FINAL REPORT

DEVELOPMENT OF A NEW GENERATION SOLID ROCKET MOTOR

IGNITION COMPUTER CODE

by

Winfred A. Foster, Jr.
Associate Professor

Ronald M. Jenkins
Assistant Professor

Alessandro Ciucci
Graduate Research Assistant

Shelby D. Johnson
Graduate Research Assistant

Prepared under

NASA Grant No. NAG8-683

between

George C. Marshall Space Flight Center
National Aeronautics and Space Administration

Auburn University
Engineering Experiment Station
Auburn University, Alabama 36849

February 10, 1994

(NASA-CR-195734) DEVELOPMENT OF A NEW GENERATION SOLID ROCKET MOTOR IGNITION COMPUTER CODE Final Report (Auburn Univ.) 173 p

Unclas
ABSTRACT

This report presents the results of experimental and numerical investigations of the flow field in the head-end star grain slots of the Space Shuttle Solid Rocket Motor. This work provided the basis for the development of an improved solid rocket motor ignition transient code which is also described in this report. The correlation between the experimental and numerical results is excellent and provides a firm basis for the development of a fully three-dimensional solid rocket motor ignition transient computer code.
ACKNOWLEDGEMENTS

The authors express appreciation to personnel of the George C. Marshall Space Flight Center (MSFC) for their support and assistance in this project. In particular, Mr. John E. Hengel, whose overall support of this project was an essential factor in its overall success and to Mr. Andrew W. Smith for his technical assistance. The authors also wish to thank to Mr. Billy H. Holbrook for his work in constructing the experimental models.
# TABLE OF CONTENTS

**ABSTRACT** ........................................................................................................ ii

**ACKNOWLEDGEMENTS** .................................................................................. iii

**LIST OF FIGURES** .......................................................................................... v

**LIST OF TABLES** ............................................................................................... vii

**NOMENCLATURE** .............................................................................................. viii

I. **INTRODUCTION** ............................................................................................. I-1

II. **EXPERIMENTAL ANALYSIS** .......................................................................... II-1
    Experimental Apparatus .................................................................................. II-2
    Test Facility .................................................................................................. II-11
    Test Plan ........................................................................................................ II-11
    Oil Smear Results ......................................................................................... II-12
    Results From Static Pressure Measurements ............................................. II-13
    Results From Heat Transfer Measurements ................................................ II-13
    References ..................................................................................................... II-29

III. **THEORETICAL/NUMERICAL ANALYSIS** .................................................. III-1
    Introduction ................................................................................................... III-1
    Conservation Equations ................................................................................ III-2
    Turbulence Model ......................................................................................... III-6
    Heat Transfer Relations .............................................................................. III-8
    Numerical Technique ................................................................................... III-10
    Initial and Boundary Conditions ................................................................ III-11
    Sample Results ............................................................................................. III-13
    References ..................................................................................................... III-23

IV. **CAVENY PROGRAM MODIFICATIONS** ..................................................... IV-1
    References ..................................................................................................... IV-7

V. **CAVENY PROGRAM INTERFACE** ............................................................... V-1

VI. **X PLOT PROGRAM DESCRIPTION** ............................................................ VI-1

VII. **X PLOT USER INSTRUCTIONS** ................................................................. VII-1
    APPENDIX A Sample Caveny Input File ..................................................... A-1
    APPENDIX B Caveny Program Makefile ..................................................... B-1
    APPENDIX C xplot Source Program .......................................................... C-1

iv
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIG.</th>
<th>TITLE</th>
<th>PAGE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>II-1</td>
<td>Schematic of star slot</td>
<td>II-4</td>
</tr>
<tr>
<td>II-2</td>
<td>Cross-section of star slot region</td>
<td>II-4</td>
</tr>
<tr>
<td>II-3</td>
<td>Schematic of head-end section</td>
<td>II-8</td>
</tr>
<tr>
<td>II-4</td>
<td>Schematic of single-port and multi-port igniters</td>
<td>II-8</td>
</tr>
<tr>
<td>II-5</td>
<td>Head-end section model</td>
<td>II-9</td>
</tr>
<tr>
<td>II-6</td>
<td>Igniters models</td>
<td>II-9</td>
</tr>
<tr>
<td>II-7</td>
<td>Location of pressure taps and calorimeters</td>
<td>II-10</td>
</tr>
<tr>
<td>II-8</td>
<td>Detail of calorimeter installation</td>
<td>II-10</td>
</tr>
<tr>
<td>II-9</td>
<td>Igniter 1 (100 psi)</td>
<td>II-14</td>
</tr>
<tr>
<td>II-10</td>
<td>Igniter 2 (100 psi)</td>
<td>II-14</td>
</tr>
<tr>
<td>II-11</td>
<td>Igniter 3 (100 psi)</td>
<td>II-14</td>
</tr>
<tr>
<td>II-12</td>
<td>Igniter 1 (500 psi)</td>
<td>II-15</td>
</tr>
<tr>
<td>II-13</td>
<td>Igniter 2 (500 psi)</td>
<td>II-15</td>
</tr>
<tr>
<td>II-14</td>
<td>Igniter 3 (500 psi)</td>
<td>II-15</td>
</tr>
<tr>
<td>II-15</td>
<td>Igniter 1 (1000 psi)</td>
<td>II-16</td>
</tr>
<tr>
<td>II-16</td>
<td>Igniter 2 (1000 psi)</td>
<td>II-16</td>
</tr>
<tr>
<td>II-17</td>
<td>Igniter 3 (1000 psi)</td>
<td>II-16</td>
</tr>
<tr>
<td>II-18</td>
<td>Igniter 1 (1500 psi)</td>
<td>II-17</td>
</tr>
<tr>
<td>II-19</td>
<td>Igniter 2 (1500 psi)</td>
<td>II-17</td>
</tr>
<tr>
<td>II-20</td>
<td>Igniter 3 (1500 psi)</td>
<td>II-17</td>
</tr>
<tr>
<td>II-21</td>
<td>Igniter 1 (1800 psi)</td>
<td>II-18</td>
</tr>
<tr>
<td>II-22</td>
<td>Igniter 2 (1800 psi)</td>
<td>II-18</td>
</tr>
<tr>
<td>II-23</td>
<td>Igniter 3 (1800 psi)</td>
<td>II-18</td>
</tr>
<tr>
<td>II-24</td>
<td>Igniter 1 (100 psi)</td>
<td>II-19</td>
</tr>
<tr>
<td>II-25</td>
<td>Igniter 2 (100 psi)</td>
<td>II-19</td>
</tr>
<tr>
<td>II-26</td>
<td>Igniter 3 (100 psi)</td>
<td>II-19</td>
</tr>
<tr>
<td>II-27</td>
<td>Igniter 1 (500 psi)</td>
<td>II-20</td>
</tr>
<tr>
<td>II-28</td>
<td>Igniter 2 (500 psi)</td>
<td>II-20</td>
</tr>
<tr>
<td>II-29</td>
<td>Igniter 3 (500 psi)</td>
<td>II-20</td>
</tr>
<tr>
<td>II-30</td>
<td>Igniter 1 (1000 psi)</td>
<td>II-21</td>
</tr>
<tr>
<td>II-31</td>
<td>Igniter 2 (1000 psi)</td>
<td>II-21</td>
</tr>
<tr>
<td>II-32</td>
<td>Igniter 3 (1000 psi)</td>
<td>II-21</td>
</tr>
<tr>
<td>II-33</td>
<td>Igniter 1 (1500 psi)</td>
<td>II-22</td>
</tr>
<tr>
<td>II-34</td>
<td>Igniter 2 (1500 psi)</td>
<td>II-22</td>
</tr>
<tr>
<td>II-35</td>
<td>Igniter 3 (1500 psi)</td>
<td>II-22</td>
</tr>
<tr>
<td>II-36</td>
<td>Igniter 1 (1800 psi)</td>
<td>II-23</td>
</tr>
<tr>
<td>II-37</td>
<td>Igniter 2 (1800 psi)</td>
<td>II-23</td>
</tr>
<tr>
<td>II-38</td>
<td>Igniter 3 (1800 psi)</td>
<td>II-23</td>
</tr>
<tr>
<td>II-39</td>
<td>Igniter 1 (100 psi)</td>
<td>II-24</td>
</tr>
<tr>
<td>II-40</td>
<td>Igniter 2 (100 psi)</td>
<td>II-24</td>
</tr>
<tr>
<td>II-41</td>
<td>Igniter 3 (100 psi)</td>
<td>II-24</td>
</tr>
<tr>
<td>II-42</td>
<td>Igniter 1 (500 psi)</td>
<td>II-25</td>
</tr>
<tr>
<td>II-43</td>
<td>Igniter 2 (500 psi)</td>
<td>II-25</td>
</tr>
<tr>
<td>II-44</td>
<td>Igniter 3 (500 psi)</td>
<td>II-25</td>
</tr>
<tr>
<td>II-45</td>
<td>Igniter 1 (1000 psi)</td>
<td>II-26</td>
</tr>
<tr>
<td>II-46</td>
<td>Igniter 2 (1000 psi)</td>
<td>II-26</td>
</tr>
<tr>
<td>II-47</td>
<td>Igniter 3 (1000 psi)</td>
<td>II-26</td>
</tr>
<tr>
<td>FIG.</td>
<td>TITLE</td>
<td>PAGE NO.</td>
</tr>
<tr>
<td>--------</td>
<td>----------------------------------------------------------------------</td>
<td>----------</td>
</tr>
<tr>
<td>II-48</td>
<td>Igniter 1 (1500 psi)</td>
<td>II-27</td>
</tr>
<tr>
<td>II-49</td>
<td>Igniter 2 (1500 psi)</td>
<td>II-27</td>
</tr>
<tr>
<td>II-50</td>
<td>Igniter 3 (1500 psi)</td>
<td>II-27</td>
</tr>
<tr>
<td>II-51</td>
<td>Igniter 1 (1800 psi)</td>
<td>II-28</td>
</tr>
<tr>
<td>II-52</td>
<td>Igniter 2 (1800 psi)</td>
<td>II-28</td>
</tr>
<tr>
<td>II-53</td>
<td>Igniter 3 (1800 psi)</td>
<td>II-28</td>
</tr>
<tr>
<td>III-1(a)</td>
<td>Space shuttle SRM.</td>
<td>III-3</td>
</tr>
<tr>
<td>III-2</td>
<td>SRM star grain cross section</td>
<td>III-3</td>
</tr>
</tbody>
</table>
| III-3
|        | SRM star grain calculation domain                                    | III-12   |
| III-4
|        | Space Shuttle SRM igniter flow vs. time trace                         | III-12   |
| III-5
|        | Igniter flow-field comparison (cold flow)                            | III-15   |
| III-6
|        | Single port igniter cold-flow, $P_{\text{igniter}} = 6.8$ atm        | III-16   |
| III-7
|        | 45 degree igniter cold-flow, $P_{\text{igniter}} = 6.8$ atm          | III-17   |
| III-8
|        | Single port igniter cold-flow, $P_{\text{igniter}} = 34$ atm         | III-18   |
| III-9
<p>|        | 45 degree igniter cold-flow, $P_{\text{igniter}} = 34$ atm          | III-19   |
| III-10 | Typical slot area burn sequence (predicted)                          | III-20   |
| III-11 | Calculated Nu Contours (cold flow) single port igniter               | III-21   |
| III-12 | (Early) Ignition transient head-end pressure rise                    | III-22   |
| VI-1   | Two-dimensional data organization                                     | VI-4     |
| VI-2   | One-dimensional data organization                                     | VI-5     |
| VI-3   | X toolkit application design model                                    | VI-6     |
| VI-4   | Plot layout design                                                   | VI-8     |
| VI-5   | Example dialog diagram                                               | VI-12    |
| VII-1  | xplot top level dialog                                               | VII-1    |
| VII-2  | xplot input file editor dialog                                        | VII-2    |
| VII-3  | xplot execution setup dialog                                         | VII-4    |
| VII-4  | Plot manager dialog                                                  | VII-6    |
| VII-5  | xplot format dialog                                                  | VII-7    |
| VII-6  | xplot data attributes dialog                                         | VII-10   |
| VII-7  | xplot file editor dialog                                             | VII-11   |
| VII-8  | xplot view dataset dialog                                            | VII-12   |
| VII-9  | Pressure versus location at time=30 ms                                | VII-17   |
| VII-10 | Pressure versus location at time=60 ms                                | VII-17   |
| VII-11 | Pressure versus location at time=90 ms                                | VII-18   |
| VII-12 | Pressure versus location at time=120 ms                               | VII-18   |
| VII-13 | Pressure versus time at distance=3 cm                                 | VII-20   |
| VII-14 | Pressure versus time at distance=1330 cm                              | VII-20   |
| VII-15 | Mach number &amp; burn rate versus distance at time=90 ms                | VII-21   |
| VII-16 | Mach number &amp; burn rate versus distance at time=180 ms               | VII-22   |
| VII-17 | Reynolds number at exit versus time                                  | VII-23   |
| VII-18 | Motor thrust versus time                                              | VII-23   |</p>
<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>PAGE NO.</th>
</tr>
</thead>
<tbody>
<tr>
<td>II-1</td>
<td>Test matrix ...........................................................................</td>
<td>II-1</td>
</tr>
<tr>
<td>III-1</td>
<td>Constants used in burning rate law .........................................</td>
<td>III-9</td>
</tr>
<tr>
<td>III-2</td>
<td>Solid propellant properties ..................................................</td>
<td>III-10</td>
</tr>
<tr>
<td>IV-1</td>
<td>Sample burning perimeter fraction table ....................................</td>
<td>IV-3</td>
</tr>
<tr>
<td>IV-2</td>
<td>Caveny code input tables ....................................................</td>
<td>IV-3</td>
</tr>
<tr>
<td>IV-3</td>
<td>Circular perforated grain variables .........................................</td>
<td>IV-4</td>
</tr>
<tr>
<td>IV-4</td>
<td>Slotted grain variables ......................................................</td>
<td>IV-4</td>
</tr>
<tr>
<td>IV-5</td>
<td>Star grain variables ................................................................</td>
<td>IV-4</td>
</tr>
<tr>
<td>IV-6</td>
<td>Wagonwheel grain variables ...................................................</td>
<td>IV-5</td>
</tr>
<tr>
<td>IV-7</td>
<td>Dogbone grain variables ........................................................</td>
<td>IV-5</td>
</tr>
<tr>
<td>IV-8</td>
<td>Sample burning geometry table ................................................</td>
<td>IV-5</td>
</tr>
<tr>
<td>IV-9</td>
<td>Variables saved in caveny.plot file .........................................</td>
<td>IV-6</td>
</tr>
<tr>
<td>VI-1</td>
<td>Widgets used in xplot ................................................................</td>
<td>VI-11</td>
</tr>
<tr>
<td>VI-2</td>
<td>xplot source files ..................................................................</td>
<td>VI-14</td>
</tr>
<tr>
<td>VI-3</td>
<td>xplot header files ..................................................................</td>
<td>VI-14</td>
</tr>
<tr>
<td>VII-1</td>
<td>Input file editor dialog commands .............................................</td>
<td>VII-4</td>
</tr>
<tr>
<td>VII-2</td>
<td>Execution dialog commands ......................................................</td>
<td>VII-5</td>
</tr>
<tr>
<td>VII-3</td>
<td>Plot manager dialog commands ..................................................</td>
<td>VII-6</td>
</tr>
<tr>
<td>VII-4</td>
<td>Format dialog commands ................................................................</td>
<td>VII-9</td>
</tr>
<tr>
<td>VII-5</td>
<td>File editor dialog commands ...................................................</td>
<td>VII-12</td>
</tr>
<tr>
<td>VII-7</td>
<td>Command-line interface verbs .................................................</td>
<td>VII-14</td>
</tr>
<tr>
<td>VII-8</td>
<td>Xplot.clr file format ................................................................</td>
<td>VII-15</td>
</tr>
<tr>
<td>VII-9</td>
<td>Xplot.var file format ................................................................</td>
<td>VII-15</td>
</tr>
<tr>
<td>VII-10</td>
<td>Sample Xplot.def file ................................................................</td>
<td>VII-16</td>
</tr>
</tbody>
</table>
NOMENCLATURE

