DEVELOPMENT AND IMPLEMENTATION OF SOFTWARE FOR VISUALIZING AND EDITING MULTIDIMENSIONAL FLIGHT SIMULATION INPUT DATA

A Thesis

presented to

the Faculty of the College of Engineering

California Polytechnic State University

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Todd Michael Whelan

September 1996

APPROVAL PAGE

TITLE: DEVELOPMENT AND IMPLEMENTATION OF SOFTWARE FOR VISUALIZING AND EDITING MULTIDIMENSIONAL FLIGHT SIMULATION INPUT DATA

AUTHOR: Todd Michael Whelan

DATE SUBMITTED: September 23, 1996

Dr. Daniel J. Biezad

Adviser _____ Signature _____

Dr. Jin Tso

Committee Member _____ Signature _____

Dr. Estelle Basor

Committee Member _____ Signature _____

Abstract

In a real-time or batch mode simulation that is designed to model aircraft dynamics over a wide range of flight conditions, a table look-up scheme is implemented to determine the forces and moments on the vehicle based upon the values of parameters such as angle of attack, altitude, Mach number, and control surface deflections. Simulation Aerodynamic Variable Interface (SAVI) is a graphical user interface to the flight simulation input data, designed to operate on workstations that support X Windows. The purpose of the application is to provide two and three dimensional visualization of the data, to allow an intuitive sense of the data set. SAVI also allows the user to manipulate the data, either to conduct an interactive study of the influence of changes on the vehicle dynamics, or to make revisions to the data set based on new information such as flight test. This paper discusses the reasons for developing the application, provides an overview of its capabilities, and outlines the software architecture and operating environment.

Acknowledgments

I owe many thanks to those that provided insightful suggestions, answered numerous questions, and attended countless progress briefings as the product evolved. In particular **Lawrence Schilling**, who instigating the project; my advisor **Dr. Daniel J. Biezad**; **Ken Norlin**, **Jeanette Le-Antoniewicz**, and **Marlin Pickett**, who each had a turn receiving the majority of my questions; and the rest of the **XFE/Simulation branch** at NASA Dryden for their support. I also owe a special thanks to **Nicholas Kantartzis**, who allowed me to get my foot in the door at NASA Dryden in another program. That opportunity has led to the fantastic experiences I have had in a total of six summer quarters and five winter breaks at the Flight Research Center.

This work was funded by NASA grant NAG 2-833, in response to the proposal titled "Intuitive Generic Displays for Real Time Selection of Aerodynamic Coefficients During Flight Simulation." The Principal Investigator was Dr. Daniel J. Biezad, professor of Aeronautical Engineering at California Polytechnic State University at San Luis Obispo.

SAVI is available upon request from the National Aeronautics and Space Administration. For more information, contact:

Dr. Daniel J. Biezad (805) 756-5126 dbiezad@oboe.calpoly.edu

Todd Whelan twhelan@daniel.aero.calpoly.edu

Table of Contents

Introduction

Background

The Simulation Aerodynamic Variable Interface, or SAVI (pronounced "savvy"), is a software application that has been designed to graphically visualize and manipulate multidimensional sets of data. It has been developed for the flight simulation department at the National Aeronautics and Space Administration (NASA) Dryden Flight Research Center. Dryden is located at Edwards Air Force Base, California, home of the Air Force Flight Test Center. SAVI allows the engineers to visualize and manipulate the multidimensional data sets that form the input for the flight simulators and model the aircraft dynamics.

The NASA simulators at Dryden are fixed-base, i.e. without a motion system to move the cockpit and mimic the simulated aircraft's attitude. The simulators have forward-looking visuals that consist of a television projection system for out-the-window graphics and a heads-up display (HUD) overlay. These are engineering simulators, used for the development and testing of research aircraft. The research test pilots occasionally rehearse maneuvers before a test flight, or evaluate the handling qualities of proposed flight control system or airframe modifications. They are high fidelity, six-degree-of-freedom simulators, capable of accurately modeling nonlinear responses over a large envelope of operating conditions.

The ability to model these dynamic responses so well is the result of a table-look-up scheme used to provide the inputs to the equations of motion. For example, to determine the forces and moments on the aircraft due to aerodynamic loads, the contributions of different components will be added together. Consider the change in the lift force coefficient that is produced by a particular control surface deflection. This effect may have been measured in a wind tunnel as a function of angle of attack, Mach number and surface deflection angle. This information would be stored in computer memory as a three-

dimensional array, and using the current values of these three independent variables the simulator can interpolate between the provided data points to calculate the force contribution. While this technique can allow the simulator to model complex nonlinear dynamics, it requires large amounts of input data. In addition, when the tables of values are a function of more than two independent variables, the input files become more and more difficult to interpret in text form.

Until now, the input data that determines the forces and moments in the flight simulators at the Dryden Flight Research Facility were accessible only as text files. If the data was to be modified or updated in any way, a particular value had to be found in a large, multidimensional table and changed by hand. This was a time consuming process, and prone to errors. In addition, it was impossible to visualize the data set, which could provide insight into the nature of an air vehicle's dynamics. It was also difficult to confirm that the data was an accurate representation of information originally provided to the simulation engineer as graphs or charts.

In response to these challenges, NASA requested a solution that would enable interactive studies to be conducted with the simulators. NASA envisioned the simulation engineer being able to halt the simulator temporarily, rapidly modify a region of the data set, and immediately allow a pilot to evaluate the effects. In this way, an intuitive sense of the critical parameters effecting aircraft performance could be cultivated, and theoretical studies could be undertaken.

Occasionally, however, the simulated behavior of an aircraft needs to be modified in a different way than striving for a qualitative, theoretical optimum. For example, a prototype aircraft may exhibit a phenomenon in flight test that was not predicted by the simulation. It becomes necessary to determine what part of the data set is not providing an accurate model, and an iterative process is applied in

order to produce simulation time histories that will overlay those observed in flight test. This process will also benefit greatly from the interactive, graphical interface to the input data that SAVI provides.

Although SAVI has been designed for immediate use in flight simulation, the tool can be applied to any task that would benefit from the capability to visualize and edit a multidimensional data set. SAVI's power is in its ability to portray and manipulate data in which the dependent variables are a function of one or many parameters.

