A98-32468

AIAA-98-2790

The Michigan Aero Instructional Software Project — TODOR Meets the World-Wide Web

Kenneth G. Powell*
Vincent T. Coppola†
Department of Aerospace Engineering
The University of Michigan
Ann Arbor, MI 48109

April, 1998

Abstract

This paper describes an initiative begun recently in the Aerospace Engineering Department at the University of Michigan. It concerns the development of Internet-based educational software for aerospace engineering. The Java programming language is used to develop "applets" — programs that can be accessed via web browsers such as Netscape Navigator or Internet Explorer from anywhere on the world-wide web (WWW). Each applet is a laboratory-like module, in which students can learn interactively about aerodynamics, astrodynamics, and other aerospace disciplines. For instance, in one applet, students can distribute point-singularities and visualize the resulting potential flow via various plots and animations. The history behind the decision to start this project, the choice of paradigm for the software, and a list of the modules developed to date are detailed. More information on the project, and the modules themselves, can be accessed through the home page of the University of Michigan Aerospace Engineering Department's home page [1].

What Was TODOR?

In 1985, a project was begun in the Aeronautics and Astronautics Department at MIT to develop workstation-based educational software for teaching aerospace engineering. The work was funded by NSF and MIT's Project Athena, and was named TODOR[3], in honor of Theodore von Kármán. In 1988, an NSF-sponsored workshop was held to disseminate the TODOR software to other aerospace engineering departments around the country. One of the departments represented at the meeting was the Michigan Aerospace Engineering department; the TODOR software was installed at the University of Michigan in 1988, and used extensively in undergraduate classes until 1995.

TODOR was a package of software modules, all of which related to aerospace engineering. Some of the modules were tutorial in nature; others were set up more like ex-

*Associate Professor, AIAA Senior Member
†Assistant Professor, AIAA Member

perimental laboratories, in that the students provided various input parameters, and the output resulting from solving the problems the student set up were plotted on the screen. The software ran on UNIX workstations. The TODOR package included:

- POTFLOW — A potential-flow laboratory, in which students could place point singularities at various places on the screen, and visualize the resulting flow by plotting streamlines, tracking bubbles through the flow, etc.
- MULTI — A panel method for multi-element airfoils, including a prediction of incipient separation. Students chose sections, sizes and placement for slats, main elements and flaps, and "flew" the resulting airfoils, plotting the results.
- 1D NOZZLE — A quasi-one-dimensional nozzle module with variable area and back pressure/reservoir pressure ratios.

These were, at least in the experience of using TODOR in undergraduate classes at Michigan, the most useful of the modules; there were nine other modules of varying utility/popularity.

TODOR was ahead of its time, both in a good sense and in a bad sense. In the good sense, TODOR made innovative use of workstations to allow students to learn by doing. The aerodynamic modules, in particular, allowed students an experience that combined the best of analytical techniques and wind-tunnel testing. They could design an airfoil experiment, run it in a few seconds, and visualize the results using simulated smoke flow, pitot probes, etc. They could see the stream-function for complicated collections of singularities, doing in minutes what would take hours with pencil and paper. Throughout, the graphical output of the modules helped the students build their intuitive skills about fluids.

In the bad sense, TODOR was hampered in its development by the relatively crude state of graphical user interface (GUI) software available on UNIX workstations at the time. As a result, TODOR was difficult to port to new architectures, in part due to a proprietary layer of software in TODOR that handled some of the GUI issues.

Ultimately, TODOR was unmaintainable, at MIT as well as Michigan and other schools. Reasons included the difficulty of maintaining object files for multiple platforms, the proprietary layer of GUI software, the necessity of rebuilding the software each time the operating system was upgraded on any of the computers at any of the schools using the software, and the difficulties associated with distribution of the software from MIT to the other schools.

Possible Models for a Follow-on to TODOR

The Michigan Aero Instructional Software (MAIS) project is an attempt to capture the best elements of TODOR, while transcending as many as possible of the aspects that led to TODOR's demise. Given recent developments in the computer world, many of TODOR's shortcomings can currently be addressed. Three possible models for developing MAIS were considered; their relative strengths and weaknesses are discussed below.