\[ b \] = star slot width
\[ C_1, C_2 \] = empirical constants appearing in turbulence model
\[ c_p \] = specific heat
\[ e \] = gas internal energy
\[ e_{\text{gp}} \] = internal energy of the gaseous burning propellant
\[ h_c \] = convective heat transfer coefficient
\[ k \] = kinetic energy of turbulent velocity fluctuations
\[ K \] = thermal conductivity
\[ M \] = Mach number
\[ Nu \] = Nusselt number
\[ p \] = pressure
\[ Pr \] = Prandtl number
\[ r \] = propellant burn rate
\[ Re \] = Reynolds number
\[ S \] = source term(s) in governing conservation equations
\[ t \] = time
\[ T \] = temperature
\[ u, v, w \] = velocity components
\[ V \] = \((u^2 + v^2)\)^{1/2}
\[ x, y, z \] = spatial coordinates
\[ \alpha \] = thermal diffusivity
\[ \gamma \] = ratio of specific heats
\[ \varepsilon \] = rate of dissipation of turbulent kinetic energy
\[ \sigma_k, \sigma_t \] = constants appearing in turbulence model
\[ \sigma_p \] = constant appearing in propellant burn rate equation

subscripts

\[ c \] = mass
\[ e \] = energy
\[ i \] = initial value
\[ \ell \] = laminar
\[ P \] = propellant
\[ \text{ref}, R \] = reference condition
\[ \text{star tip} \] = propellant grain star tip
\[ t \] = turbulent
\[ \text{wa} \] = adiabatic wall
\[ x_m, y_m \] = \(x, y\) components of linear momentum

superscript

\[ * \] = dimensionless quantity
I. INTRODUCTION

This report describes the results obtained from experimental and numerical analyses conducted for the purpose of developing an improved solid rocket motor ignition transient code. The specific objective of this work was to improve ability to predict the influence of the star grain on the ignition transient; in particular, the calculation of the flame spreading rate on the propellant surfaces inside the star slot. This report is divided into three main subject areas: 1) experimental analysis, 2) numerical analysis and 3) computer code modification, development and documentation.

Section II presents the results of a series of experiments conducted at NASA's George C. Marshall Space Flight Center (MSFC). These experiments utilized a model designed and constructed in the Aerospace Engineering Department at Auburn University. This model is a one-tenth scale simulation of the Space Shuttle Solid Rocket Motor (SRM) head-end section. The model was tested in the special test section of the 14"x14" trisonic wind tunnel at MSFC. The tests were cold-flow, using air, simulations of the internal flow through the igniter nozzle and the head-end section of the SRM. The air was supplied through a special high pressure line connected to the wind tunnel. There was no external flow around the model. The tests were designed to provide both qualitative and quantitative data on the interaction between the igniter plume and the star slots and the flow field within the star slots. Qualitative measurements were made using oil smears, Schlieren photography and by seeding the flow field with aluminum particles which were illuminated with a laser system and recorded on video tape. Quantitative measurements were made of the pressure distribution within a slot and of the heat transfer rates to the wall of a slot. The correlation of the data between the various experiments performed is excellent.

Section III provides the theoretical basis for the Computational Fluid Dynamic (CFD) model which was developed to analyze the flow field in a star slot and the results obtained from the CFD analysis. The CFD model was verified using the experimental data obtained from the cold flow tests described above. The primary objective of the CFD model was to provide data regarding the spread of the combustion process throughout the star slots. Based on these calculations a burning surface area versus time model is obtained for use in the ignition transient performance model of an SRM. The results of the analysis are compared to existing motor data and are shown to be in good agreement.

Sections IV-VII describe the modifications, interface, program description, and instructions for an improved ignition
transient computer code. The modifications and interface are to an existing one-dimensional ignition transient computer code. The code was chosen because it is well documented and provides an efficient means of implementing the modifications to the burning surface versus time calculations as determined from the analyses described above. The ability to account for the flame spreading in the star slot of the head-end section represents a significant step forward in the ability to accurately model the early portion an SRM's ignition transient. The last two sections describe the development of a pre- and post-processor for an ignition transient code. The code presented here uses a modified one-dimensional ignition transient code modified to account for the flame spreading in the star slots to solve for the ignition transient performance of an SRM. However, it would be relatively straightforward to modify the current version of the code to use a more sophisticated ignition transient model as a solver. The computer code is currently implemented on a SUN workstation using X-Windows. Instructions necessary for using the program in an X-Windows environment are included in Section VII. A sample input file and the code for the pre- and post-processor are included in the Appendices.

The work described in this report provides an excellent base for the development of a fully three-dimensional ignition transient performance prediction code using CFD techniques. The experimental database which has been generated from this work provides significant new insight into flow field phenomena occurring in the star slots of SRM's which have head-end star grains and head-end igniters.
II. EXPERIMENTAL ANALYSIS

One of the primary limitations of existing ignition transient prediction computer codes is an overestimation of the ignition delay and departure of the predicted pressure-time history curve from measured data in the early part of the transient. One of the reasons for this is the lack of data on the flow field in star grain sections of solid rocket motors. The experiments described in this section focused on the flow field in the star grain section of the Space Shuttle solid rocket motor (RSRM). The purpose of the experiments was to obtain a credible data base for the flow field patterns and heat transfer rates within the star slots. A one-tenth scale model of the Space Shuttle RSRM head-end star grain was designed and constructed for the experimental study. The availability of experimental flow field data during the ignition transient in solid rocket motors is very scarce. Conover conducted cold flow tests using a one-tenth scale model of the Space Shuttle solid rocket motor's head-end star grain section. Conover's tests used both a single port igniter, such as found on the RSRM and a three port igniter. The series of tests included Schlieren photographs of the igniter mounted in a plenum, oil smear data, pressure data and heat transfer coefficient measurements. Fifteen pressure ports were fitted in one side plate of a slot, while fifteen calorimeters to measure heat transfer coefficients were located in another slot. The data were taken at pressure levels from 100 to 1500 psi in 100 psi increments. The actual Space Shuttle igniter operates at approximately 2000 psi. Limits on the test facility prevented taking data at higher igniter chamber pressures. The static pressure data obtained provide some qualitative trends, but there was considerable scatter in the data when the igniter chamber pressure exceeded 1100 psi. This was probably due to the fact, as Conover states, that "above this pressure the side plates used to form the star grain slots were deflected to produce a one-sixteenth-inch gap where the plates come together at the star points," thus causing some leakage. There were also some apparent inconsistencies in the temperature data due to model warm-up during the test. However, the oil smear data provide a good indication of the recirculating flow pattern inside a slot. Even though Conover's experimental data provide useful information on the flow field inside the star slot, they should be considered preliminary in nature, and a starting point for a more in-depth investigation.

The present work describes a series of tests directed toward the collection of qualitative and quantitative data documenting the main characteristics of the flow in the head-end section of a solid rocket motor. In particular, the following objectives were to be accomplished with the test program.
1) Obtain information on the igniter plume structure and shape and its interaction with the star grain geometry.

2) Determine the region of the igniter plume impingement on the side walls of the slots of the RSRM star grain section model.

3) Determine the flow field characteristics of the subsonic, recirculating flow within the slot.

4) Measure the heat transfer coefficients at several locations inside the slot.

The above objectives were to be accomplished using the following techniques:

1) Flow visualization.

2) Oil smears

3) Static pressure measurements

4) Heat transfer coefficient measurements

5) Velocity measurements

Experimental Apparatus

The test article used in this investigation was a one-tenth scale, cold flow model based on the geometry of the Space Shuttle RSRM head-end section. The test article had four slots, as opposed to eleven slots in the actual Space Shuttle motor. A single port igniter model and two four-port igniter models were used in the tests.

The scale factor of 1:10 was derived from an analysis which matches the Reynolds number between the model flow and the full scale flow in the star grain section. Besides geometrical similitude, the primary scaling parameter to ensure proper similarity between the cold flow model and the real, full scale motor is the Reynolds number. Compressibility effects are important only in the igniter plume region, since the flow inside a star slot is essentially subsonic. However, the igniter mass flow rate is an important parameter, since it is thought to be responsible for entrainment of the flow, which determines the recirculation pattern. The value of the Reynolds number determines the nature of the viscous effects. The viscous effects in turn are related to the local convective heat transfer coefficient and therefore to the amount of heat transfer from the hot gases to the solid propellant. An exact match of Reynolds number between the model and full scale flow field is not necessary for similarity. Instead, generally good agreement between Reynolds numbers is a sufficient condition for the general studies of these flow fields.
as has been extensively documented in the literature.\textsuperscript{2-5}

Because of the complicated nature of the flow in the head-end section, it is difficult to define an overall Reynolds number for the entire slot. However, it seems appropriate to consider a representative Reynolds number, which can be defined at any point in the star slot, as:

\[ R_s = \frac{\rho V L}{\mu} \]

where \( \rho, V \) and \( \mu \) are the local density, velocity and viscosity, respectively, and \( L \) is a geometric reference length (for instance, the distance from a given point to the motor centerline). The product \( \rho V \) can be viewed as the local mass flux. It is assumed that this local mass flux is proportional to the overall mass flux that enters a star slot, which in turn is proportional to the mass flow rate that enters the star slot divided by the open area of the slot, so that:

\[ \rho V = \frac{m_{\text{slot,act}}}{bl} \]

where \( b \) and \( l \) are the slot width and slot length, respectively, as shown in Figure II-1.

The mass flow rate that actually enters a given slot depends on several factors, such as igniter mass flow rate, \( m_{ig} \), geometric dimensions of the slot and motor, and igniter plume shape. The functional dependency between these parameters is usually not known, but can be generally expressed as:

\[ m_{\text{slot,act}} = m_{\text{slot,avail}} f(\text{plume shape}) \]

To simplify the analysis, it can be assumed that three-dimensional effects are negligible and the mass flow available to the slot, \( m_{\text{slot,avail}} \), is the portion of igniter mass flow encompassed in the circular sector facing the same slot (see Fig. II-2), or

\[ m_{\text{slot,avail}} = m_{ig} \frac{\theta}{2\pi} = m_{ig} \frac{b}{2\pi r} \]

The factor \( b/(2\pi r) \) can be considered as the fraction of the port perimeter occupied by a slot.

Using Eqs. (II-2), (II-3) and (II-4), the Reynolds number can be expressed as:

\[ II-3 \]
Fig. II-1 Schematic of star slot

Fig. II-2 Cross-section of star slot region
Note that the Reynolds number, and thus the flow pattern inside the slot, is independent of the number of slots.

For proper scaling, let $R_e_1 = R_e_2$, where the subscript 1 refers to the actual motor and 2 refers to the model. Then the following assumptions are made:

1) All dimensions are exactly scaled.
2) The igniter plume shape is scaled for 1 and 2.

Note that the plume shape is determined mainly by the igniter nozzle area ratio, igniter stagnation pressure, $P_0$, and back pressure. When all dimensions are scaled, the area ratio is the same for the real srm and the scaled model. The back pressure is also the same, and equal to the ambient value. This latter condition is true at least until the point in time when a first ignition occurs. This suggests that the igniter total pressure ideally should be the same, that is 2000 psi.

Under the assumptions above, $f(\text{plume shape})$ drops out when equating the Reynolds numbers, and the following relationship is obtained:

$$ \frac{r_2 L_2}{r_1 L_1} = \frac{m_{i g 2}}{m_{i g 1}} \frac{\mu_1}{\mu_2} $$ \hspace{1cm} \text{(II-6)}

Then, defining the scaling factor as $S$, so that

$$ S = \frac{L_2}{L_1} = \frac{L_2}{L_2} = \frac{r_2}{r_1} \hspace{1cm} \text{(II-7)}

Equation (II-6) is reduced to:

$$ S = \frac{m_{i g 2}}{m_{i g 1}} \frac{\mu_1}{\mu_2} \hspace{1cm} \text{(II-8)}

At this point, an appropriate viscosity-temperature relationship must be chosen. Following Caveny, an empirical curve fit for

\hspace{1cm} II-5
typical combustion gases is such that viscosity, \( \mu \), is proportional to the gas temperature, \( T \), as

\[
\mu = T^{0.6}
\]

so that Eq. (II-8) becomes:

\[
S = \frac{m_{i2}}{m_{i1}} \left( \frac{T_1}{T_2} \right)^{0.6}
\]

Since \( m_{i1} \), \( T_1 \), and \( T_2 \) are known values, it appears from Eq. (II-10) that the scale factor, \( S \), can be arbitrarily chosen simply by varying the igniter model mass flow rate, \( m_{i1}' \). However, the igniter throat area is dependent on the scale factor to be determined. Furthermore, the igniter chamber pressure should be close to 2000 psi because of the assumption that the plume shape does not change from the real motor to the scaled model. This is more evident if Eq. (II-10) is rewritten in terms of stagnation pressures and temperatures, as:

\[
S = \frac{P_{o1}}{P_{o2}} \sqrt{\frac{R_2 T_{o2}}{R_1 T_{o1}}} \left( \frac{T_2}{T_1} \right)^{0.6} \sqrt{\frac{V_1}{V_2} \left( \frac{V_1+1}{V_1-1} \right)^{\frac{2}{3(V_1-1)}}} \left( \frac{V_2+1}{V_2-1} \right)^{\frac{2}{3(V_2-1)}}}
\]

Note that the term \( A_2'/A_1' \) is, itself, \( S^2 \).

For the cold-flow tests, the values of \( R_2 \) and \( V_2 \) for air are chosen for the model, and typical values of the temperatures are \( T_{o2} = 310^\text{o}K \), \( T_2 = 298^\text{o}K \). Since \( P_{o1} = P_{o2} = 2000 \) psi, the pressure term does not contribute to the value of \( S \). Representative values of the variables for the flow in the head-end section of the actual SRM are taken from Ref. 5.

Substituting in Eq. (II-11), a value equal to 0.098 is obtained for \( S \), which is very close to the chosen scale factor 1:10. Fortuitously, a one-tenth scale also defines a model size which is close to the largest size that would fit in the dimensions of the test section of the 14x14-inch tunnel employed for the tests.

The number of slots was determined based on different criteria. According to the previous analysis, the Reynolds number is independent of the number of slots. Therefore, one can ideally choose any convenient number. Even though the actual SRM head-end
section has eleven slots, only four slots are used in the scaled model. This represents a tradeoff between the desirability of fluid visualization in one (transparent) star slot and the requirements for all other test instrumentation.

The entire star grain section model, as well as the three igniter models, were fabricated from aluminum, except for the transparent slot which consists of two plexiglas plates. The total length of the model is 19.72 inches; the largest diameter is 16 inches. Figures II-3 and II-4 show a schematic representation of the entire model and of the igniters, respectively.