Educational Application of SAVI

In addition to its utilization at NASA Dryden, SAVI is being implemented as a member of the Pangloss software suite. The Pangloss project is an ongoing collaborative effort being undertaken by undergraduate and graduate students at Cal Poly. The majority of the students involved are from within the Aeronautical Engineering department, but the group encompasses other academic disciplines as well. The main effort of the Pangloss project is to automate the preliminary design process of aircraft as much as possible, by developing new software tools and creating interfaces between existing ones. Primary efforts include implementing multidisciplinary design principles, and avoiding "black box" tools that detract from the software's educational value. The components of Pangloss that are related to SAVI include ACSYNT, PREDAVOR, and ACSIM.

ACSYNT, which is an acronym for Aircraft Synthesis, is an aircraft design and optimization application. It includes a geometry package that can be used to quickly generate complex aircraft shapes from basic components. This geometry is used within ACSYNT, in conjunction with aerodynamic, performance, economic and other input parameters, to analyze the aircraft design. The geometry information can also, however, be used by other programs.

PREDAVOR is an example of such a program, which uses the ACSYNT geometry output as input to a vortex lattice analysis. PREDAVOR calculates the forces and moments on the geometry at different airspeeds, angles of attack and sideslip, and control deflections, generating tables of stability and control derivatives. These tables can then be fed into the flight simulation program that was developed for the Pangloss project, ACSIM.

ACSIM is a six-degree-of-freedom real-time flight simulation program. It was originally designed to work with a single set of stability and control data in a linearized model about an equilibrium flight condition. ACSIM is being expanded, however, to implement a table-look-up scheme similar to the one used at Dryden, and therefore will be able to model an aircraft's dynamics over a wider operating envelope. The primary strength of ACSIM is the inclusion of two handling quality evaluation tasks. The first task, shown in Figure 1, is an up-and-away tracking task in which the simulator pilot attempts to point the nose of the aircraft at a target moving through space. The target is in the shape of a cross, with lights that alternate randomly at the ends of the arms. The second task, shown in Figure 2, is a landing approach in which the simulator pilot flies to a runway and is graded on deviating from the extended centerline of the runway, and flying above or below the desired glidepath angle.

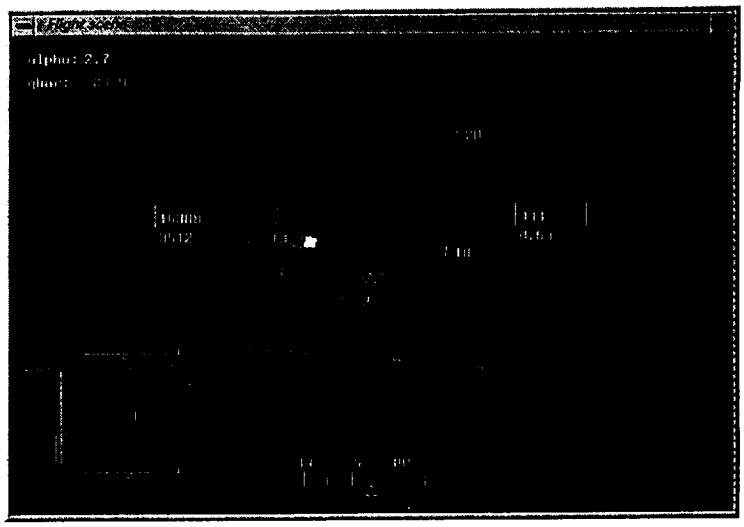

Figure 1

ACSIM Handling Qualities Evaluation Up-and-Away Task

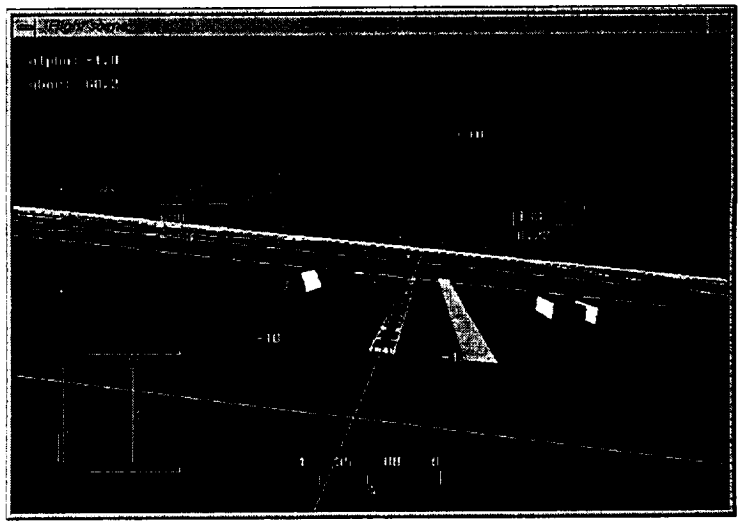

Figure 2

ACSIM Handling Qualities Evaluation Landing Task

Using SAVI, the student will be able to visualize the stability and control data set coming from PREDAVOR, gaining insight into its trends and how the aircraft dynamics parameters relate to each other. The student may also make modifications to portions of the flight envelope, and then evaluate the effects of the changes by flying the handling qualities evaluation tasks in ACSIM.

Methodology

Development of the project began by researching the needs and desires of the engineers that would be the end users, inventing techniques that would be useful for looking at the data, and creating methods for changing the values. Emphasis was placed upon producing an interface that is intuitive and consistent with other computer applications, and consideration was given to portability of the source code to various computer platforms. The program must also be conducive to future additions and further improvements by another programmer.

It was realized from the beginning that the program would have to be very flexible. It must be compatible with any of the flight simulations at Dryden, and be able to accept any number of dependent variables within the data set. Each variable may be a function of any number of independent parameters, which would produce a large and drastically varying number of data points for each variable. However, sizing every array to accomodate the largest number of data points expected in a variable would require far too much computer memory. Therefore, dynamic memory allocation became a requirement for the SAVI data architecture.

Research of data management techniques led to the implementation of the "linked list" concept, a common technique for dynamically storing and manipulating sets of information of different sizes (Tenenbaum). A linked list makes extensive use of the "pointer", a variable type used in the C programming language that stores the memory location of another variable. These are represented in Figure 3 by the boxes that have arrows emanating from them, "pointing" to other variables. Another entity used to build the linked list is a "structure", which is an encapsulation of several variables. The variables within the structure may be of different types, such as floating point values, integers, character strings, or pointers. By including a pointer within a

structure that can point to another structure of its own type, the structures can be linked together into a chain that can have links added or taken away.