Platform-Dependent Machine Code

This consists of a compiled Language, a graphics library, a GUI library and a numerical library. This is the mode in which TODOR was written. FORTRAN was used as the compiled language, and the X-Windows toolkit and a proprietary layer of software called BLOX were used for the graphics and the graphical interface. The

modules of TODOR did not explicitly make use of a numerical library; necessary numerical tools (linear algebra routines, ODE solvers, etc.) were written from scratch. If the MAIS project were done in this mode, the best choices for the pieces might be:

- C++ for the compiled language;
- X-Windows toolkit for the graphics library;
- Motif for the GUI library;
- Numerical recipes for the numerical library.

If MAIS were programmed in this mode, it would address one important shortcoming of TODOR; there would be no proprietary layer of software that needed to be ported each time a new platform was employed. TODOR was developed in the very early days of X-Windows, and Motif was unavailable. However, many of the other difficulties associated with maintaining TODOR would not be addressed by adopting this model.

Commercial Engineering Package

There would be many advantages to using MATLAB (or Maple or Mathematica) as the base for an instructional software project. MATLAB has graphics, GUI and numerical libraries combined in one package. Educational software written in MATLAB could be run by anyone who has a MATLAB license (many engineering colleges have site licenses for MATLAB). It could be run on a variety of platforms, with no need to rebuild object libraries, since MATLAB is an interpreted, rather than compiled language. All of these are very strong reasons to base educational software on MATLAB. Indeed, there are only three shortcomings to this approach, and they are minor compared to those overcome by the approach:

1. As a programming language, MATLAB is slow and somewhat limited (e.g. it has no support for multi-dimensional arrays);
2. MATLAB source files would need to be distributed to anyone wanting to use the software;
3. The software could not be run by anyone who did not have access to MATLAB.

Neither of these shortcomings is very onerous, and there is a lot of engineering educational software being developed in MATLAB currently. However, there is one more mode for educational software, only recently available, that is even more attractive.

Platform-Independent Interpreted Code

Java is an object-oriented programming language, developed at Sun Microsystems, beginning in 1991 [2]. While it was originally envisioned as a language for programming consumer electronic devices, it grew into a full-blown computer language that incorporates its own graphics, GUI and networking libraries. There are two characteristics of the language that make it an excellent choice for educational software. First, Java is an interpreted, rather than a compiled language. Thus, a single version of the source code can be developed that will run on any machine that can run a Java interpreter. Second, a certain class of Java programs, called applets, can be embedded in WWW home pages. Given these two characteristics, and the fact that the leading web browsers (e.g. Netscape Navigator and Microsoft Internet Explorer) incorporate Java interpreters, the following is true:

Anyone in the world with a web browser and access to the world-wide

web can link to the computer that contains the educational software, "check out" a copy of the executable code, and run it locally inside a web browser.

A Java applet works in the following way:

- Source code is written in Java, a language that resembles C++.
- The source code is converted into machine-independent bytecodes.
- The bytecodes are stored on a WWW server.
- A remote user clicks on a web-page hypertext link.
- The bytecodes are copied across the net, and interpreted by the Java interpreter of the web browser being used.

In one fell swoop, Java solves the code-portability and code-distribution problems associated with most educational software. Also, given that web browsers and internet access are more prevalent than MATLAB site licenses, many more people have access to the software than would under the MATLAB mode. K-12 schools, in particular, are not likely to have access to MATLAB, but, more and more, do have WWW access and Java-capable web browsers.

Java does have some disadvantages, relative to the other two modes. None of these seems insurmountable, however. In particular:

- Java is a new language, and there is a smaller pool of competent programmers than for more established languages. However, it is relatively easy to learn, especially for those familiar with C++. And the pool of competent programmers is growing at a dramatic rate.
- Java lacks much of the numerical and scientific graphics software available in MATLAB. However, development is going on in Java across the country, and many of these capabilities will soon be available. In particular, the Java 2D suite of classes in the 1.2 release of Java, and the upcoming Java 3D suite of classes, provide support for vectors, matrices, quaternions, and 2D and 3D graphics.
- Java is slower than compiled languages such as C++. However, whereas two years ago Java was an order of magnitude slower on scientific applications, the advent of so-called "just-in-time compilers" has dramatically narrowed the performance gap.