Each star slot is formed from two plates separated by a bottom spacer of suitable thickness. An insert at the head-end simulates the actual grain surface. The circular port is formed from four contoured pieces connected to the outer surface of the slot plates. Two plates are located at the upstream and downstream end to close the slots, and provide the attachment points for all the parts. The igniter is connected to the inner surface of the head-end plate. The single port igniter has a insert into which the nozzle has been cut. The single port igniter has a throat diameter of 0.6025 inches and a conical shape with a half-angle of 27.2 degrees. The area ratio is 1.428. The four-port igniters are simply four straight holes drilled in the igniter casing, each with a 0.3012 diameter. The four-port igniters are oriented so that each of the four jets centerlines are directed into a star slot. Figure II-5 shows the whole star grain section model. The three igniters used are shown in Fig. II-6.

The model is fully instrumented for measuring the parameters of interest as defined above.

Static pressure measurements are taken inside a single star slot and along two of the four contoured sectors forming the circular port of the model. Three pressure ports are located along each of the two contoured sectors. Twenty-eight static pressure ports are provided in one wall of a star slot. The measurements are taken at three different slot depths and consist of eight, ten, and eight pressure taps, respectively, as shown in Fig. II-7. The three depths are equally spaced along the height of the slot.

In the plate forming the wall adjacent to the one containing the pressure taps, twenty-eight calorimeters are installed to obtain heat transfer coefficient data. The calorimeters are placed at the same geometric locations as the pressure ports (Fig. II-9). Each calorimeter is mounted in a plug, flush with the inner surface of the slot plate. A detail of the calorimeter installation is given in Fig. II-8. In order to get accurate measurements of the heat transfer coefficients, the calorimeters were pre-heated before each measurement was taken.

A second star is used to obtain oil smear data. A silicone-
Fig. II-3 Schematic of head-end section

Fig. II-4 Schematic of single-port and multi-port igniters
Fig. II-5  Head-end section model

Fig. II-6  Igniter models

II-9
Fig. II-7  Location of pressure taps and calorimeters

Fig. II-8  Detail of calorimeter installation
based oil was used to apply a matrix of oil drops to the finished surface of one of the plates in the slot. After each run, well defined marks or smears indicated the local direction of the flow and gave a good overall picture of the flow field.

The transparent slot previously mentioned was used for real time flow visualization. A laser sheet was projected from the aft-end of the model and illuminated most of the transparent slot. Aluminum particles mixed with pure alcohol were injected into the slot and the aluminum particles were illuminated by the laser sheet. The movement of the particles was clearly visible in the transparent slot and provided an excellent qualitative measurement of the behavior of the flow field in the slot. The flow visualization obtained using the aluminum particles gave a more detailed picture of the flow patterns in the slot than was possible with the oil smears. In addition, the real time nature of measurements provided a means for studying the dynamic characteristics of the flow field. Video tape recordings of these experiments were made to document the measurements.

The fourth slot was initially intended for making hot-wire anemometry measurements. However, because of difficulties associated with flow blockage in the small slot and difficulties in making measurements in or near the high speed (high subsonic or low supersonic) plume, this measurement was abandoned for the present investigation.

**Test Facility**

The cold-flow tests used the special test section in the NASA Marshall Space Flight Center 14x14-inch trisonic wind tunnel. The tunnel operated as an intermittent blow-down wind tunnel from storage pressure to atmospheric exhaust. The full Mach number capability was not needed for the test program which was carried out. Instead only a high pressure internal flow through the special test section was required. The high pressure air passed through the hollow centerbody of the tunnel, into a pipe connected to the head-end plate of the model. It was then exhausted into the special test section through the igniter models. There was no external flow around the model. A venturi was installed upstream of the model to determine the mass flow rate through the igniter. Additional information regarding the NASA/MSFC trisonic wind tunnel is given in Ref. 6.

**Test Plan**

The test program included two main series of tests. Table II-1 shows the test matrix which was used for each series of tests. In the first series each igniter model was placed in the test section without the star grain portion of the model. Air flow at pressures of 100, 500, 1000, 1500 and 1800 psi passed through each of the igniters used in the experiments and mass
flow rates corresponding to each pressure were recorded. A Schlieren system and video tape recorder were used during each run to examine the plume shape at various pressures. This was done to establish a reference for the plume geometry which could be compared with the plume geometry observed with the star grain in place.

The second series of tests used the entire head-end section model along with each of the three igniters. Oil smear, flow visualization, static pressure and heat transfer coefficient measurements were made at each condition shown in the test matrix of Table II-1. It should be noted that the test condition at 1800 psi approximates the design condition of 2000 psi at which similitude between the one-tenth scale model and the actual Space Shuttle RSRM flow field is achieved. The value of 1800 psi was used because it represents the upper limit on the facility at the mass flow rates necessary for the tests.

<table>
<thead>
<tr>
<th>Igniter Angle (deg)</th>
<th>Chamber Pressure (psi)</th>
<th>100</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>1800</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>22.5</td>
<td></td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>45</td>
<td></td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Oil Smear Test Results

Oil smears were taken for each of the fifteen test conditions shown in Table II-1. The oil smears were generated from a pattern of oil droplets placed on one side of a slot. The spacing between the droplets was approximately one-half inch. Figs. II-9 through II-23 show photographs of the oil smears generated for each of the fifteen test conditions. The oil smears provide considerable detail regarding the direction of the primary flow in the slots, the region of the igniter plume impingement, and the recirculation patterns which occur in the slot. Although qualitative in nature, this data showed good agreement with the data from the CFD analyses presented in refs. 7-9, which are summarized in Section III of this report.
Results From Static Pressure Measurements

Static pressure measurements were made at 27 locations on the surface of one of the slots. Fig. II-7 shows the location and numbering scheme used for the static pressure ports. Static pressure distributions obtained from these measurements are shown in Figs. II-24 through II-38. This data confirms the quantitative data obtained from the oil smear tests with regard to the location of the main flow paths, recirculation regions, stagnation points and "dead" regions within the slot. The data obtained agrees well with the CFD results which will be discussed in Section III, where a comparison will be given between the experimental data and the CFD results.

Results From Heat Transfer Measurements

Calorimeters were placed in a slot face adjacent to the slot face where the static pressure measurements were made. The calorimeters were located at points corresponding to the 27 locations shown in Fig. II-7. Because of the lack of a sufficient number of calorimeters to measure data at all 27 locations simultaneously, two runs were made using 15 calorimeters in each run. Three calorimeters were not moved between runs to insure that consistent data were being obtained between the two runs. The results for the measured heat transfer coefficients are shown in Figs. II-39 through II-53.
Fig. II-9 Igniter 1 (100 psi)

Fig. II-10 Igniter 2 (100 psi)

Fig. II-11 Igniter 3 (100 psi)
Fig. II-15 Igniter 1 (1000 psi)

Fig. II-16 Igniter 2 (1000 psi)

Fig. II-17 Igniter 3 (1000 psi)
Fig. II-21  Igniter 1 (1800 psi)

Fig. II-22  Igniter 2 (1800 psi)

Fig. II-23  Igniter 3 (1800 psi)
Fig. II-24 Igniter 1 (100 psi)

Fig. II-25 Igniter 2 (100 psi)

Fig. II-26 Igniter 3 (100 psi)
Measured Pressure, Single Port Igniter

\[ P_{\text{igniter}} = 500 \text{ psi} \]

Fig. II-27  Igniter 1 (500 psi)

Measured Pressure, 22 deg Igniter

\[ P_{\text{igniter}} = 500 \text{ psi} \]

Fig. II-28  Igniter 2 (500 psi)

Measured Pressure, 45 deg Igniter

\[ P_{\text{igniter}} = 100 \text{ psi} \]

Fig. II-29  Igniter 3 (500 psi)

II-20
Fig. II-30  Igniter 1 (1000 psi)

Fig. II-31  Igniter 2 (1000 psi)

Fig. II-32  Igniter 3 (1000 psi)
Measured Pressure, Single Port Igniter
$P_{\text{igniter}} = 1500$ psi

Fig. II-33 Igniter 1 (1500 psi)

Measured Pressure, 22 deg Igniter
$P_{\text{igniter}} = 1500$ psi

Fig. II-34 Igniter 2 (1500 psi)

Measured Pressure, 45 deg Igniter
$P_{\text{igniter}} = 1500$ psi

Fig. II-35 Igniter 3 (1500 psi)

II-22
Measured Pressure, Single Port Igniter
$P_{\text{igniter}} = 1800 \text{ psi}$

Fig. II-36 Igniter 1 (1800 psi)

Measured Pressure, 22 deg Igniter
$P_{\text{igniter}} = 1800 \text{ psi}$

Fig. II-37 Igniter 2 (1800 psi)

Measured Pressure, 45 deg Igniter
$P_{\text{igniter}} = 1800 \text{ psi}$

Fig. II-38 Igniter 3 (1800 psi)

II-23
Measured Heat Transfer, Single Port Igniter
\[ P_{\text{Igniter}} = 6.8 \text{ atm (100 psi)} \]

Fig. II-39 Igniter 1 (100 psi)

Measured Heat Transfer, 22.5° Igniter
\[ P_{\text{Igniter}} = 100 \text{ psi} \]

Fig. II-40 Igniter 2 (100 psi)

Measured Heat Transfer, 45° Igniter
\[ P_{\text{Igniter}} = 6.8 \text{ atm (100 psi)} \]

Fig. II-41 Igniter 3 (100 psi)
Fig. II-42  Igniter 1 (500 psi)

Fig. II-43  Igniter 2 (500 psi)

Fig. II-44  Igniter 3 (500 psi)
Fig. II-45 Igniter 1 (1000 psi)

Fig. II-46 Igniter 2 (1000 psi)

Fig. II-47 Igniter 3 (1000 psi)
Fig. II-48 Igniter 1 (1500 psi)

Fig. II-49 Igniter 2 (1500 psi)

Fig. II-50 Igniter 3 (1500 psi)
Fig. II-51 Igniter 1 (1800 psi)

Fig. II-52 Igniter 2 (1800 psi)

Fig. II-53 Igniter 3 (1800 psi)
References


III. THEORETICAL/NUMERICAL ANALYSIS

Introduction

The ignition transient of an SRM employing a pyrogen igniter can be defined as the time interval from the onset of the igniter flow to the time a quasi-steady flow develops. The starting transient is traditionally divided into three phases: the induction interval, or ignition lag; flame spreading; and chamber filling. The induction interval begins when the igniter flow is initiated and ends when a point on the propellant surface reaches a critical ignition temperature, and a flame first appears. The flame spreading phase follows, ending when the entire propellant surface is ignited. Following this is the chamber filling phase, during which rapid chamber pressurization occurs due to the energy and mass addition from the burning propellant. A peak pressure may occur, followed by a pressure decrease towards an equilibrium value, attained when mass production by the propellant equals the mass outflow from the motor nozzle. Numerous studies have been directed at the analysis of SRM ignition transient phenomena. As discussed by Peretz et al., these analyses can generally be categorized into three groups: (1) lumped chamber parameter, \( P(t) \) models; (2) one dimensional, quasi-steady flow, \( P(x) \) models; and (3) temporal and one-dimensional flow field, \( P(x,t) \) models. The simplest analyses fall into the first group; a uniform chamber pressure is assumed and an equation for \( \frac{dP_{chamber}}{dt} \) is derived and integrated to obtain a pressure-time trace. The flame spreading speed is assumed to be a known constant. In \( P(x) \) type models, flow property distributions are considered at each instant of time and one-dimensional steady state conservation equations are solved along the motor axis. As in the \( P(t) \) type models, the flame spreading speed is not part of the solution but rather must be input. In \( P(x,t) \) type models both spatial and temporal property variations are considered. A series of control volume increments are assumed along the motor axis and a set of time dependent one-dimensional conservation equations are solved. The flame spreading speed can be obtained as part of the solution if convective heat transfer to the propellant grain is taken into account. A widely used model of this type is that developed by Caveny and Kuo.10

Jasper Lal et al., have recently developed a one-dimensional model which takes canted pyrogen igniters into account by modifying the heat transfer analysis to include direct igniter plume impingement on the solid propellant surface. Even more recently, Johnston has presented a numerical procedure for the analysis of internal flows in a solid rocket motor wherein an unsteady, axisymmetric solution of the Euler equations is combined with simple convective and radiative fluid heat transfer models and an unsteady one-dimensional heat conduction solution for the propellant grain. Flow in the star grain slots is not directly calculated. Instead, burn rate constants are adjusted to account for the variable area in the star grain region. Results are presented for a Titan 5 segment SRM, a Titan 7 segment SRM, and the Space Shuttle SRM. Of these three motors, the Space Shuttle
SRM has the more pronounced axial grooves in the star grain, and agreement of Johnston's model with pressure-time data in the head-end star grain section of this motor is acknowledged to be poor.

In general, it can be argued that predictions agree quantitatively well with test data for motors such as those used on the Space Shuttle, with the exception of the time period which directly involves burning of the head-end star grain segment. It may be argued that discrepancies arise primarily from three factors: (1) the flow field is usually assumed to be one dimensional; (2) the star geometry in the head-end segment is approximated by variations in port area and burning perimeter of the grain; and (3) the igniter flow field is not taken into account. The present analysis seeks to address these issues.

Conservation Equations

In this investigation, the Space Shuttle solid rocket motor (SRM) is taken as the reference motor design. It is characterized by a large length-to-diameter ratio and by a small port-to-nozzle throat area ratio. The reference motor is divided into four segments, as shown in Fig. III-1(a). The head-end, star shaped region of the solid propellant grain contains eleven slots; a cross section of the head-end segment is shown in Fig. III-1(b).

The flow field to be analyzed is extremely complex; it is unsteady, multi-dimensional, turbulent, and compressible. Further, the flow field is divided into a supersonic core region, defined by the expanding gases from the igniter, and a subsonic region inside the star slot. The two-dimensional, unsteady, compressible Navier-Stokes equations, neglecting body forces and heat source terms, are employed. The flow field is described by adopting a cylindrical coordinate system for the port region from the motor centerline to the star grain tips, and a rectangular cartesian coordinate system for the region of the flow inside the star slot.

Although the flow-field is three dimensional in the head-end section of the motor, the star grain cross sectional shape of the propellant segment implies that a certain number of planes of symmetry exist, so that it is possible to restrict the domain to a single sector, as shown in Fig. III-2. The two-dimensional Navier-Stokes equations are obtained by averaging the full, three-dimensional Navier-Stokes equations (with a formal integration) in a direction perpendicular to the plane of symmetry of the star slot. Averaging is carried out along the azimuthal coordinate, $\theta$, in the port region, and in the z-direction inside the star slot. The calculated flow field variables thus represent an average value in the domain considered (area shaded in Fig. III-2).

The governing equations can most conveniently be solved in dimensionless form. For the flow under investigation, no "natural" free stream parameters exist. Consequently, the values of the igniter exit parameters at the condition of maximum igniter mass flow have been chosen as reference conditions, with the distance from the motor centerline to the bottom of the slot taken as the...
Figure III-1: (a) Space Shuttle SRM; (b) SRM Star Grain Cross Section

Figure III-2: SRM Star Grain Calculation Domain
reference length. It should be noted that the igniter mass flow is itself a function of time; this will be discussed in more detail later.