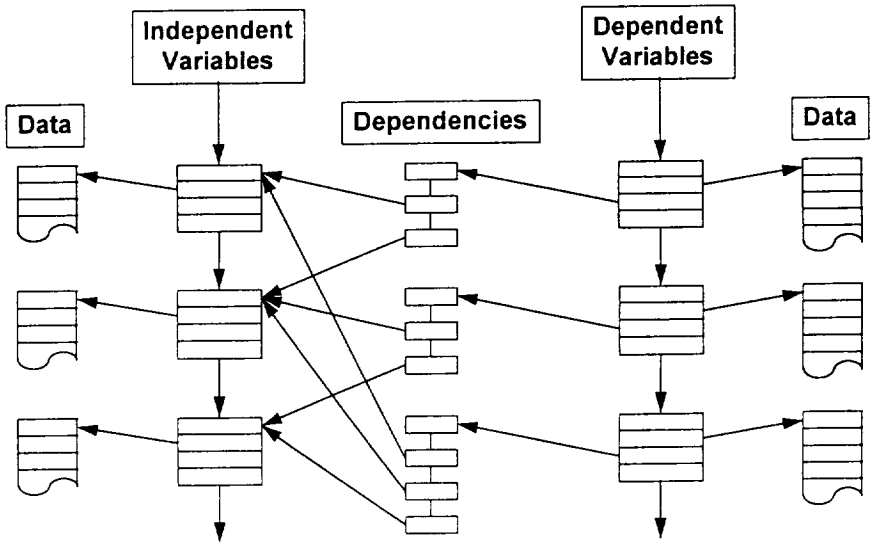

Figure 3

SAVI Data Architecture in Memory

The next issue to consider was the implications of the windowing environment in developing the SAVI application. X-Windows appears to be the standard windowing environment for workstation computers. This is the case for the Sun and Silicon Graphics systems currently in place at Dryden, and it appears that any future alternative platform will be the same in this regard.

Within the X-Windows environment, there are different sets of "toolkits", or libraries of programming objects. These toolkits create the elements of the graphical interface, such as windows, buttons, menus, and text fields, as well as complex combinations like file browsers. The programming objects "know" how to handle their own basic

behaviors automatically, such as popping up a menu when a particular button is clicked, or redrawing itself when it is covered by another window and then uncovered again. By adopting a standard for the toolkit and style to be used in graphical interface programs, it helps the different applications emulate the same "look and feel". In this way, a computer user that is familiar with one application can very easily start using another, because basic things look and act the same. The true test of a graphical interface design is how easily a new user can begin using it, without having to refer to a manual or to help documentation.

At the beginning of this project, OpenLook and Motif had emerged as the dominant toolkits for X-Windows programs with graphical interfaces. OpenLook had been gaining in popularity, but Motif was still more widely used. Motif was also being used for other applications being developed at Dryden. The decision was therefore made to proceed with the Motif library, and almost all of the new software being developed for workstations today is based on the same decision.

Some consideration was given to choose the programming language for developing SAVI. The most appropriate options include FORTRAN, C++, and C. Although FORTRAN is used extensively in the simulations at Dryden, it was not considered a viable option, since the X-Windows and Motif routines and include files are written in C. All of the programming examples in the reference manuals are written in C, as well. Another option would be to use the C++ programming language, which is a superset of C and would support calls to the X-Windows and Motif subroutines. C++ has been gaining in popularity and acceptance over C; however, C code is still much more common and widely known at Dryden, and the C compilers are more readily available. These facts lead to the decision to develop the application in C.

Implementation

When possible, the graphical interface portions of SAVI were built with the Motif toolkit, which is a standardized set of X Windows routines that create objects called "widgets", such as pulldown menus, scroll bars, buttons, and more complex combinations like file browsers. These routines encapsulate many operations in X Windows, create programming objects that handle many operations automatically, and provide a consistent look and feel between different graphical applications. The control window for SAVI, shown in Figure 4, is an example of a collection of Motif widgets. A software application called X-Designer was utilized to build the initial skeleton code for SAVI. X-Designer is an interface builder, and will generate the source code to construct the menus, buttons, and other widgets, leaving "stubs" for a programmer to add customized text, color, and functionality to the interface. In many cases the code that was generated by X-Designer was used as examples to learn from and expand upon. Another source of examples was the set of programming and reference manuals published by O'Reilly & Associates. These books also provided the information necessary to learn and implement the custom features not part of the Motif toolkit.

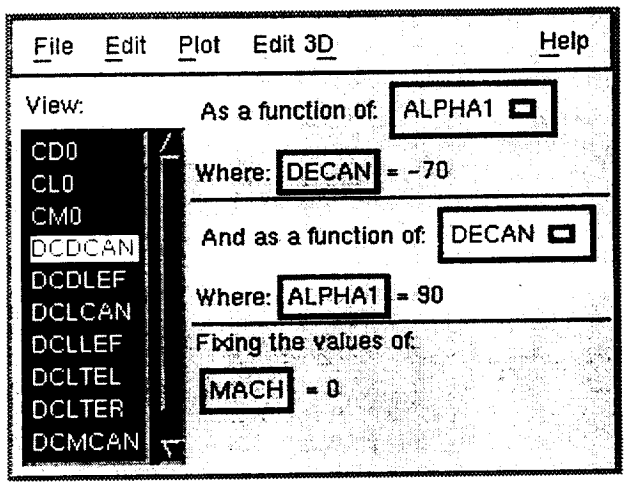

Figure 4

SAVI Control Window

The actual plotting and graphing of the curves needed to be drawn line by line with calls to basic X Windows graphics routines. The code to handle the mouse events generated by clicks and drags within the plot windows also had to be written by hand. Routines were written to plot the curves and label the axes for the two dimensional graphs and the three-dimensional plot, shown in Figure 5. A transformation matrix is maintained for the three-dimensional view, allowing the user to rotate the plot with the scrollbars beside and below the plot. The effect is an animated motion of the surface, which helps the user visualize the three-dimensional shape. As an additional aid to comprehending the shape of the data, routines have been implemented to convert the wireframe into a collection of polygon "tiles". By sorting the tiles from back to front according to current plot rotation and drawing them in order, nearer tiles are drawn over parts of previously drawn distant ones to effectively portray a solid-looking surface.