Design of the MAIS Package

The MAIS package is therefore planned as a library of Java applets, all of which can be accessed from a home page in the Aerospace Engineering Department at the University of Michigan [1]. Michigan students, and students at other colleges (and even high schools) will gain access to it through their web browsers, simply by knowing the location of the hyptertext link.

A Sample Applet — Potential Flow

One of the modules currently implemented is a potential-flow laboratory, in which point singularities can be introduced, and the resulting flow analyzed via visualizations and animations. This module borrows heavily from ideas implemented in TODOR's POTFLOW module, and is really a tribute to TODOR. Figures 1–5 demonstrate a simple session with the Potential flow module. Figure 1 shows the first screen of the module. Figure 2 shows the screen reached by press-

ing the "Build New Flow" button. Here, the student has built up a flow from a collection of elements — a free-stream, a vortex and a doublet, in this example — by clicking on the appropriate buttons and placing the items in position with the mouse. Directly underneath the plotting window is a line that contains the current location of the mouse, and the computed velocity and pressure at that location; this information is updated continuously as the mouse is moved. Figure 3 shows the screen reached by returning to the main menu and pressing the "Analyze Flow" button. Figure 4 shows the screen reached by pressing "Plot Flow Quantities;" here, the student has plotted a number of streamlines by pressing the appropriate button and using the mouse. Finally, Figure 5 shows the final frame of an animation in which the student has released a "bubble array" starting at a location chosen with the mouse.

Other Applets

Other modules currently implemented or in development include:

- An interplanetary space-flight module, in which students can define origin and destination planets and dates, and analyze the resulting transfer orbit.
- A rotational kinematics module, in which multiple transformations of an object using various rotation methods (Euler, Axis-Angle, Quaternion) can be viewed and compared.
- A potential-flow plus boundary-layer module for airfoils, in which students can "build" airfoils (including multi-element systems), and compute and plot the C_p, lift and drag via a viscous/inviscid coupling method.
- A 3D wing incompressible flow module, based on a vortex-lattice method.
- A nozzle-flow code, in which students can study the effect of area distribution on the pressure and Mach-number distribution in a nozzle.

User-Interface Issues

This project is in its earliest phases, and is presented here in the hopes of soliciting interest and feedback that can help in making the end-product as useful as possible.

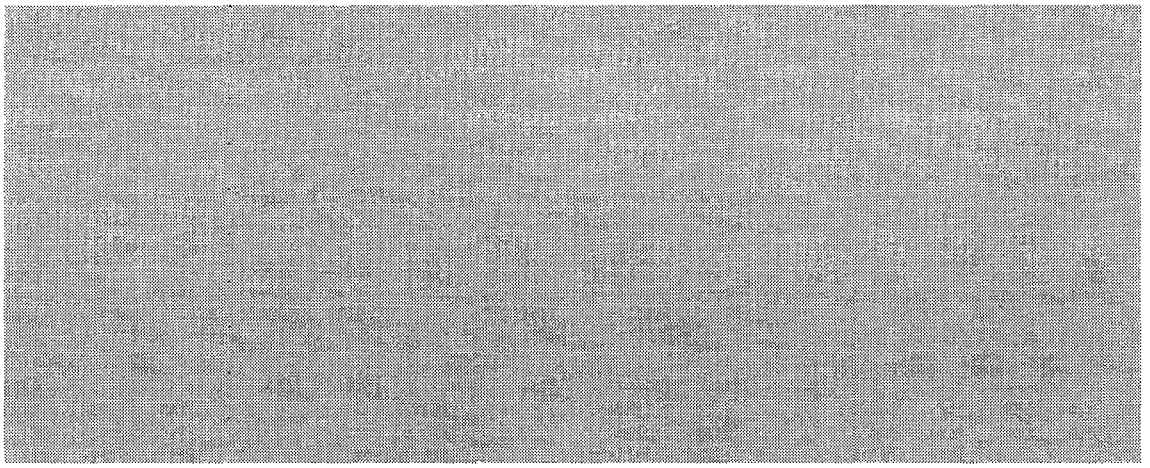
Some design choices have been made so as to give the modules a consistent look and feel, and allow for reusability of many of the Java classes developed for the modules. Each module consists of:

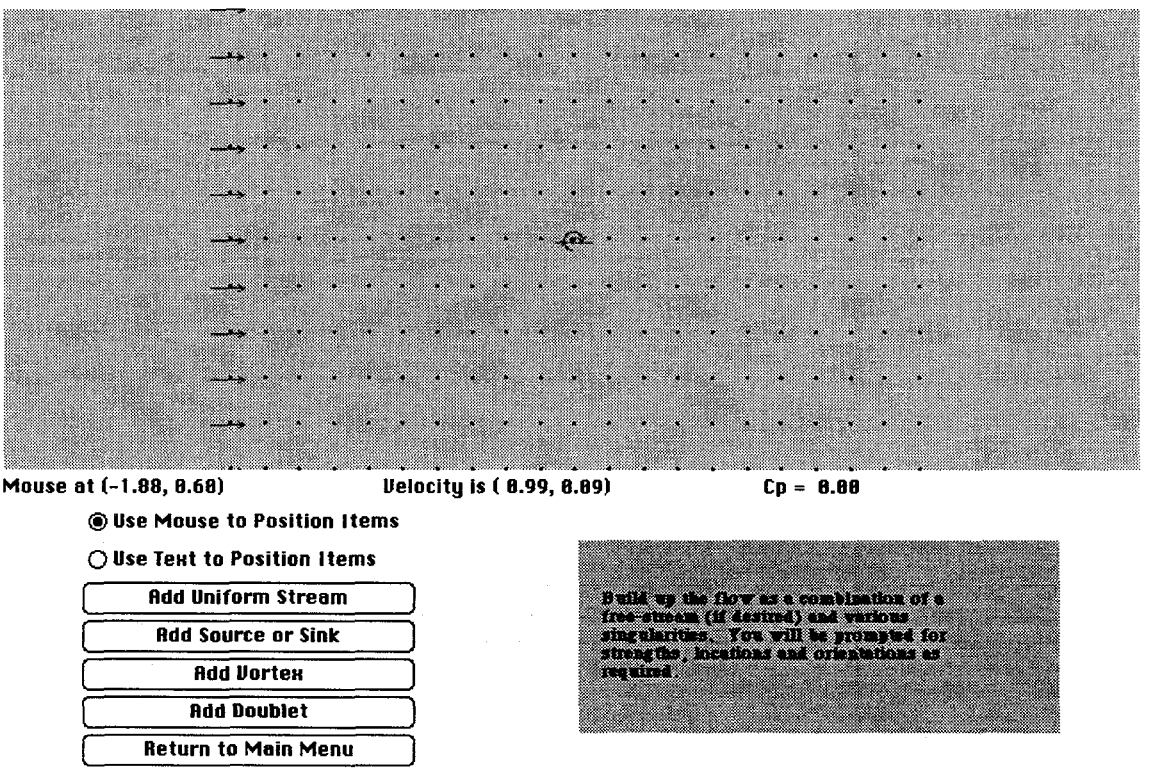
- A plotting window, contained in the top half of the applet;
- A panel of buttons, contained in the bottom left corner of the applet;
- A window containing informational messages and/or input fields, contained in the bottom right corner of the applet.