The dimensionless governing equations then take the following form:

**continuity:**
\[
\frac{\partial p^*}{\partial t^*} + \frac{\partial p^* u^*}{\partial x^*} + \frac{\partial p^* v^*}{\partial y^*} = S^*_g
\]
III-1

**x-momentum:**
\[
\frac{\partial p^* u^*}{\partial t^*} + \frac{\partial p^* u^* u^*}{\partial x^*} + \frac{\partial p^* u^* v^*}{\partial y^*} = -\frac{\partial p^*}{\partial x^*} + \frac{1}{Re}\frac{\partial u^*}{\partial x^*}\left[\frac{4}{3} \frac{\partial u^*}{\partial x^*} - \frac{2}{3} \frac{\partial v^*}{\partial y^*}\right] + S^*_x
\]
III-2

\[
+ \frac{1}{Re}\frac{\partial u^*}{\partial y^*}\left[\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial x^*}\right] + \frac{1}{Re}\mu^*\left[\frac{4}{3} \frac{\partial v^*}{\partial x^*} + \frac{1}{3} \frac{\partial v^*}{\partial y^*}\right] + S^*_x
\]

**y-momentum:**
\[
\frac{\partial p^* v^*}{\partial t^*} + \frac{\partial p^* u^* v^*}{\partial x^*} + \frac{\partial p^* v^* v^*}{\partial y^*} = -\frac{\partial p^*}{\partial y^*} + \frac{1}{Re}\frac{\partial v^*}{\partial y^*}\left[\frac{4}{3} \frac{\partial v^*}{\partial x^*} - \frac{2}{3} \frac{\partial u^*}{\partial x^*}\right] + S^*_y
\]
III-3

\[
+ \frac{1}{Re}\frac{\partial v^*}{\partial x^*}\left[\frac{\partial v^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*}\right] + \frac{1}{Re}\mu^*\left[\frac{4}{3} \frac{\partial u^*}{\partial x^*} + \frac{1}{3} \frac{\partial u^*}{\partial y^*}\right] + S^*_y
\]

**energy:**
\[
\frac{\partial p^* e^*}{\partial t^*} + \frac{\partial p^* u^* e^*}{\partial x^*} + \frac{\partial p^* v^* e^*}{\partial y^*} = \frac{1}{\gamma - 1} Pr Re M^2\left(\frac{\partial}{\partial x^*}(\kappa^* \frac{\partial T^*}{\partial x^*}) + \frac{\partial}{\partial y^*}(\kappa^* \frac{\partial T^*}{\partial y^*})\right) - p^*\left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*}\right) + \frac{2}{Re} \mu^*\left(\frac{\partial u^*}{\partial x^*}\right)^2 + \left(\frac{\partial v^*}{\partial y^*}\right)^2 + S^*_e
\]
III-4

\[
\frac{1}{2} \left(\frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*}\right)^2 + S^*_e
\]

III-4
equations of state:

\[ p^* = \frac{\rho^* T^*}{\gamma M^2} \]  

\[ e^* = \frac{T^*}{\gamma (\gamma - 1) M^2} \]  

The conservation equation source terms are given by:

\[ S_c = -\frac{\rho^* v^*}{y^*} \]  

\[ S_{m} = -\frac{\rho^* u^* v^*}{y^*} \left( \frac{2}{3} \frac{1}{Re} \frac{\partial \mu^*}{\partial x^*} \frac{v^*}{y^*} + \frac{1}{Re} \frac{\mu^*}{y^*} \left( \frac{1}{3} \frac{\partial v^*}{\partial x^*} + \frac{\partial u^*}{\partial y^*} \right) \right) \]  

\[ S_{y} = -\frac{\rho^* v^* v^2}{y^*} \left( \frac{2}{3} \frac{1}{Re} \frac{\partial \mu^*}{\partial y^*} \frac{v^*}{y^*} + \frac{4}{3} \frac{1}{Re} \frac{\mu^*}{y^*} \left( \frac{\partial v^*}{\partial y^*} - \frac{v^*}{y^*} \right) \right) \]  

\[ S_0 = -\frac{\rho^* e^* v^*}{y^*} + \frac{1}{\gamma - 1} \frac{1}{Pr Re M^2} \frac{K^*}{y^*} \frac{\partial T^*}{\partial y^*} - \frac{\rho^* v^*}{y^*} + \frac{2}{3} \frac{1}{Re} \frac{\mu^* (v^* y^* - v^* y^*)^2}{y^*} \]  

\[ -\frac{2}{3} \frac{1}{Re} \frac{\mu^*}{y^*} \left( \frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} \right) \frac{v^*}{y^*} \]  

for the circular port region of the computational domain. For the slot itself, these terms are given by:

\[ S_c = 2 \rho_p \frac{r^*}{b^*} \]
\[ S_{zm} = -2 \frac{1}{3} Re \frac{\partial u^*}{\partial x^*} \frac{\partial w^*}{\partial z^*} + 2 \frac{1}{b^* Re} \left( \mu^* \frac{\partial u^*}{\partial z^*} \right) \] III-12

\[ S_{ym} = -2 \frac{1}{3} Re \frac{\partial u^*}{\partial y^*} \frac{\partial w^*}{\partial z^*} + 2 \frac{1}{b^* Re} \left( \mu^* \frac{\partial v^*}{\partial z^*} \right) \] III-13

\[ S_k = \frac{2}{Re} \mu^* \left[ \left( \frac{\partial w^*}{\partial z^*} \right)^2 + \frac{1}{2} \left( \frac{\partial u^*}{\partial z^*} \right)^2 + \left( \frac{\partial v^*}{\partial z^*} \right)^2 \right] \] III-14

\[ -2 \frac{1}{3} Re \mu^* \left( \frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} + \frac{\partial w^*}{\partial z^*} \right)^2 + A \]

where

\[ A = -2 \rho_p \frac{\mu^*}{L^*} e_{\infty} \] (grain burning) III-15

\[ A = \frac{1}{\gamma - 1} \frac{1}{Pr} \frac{1}{Re} \frac{2}{b^*} \left( K^* \frac{\partial T^*}{\partial z^*} \right) \] (no grain burning) III-16

**Turbulence Model**

The low Reynolds number form of the two-equation, k-\( \varepsilon \) model developed by Jones and Launder\(^{14,15}\) is used in the present analysis. This model employs two variables, the kinetic energy of turbulent velocity fluctuations, \( k \), and the rate of dissipation of kinetic energy, \( \varepsilon \), from which the turbulent viscosity, \( \mu_t \), is calculated. The turbulent thermal conductivity, \( K_t \), is obtained by assuming a constant value of the turbulent Prandtl number, \( \sigma_t = 0.91 \).\(^{16}\) The turbulent kinetic energy and energy dissipation rate are determined from the solution of the following differential equations:

**k-equation:**

\[
\frac{\partial \rho^* k}{\partial t^*} + \frac{\partial \rho^* u^* k}{\partial x^*} + \frac{\partial \rho^* v^* k}{\partial y^*} = \frac{1}{Re} \frac{\partial}{\partial x^*} \left[ \left( \mu_t^* + \frac{\rho^*}{\sigma_k} \right) \frac{\partial k}{\partial x^*} \right] + \frac{\partial}{\partial y^*} \left[ \left( \mu_t^* + \frac{\rho^*}{\sigma_k} \right) \frac{\partial k}{\partial y^*} \right] + \frac{1}{Re} \left( P_1 + P_2 \right) - \frac{1}{Re} \rho^* e - \frac{2}{Re} \mu^* \left[ \left( \frac{\partial k}{\partial x^*} \right)^2 + \left( \frac{\partial k}{\partial y^*} \right)^2 \right] + S_k
\] III-17
\[
\frac{\partial \rho'^*}{\partial t'^*} + \frac{\partial \rho'^* u'^*}{\partial x'^*} + \frac{\partial \rho'^* v'^*}{\partial y'^*} = \frac{1}{Re} \left\{ \frac{\partial}{\partial x'^*} \left[ (\mu'^*_t + \frac{\mu'^*_\epsilon}{\alpha'_\epsilon}) \frac{\partial \epsilon'^*}{\partial x'^*} \right] + \frac{\partial}{\partial y'^*} \left[ (\mu'^*_t + \frac{\mu'^*_\epsilon}{\alpha'_\epsilon}) \frac{\partial \epsilon'^*}{\partial y'^*} \right] \right\} \\
+ \frac{1}{Re} C_1 \frac{\epsilon}{k} P_1 + C_1 \frac{\epsilon}{k} P_2 - \frac{1}{Re} C_2 \rho'^* \epsilon^2 - \frac{2}{Re} \mu'^*_t \mu'^*_\epsilon \left[ \left( \frac{\partial u'^*}{\partial x'^*} \right)^2 + \left( \frac{\partial v'^*}{\partial y'^*} \right)^2 \right] \\
- \frac{2}{Re} \frac{\mu'^*_t \mu'^*_\epsilon}{\rho'^*} \left[ \frac{(\partial^2 v'^*)}{\partial x'^* \partial y'^*} + 2 \left( \frac{\partial u'^*}{\partial x'^*} \right)^2 + 2 \left( \frac{\partial v'^*}{\partial x'^*} \right)^2 \right] + S_{\epsilon} \quad \text{III-18}
\]

where

\[
P_1 = \mu'^*_t \frac{4}{3} \left\{ \left( \frac{\partial u'^*}{\partial x'^*} \right)^2 + \left( \frac{\partial v'^*}{\partial y'^*} \right)^2 - \frac{\partial u'^*}{\partial x'^*} \frac{\partial v'^*}{\partial y'^*} \right\} \quad \text{III-19}
\]

and

\[
P_2 = -\frac{2}{3} \rho'^* k \left( \frac{\partial u'^*}{\partial x'^*} + \frac{\partial v'^*}{\partial y'^*} \right) \quad \text{III-20}
\]

The source terms are given by

\[
S_k = -\rho'^* v'^* k - \frac{1}{Re} \frac{u'^* + v'^*}{y'^*} \frac{\partial k}{\partial x'^*} - \frac{1}{Re} \frac{2}{3} \frac{u'^* + v'^*}{y'^*} \frac{\partial u'^*}{\partial x'^*} + \frac{\partial v'^*}{\partial y'^*} \quad \text{III-21}
\]

\[
S_\epsilon = -\rho'^* v'^* k - \frac{1}{Re} \frac{\mu'^*_t + \frac{\mu'^*_\epsilon}{\alpha'_\epsilon}}{y'^*} \frac{\partial \epsilon'^*}{\partial x'^*} - \frac{1}{Re} \frac{2}{3} \frac{\mu'^*_t + \frac{\mu'^*_\epsilon}{\alpha'_\epsilon}}{y'^*} \frac{\partial u'^*}{\partial x'^*} + \frac{\partial v'^*}{\partial y'^*} \quad \text{III-22}
\]

in the circular port region of the domain, and by

\[
S_k = P_v \quad \text{III-23}
\]

\[
S_\epsilon = C_1 \frac{\epsilon}{k} P_v \quad \text{III-24}
\]

III-7
Heat Transfer Relations

The goal of the present analysis is to examine the interaction between the igniter plume, the developing flow field within the head-end star grain slots, and the rate of flame spread over the grain surface. This, in turn, may provide insight into the appropriateness of a particular grain design or a particular igniter design for a given SRM. Many previous analyses utilize a convection heat transfer model, while others, such as Johnston, utilize a combined convection-radiation model. The present analysis utilizes a simple convection-only model developed by Kays and Leung, and used previously to correlate heat transfer within the O-ring gap of the Space Shuttle nozzle-to-case joint, given by

$$\text{Nu} = \frac{0.152 \text{Re}^{0.9} \text{Pr}}{0.833[2.25 \ln(0.114 \text{Re}^{0.9}) + 13.2 \text{Pr} - 5.8]}$$

where the Reynolds number is based on the hydraulic diameter of a single star grain slot. Fluid properties are based on a "reference temperature" denoted as $T_x$.

$$T_x = T + 0.5(T_{wall} - T) + 0.22(T_{wa} - T)$$

and the adiabatic wall temperature is calculated from

$$T_{wa} = T + \frac{3}{\text{Pr}} \frac{V^2}{2c_p}$$

The propellant burning rate is assumed to be of the form

$$r = r_{ref} \left( \frac{P}{P_{ref}} \right)^n \exp\left[ q_p (T_p - T_{ref}) \right]$$

The constants appearing in Eq. (27) are defined in Table III-1. Erosive burning is assumed to be negligible during the very early portion of the ignition transient.
Table III-1. Constants used in burning rate law

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{ref} )</td>
<td>0.01078</td>
<td>m sec(^{-1} )</td>
</tr>
<tr>
<td>( P_{ref} )</td>
<td>6898.2</td>
<td>KPa</td>
</tr>
<tr>
<td>( n )</td>
<td>0.35</td>
<td>-</td>
</tr>
<tr>
<td>( \sigma_p )</td>
<td>0.002</td>
<td>K(^{-1} )</td>
</tr>
<tr>
<td>( T_{ref} )</td>
<td>300</td>
<td>K</td>
</tr>
</tbody>
</table>

In order to determine when a given element of the solid propellant reaches a critical ignition temperature, one must know the surface temperature of the solid grain. This, in turn, depends on the amount of heat transferred to the grain from the hot gases.

The grain is considered to be a semi-infinite slab whose temperature is initially uniform. Heat transfer to the slab is assumed to be one-dimensional. Thus

\[
\frac{\partial T_p}{\partial t} = \sigma_p \frac{\partial^2 T_p}{\partial z^2} \tag{III-30}
\]

with the boundary conditions

\[
T_p(t, \infty) = T_{p_1} ; \quad \frac{\partial T_p(t, 0)}{\partial z} = -\frac{h_p}{K_p} [T - T_p(t, 0)] \tag{III-31}
\]

and the initial condition

\[
T_p(0, z) = T_{p_1} \tag{III-32}
\]

The assumption of one dimensional conduction heat transfer implies that ignition of adjacent grain surface elements is attributed to direct heat transfer from the hot gas only. This appears to be a reasonable assumption because of the low thermal conductivity of the solid propellant. Assumed physical properties of the propellant grain material are given in Table III-2.
Table III-2. Solid propellant properties

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\rho_p$</td>
<td>1758</td>
<td>Kg m$^{-3}$</td>
</tr>
<tr>
<td>$K_p$</td>
<td>0.4605</td>
<td>W m$^{-1}$ K$^{-1}$</td>
</tr>
<tr>
<td>$(c_p)_p$</td>
<td>1256</td>
<td>J Kg$^{-1}$ K$^{-1}$</td>
</tr>
<tr>
<td>$T_p$, critical</td>
<td>850</td>
<td>K</td>
</tr>
</tbody>
</table>

The initial propellant temperature is assumed to be $T_{pi} = 298$°K.

The (slot) wall shear stress must be approximated in order to provide closure for the governing equations. A velocity profile across the slot width (in the z-direction) is assumed for the velocity components $u$ and $v$. Pal's polynomial form of the velocity profile for a flat duct is employed. For example, the $u$ component of velocity is

$$
\frac{u}{u_{max}} = 1 - \frac{p-1}{p-1} \left( \frac{z}{b/2} \right)^2 - \frac{q-1}{p-1} \left( \frac{z}{b/2} \right)^{2p}
$$

where $u_{max}$ is the velocity at the symmetry plane of the duct (slot), $p = 11.06$, and $q = 16^{20}$. Although Eq. (III-33) applies specifically to fully developed flow, it is used here to account for wall shear stress effects even though the flow is time variant. It should be noted that frictional effects at the slot walls are accounted for only in the unignited portion of the propellant; they are neglected for the ignited portion because of the strong transverse velocity component due to mass injection from the burning propellant.

