for improving interface usability.

- So far CAMP works using some free-software packages available on the Internet. However, it would be useful to give the possibility to the user to export data in different formats, such as those support by CAD systems, or raster/vectorial formats used for GIS-based software packages.

- A GIS interface integrated in CAMP. All the data necessary for running the model selected are input manually by the user. However, for real field applications, raster or vectorial files are available, and it would be very useful to transfer automatically these data into model input files.

- The “INFO” for CAMP could achieve a better relevance if implemented as a hypertext that would directly give information on the selected topic.

Appendix A: CAMP configuration file

Rules for the Tcl/Tk filtering reader

Here are the rules that must be observed in the file to be edited in order to use correctly the tcl-based filter implemented in CAMP.

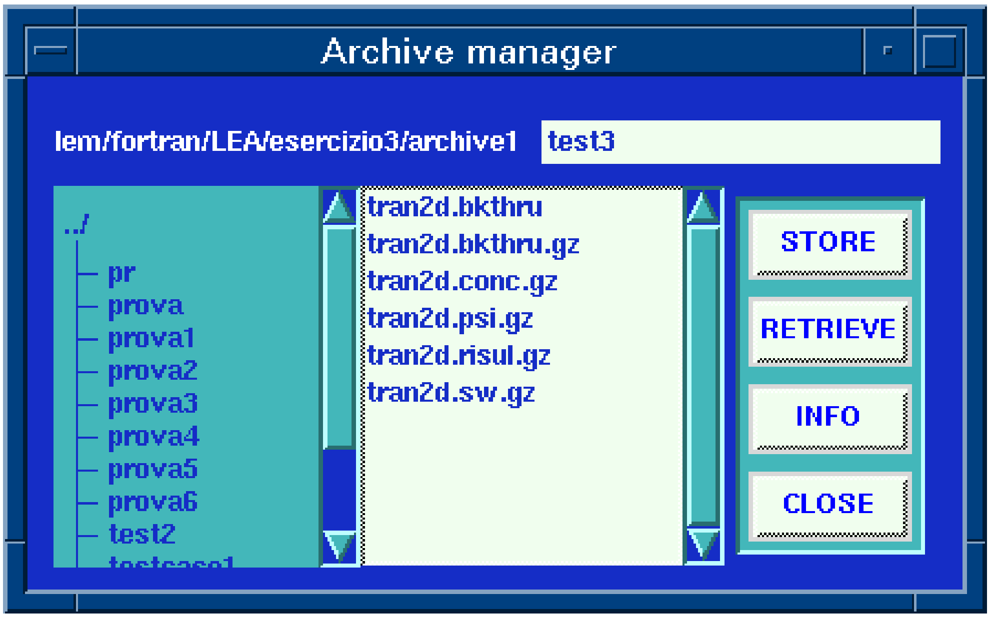
- there must be no empty rows;
- every row must have only digits, lables, or comment;
- a row of digits must be followed by a row of labels of the same number of elements;
- numbers can also be exponential or double precision;
- the comment row must begin with a "♯" character.

Configuration file sample

The following is an example of a 2D configuration file:

```
exedir:              /u/clem/fortran/LEA
exefile:             tran2d
dimension:           2D
archivedir:          /u/clem/fortran/LEA/esercizio1
datafilesprefix:     tran2d
graph:               /u/fender/bin/r6k/plotmtv plotmtv
                     conc concentration sw saturation psi pressure
graph:               /u/cxpanico/uniras/ugr unigraph sw saturation
                     bkthru breakckthrough
graph:               /u/roby/XMGR/bin/xmgr xmgr
grmesh:              /u/fender/bin/r6k/plotmtv
inoutdir:            /u/clem/fortran/LEA/esercizio1
MODEL_PARAMETERS:    .parm tcl
BOUNDARY_CONDITIONS: .bc
INITIAL_CONDITIONS:  .ic
SOIL_PROPERTIES:     .matsol tcl
GRID:                .grid
VELOCITY_FIELD:      .velelt
```

Figure 1. Main window of the CAMP program.