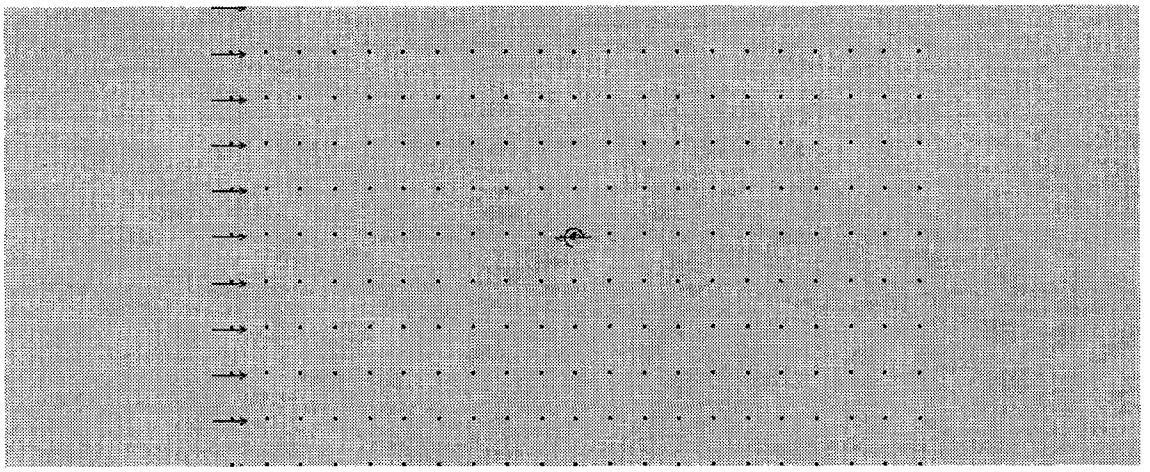
Based on the button selections made by the student, one or all of the plotting, button and info/inputs window can be updated. This simple structure has proven generally applicable to a wide variety of modules.

All of the modules have been developed in an object-oriented fashion, to promote reuse of code. Some of the classes that have been developed that may be useful to others with an interest in developing Java-based educational software include:

- PointSingularity and DistributedSingularity classes, that contain stream-function, velocity-potential and velocity-component influences for various singular solutions of Laplace's equation;
- A PotentialFlow class, that computes velocities, stream functions, and velocity







Mouse at (-1.94, 0.30) Velocity is (0.97, 0.00) Cp = 0.03

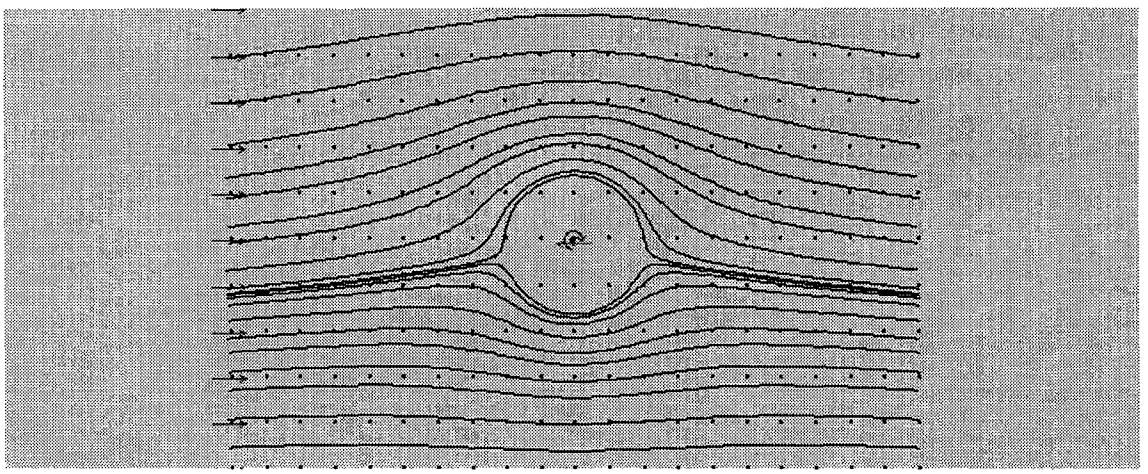
List Flow Elements

Plot Flow Quantities

Animate Flow

Return to Main Menu

Analyze the flow you have built up. You can list the elements that make up the flow, plot static pictures of streamlines, isopotential lines or velocity vectors, or make animated plots of bubbles or fluid elements.