Finally, the transverse velocity component $w(z)$ is assumed to be linear, with $w = 0$ at the slot plane of symmetry; $w$ at the slot wall can be determined from the known burning rate. Hence,

$$
w_{wall} = \frac{\rho_p x}{\rho} \quad \frac{\partial w}{\partial z} = -\frac{2}{b} w_{wall}
$$

Numerical Technique

The numerical solution of the equations of motion is obtained utilizing the explicit, time-dependent, predictor-corrector finite difference method developed by MacCormack. This method has been widely used for the numerical solution of a variety of fluid
dynamics problems, including those containing mixed subsonic-supersonic flow regions, as in the present investigation. MacCormack's predictor corrector technique is well described in the existing literature.\textsuperscript{22,23} In the present investigation, an explicit, fourth order numerical dissipation scheme has been introduced into the set of equations to damp numerical oscillations induced by the severe gradients in the flow field associated with the developing igniter flow. The fourth-order damping scheme introduced by Holst\textsuperscript{24} and modified by Berman\textsuperscript{25} and Kuruvila\textsuperscript{26} is employed. This scheme involves certain free adjustable parameters, $C_x$ and $C_y$, usually referred to as damping or dissipation coefficients. It is recommended in the literature\textsuperscript{22,23} that $C_x$, $C_y$ be such that $0 \leq C_x, C_y \leq 0.5$.

A uniform rectangular grid is employed in the portion of the domain that represents the head-end section, which includes the star slot segment plus a distance equal to the port diameter. For this region a $92 \times 63$ grid has been used, as shown in Figure III-3. A grid of unequal spacing in the $x$-direction is used in the extended portion of the domain from the head-end segment to the motor exit. The scheme of Cebeci and Smith,\textsuperscript{27} in which grid spacing is increased by a fixed percentage from an initial point, is utilized.

\textbf{Initial and Boundary Conditions}

In time marching problems, the initial conditions should in no way affect the steady state results. In theory, any initial conditions can be chosen. Of course, for an arbitrary set of initial conditions, the transient has no physical meaning and only the steady state condition is a meaningful representation of the flow field.

Since in this study the starting transient is of primary importance, the initial conditions must reflect the actual values of the flow field variables at time $t = 0$. Therefore, both components of the velocity have been set equal to zero, $u = v = 0$, and ambient values have been chosen for pressure and temperature everywhere in the flow field. Density and internal energy are obtained from the equation of state. The turbulent kinetic energy, $k$, and its dissipation rate, $\varepsilon$, are also zero initially. However, imposing this value would lead to a singularity in the $k$ and $\varepsilon$ equations; thus, a very small value has been assigned to these two variables.

Boundary conditions must also be specified for all of the dependent variables, $u$, $v$, $p$, $e$, $k$, $\varepsilon$, along with corresponding values of $p$ and $T$. Of primary importance is the specification of the time variant conditions from the developing igniter flow at the inflow boundary of the calculation domain. The conditions specified are those from the single-port igniter used in the Space Shuttle SRM. The igniter mass flow vs. time trace is shown in Figure III-4, using data from Ref. 28. From the known igniter mass flow rate vs. time trace and the geometry of the igniter nozzle, the values of the flow variables $u$, $p$, $T$ at the igniter nozzle

III-11
Figure III-3: Computational Grid

Figure III-4: Space Shuttle SRM Igniter Flow vs. Time Trace
exit are calculated using one-dimensional nozzle flow theory; the radial velocity is assumed to be zero, \( v=0 \), and the internal energy is calculated from the known temperature. Note that the numerical method employed is a shock capturing technique, so that the shock waves associated with the igniter plume are embedded in the solution. Inflow boundary conditions for \( k \) and \( \varepsilon \) must be also specified. Since no experimentally measured profiles are available, the turbulent kinetic energy is assumed to be equal to a small percentage of the freestream kinetic energy, and the energy dissipation rate is taken from an empirical relation used, for instance, in Ref. 29. Along most solid walls of the computational domain the "no slip" condition is enforced, that is, \( u = v = 0 \); also, due to the low Reynolds number form of the two-equation turbulence model, \( k = 0 \) and \( \varepsilon = 0 \). For these walls the temperature boundary condition is specified by assuming an adiabatic wall. The exception is the cylindrical port portion of the domain downstream of the star slot section. For this region, the wall temperature is determined in the same manner as in the upstream star slot, and a "blowing wall" \( (v \neq 0) \) condition is imposed to account for mass addition due to burning. The top boundary of the computational domain represents the centerline of the motor, so that a symmetry boundary condition is enforced here.

The outflow boundary at the aft end of the motor represents the final location where values of the dependent variables must be specified. The approach adopted is to place a diaphragm at the motor exit, immediately preceding a fictitious "nozzle". When the pressure differential across the diaphragm reaches some nominal value, say 1 atm, the diaphragm bursts. The nozzle is assumed to fill instantaneously. Continuous checks are made on the amount of flow approaching the exit and the amount of flow the "nozzle" could pass given the existing upstream conditions. A simple one dimensional time-dependent mass conservation calculation is then utilized to provide boundary conditions for the next time step. It should be noted that, for the first 200 msec or so following igniter firing, the downstream boundary condition is not as important as it would be later in the ignition transient.

Sample Results

Cold Flow

After verification of the overall flow model and computational technique by comparison with known solutions or measurements for problems such as supersonic laminar flow over a flat plate\(^{10}\) and developing turbulent flow in a pipe\(^{11}\), attention was turned to the igniter plume as a measure of the model's ability to predict more complex flow fields. Experimental data obtained both by Conover\(^{32}\) and in the series of tests described herein were utilized. Schlieren photographs of the exhaust plume(s) were taken for each igniter at various values of upstream total pressure. In particular, the Space Shuttle SRM igniter is a single port, conical nozzle configuration with an area ratio of 1.428 and a half angle of 27.2 degrees, with a design exit Mach number of 1.79. Figure

III-13
III-5 compares the calculated plume Mach number contours for an igniter total pressure of 102 atm (1500 psi) and a total temperature of 316 °K with the Schlieren photograph of the plume at the same conditions. A qualitative comparison indicates that the results are quite satisfactory, although the shock wave appears to be smeared over several grid points.

Next, calculations were performed for the cold flow (no burning) flow field within the star slot of the head-end section. The computed results can be compared with the oil smear data obtained in the present cold flow tests. Both the case of a single port igniter and a multi-port (canted) igniter are considered. The multi-port igniter exhaust jet is aligned with the slot, at an angle of 45 degrees with respect to the motor centerline. Typical results are shown in Figures III-6:III-9. Figures III-6 and III-7 compare results for an igniter pressure of 6.8 atm (100 psi), while Figures III-8 and III-9 compare results for 34 atm (500 psi). The igniter (not shown) is located near the upper right corner of the slot, exhausting from right to left.

**Hot Flow**

Attention was next turned to the problem of heat transfer and the calculated progression of the burning surface of the propellant grain within the star slot. As time progresses, the propellant surface temperature rises due to heat transfer from the hot gas flowing over it. Propellant ignition is assumed to occur when the temperature of the surface reaches a critical ignition temperature (850 °K). Obviously, the resulting flame spread is dependent on the heat transfer correlation assumed (eg., Eq. III-26). Figure III-10 illustrates a typical predicted burn sequence for the single port igniter presently used on the Space Shuttle SRM. First ignition of the propellant surface occurs in the vicinity of the shock located in the igniter plume, as might be expected. Interestingly, this burning progression pattern can be anticipated from calculated cold flow heat transfer contours, as can be seen in Figure III-11, which illustrates calculated Nusselt number contours for cold flow over a range of igniter pressures from 6.8 atm (100 psi) to 102 atm (1500 psi). Again, the igniter exhausts from right to left.

Figure III-12 compares the calculated head-end pressure-time trace with values obtained from measurements taken from actual motor firings31, as well as with values calculated from a typical P(x,t) model10. The comparison is made for the time interval 0≤t≤120 milliseconds, over which the first propellant grain ignition occurs and flame spreading begins on the slot surface. It can be seen that the P(x,t) model significantly underpredicts the head end pressure during this interval, while the present model matches the measured pressure trace with reasonable accuracy. During this period, the flame spread is fairly slow, as would be expected. A flame first appears on the propellant surface at 25 msec after the igniter firing, and at 120 msec approximately 20 percent of the star slot grain is burning. The rapid pressure rise shown by the P(x,t) model at approximately 115 msec corresponds to
Figure III-5: Igniter Flowfield Comparison (cold-flow)

III-15
Oil Smear

Calculated

Figure III-6: Single Port Igniter Cold-Flow, $P_{\text{Igniter}} = 6.8$ atm

III-16
Calculated

Figure III-7: 45 Degree Igniter Cold-Flow, $P_{\text{igniter}} = 6.8$ atm

III-17
Calculated

Figure III-8: Single Port Igniter Cold-Flow, $P_{\text{igniter}} = 34$ atm

III-18
Oil Smear

Calculated

Figure III-9: 45 Degree Igniter Cold-Flow, $P_{\text{igniter}} = 34$ atm

III-19
Figure III-10: Typical Slot Area Burn Sequence (predicted)
Figure III-11: Calculated Nu Contours (Cold-Flow) Single Port Igniter
**Figure III-12: (Early) Ignition Transient Head-End Pressure Rise**

- **SRM Data**
- **P(x,y) Model**
- **Present Model**

- Headend pressure, psi

- Time, seconds

III-22
the ignition of the CP portion of the grain in that model. First ignition of the CP portion is predicted to be at approximately 80 msec in the present model. It should be noted that the present model tends to under-predict the pressure rise in the head-end for $t > 120$ msec. This suggests that the predicted flame spread for time $t > 120$ msec is too low. This may be due to one or more of several effects, including under-prediction of the convection heat transfer coefficient, the presence of significant radiative heat transfer, or the significance of 3-D conduction effects within the propellant grain.

References


IV. CAVENY PROGRAM MODIFICATIONS

To incorporate the results obtained from the experimental and analytical work modifications were made to the original solid rocket motor ignition code. These modifications proceeded along two independent paths: changing the code so that it could work with the interface program, and changing the code to improve its accuracy when used with star grain segments.

To significantly increase the accuracy of the prediction codes used by the Caveny program, two of its fundamental assumptions had to be discarded and replaced with other computational models. First, the original code assumed that the propellant grain ignited at a given temperature, that the entire segment was not burning until it reached this temperature, and that after reaching that temperature the whole segment would burn evenly. Second, rather than calculating the burning perimeters of the segments based on input motor geometries, the Caveny program accepted an equivalent effective perimeter of the segment at various stations along the grain. This effective perimeter was modeled as a circular perforated grain, and the gas dynamics equations assumed one dimensional flow in a cylinder. These assumptions led to inaccuracies for star grains.

To correct the inadequacies of the first assumption, a new model for the burning area was introduced. A system was introduced in which the fraction of the segment burning was a function of simulation time rather than local temperature. A table of percentage burning surface as a function of time could be specified for any of the segments. This is obtained from the CFD model described in Section III. A variable NBPTAB (number of burning perimeter tables) was added to the NAME namelist. The variables NBPENT, BPTIME, and BPFRAC were added to the INPUT common block, respectively representing up to thirty burning perimeter entries in a given table and up to twenty values of time and burning surface fraction per station. These tables contain discrete values of burning surface fraction as a function of time. A linear interpolation function INTERP was also added to the Caveny code to calculate intermediate values. In addition, a new model for turning on burning of the grain had to be introduced to the code as well. As previously stated, in the previous model, the grain segments were assumed to begin burning when they reached the ignition temperature.

A subroutine to calculate burning areas as functions of distance burned was also added. The original version of the Caveny program held the burning perimeter constant during the course of a run. This assumption was considered a valid approximation since the simulated burning times and burn distances would be small in comparison to the total burn time and overall size of the motor.
To correct the shortcomings of this second assumption, a new model for calculating the burning perimeter was introduced. In this model, a number of different grain geometries can be specified in the input case file. A variable NBGTab (number of burning geometry tables) was added to the NAME namelist. The variables NBGENT and BGVAL were added to the INPUT common block, respectively representing the number of burning geometry variables in a given table and up to ten variables that have different meanings depending on the geometry model selected. A subroutine to integrate the burning rate, BDCALC, and a variable representing the total burn distance at an axial station, BDIST, were also added to the Caveny program.

The Caveny program reads a single input file that specifies all parameters for a particular case. This file consists of at least one namelist, followed by a number of tables, followed by a number of optional namelists. The program reads the first namelist, and from it modifies the default values set in the program. This namelist includes variables describing the mgtor geometry, simulation parameters such as time step and end time, options for printing results, etc. Following this initial namelist are the tables for igniter mass flow and motor geometry. After these tables are a number of optional NAME namelists, each of which can be used to modify any of the simulation parameters listed in NAME. These subsequent namelists are read when the Caveny program has reached the end of a run. The new set of variables is generally restricted to output parameters such as time increment, print interval, and updated end time.

The tables associated with the new burning fraction function are specified by a single integer value for NBPENT(I) (read using a 1I10 format) between 1 and 20 representing the number of burning fraction versus time pairs for that location. These pairs of values for BPTIME(I,J) and BPFRAC(I,J) (read using a 2F10.4 format) immediately follow, one per line. An example set of burning fraction tables is given in Table IV.1.

In each of the following example listings, a vertical bar and all text following it should be interpreted as comments and should not be inserted into the input files.
Table IV.1: Sample Burning Perimeter Fraction Table

\&NAME
\&NBPTAB = 4
\&END
5
0.0000 0.0000
0.2500 0.1000
0.5000 0.2000
0.7500 0.4000
1.0000 0.8000

These tables must be placed in the input file immediately following the initial NAME namelist, but before any of the other tables (such as igniter mass flow, grain geometry, etc.) that can be specified. The number of tables provided must correspond to the number specified in NBPTAB, and the number of entries per table must correspond to the value provided for NBPENT(J). The tables associated with the new burning geometry function are specified in a single integer value for NBGENT(I) (read using a 1I10 format) between 1 and 10 representing the number of variables in the table for that location. These variables are interpreted differently by the various motor geometries that can be specified. In all cases, the first variable BGVAL(J,1) indicates the type of geometry specified at the station. The number of variables that follow and their meanings are determined by this first value. Table IV.2 shows the valid values and their meanings. Tables IV.3 through IV.7 detail the meanings of the remaining variables in the table for each type of available geometry. When possible, variable definitions are linked to corresponding quantities between various grain configurations.