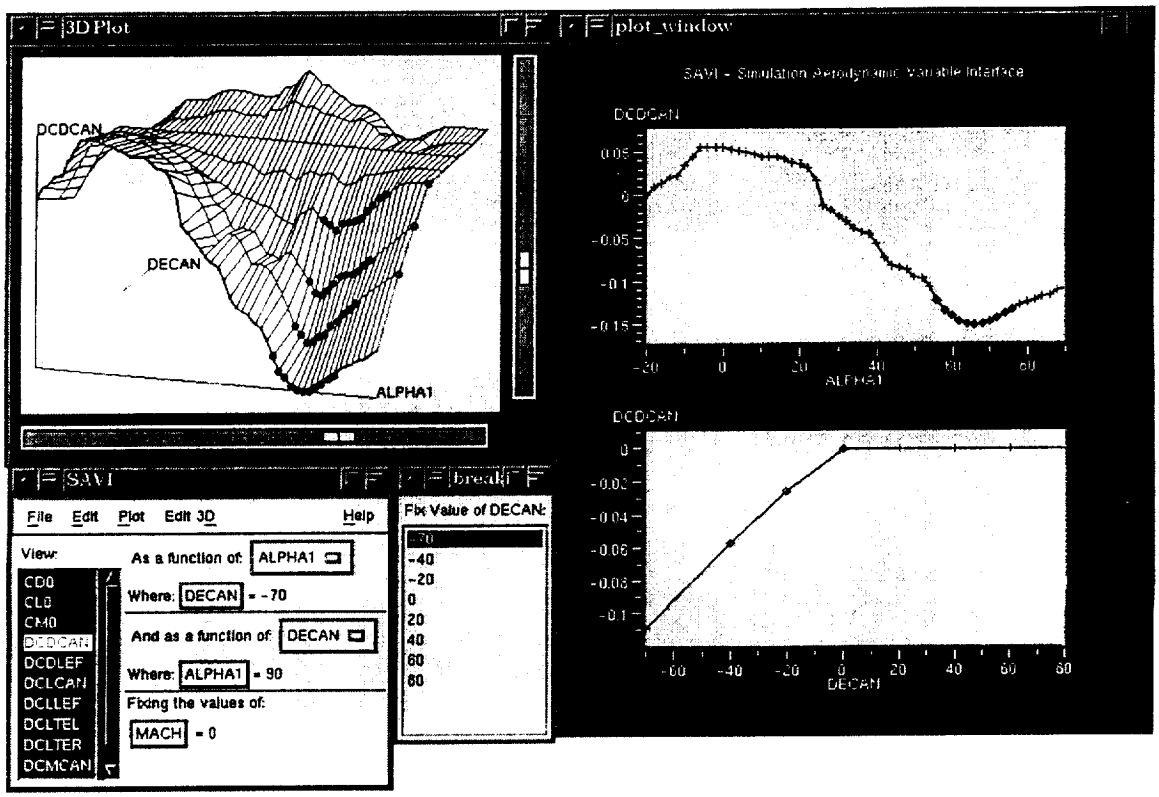

Figure 5

SAVI Desktop

Development of the user interface portions of SAVI began with X-Designer, a graphical user interface builder. X-Designer was used to construct the skeleton of the code that defines the basic layout of the pulldown menus in the control window, and the layout of the plot windows. X-Designer will generate the C code to create buttons, windows, slider bars, menus, and other graphical interface objects. The programmer then writes the software routines that give the widgets customized functionality. After some preliminary work in generating basic windows and widgets and studying the resulting code, a point was reached where custom layouts had to be written by hand.

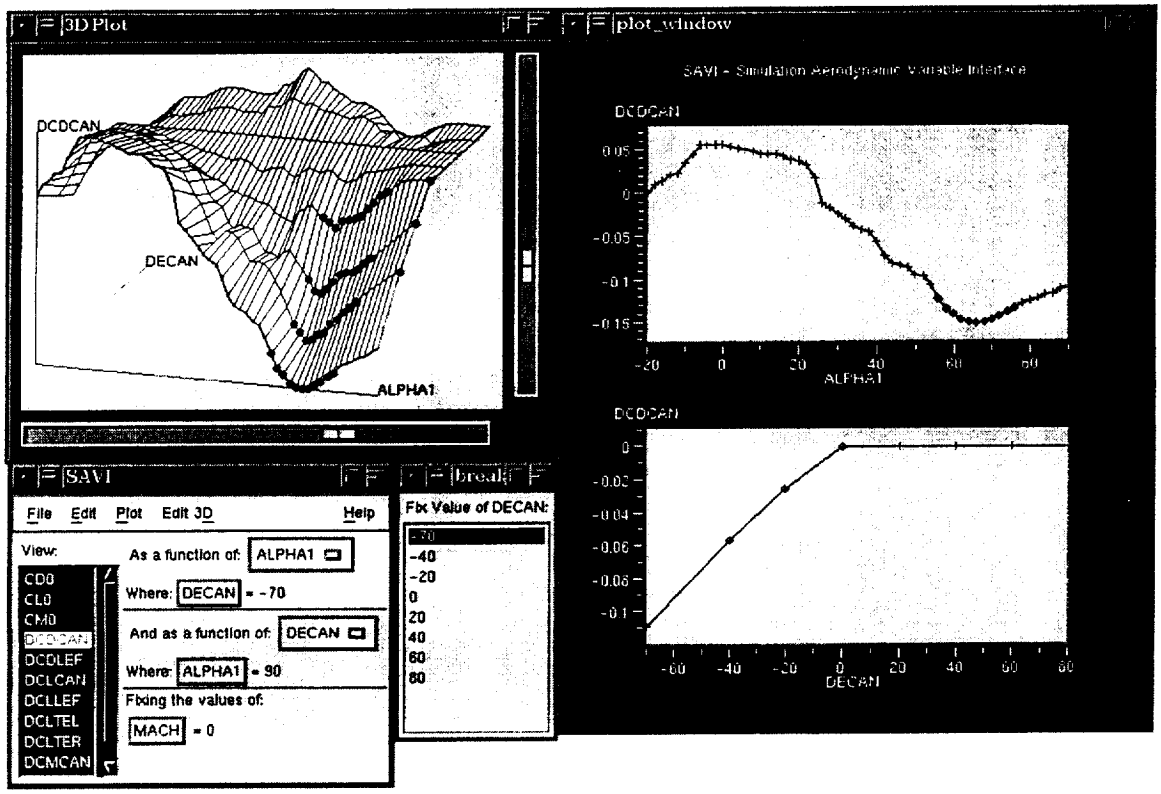

Figure 5

SAVI Desktop

Development of the user interface portions of SAVI began with X-Designer, a graphical user interface builder. X-Designer was used to construct the skeleton of the code that defines the basic layout of the pulldown menus in the control window, and the layout of the plot windows. X-Designer will generate the C code to create buttons, windows, slider bars, menus, and other graphical interface objects. The programmer then writes the software routines that give the widgets customized functionality. After some preliminary work in generating basic windows and widgets and studying the resulting code, a point was reached where custom layouts had to be written by hand.