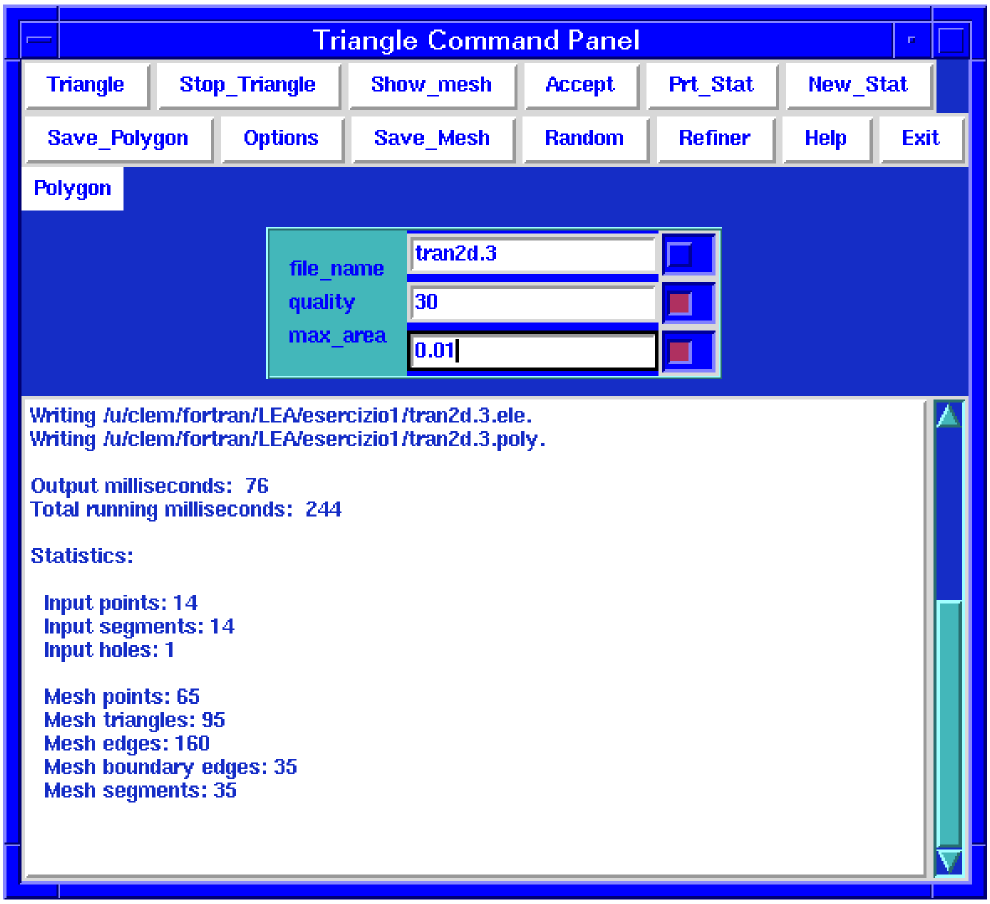


lem/fortran/LEA/esercizio3/archive1 test3

../
- pr
- prova
- prova1
- prova2
- prova3
- prova4
- prova5
- prova6
- test2

tran2d.bkthru
tran2d.bkthru.gz
tran2d.conc.gz
tran2d.psi.gz
tran2d.risul.gz
tran2d.sw.gz

Figure 2. Archive manager window of the CAMP program.



file_name tran2d.3
quality 30
max_area 0.01

```
Writing /u/clem/fortran/LEA/esercizio1/tran2d.3.ele.
Writing /u/clem/fortran/LEA/esercizio1/tran2d.3.poly.

Output milliseconds:  76
Total running milliseconds:  244

Statistics:

  Input points: 14
  Input segments: 14
  Input holes: 1

  Mesh points: 65
  Mesh triangles: 95
  Mesh edges: 160
  Mesh boundary edges: 35
  Mesh segments: 35
```

Figure 3. Triangle command panel window of the CAMP program.