Table IV.2: Caveny Code Input Tables

<table>
<thead>
<tr>
<th>BGVAL(I,1)</th>
<th>Grain type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>Circular perforated</td>
</tr>
<tr>
<td>2.0</td>
<td>Slotted</td>
</tr>
<tr>
<td>3.0</td>
<td>Star</td>
</tr>
<tr>
<td>4.0</td>
<td>Wagonwheel</td>
</tr>
<tr>
<td>5.0</td>
<td>Dogbone</td>
</tr>
</tbody>
</table>
### Table IV.3: Circular Perforated Grain Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Variable Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGVAL(I,1)</td>
<td>none</td>
<td>1.0 = Circular perforated grain</td>
</tr>
<tr>
<td>BGVAL(I,2)</td>
<td>inches</td>
<td>Interior diameter (D1)</td>
</tr>
<tr>
<td>BGVAL(I,3)</td>
<td>inches</td>
<td>Exterior diameter (D2)</td>
</tr>
</tbody>
</table>

### Table IV.4: Slotted Grain Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Variable Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGVAL(I,1)</td>
<td>none</td>
<td>2.0 = Slotted grain</td>
</tr>
<tr>
<td>BGVAL(I,2)</td>
<td>inches</td>
<td>Interior diameter (D1)</td>
</tr>
<tr>
<td>BGVAL(I,3)</td>
<td>inches</td>
<td>Exterior diameter (D2)</td>
</tr>
<tr>
<td>BGVAL(I,4)</td>
<td>none</td>
<td>Number of star points (N)</td>
</tr>
<tr>
<td>BGVAL(I,5)</td>
<td>inches</td>
<td>Slot width</td>
</tr>
</tbody>
</table>

### Table IV.5: Star Grain Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Variable Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGVAL(I,1)</td>
<td>none</td>
<td>3.0 = Star grain</td>
</tr>
<tr>
<td>BGVAL(I,2)</td>
<td>inches</td>
<td>Interior diameter (D1)</td>
</tr>
<tr>
<td>BGVAL(I,3)</td>
<td>inches</td>
<td>Exterior diameter (D2)</td>
</tr>
<tr>
<td>BGVAL(I,4)</td>
<td>none</td>
<td>Number of star points (N)</td>
</tr>
<tr>
<td>BGVAL(I,5)</td>
<td>degrees</td>
<td>Star half-angle</td>
</tr>
</tbody>
</table>
Table IV.6: Wagonwheel Grain Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Variable Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGVAL(I,1)</td>
<td>none</td>
<td>4.0 = Wagonwheel grain</td>
</tr>
<tr>
<td>BGVAL(I,2)</td>
<td>inches</td>
<td>Interior diameter (D1)</td>
</tr>
<tr>
<td>BGVAL(I,3)</td>
<td>inches</td>
<td>Exterior diameter (D2)</td>
</tr>
<tr>
<td>BGVAL(I,4)</td>
<td>none</td>
<td>Number of star points (N)</td>
</tr>
<tr>
<td>BGVAL(I,5)</td>
<td>degrees</td>
<td>Star half-angle</td>
</tr>
</tbody>
</table>

Table IV.7: Dogbone Grain Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Units</th>
<th>Variable Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGVAL(I,1)</td>
<td>none</td>
<td>5.0 = Dogbone grain</td>
</tr>
<tr>
<td>BGVAL(I,2)</td>
<td>inches</td>
<td>Interior diameter (D1)</td>
</tr>
<tr>
<td>BGVAL(I,3)</td>
<td>inches</td>
<td>Exterior diameter (D2)</td>
</tr>
<tr>
<td>BGVAL(I,4)</td>
<td>none</td>
<td>Number of star points (N)</td>
</tr>
<tr>
<td>BGVAL(I,5)</td>
<td>degrees</td>
<td>Star half-angle</td>
</tr>
</tbody>
</table>

The routine to calculate the burning perimeters as functions of time were based on the work presented in Reference 2. A sample set of geometry description tables is given in Table IV.8.

Table IV.8: Sample Burning Geometry Table

<table>
<thead>
<tr>
<th>&amp;NAME</th>
<th>NBGTAB = 2</th>
<th>&amp;END</th>
<th>Table #1: CP Grain (3 vars)</th>
<th>Table #2: Star Grain (5 vars)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>Inner diameter (inches)</td>
<td>Inner diameter (inches)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Outer diameter (inches)</td>
<td>Outer diameter (inches)</td>
</tr>
<tr>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
<td>Number of star points (-)</td>
</tr>
<tr>
<td>54.345</td>
<td></td>
<td></td>
<td></td>
<td>Star half-angle (degree)</td>
</tr>
<tr>
<td>23.456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>54.345</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>123.456</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The original Caveny program produced two types of output files: a 132-column file reflecting key simulation variables as functions of time and (where appropriate) space, and a plot file that was designed specifically to work with the Princeton University plotter.

Rather than changing the format of the default output file of the Caveny program, new routines to generate a separate plot file were added instead. Of all the simulation variables calculated and printed in the original program, only the variables listed in Table IV.9 were considered to be of interest for plotting.

### Table IV.9: Variables Saved in caveny.plot File

<table>
<thead>
<tr>
<th>Time and Space Dependent Variables</th>
<th>Time Dependent Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td><strong>Meaning</strong></td>
</tr>
<tr>
<td>PSIA</td>
<td>Pressure</td>
</tr>
<tr>
<td>T</td>
<td>Temperature</td>
</tr>
<tr>
<td>M</td>
<td>Mach Number</td>
</tr>
<tr>
<td>BR</td>
<td>Burn Rate</td>
</tr>
<tr>
<td>TAUB</td>
<td>Burn Distance</td>
</tr>
<tr>
<td>XMSA</td>
<td>Mass Flow Rate</td>
</tr>
<tr>
<td>TPS</td>
<td>Surface Temperature</td>
</tr>
<tr>
<td>REP</td>
<td>Reynolds Number</td>
</tr>
</tbody>
</table>

To make the Caveny program more maintainable and modifiable, it was restructured to take advantage of the program development features offered by the Unix platform. In its original form, the program was approximately 3500 lines of FORTRAN code in a single file. All common blocks and namelists had to be repeated in each subroutine that used them. These common blocks and namelists often had incompatible lengths and variable names substituted between different subroutines.

To make the code more readable and more easily edited, the source was broken up into separate files that contained only one subroutine each. The filenames had a basename that was the same as the subroutine they contained and a .f suffix. In addition, all common blocks and namelists were also placed in individual
files. These files were given names corresponding to the name of
the common block or namelist and a .com or .nam suffix
respectively. These files were then referenced in the code
through INCLUDE statements. This organization insured that
identical copies of the common blocks and namelists were being
used by all subroutines.

This new organization offered another advantage in program
development. With each subroutine placed in only one file, only
the files that had been changed since the previous build had to
be recompiled, rather than compiling all modules after any
change. This change was further aided through use of the Unix
make utility. An appropriate makefile that was suitable for the
whole project was created. This program organization made
modifying the program much easier. A complete listing of this
makefile is provided in Appendix B.

References

1. Sforzini, R. H., "Design and Performance Analysis of Solid-
Propellant Rocket Motors Using a Simplified Computer

2. Caveny, L. H., and Kuo, K. K., "Ignition Transients of Large
Segmented Rocket Boosters," April 1976, NASA Contractor
Center.
V. CAVENY PROGRAM INTERFACE

The interface application is designed to make running the Caveny program easier for potential users, both novice and expert. The three natural divisions of using the program were: building the input data files, executing the program, and collecting and interpreting the results it generated. The principal parts of the interface program are designed to correspond to these same three areas.

An input file editor which is included is more than a simple text processor. Ideally, it should prevent the user from creating an input file that the program could not run. In practice, this would be nearly impossible, but it could ensure that all variables fall within certain constraints. From within the interface application, the Caveny program runs in the background. The interface program automatically reads in the data generated from the most recent execution. It then be able to produces a number of plots simultaneously. These plots are dynamic, in that the user configures the variables displayed, the range of values over which they are shown, and all aspects of the plots' appearance.

The parts of the interface program that generate input files for the Caveny program are the most crucial in making the program easy to use. Configuring input files proved to be the most daunting and error-prone aspect of running the Caveny program. As such, attention was paid to providing the user with a number of features to make editing these files more convenient.

The input file editor is designed to provide two critical functions: on-line documentation of all simulation variables and error-checking of the values provided for those variables. The documentation for the variables includes (but is not necessarily limited to) definitions and descriptive text, units in which the variables should be input, and default values for each. Limited error-checking is provided by comparing the input values against known valid minimum and maximum values. While this technique does not insure that a case is consistent and will run properly, it does provide a safeguard against obvious errors.

A simple text file format has been chosen to support this part of the program. A file containing all data about the variables that could be modified in the Caveny program input case files would have significant advantages over compiling this information into the interface program. Descriptive text and default values could be changed by a user without recompiling the code, and information tailored to different audiences could be maintained. More importantly, if new variables are added to the Caveny program in the future, these variables can be easily
incorporated into the input file generation part of the interface program.

The output capability of the interface program is provided to reduce the amount of data generated by the Caveny program and to present the data in a more easily recognizable and understandable fashion than simple printed data.

The ability to generate plots of the data presents a significant problem. First, the data is not organized in a fashion that would be well-suited to work with an existing plotting package. Although the output file can be changed to contain only numbers, there are no existing programs that could accommodate the volume of data produced by the Caveny program. One of the main considerations has to be reducing the amount of data provided to the user. The most efficient way to do this is to create a number of plots that allows the user to view a lot of data at one time. These plots are dynamic, and the variables they display and the ranges over which the data are displayed can be changed. More then one plot can be maintained at a time by the program.

An integrated environment from which the user does not need to exit offers the most significant advantages in ease of use. The goal is to provide a shell program that encourages use by those not familiar with the Caveny program or the Unix operating system. To this end, a number of features that were not critical to using the program were incorporated into the interface. A simple text editor based on the xedit sample application has been designed and included.
VI. XPLOT PROGRAM DESCRIPTION

The specifications set forth in Chapter V led to the development of a stand-alone application named xplot, a graphical user interface for the Caveny program.

A number of issues had to be resolved at the outset, since they would have significant impact on program development. The computers available in the College of Engineering at the time the project began were the IBM mainframe, Sun workstations, and a number of IBM PC compatibles. The mainframe offered no graphics capability. The PCs did not have the memory, storage, or processing capability necessary to run even the original Caveny program. The Sun workstations offered the power and graphic output capability needed. Since the operating system used on these machines was Unix, that would be the operating system for which xplot would be written.

The X Window System from the X Consortium at MIT was chosen as the graphics package for the interface program. Competitive packages available for Unix workstations that were considered include: Pixrect (Sun Microsystems' device-dependent screen drawing mechanism), PHIGS (a Programmable, Hierarchical Interactive Graphics System - designed for creating, manipulating, and rendering objects in both two- and three-dimensional space), and GL (a multi-vendor general-purpose drawing environment from Silicon Graphics). X is more appropriate for the specific application for a number of important reasons. X is widely available on most Unix platforms that support graphics output. The calls that are made to X routines are platform-independent; as such, programs written on a machine that supports X should only require recompilation to work on a different machine. Revisions of X have been available at no charge for both internal use and worldwide distribution. Most important, however, X is the only system available that provides any support for developing full-scale, interactive user applications; most of the other systems only offer libraries of drawing calls.

The choice of Unix for the operating system and X for the graphics system made C the most natural programming language to use in writing xplot. At the project's inception, C compilers were provided for use with the Sun workstations. More important, however, was the need for a language offering both data structures and dynamic memory allocation to use in conjunction with subroutine calls to X. X routines require data to be passed and returned using data structures, and they often require the calling application to both allocate and free memory. Since FORTRAN does not support either capability, C was the obvious choice.
From the outset of the program, xplot and the Caveny program were designed to work together but remain two separate processes. The Unix environment supports true preemptive multitasking, so it was possible to create an interface shell that executes the underlying solid rocket motor prediction code. From the user's standpoint, there is no need to exit from xplot. All the details of building the input file, running the Caveny program, and loading the resulting data were made transparent.

There are several reasons that support making the interface program a separate process. There is no reason why the user should have to wait for the Caveny program to finish executing before using other parts of xplot. Had the programs been integrated, there would have been no way for the user to continue building input datasets are creating plots from previous runs. The user would have had to wait for the simulation to finish before continuing the analysis. It would also have added to the complexity of the two programs if C and FORTRAN routines had been used together. The way in which arguments are passed to subroutines differs between the two languages, and special libraries that support both languages must also be used. Finally, the Caveny program could be left unmodified if two processes were used; any other implementation would have required making changes to the existing code.

The Unix operating system provides true preemptive multitasking for multiple processes on the same machine. Any number of executables can be running concurrently, and time-sharing between them is handled by the operating system. Associated with each instance of an executable is the environment under which it is run. Collectively, the executable code, its environmental variables, and the memory it has reserved for its internal data are referred to as a process.

Under the Unix operating system, there is only one way to start a new process. The fork system command creates a duplicate of the process that is currently executing, including all of its environmental variables and data. The fork command is unique in that it has a single entry point and two return points, one in the calling process and one in the process it creates. These two processes are identical, but can distinguish themselves from one another through the value returned from fork. The parent process is returned the process ID of the child process if the call succeeds or -1 if the call fails, and the child process is returned a value of zero.

The fork command only creates a duplicate of the parent process; it does not run the desired executable. For this to occur, one of the exec family of system routines must be used to change the executable associated with the child process. Note that all environment variables remain the same across a call to one of the exec functions.
The fork and exec functions are almost always used with one another. Typically, all a child process will do upon returning from a fork is transform itself into the appropriate executable using one of the exec calls. Assuming that the child process is performing some task for the parent process, the parent will often block until the child has finished executing. Under Unix, the wait command provides this facility.

The fork-exec-wait combination is used in conjunction with xplot and the Caveny code. When the user chooses to execute the Caveny code after building an input case, the interface program forks to create a clone process. The child process then immediately transforms itself to the executable name stipulated in a corresponding field in the execution dialog. This executable is named caveny by default. xplot does not wait until the child process has finished executing to restore control to the user. Note, however, that the current dataset, input files, and plots will not necessarily be updated unless specifically requested by the user. Although similar results could have been achieved using the Unix system command, system is, in general, less reliable and requires much more overhead than a fork-exec-wait combination does.

No direct communication takes place between xplot and the Caveny program. Instead, xplot is used only as a pre- and post-processing application. Input files can be generated using xplot's built-in editor. When the user selects to run the Caveny program, a file name must be specified in the execution dialog. xplot renames this file to the name of the input file expected by the Caveny code, caveny.in. The Caveny code is then run, and it generates two output files named caveny.out and caveny.plot. Either file can be viewed using the file editor dialog, and the plot file can then be loaded as the current dataset and used to create new plots. The two programs communicate through these three files only.

The organization of data generated by the Caveny program was reflected in xplot, both in the features it offered to the user and in the way it stored and manipulated the data internally. In general, the Caveny program produces two types of data: variables that are functions of both distance along the motor and time elapsed in the simulation, and variables that are functions of time alone. The model chosen for the interface had to be flexible enough to allow for both of these types of data. Instead of modifying the output file of the Caveny program, write statements were simply added at appropriate points in the code to produce a plot file with the desired format (see 2.5).

Data were stored in a three-dimensional floating point array named plotData. The X-dimension was used as discrete locations along the grain, so this dimension was limited to 30 values. The Y-dimension was used for the different variables output to the
plot file. Since the maximum number of variables that could be output was limited to 8 that were functions of both space and time and 8 that were functions of time alone, this dimension was limited to 16 values. The Z-dimension was used as an index into the time at which the variables had been written. Since this number was dependent on a number of factors (i.e., the total run duration, step size, and print interval of the simulation), the Z coordinate had to have the greatest dimension. Ultimately, this value was limited to 1024 points, a value that was generally appropriate for runs of less than one second. Figures VI.1 and VI.2 graphically show this data organization.

Figure VI.1: Two-Dimensional Data Organization
Figure VI.2: One-Dimensional Data Organization

In Figure VI.1, the two-dimensional sets of data (used for quantities such as pressure, temperature, and Mach number) have values (Y axis) for all pairs of space (X) and time (Z). The increasing values of time are shown as planes of (X, Y) pairs, which is consistent with the way the data is written to the file. In Figure VI-2, one-dimensional data (such as exit pressure, mass burned, and thrust) is shown. Here, data is a function of time (Z) alone.

This storage technique made it possible to change the way in which two-dimensional data was presented in plots. The curves could be drawn at various points in time, with the variable shown as a function of distance along the grain. However, the data "cube" could also be rotated, so the data could be drawn at various "slices" of distance, with each variable shown as a function of simulation time. Example plots that reflect this capability are given in section VII of this report.

The X Window System is a set of library subroutines that provide graphics and interface capabilities on Unix platforms. X is based on the client-server model; client applications make requests to an X server to perform certain actions, and the X server notifies the client about particular events that are of interest. X works equally well on a single computer or across a network. Each host computer can have an X server running on it, so it is possible for applications to make requests on both local
and remote hosts. X provides a platform-independent set of subroutines that client applications can use. All machine-dependent drawing and event scheduling are handled through the X server, an implementation tailored specifically for the hardware on which it is running.