Platform-specific routines were avoided in creating SAVI, and the C code was written to the lowest common denominator of compilers. Originally, the code was written to the ANSI standard in the expectation that virtually any compiler would support it. This was not the case, as was discovered when using the older Sun C compiler. The code was then reverted to the style originally suggested by Kernighan and Ritchie.

The particular architecture used by SAVI to store the data in memory is shown in Figure 3. To illustrate how the information is arranged, begin with the box marked "Dependent Variables". This box represents a pointer to the first member in the linked list of dependent variables. Starting with this pointer and the name of a dependent variable of interest, the list can be descended, checking the name stored in each structure. When a match is found, a member of that structure will point to a list of the independent parameters of which the variable is a function. This list is a linked list of pointers, each of which leads to a structure that contains all the information associated with the applicable independent variable.

For example, suppose information is needed about the dependent variable called CmAlpha. The dependent variable linked list is descended until the structure containing the name "CmAlpha" is encountered. Following the pointer to its list of independent variables, and following each of the pointers in this list in turn, structures for "Alpha", "Beta", "Mach", and "Altitude" would be found. The arrays of data for each of these independent variables would indicate the position on the horizontal axis above which to plot the CmAlpha values.

The source code has been compiled and executed successfully on both Sun and Silicon Graphics platforms, using three different C compilers. Theoretically, the application could also be used on a personal computer that is running the LINUX operating system, with X Windows and the Motif environment.

Results

The windows of the SAVI desktop are shown in Figure 5. The individual windows are shown in Figures 4, 6 and 7. When the application is launched, the windows appear on the terminal screen with blank plots. Data can be accessed for display and editing in two ways, either from a data file or by interfacing with simulation memory. A data file is opened by selecting the "Open" button in the "File" pulldown menu, which creates a file browser for selecting the file. The input file is interpreted, and a list of the dependent variables appears in the scrolled list shown in Figure 4. The simulation memory interface is set up by selected "Simulation Interface" from the "File" pulldown menu, and the list of dependent variables again appears in the scrolled list shown in Figure 4. When a variable is selected from the list, other menus appear in the control window, which allow the independent parameters to be chosen to plot against. In the case of variables that are a function of more than two parameters, additional menus allow the user to fix the values of the other parameters, selecting the appropriate data surface to view.

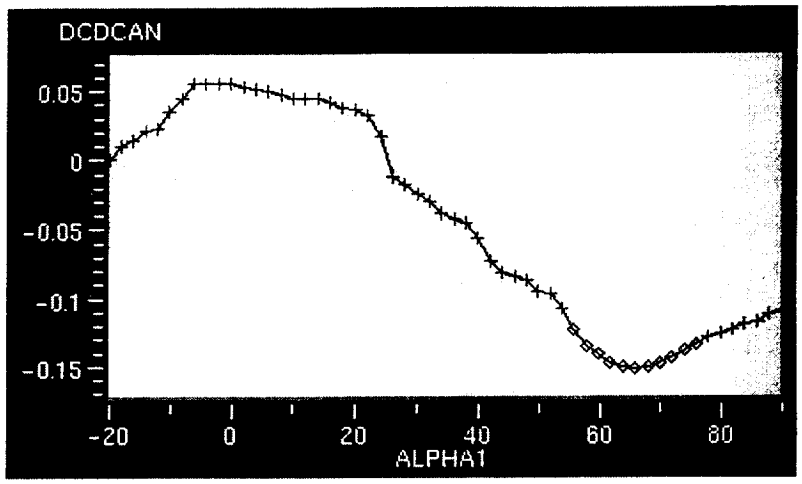

Figure 6

SAVI Two-Dimensional Plot Window

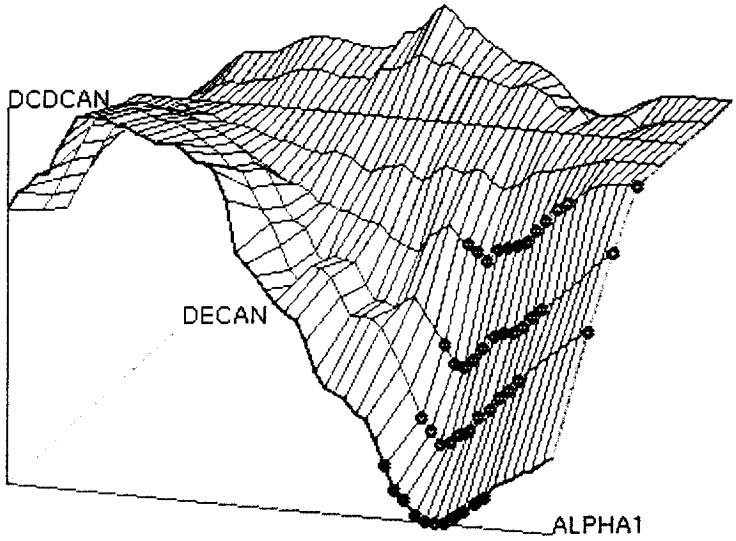

Figure 7

Savi Three-Dimensional Plot Window

If the dependent variable is a function of only one parameter, the appropriate curve is drawn in the primary two-dimensional plot window, as shown in Figure 6. For variables that are a function of two or more parameters, another curve is shown in the secondary two-dimensional plot window, as shown in Figure 5. This curve is orthogonal to the first, and they are each highlighted in different colors along the appropriate lines in the three-dimensional plot window, as shown in Figure 7. The curves to be shown in the two-dimensional plots can be chosen with the menus of dependency values in the control window, or the user can click on the three-dimensional surface and the nearest intersection is used to generate appropriate curves.

The SAVI Desktop

The control window displays a list of the dependent variables that were found in the data file, or that were found in simulation shared memory. The control window also contains the menus that control file input and output, modify the plot appearance, select the current editing function, and launch the interactive on-line help.

The primary two-dimensional plot is color-coded, with the data points designated as red '+' symbols and the selected ones as blue diamonds. The user can zoom in to look at a region of the curve close up, and it is in this window that the user may apply the two-dimensional editing algorithms.

For variables that are a function of more than one parameter, a secondary two-dimensional plot is generated. It also is color-coded with the points marked with green '+' symbols, and the selected points marked with blue diamonds. If points are selected in both two-dimensional plots, it is analogous to selecting rows and columns of the matrix of points that make up the three-dimensional plot. These selected rows and columns define the rectangular region in the matrix that will be the target of the three-dimensional editing algorithms.

and column is selected become highlighted, and subsequent editing commands are applied to those points.