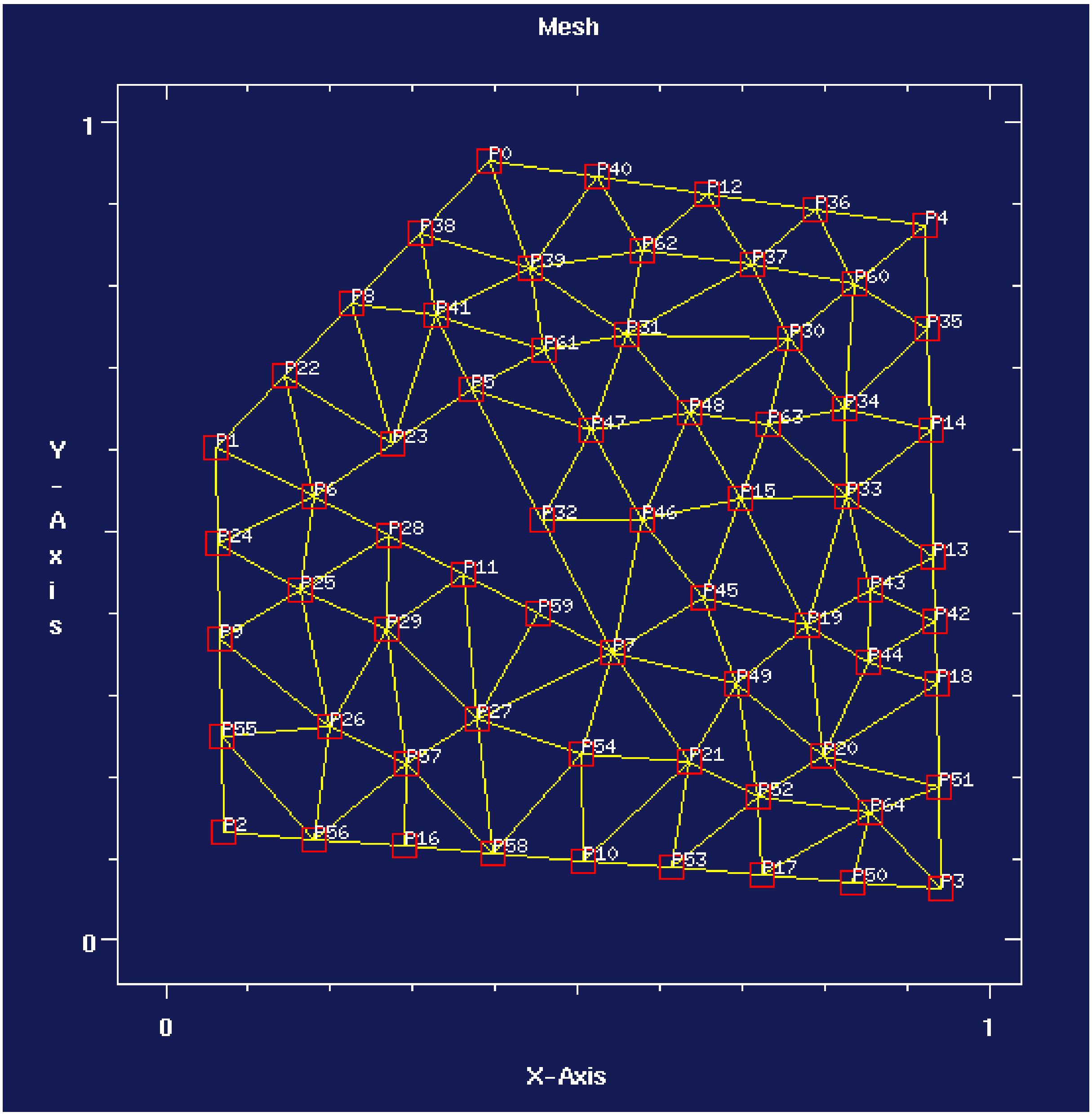
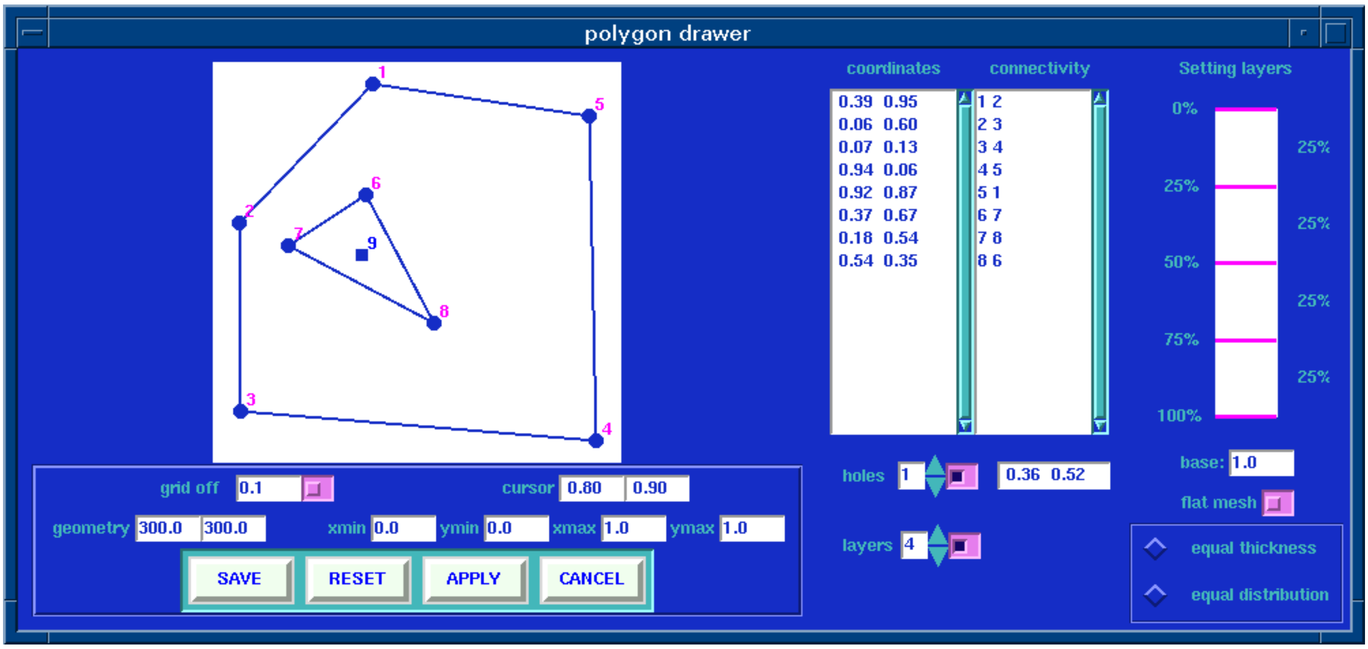