X actually provides three different levels of possible interaction with client applications. At the lowest level is the X library (Xlib), a library that provides extensive routines for drawing graphics primitives (lines, rectangles, polygons, arcs, text, etc.) and basic windowing environment routines. Xlib alone is generally used only for basic drawing and as the foundation for more elaborate user interface mechanisms. The X Toolkit (Xt) is an example of one such mechanism. It provides the framework for manipulating complex graphic objects that have specific appearance and behavior called widgets. Examples of widgets include push buttons, toggle buttons, menus, scroll bars, etc. Xt is an event-driven system, in that the client application creates objects, modifies their appearance and customizes their behavior appropriate for their use. The Xt application main loop performs all polling and dispatching of events. The third level of interaction is a widget set, which greatly affects the functionality of the user interface. Short of writing custom widgets, the things that can be done in a program are limited by the functionality of the widgets employed. This three-tiered organization of interface code is shown in Figure VI.3.

![Figure VI.3: X Toolkit Application Design Model](image-url)
The client application can interact with any or all levels of interface provided: it can change the value, appearance, or behavior of one or more widgets through the widget set, control the overall application through Xt, issue drawing requests in a window through Xlib, or perform any operating-system specific calls concurrently.

The xplot interface program uses all three levels of X. The widget set chosen was the Athena widgets provided as examples with the distribution from the X Consortium. Although Athena widgets do not have as nice an appearance or quite the functionality of commercial widget sets, they are adequate for the somewhat limited needs of the program. Further discussion of the widgets used in xplot and Xt in general is given below.

As previously stated, the X environment is event-driven. Rather than performing certain actions in a specified order, as is customary in traditional programs, X applications must respond to events, which are usually user inputs to the program, either directly or indirectly. For example, at the Xlib level, a plot must be redrawn not only when created but also whenever it is exposed (from underneath other windows) or resized. All of the techniques discussed for drawing plots really refers to the actions taken whenever one of them must be updated.

A simple layout was chosen for the plots generated by xplot. Only simple, rectangular 2D scatter or line plots are provided. Many simplifying assumptions about the kind of data being displayed can be made since the application is designed specifically to work with the output of the Caveny program. The independent variables are either axial distance along the grain or simulation time. The number of variable data traces displayed are limited to sixteen for time-dependent and eight for distance-dependent plots. The general layout characteristic of these plots is shown in Figure VI.4.
The size of a plot depends on both the size of the window enclosing it and the borders chosen for it. Only the data curves are drawn in the data area; all other plot components are drawn in the borders around it. The width and height of the plot then can be expressed as:

\[
W_{\text{plot}} = W_{\text{window}} - B_{\text{left}} - B_{\text{right}} \\
H_{\text{plot}} = H_{\text{window}} - B_{\text{top}} - B_{\text{bottom}}
\]

The elements of the plot are drawn in the following order: plot title, plot legend, x-axis title, axis labels, axes, gridlines, and then the plot data. This order was chosen to insure that various elements of the plot neither obscure nor erase the others.

The plot title is centered over the main plot area. The font used for the title is Helvetica 18 bold. The width of the text string is computed using the XStringWidth function. Once the
width of the text string is known, the x-component of the title can be calculated using the following equation:

$$X_{title} = Bleft + \frac{W_{plot} - W_{title}}{2}$$

The plot legend is placed across the bottom of the plot area. Each element of the legend is drawn as a small rectangle having the same color as the data it represents, followed by a string label corresponding to the same data. Up to sixteen sets of data can be shown in the plot legend area, with each line containing as many as four identifying markers.

The plot labels are drawn using successive calls to XDrawString. The sprintf function from the C standard library was used to print the text to a string in the format chosen. These axis labels are drawn at regular intervals along the appropriate axis.

An axis title is drawn for the x-axis only, since X does not support drawing text rotated to arbitrary angles. The x-axis title consists of the independent variable, either axial location or time, drawn in Helvetica 10 Bold, centered below the main plot area. It also indicates which slice of either time or space is currently being displayed.

The plot axes are drawn using a 1 point black line along the bottom and left edges of the plot area. Since all of the data that could be plotted, including dependent and independent data, contained only positive values, no provision was made for drawing the lines where Y = 0 or X = 0 inside the plot area.

The grids on the plot are drawn at regular intervals corresponding to the number of axis labels. These lines are drawn as 0 width lines inside the bounding rectangle that formed the main plot area.

The plot data is the most complicated aspect of the plot to draw. Each trace of data has its own attributes, and there can be up to 16 traces of data on a single plot. The location of any single data point can be calculated from interpolation:

$$X_{data[i]} = Bleft + \frac{(X[i] - X_{min})}{X_{range}} * W_{plot}$$

$$Y_{data[i]} = Bbottom - \frac{(Y[i] - Y_{min})}{Y_{range}} * H_{plot}$$

Using these equations, data can lie outside the plot area rectangle if the scales have been manually set. Such values of data are either less than the minimum or greater than the maximum values of the scales. To insure that the data drawn in the main data area of the plot frame does not exceed the limits of the
area, the calls for drawing data were clipped to the plot data area. The clip mask of the current graphics context is set to the rectangle \( R(B_{\text{left}}, B_{\text{top}}, W_{\text{plot}}, H_{\text{plot}}) \). Explicitly setting the clip mask overrides the mask that is set by an expose event. Rather then have the program maintain a list of all areas exposed, combine them with the plot area into a single clip mask, and then redraw the plot, xplot simply redraws the entire plot on any expose event.

Xlib provides line attributes as part of the graphics context, or current drawing environment. These attributes include foreground and background drawing colors, line width, style (solid, on-off dashed, double-dashed), join mode (rounded, mitered, or beveled), and end cap style (butt or rounded). Implementing the various types of lines that can be specified in xplot only requires calling XSetLineAttributes with the appropriate values before any drawing commands.

Since X is implemented using client-server methodology, the speed with which it can perform drawing operations is as limited by the rate it can receive commands as it is by the speed of the processor. To reduce the amount of communication (often network) traffic, Xlib provides routines for drawing arrays of objects as well as the routines used to draw single objects. Points, lines, line segments, rectangles, and arcs, can all be drawn either individually or in groups. To maximize refresh rate, xplot assembles a table of all the points for a given trace of data, then calls XDrawLines.

Descriptions of the Athena widgets used in creating the xplot application are given in Table VI-1.
<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box</td>
<td>Geometry management of arbitrary widgets in a box of specified dimension.</td>
<td>Most dialogs, to organize buttons and text fields into logical groupings.</td>
</tr>
<tr>
<td>Command</td>
<td>A rectangular button used to invoke an action.</td>
<td>In all dialogs to perform any necessary actions.</td>
</tr>
<tr>
<td>Dialog</td>
<td>A window, used to hold other widgets, that can be placed and iconified by the window manager.</td>
<td>All pop-up selection panels are instanced from dialogs.</td>
</tr>
<tr>
<td>Form</td>
<td>Organizes the layout and size of its child widgets.</td>
<td>Organizing sub-objects in all dialogs.</td>
</tr>
<tr>
<td>Label</td>
<td>A rectangular area containing descriptive text.</td>
<td>All dialogs, to indicate the function of an area or an associated text field or button.</td>
</tr>
<tr>
<td>List</td>
<td>An array of text entries, drawn in a column, from which one or more can be selected.</td>
<td>Scrolling lists, such as the command and history summary and the input file variables.</td>
</tr>
<tr>
<td>MenuButton</td>
<td>A rectangular button from which a pull-down menu can be displayed.</td>
<td>Only used in the Top Level menu to contain all menus.</td>
</tr>
<tr>
<td>Text</td>
<td>An area containing text that can be either display-only or editable.</td>
<td>All dialogs, whenever the user is prompted for data entry; in dialogs to display large amounts of read-only text.</td>
</tr>
<tr>
<td>Toggle</td>
<td>A button that can be in one of two states, either on or off.</td>
<td>Setting options about the appearance of plots.</td>
</tr>
<tr>
<td>Viewport</td>
<td>An area with scroll bars that allows the user to view an object larger than the area provided.</td>
<td>Scrolling lists, such as the command and history summary and the input file variables.</td>
</tr>
</tbody>
</table>

The dialogs used in xplot are all constructed using similar techniques. A shell widget interacts with the window manager, allows a window to be moved, resized, restacked, or iconified, and can contain only a single child widget. A dialog shell widget was used for all of the dialogs other than the Top Level. A form widget was created as the shell's only child, and it handles geometry management of all the child widgets. Normally, these children were arranged by function into related groups.

The way the Format dialog is constructed is representative of the dialogs used in xplot. A diagram of this dialog and its components is given as Figure VI-5. A single Form widget is the only child of a dialog shell. A number of box widgets are its children, arranged in an order that logically divides functions of the dialogs. These boxes are attached to one another using
the geometry management facilities provided by the form widget, so the dialog is only as large as it needs to be to contain all of its children. The text fields and command buttons inside the boxes are organized in columns to minimize the space they occupy.

Only a fraction of the widgets created for an Xt application need to be modified during program execution. The values of some of these widgets will be modified by the user, and some will be changed by the application to reflect the current state of the data it maintains. In general, widgets that need to be modified include text fields (used for data entry and presentation), command buttons (used to reflect currently available or acceptable choices), and toggle buttons (used to reflect a current state of a selection).

Figure VI.5: Example dialog design
In an Xt application, the ID associated with a particular widget is returned by the function used to create it. For most of the widgets used in xplot, that ID is retained only as long as needed. For instance, all widget creation routines require the widget's parent to be passed as one of the arguments. However, for decorative and geometry-management widgets (like form, box, or label), once their children have been created, there is no reason to maintain their ID, since it is not necessary to modify them. For the widgets that did need to have their values read and set, the variable-arguments form of the Xt get and set functions, XtVaGetValues and XtVaSetValues, are used.

Xt provides higher-level event processing than is possible through Xlib alone. Instead of notifying the client application that the user has pressed and released one of the mouse buttons at a certain location on the screen, it might instead notify that a button has been selected, a menu has popped up, or a scroll bar had been paged. Xt provides these higher-level events through a mechanism it defines as callback routines. Callback routines tell a widget how to respond to certain well-defined events. Most events will not be of interest to the application, and the widget's default behavior will be the only action that is either necessary or appropriate.

In addition to the graphical interface provided by xplot, a command-line language was written. The grammar supported by xplot is very small. Since there are less than fifty possible legal tokens in the vocabulary, there is no need for a separate lexical analyzer or parser (such as those provided by the Unix utility programs lex and yacc). Instead, each line is simply broken down into its constituent components by a single function call. This function uses string comparison routines to arrive at allowable input. To keep the code as straightforward as possible, the actions and options that can be set from the command-line are mapped to equivalent functions already present from within the graphical interface.

The source code for xplot is organized into files, grouping subroutines that deal with related aspects of the application. This organization is used for two main reasons. First, to encapsulate the data and routines into a similar area, so that as little global data and global subroutine calls are necessary. Second, to make compilation both more efficient and quicker during development. The source code files correspond to the various main aspects of the program, i.e. the command-line interface, the internal data, the plot formats, drawing the plots, etc.

A complete listing of the source files used to build xplot and their associated functions is given in Tables VI-2 and VI-3. All source files have a .c suffix, and all header files have a .h suffix.
### Table VI-2: xplot Source Files

<table>
<thead>
<tr>
<th>Filename</th>
<th>Lines</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpcommands.c</td>
<td>248</td>
<td>Passes input from the command-line, translates commands into their graphic equivalents.</td>
</tr>
<tr>
<td>xpdata.c</td>
<td>250</td>
<td>Manipulates all of the internal data.</td>
</tr>
<tr>
<td>xpformats.c</td>
<td>358</td>
<td>Loads and saves formats from files.</td>
</tr>
<tr>
<td>xpgeneral.c</td>
<td>137</td>
<td>Creates graphic contexts, loads colors and fonts.</td>
</tr>
<tr>
<td>xpgeometry.c</td>
<td>169</td>
<td>Responds to all configure-notify and expose events generated by the window manager.</td>
</tr>
<tr>
<td>xpinput.c</td>
<td>716</td>
<td>Loads all input datasets.</td>
</tr>
<tr>
<td>xplot.c</td>
<td>69</td>
<td>Initializes all of the Xt interface, calls all the subroutines to create all of the widgets, then calls XtAppMainLoop.</td>
</tr>
<tr>
<td>xpmenus.c</td>
<td>582</td>
<td>Creates menus in the Top-Level dialog and responds to commands issued from these menus.</td>
</tr>
<tr>
<td>xpplots.c</td>
<td>731</td>
<td>Draws all of the aspects of the plots.</td>
</tr>
<tr>
<td>xpwidgets.c</td>
<td>907</td>
<td>Creates all widgets in the application.</td>
</tr>
</tbody>
</table>

### Table VI-3: xplot Header Files

<table>
<thead>
<tr>
<th>Filename</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>xpcommands.h</td>
<td>13</td>
</tr>
<tr>
<td>xpdata.h</td>
<td>25</td>
</tr>
<tr>
<td>xpformats.h</td>
<td>13</td>
</tr>
<tr>
<td>xpgeneral.h</td>
<td>23</td>
</tr>
<tr>
<td>xpgeometry.h</td>
<td>8</td>
</tr>
<tr>
<td>xpinput.h</td>
<td>9</td>
</tr>
<tr>
<td>xpmenus.h</td>
<td>14</td>
</tr>
<tr>
<td>xpplots.h</td>
<td>23</td>
</tr>
<tr>
<td>xpwidgets.h</td>
<td>47</td>
</tr>
</tbody>
</table>

VI-14
VII. XPLLOT USER INSTRUCTIONS

The xplot program is designed to be usable both by those familiar with graphical interfaces as well as those more comfortable with command-line interfaces. Although it is graphical in design, xplot does provide rudimentary support for a command-line language. A discussion of the graphical interface and the commands are presented first, then each of the text commands are discussed in terms of their graphic equivalents.

The top-level dialog of xplot, the window that is displayed when the program is first run, contains controls to access all capabilities of the program. This dialog is shown in Figure VII.1.

![Figure VII.1: xplot Top Level Dialog](image)

A menu bar, located across the top of the dialog, provides menus for performing certain functions. To the immediate right of the dialog is an area for entering the pathnames associated with certain files recognizable by xplot, such as datasets, plot format files, etc. Below the menu bar are three areas used in conjunction with the command-line interface. To the left is a scrolling list containing all command verbs that xplot recognizes. To the right is a scrolling list containing the last fifteen commands entered by the user. Below these two scrolling lists is the text field in which all commands must be entered.
Selecting either the verbs from the command list or previous commands from the history list will place them in the command text field, but it will not execute the command. To execute a command, the user must press either return or enter in the text field.

The top-level menus "File", "Input", "Execute", "Dataset", "Format", "Plot", and "Windows" represent the logical divisions of the program. The "File" menu allows the user to select files for viewing, save the current file, or quit the application. The "Input", "Dataset", and "Format" menus are identical in layout, but each deals with a different kind of file that can be read by xplot.

The Input File dialog allows the user to create and modify input files for the Caveny program without actually editing the text files it uses for input. This dialog is shown in Figure VII.2.

![Input File Editor Dialog](image)

Figure VII.2: xplot Input File Editor Dialog
On the right side of the dialog a scrolling list denoted Variable List shows all of the case description variables and their current values. If the user selects one of these list elements, the four fields of the dialog described below will be updated to reflect the properties associated with this variable.