Editing Algorithms

Developing Editing Algorithms

In developing the different algorithms for editing values within the data set, two design philosophies were applied. The first was to strike a balance between the power and the complexity of the editing operation, to give the user as many options as possible without losing the intuitive sense of the application. It is undesirable to have data affected in ways that were not intended, even worse to change it in ways that the user in unaware of. Therefore, all of the editing routines operate on data that can be seen at one time in a single curve or on a single data surface.

The second philosophy in designing the editing algorithms was to create simple techniques that could be combined like building blocks to construct complex results. This is demonstrated by the surface shown in Figure 8, which was created with three simple operations.

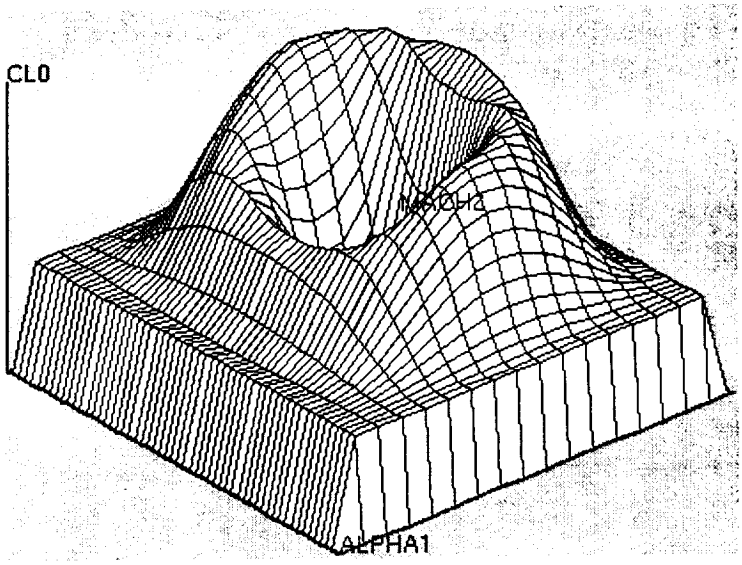

Figure 8

Complex Surface Generated from Simple Editing Operations

Editing Algorithms for the Two-Dimensional Plot

For the purpose of displaying the results of the editing operations, consider a set of points at a value of 1.0 as shown in Figure 9. Holding the <shift> key down on the keyboard and depressing the left mouse button while the pointer is in the primary two-dimensional plot window will move any of the selected points onto the imaginary horizontal line that passes through the pointer, as shown in Figure 10. In this way, one or more values can be "dragged" to a desired value. While the drag operation is taking place, a digital readout above the two-dimensional plot will indicate the current value that the mouse is pointing to. This particular feature will work outside the drag operation, also; if no points are selected and the <shift><left mouse> combination is used over a data point, its value can be more accurately read from the graph.

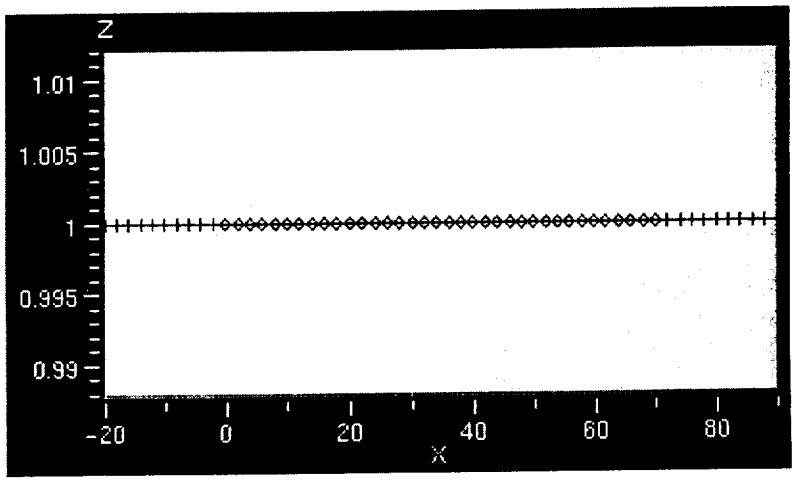

Figure 9

Two-Dimensional Data Before the Editing Operations

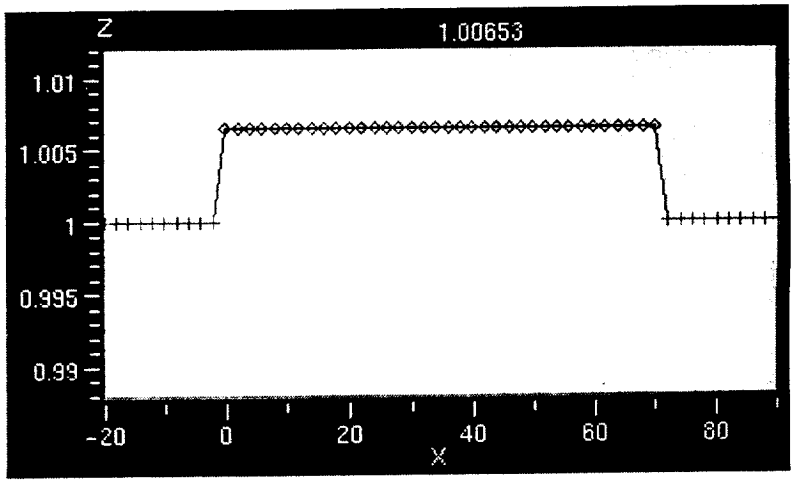

Figure 10

Two-Dimensional Data After a Drag, Input Value, Constant Delta, or Constant Multiple Editing Operation

If the value of a selected point or points needs to be set more accurately than by the click-and-drag method, the value can be input