Mouse at (-0.74,-0.31) Velocity is (0.79, 0.02) Cp = 0.36

Plot Velocity Vectors

Plot Single Streamline

Plot Streamline Array

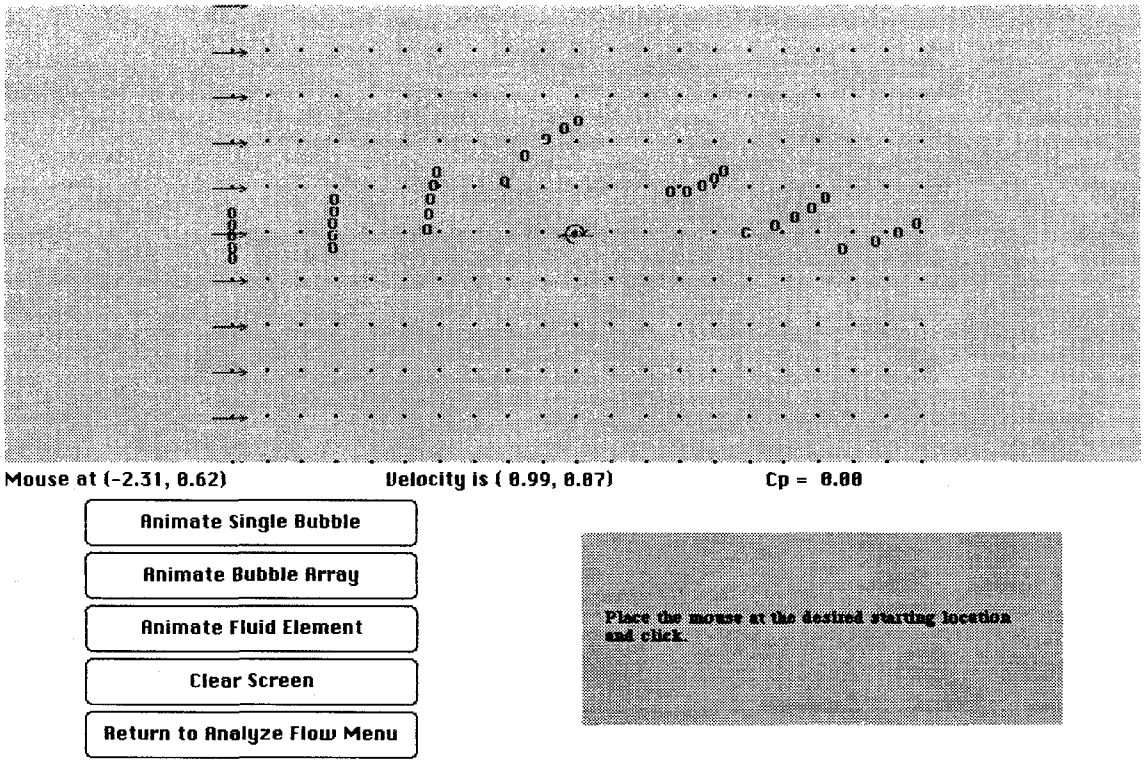
Plot Isopotential Line

Plot Isopotential Array

Clear Screen

Return to Analyze Flow Menu

Place the mouse at the desired starting location and click.



potentials for flows described as collections of singularities;

- An OrbitElements class, that can compute the elements of an orbit from a variety of inputs, and includes the orbit elements for the planets and some major asteroids and comets;

- An Orbit class, that can compute inertial positions and velocities on an orbit as a function of time, and can compute Lambert transfers.

- A number of classes in support of computing and displaying rotational kinematics (Quaternions, Euler angles, etc.);

- A wide variety of support classes for scientific graphics and numerical methods.

Source code for these classes can be obtained by e-mailing the first author (powell@umich.edu). Any feedback on the modules (including bug reports!) can be sent to the same address.

References

[1] University of Michigan Department of Aerospace Engineering home page, http://www.engin.umich.edu/dept/aero.

[2] P. Naughton, *The Java Handbook*, McGraw Hill, 1996.

[3] E. M. Murman, A. R. LaVin and S. C. Ellis, *Enhancing Fluid Mechanics Education with Workstation-Based Software*, AIAA Paper AIAA-88-0001, 1988.