Figure 4. The mesh visualized with plotmtv.



coordinates	connectivity
0.39 0.95	1 2
0.06 0.60	2 3
0.07 0.13	3 4
0.94 0.06	4 5
0.92 0.87	5 1
0.37 0.67	6 7
0.18 0.54	7 8
0.54 0.35	8 6

grid off 0.1 ☐
cursor 0.80 0.90
geometry 300.0 300.0
xmin 0.0 ymin 0.0 xmax 1.0 ymax 1.0
holes 1 ■ 0.36 0.52
layers 4 ■
base: 1.0
flat mesh ☐
◇ equal thickness
◇ equal distribution

Figure 5. Polygon and layer drawer.

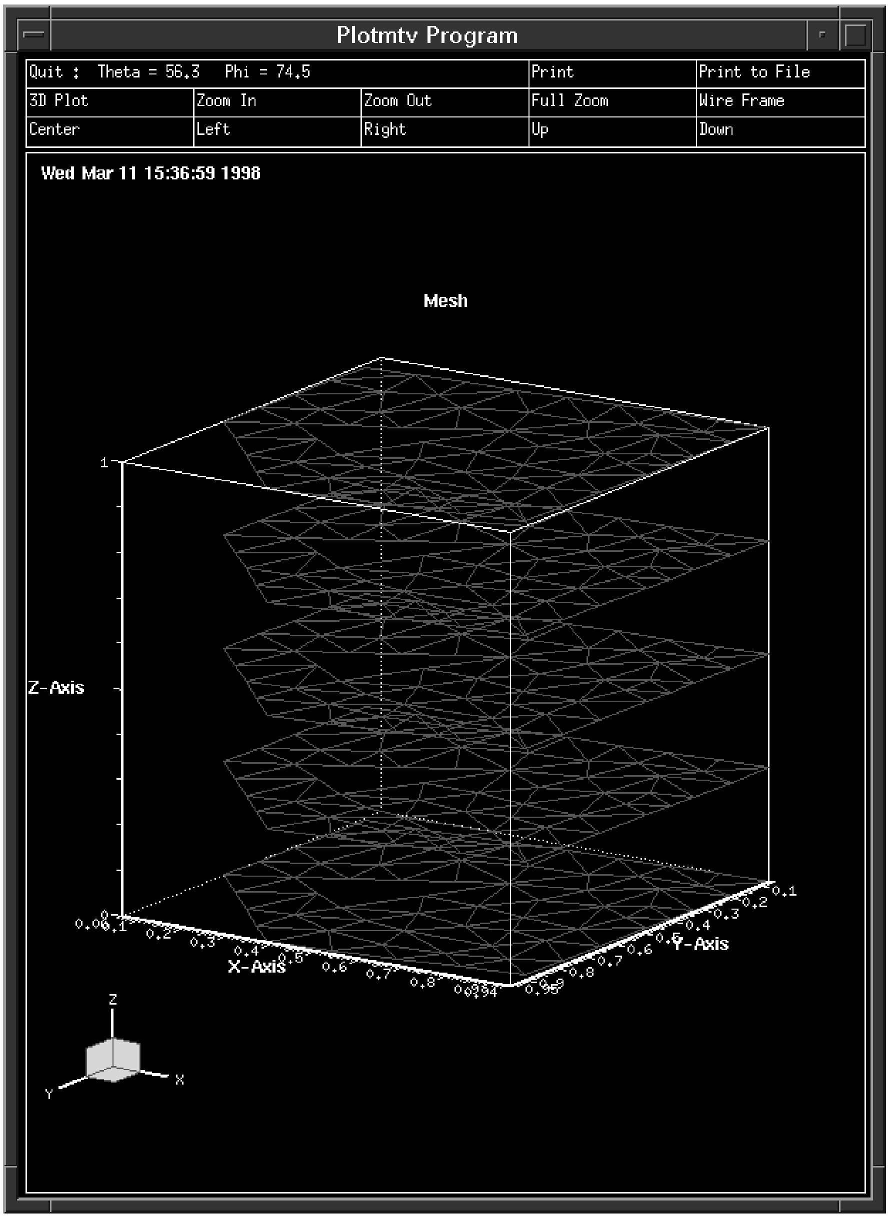
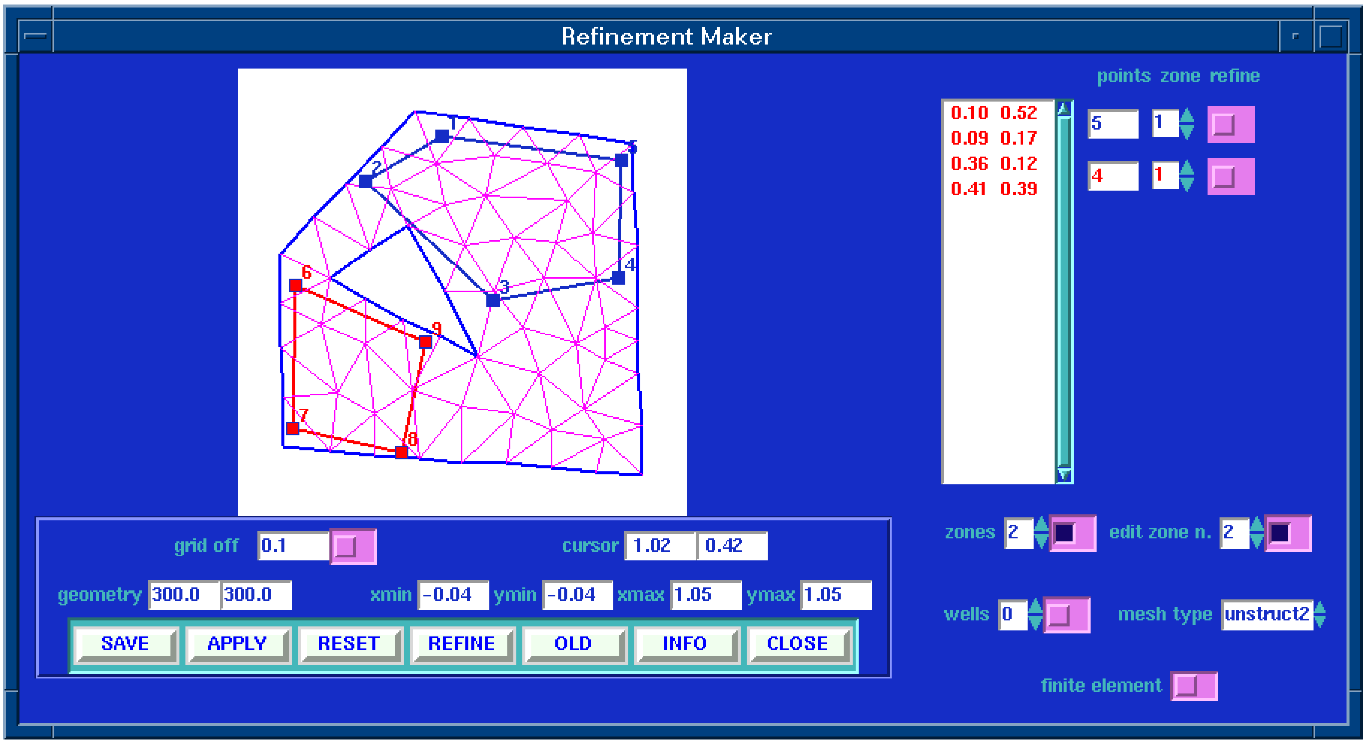


Figure 6. The mesh in 3D visualized with plotmtv.



0.10	0.52
0.09	0.17
0.36	0.12
0.41	0.39

points	zone	refine
5	1	☐
4	1	☐

grid off 0.1 ☐

cursor 1.02 0.42

geometry 300.0 300.0

xmin -0.04 ymin -0.04 xmax 1.05 ymax 1.05

zones 2 ■

edit zone n. 2 ■

wells 0 ☐

mesh type unstruct2

finite element ☐

Figure 7. Refiner widget.