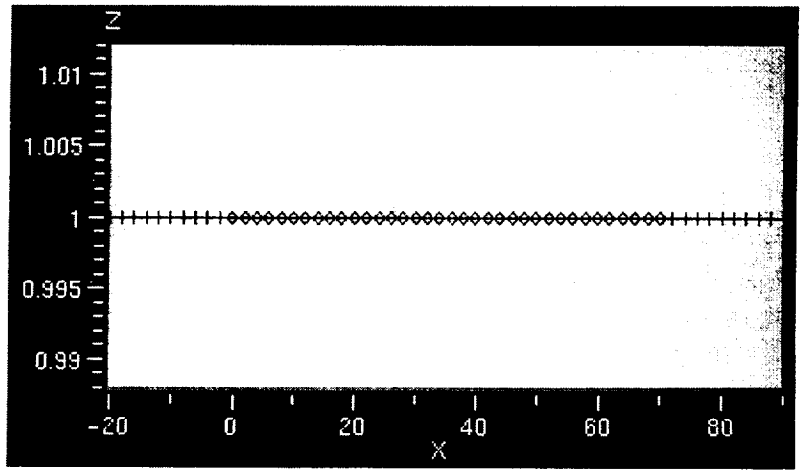

Figure 9

Two-Dimensional Data Before the Editing Operations

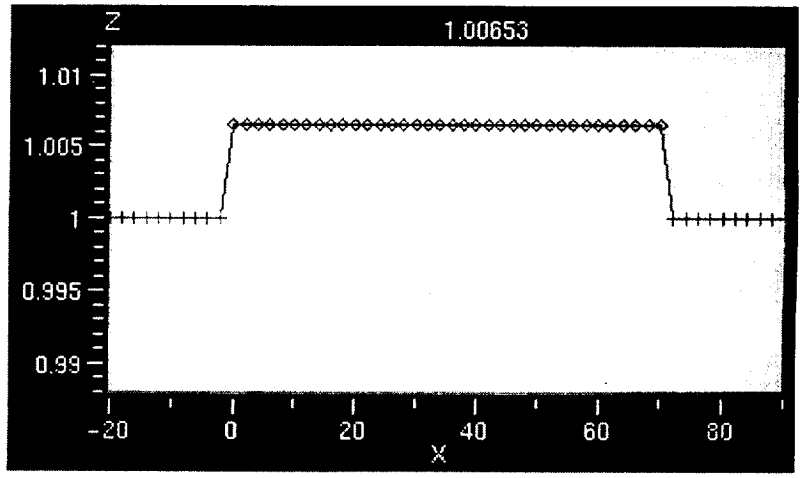

Figure 10

Two-Dimensional Data After a Drag, Input Value, Constant Delta, or Constant Multiple Editing Operation

If the value of a selected point or points needs to be set more accurately than by the click-and-drag method, the value can be input

directly from the keyboard. This operation is another way to produce the results shown in Figure 10.

A set of selected points can be increased or decreased by a fixed amount by providing a positive or negative additive delta that is added to each point in turn. This operation is a third way to produce the results shown in Figure 10. The same effect can be achieved a fourth way with the application of a constant multiplier, in which all of the points are multiplied by a coefficient input from the keyboard.

After these initial editing capabilities where added, it was realized that more sophisticated algorithms needed to be developed. Consider that for each particular flight state of the aircraft, or each combination of values of the independent variables, the simulator table-look-up scheme will determine a value by interpolating between the points in the immediate vicinity. If the data is changed in a small region to match a flight characteristic, or if the change is not blended smoothly into the surrounding region, the simulated dynamics will work well at the modified flight state but change quickly back to the old behavior within a single interval of data. Therefore, some editing algorithms were developed that would apply the full delta or multiple at a point, and then blend smoothly to the old values over the selected interval. One method tapers the changes with a linear profile, and the other uses a sinusoid curve.

The first class of blended editing methods is termed the "variable delta". As shown in Figure 11, the center of the region is changed by the full value, with the delta decreasing linearly to zero and leaving the endpoints unaffected. The second type of delta is demonstrated in Figure 12, in which the function added to the region is one period of a sinusoid. The sinusoid is scaled such that the minima of zero occur at the endpoints, and the maximum occurs at the center of the region.

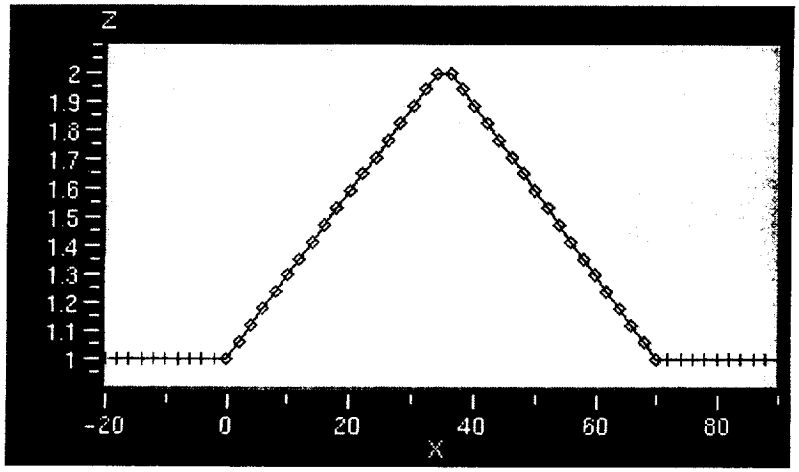

Figure 11

Two-Dimensional Linear Delta Editing Operation

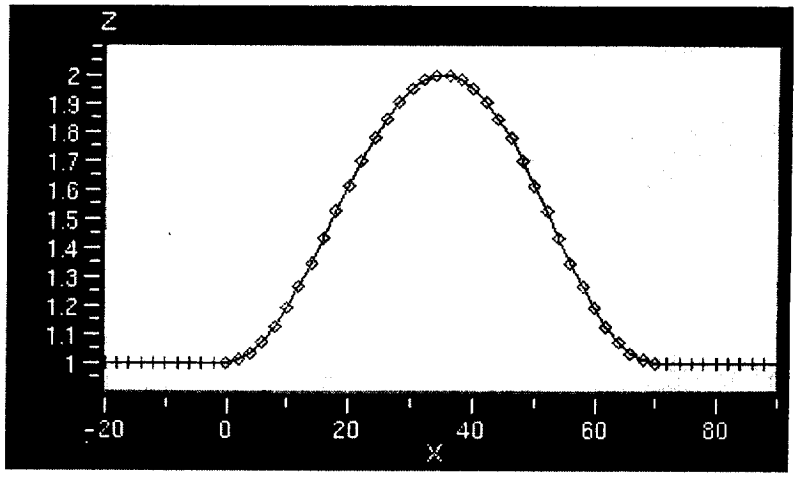

Figure 12

Two-Dimensional Sinusoid Delta Editing Operation

The variable multiplier is very similar to the variable delta, with the center of the region receiving the full affect but the endpoints remain where they were. In this case the editing function is multiplied

by the original curve, with endpoints at a value of one and the center at the full multiplier. The linear multiplier produces the same effect as shown in Figure 11, and the sinusoid multiplier produces the same effect as shown in Figure 12.

Editing Algorithms for the Three-Dimensional Plot

Once points have been selected in both two-dimensional plots to define a region on the three-dimensional surface, the three-dimensional editing algorithms can be applied to these points.

There is no direct correlation to the click-and-drag operation, but the user can input a value for the points directly, apply a constant delta, or a constant multiple in the same manner as the two-dimensional plots.

The variable delta and variable multiple work slightly differently than the two-dimensional case, however. As shown in Figure 13, applying the three dimensional linear delta to a region adds a pyramid shaped function, increasing the point in the center by the full amount of the delta and tapering down linearly to the old values at the edge. The effect of the sinusoid delta is shown in Figure 14, in which the added function is a sinusoid in both directions.

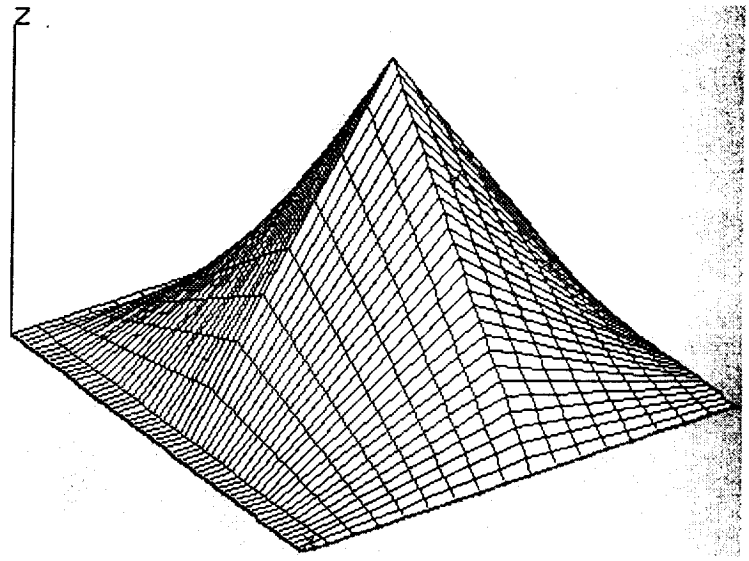

Figure 13

Three-Dimensional Linear Delta Editing Operation

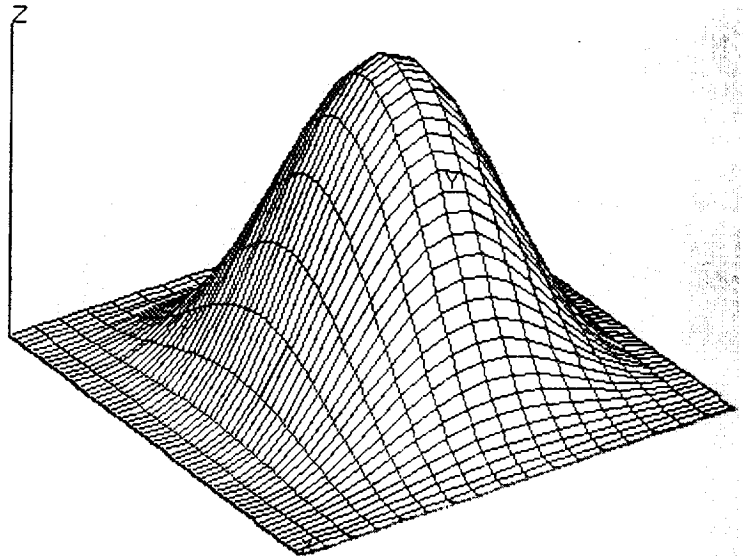

Figure 14

Three-Dimensional Sinusoid Delta Editing Operation

The linear and sinusoid delta can be combined, as shown in Figure 15, in which the blend is linear in one direction and sinusoidal in the other.

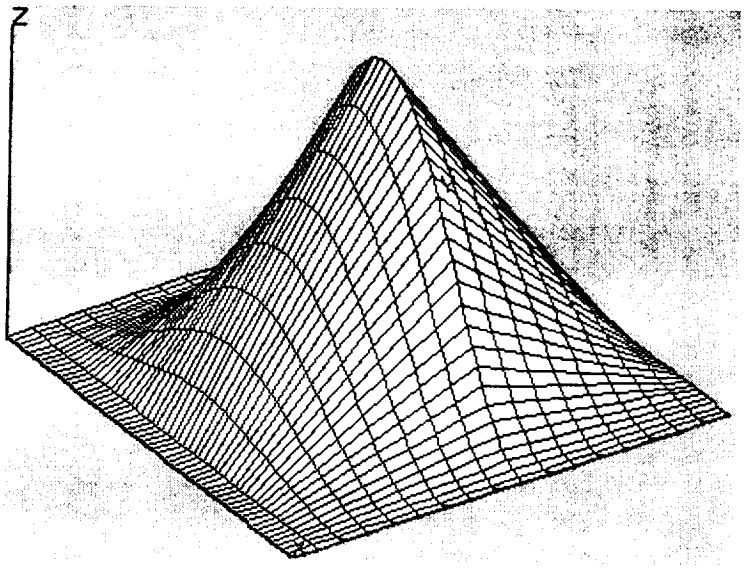

Figure 15

Three-Dimensional Combination of Linear and Sinusoid Editing Operations

Unevenly Spaced Data

During development, when dealing with unevenly spaced or sparsely spaced data, it was often noted that none of the points were receiving the full effect of the variable delta or variable multiple edit operations. This was at first misinterpreted as a problem with the code, but when it was realized that a point simply did not correspond with the location of the peak, it was assumed that the end user may feel similar confusion. The "center" of the region in the variable delta and variable multiple editing algorithms is now determined by counting the number of selected points, rather that averaging the horizontal values

User's Manual

Finally, on-line interactive help has been installed for the user, in the form of HTML with hypertext. It is in the same format at World Wide Web (WWW) pages, and is accessed from within the application by launching a web browser to view the help file.

Conclusions

SAVI is a powerful, intuitive graphical user interface to multidimensional data sets, such as flight simulation input data. It presents the user with a look and feel consistent with other applications, and is written with C code that is portable to various computer platforms. SAVI will be a useful and productive tool for the simulation engineers at Dryden, as well as members of other groups at NASA such as aerodynamics, propulsion, and flight controls engineers, who often provide the data models for the aircraft simulations. For the first time, the data can be viewed as a function of any one or two of its dependencies, and the values can be quickly modified in an intuitive way.

SAVI will also have educational value at the university level, and is in the process of being implemented as a tool in the undergraduate Aeronautical Engineering curriculum. It will enable students to visualize the nature of an aircraft simulation data set, and begin to develop an intuitive sense of aircraft dynamics. In addition, SAVI will be used as a tool for working with data generated by students in their own preliminary design of aircraft. It has the potential to be used in assessing the effects of design changes on performance, stability and control, and handling qualities, or for updating a data set with the results of different prediction techniques.