Once the user has selected a variable to edit, either through the scrolling list or with one of the commands located across the bottom of the dialog, the top four text fields will be filled in with the current data about that variable. The FORTRAN name of the input parameter will be displayed in the Variable field. A brief (one-line) description of the parameter's significance to the Caveny program will be visible in the Definition field. The Explanation section will contain up to six lines of pertinent information about the variable. This information will usually consist of the internal representation of the variable (i.e., INTEGER, REAL, etc.), the FORTRAN format in which the variable will be read (i.e., F10.4), the units that it should be expressed in terms of, limits on the values it can take on, etc. All three of these sections are read-only.

The contents of the Entry section can vary greatly depending on the variable selected. For simple scalar quantities, which are summarized in the Variable List, a single value will be displayed.

The Filename field is a read-write text field that contains the pathname to be used in all read/write operations. When the dialog is first raised, this field will be empty. At this point, the user can either start a new input case file or enter a valid pathname in this field. The value provided in this field will be used for all read and write operations. As such it is possible for a user to load in a new input dataset without saving the changes made, save the new values in place of the old file, or save the current values to a new filename. The messages text field will contain pertinent information about the operations that the dialog performs and diagnostics about errors that it may encounter in attempting to carry out the user commands.

The commands available in the Input File dialog are summarized in Table VII.1. For more information about the format of the Caveny program input file and the variables defined in namelist NAME, see Section IV.

VII-3
Table VII.1: Input File Editor Dialog Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Returns to the first variable available and selects the first variable name in the scrolling list.</td>
</tr>
<tr>
<td>Prev</td>
<td>Advances to the previous/next variable available and highlights the previous/next variable displayed in the scrolling list. When either of these commands are selected, the value in the Entry field will be substituted for the previous value of the current variable, and this value will be reflected in the Variable List.</td>
</tr>
<tr>
<td>Next</td>
<td></td>
</tr>
<tr>
<td>Write</td>
<td>Saves the current values to the filename specified in the Filename text field.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Dismisses this dialog, but maintains all of the current settings.</td>
</tr>
</tbody>
</table>

The Execution dialog contains text fields that allow a user to configure the specifics of running the Caveny program from within xplot. This dialog is shown in Figure VII.3.

Figure VII.3: xplot Execution Setup Dialog
The Execution dialog consists of four text fields. The Path field displays the pathname, either global or local, that is the directory that will be searched for the Caveny program. The Command field displays the name of the Caveny program executable. By default, the path is simply the local path "./" and the executable is named "caveny". The Input File and Output File fields name, respectively, the files expected by the Caveny program as input and the filename that will be used for its output. By changing these filenames, various input cases and output datasets can be maintained by the user. The user can also run the Caveny program from this dialog, once he has selected the proper configuration, using the Execute button. The commands available with the Execution dialog are summarized in Table 14.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cancel</td>
<td>Dismisses this dialog, saving all of its current settings.</td>
</tr>
<tr>
<td>Accept</td>
<td>Dismisses this dialog, saving all of its current settings to internal values for these parameters.</td>
</tr>
<tr>
<td>Execute</td>
<td>Immediately runs the Caveny program using the parameters currently specified in the dialog; saves all current values.</td>
</tr>
</tbody>
</table>

The Plot Manager dialog contains controls that allow the user to configure up to eight plots simultaneously. This dialog is shown in Figure VII.4.
The Commands column allows the user to switch the independent variable, move either along the grain or in time, or animate any of the eight available plots. The commands available in the Plot Manager dialog are summarized in Table VII.3.

Table VII.3: Plot Manager Dialog Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Space Plot/Time Plot</td>
<td>Changes the orientation of the currently active plot. Toggles between the plot being displayed as variables as functions of time or axial distance.</td>
</tr>
<tr>
<td>Increment Decrement</td>
<td>Changes the view to the next/previous time or spatial plane location.</td>
</tr>
<tr>
<td>Animate &gt;&gt; Animate &lt;&lt;</td>
<td>Quickly increments/decrements the view to the beginning/end of the time or spatial stations.</td>
</tr>
<tr>
<td>Edit Format</td>
<td>Brings up the Format dialog, with the values shown corresponding to the currently active plot. If the format dialog is already visible, then change the dialog so that it displays the values associated with the currently active plot.</td>
</tr>
<tr>
<td>Cancel</td>
<td>Dismisses this dialog, saving all of its current settings.</td>
</tr>
</tbody>
</table>
The Format Dialog allows the user to edit the format used to display any of the eight plots available. This dialog is shown in Figure VII.5.
The format dialog will allow the user to edit the formats of plots other than the active plot, and these plots need not be either created or visible to edit their format. These formats can be both loaded from and saved to external files for future use. This dialog works in conjunction with the Data Attributes dialog to modify the internal state of the formats of the plots.

Formats can be saved or loaded just like datasets. When the program is begun, a default set of format values will be installed for each available plot. The user can edit the format to the style preferred, then save and recall these settings using the Filename field. The Title field specifies the label that will appear centered across the top of the plot.

There are two columns of toggle buttons labeled Variables. The left column contains all of the variables in the current dataset that are functions of both position and time. The variables listed in the right column are functions of time alone. Note that nothing will be displayed on the plot if the user selects only time-dependent data from the Format dialog and attempts to create a space plot. The organization of data in a plot file is discussed in Section VI.

By selecting one of the toggle buttons in the Options column, the user can determine which features to use on the current plot. The first option, "Scale Data", forces the Y axis to take on minimum and maximum values corresponding to the minimum and maximum values of the data displayed on the plot. The second option, "Smooth Data", sets the minimum and maximum values and the number of gridlines to even numbers that are more appealing. None of the other six options are currently defined.

Values for both the x- and y-axes can be set using the text fields at the bottom of the dialog. If the "Scale Data" option is not active, then the minimum and maximum values of either axis can be set manually. Note, however, that both minimum and maximum must be set for each if any are to be set. The number of gridlines and labels for either axis can be set in the same way. The minimum number of gridlines is 1, and the maximum is 20. The borders affect the way the plot is sized and drawn. The text fields in the x-axis area control the size of the left and right borders, and the text fields in the y-axis area control the bottom and top borders. All plot decorations appear outside the main plot area, so the borders may have to be resized to give the plot an appropriate appearance.

The Format field dictates the number of decimal points and total field width that is used in displaying the axis labels. These values should be specified in terms of standard C formatting patterns as used with the standard input/output
function sprintf. This format is very similar to the syntax used for REAL variables in FORTRAN FORMAT specifications. The commands available in the Format Dialog are summarized in Table VII.4.

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prev</td>
<td>Switches to the previous or next format specification. Any of the formats numbered [0-7] can be edited at any time.</td>
</tr>
<tr>
<td>Next</td>
<td></td>
</tr>
<tr>
<td>Load</td>
<td>Loads or saves the particular format being displayed to or from the filename currently in the Filename field.</td>
</tr>
<tr>
<td>Save</td>
<td></td>
</tr>
<tr>
<td>Cancel</td>
<td>Dismisses the Format dialog. Retains the current contents of the dialog, but does not save those changes to the plot or to a file.</td>
</tr>
<tr>
<td>Accept</td>
<td>Changes the properties of the plot corresponding to the current format to the values input.</td>
</tr>
<tr>
<td>Revert</td>
<td>Changes the values in the dialog to either those values last saved or the default values. This function is essentially the same as switching to either the previous or next format and then switching back.</td>
</tr>
</tbody>
</table>

The Data Attributes dialog allows the user to customize the appearance of the data displayed in the main content area of a plot. This dialog is shown in Figure VII.6.
The first column contains an index into the colors specified in the colors files. The color of the label containing the variable name will be changed to this color whenever the dialog is updated. The number of dashes can be specified in the second column. The format for entries in this column is the number of pixels that should be drawn "on" followed by the number of pixels that should be drawn "off". For instance, 20 pixels on followed by 10 pixels off would be entered as "20 10". There are no commands associated with the Data Attributes dialog, since it only reflects additional information about the format currently selected with the Format dialog.

The File Edit dialog provides an editable text field in which the user can load, view, make changes to, and save ASCII text files. This dialog is shown in Figure VII.7.

<table>
<thead>
<tr>
<th>Data</th>
<th>Colors</th>
<th>Dashes</th>
<th>Colors</th>
<th>Dashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pressure</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Temperature</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mach Number</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burn Rate</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distance</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass Flow</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Surface Temp</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reynolds</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exit Pressure</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mass Burned</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mach Nozzle</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thrust</td>
<td></td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>13</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>15</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure VII.6: xplot DAta Attributes Dialog
The dialog is modeled after the xedit application included in the standard distribution of the X Windows from MIT. This rather simple dialog contains three action buttons, a single-line text field for entering filenames, and a large text field for displaying the contents of text files. This dialog is included to provide the user a simple capability of editing text files from within the application, without having to quit and restart the interface program or edit field in another shell tool. The load and save buttons read in or write out their contents based on the filename specified in the single-line text field. The contents either of the text buffer or the external text file will be overwritten upon executing one of these commands. The commands available in the File Edit dialog are summarized in Table VII.5.

Figure VII.7: xplot File Editor Dialog
Table VII.5: File Editor Dialog Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Done</td>
<td>Dismisses the dialog. Maintains the current state of the text displayed, but does not save those changes to a file.</td>
</tr>
<tr>
<td>Save</td>
<td>Saves the current state of the text to the filename given in the Filename text field.</td>
</tr>
<tr>
<td>Load</td>
<td>Loads the contents of the filename specified in the Filename field into the current text buffer. Overwrites the contents of the text buffer with the contents of the file specified in the Filename field. If the file does not exist (i.e., if the filename represents a new file), then that file is opened (if possible) and the contents of the text buffer are cleared.</td>
</tr>
</tbody>
</table>

The View Dataset dialog provides a simple read-only text field area displaying the currently loaded dataset. This dialog is shown in Figure VII.8.

![Figure VII.8: Dataset: ./sourceAcaveny.plot](image)

This dialog is provided only to allow the user to easily view the values as data in the plots created. This dialog functions just like the File Edit dialog, except it always contains the text of the current dataset, and the text in it can...
be neither edited nor saved. The dialog can be resized according to the width of data it displays. There are no commands associated with the View Dataset dialog. It must be shown or hidden from either the Dataset menu or the Window Manager dialog.

A simple command-line interface (or CLI) is provided for those users more comfortable with the more conventional means of entering commands. Note that each of the commands available through the CLI corresponds to one of the commands available in the graphical interface. A summary of the commands available is given in Table VII.7. In Table VII.7, the notation $i$ refers to the first argument, $2$ to the second, etc.
<table>
<thead>
<tr>
<th>Command</th>
<th>Args</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>plot</td>
<td>Changes the active plot to the one specified in $1.</td>
</tr>
<tr>
<td>color</td>
<td>data</td>
<td>Changes the color associated with the data specified in $1 to the color specified in $2. Equivalent to changing the appropriate &quot;color&quot; field on the Line Format dialog.</td>
</tr>
<tr>
<td>data</td>
<td>file</td>
<td>Loads the file specified in $1 as the current dataset.</td>
</tr>
<tr>
<td>decrement (dec)</td>
<td>plot</td>
<td>Decrements the plot specified in $1. If no argument is given, the active plot is decremented. Equivalent to selecting &quot;Decrement&quot; in the Plot Manager dialog.</td>
</tr>
<tr>
<td>edit</td>
<td>file</td>
<td>Loads the file specified in $1 as the text to be edited in the File Edit dialog and raises this dialog. If no file is given, then the previous filename is used.</td>
</tr>
<tr>
<td>execute (exec)</td>
<td>path</td>
<td>Executes the Caveny program from the path specified in $1 and the application name specified in $2. If neither argument is given, the defaults are taken from the values most recently specified in the Execute dialog.</td>
</tr>
<tr>
<td>format</td>
<td>format</td>
<td>Changes the format being edited in the Format dialog to the value specified in $1.</td>
</tr>
<tr>
<td>hide</td>
<td>plot</td>
<td>Hides the plot specified in $1. Equivalent to selecting &quot;Hide&quot; for the appropriate plot in the Plot Manager dialog.</td>
</tr>
<tr>
<td>increment (inc)</td>
<td>plot</td>
<td>Increments the plot specified in $1. If no argument is given, the active plot is incremented. Equivalent to selecting &quot;Increment&quot; in the Plot Manager dialog.</td>
</tr>
<tr>
<td>input</td>
<td>-</td>
<td>Loads the file specified in $1 as the input dataset to be used in the Input File Editor dialog and raises this Dialog. If no file is specified, then the dialog is raised with a new input dataset.</td>
</tr>
<tr>
<td>load</td>
<td>path</td>
<td>Loads a dataset from the full pathname given in $1. If no argument is given, it attempts to re-load the dataset most recently used with a &quot;load&quot; command from the &quot;Dataset&quot; menu.</td>
</tr>
<tr>
<td>plane</td>
<td>-</td>
<td>Sets a time/space plane on the currently active plot to $1. If the value specified in $1 is less than one or more than the time/space planes available, no action is taken.</td>
</tr>
<tr>
<td>plot</td>
<td>-</td>
<td>Creates the plot specified in $1.</td>
</tr>
<tr>
<td>quit</td>
<td>-</td>
<td>Terminates execution of xplot.</td>
</tr>
<tr>
<td>save</td>
<td>format</td>
<td>Saves the current format specified in $1.</td>
</tr>
<tr>
<td>show</td>
<td>plot</td>
<td>Shows the plot specified in $1. Equivalent to selecting &quot;Show&quot; for the appropriate plot in the Plot Manager dialog.</td>
</tr>
<tr>
<td>space</td>
<td>plot</td>
<td>Toggles the plot specified in $1 to being a space plot. If no argument is given, the active plot is toggled. Equivalent to toggling the &quot;Space plot/Time Plot&quot; button on the Plot Manager Dialog.</td>
</tr>
<tr>
<td>time</td>
<td>plot</td>
<td>Toggles the plot specified in $1 to being a time plot. If no argument is given, the active plot is toggled. Equivalent to toggling the &quot;Space plot/Time Plot&quot; button on the Plot Manager Dialog.</td>
</tr>
</tbody>
</table>

VII-14
The xplot application has a number of external files that must be present to run properly. In addition, a number of additional files can be used to configure the application according to the user's preferences, or to provide additional support for interactions with other programs.

In each of the following example listings, a vertical bar and all text following it should be interpreted as comments and should not be inserted into the input files.

XPlot.clr. This file specifies the colors available to be used in the line format dialog. The file is composed of a number of lines containing a number and a valid color name as specified in the X11R4 color database. Blank lines in the input file are ignored. These number-color pairs allow the user to define the colors that will be available for the various traces of data. An example file excerpt is given as Table VII.8:

| 1  | White          | Color #1 = White (255, 255, 255) |
| 2  | Black          | Color #2 = Black ( 0, 0, 0)       |
| ...|                |                                 |
| 18 | Red            | Color #18= Red (255, 0, 0)        |
| 19 | Green          | Color #19= Red ( 0, 255, 0)       |
| 20 | Blue           | Color #20= Red ( 0, 0, 255)       |

The XPlot.var file contains entries for all of the data to be placed in a Caveny program input dataset generated by the xplot program. Each variable entry in this file consists of a number of parameters surrounded by a leading and a trailing double-quote. The separate fields are terminated by a trailing slash (/). The fields of a variable entry are (in order): the variable list index, the FORTRAN name of the variable, a short definition of the variable, its default value (in appropriate units), and an optional long description. An example format of the entries is given in Table VII.9.

<table>
<thead>
<tr>
<th>0/</th>
<th>NDELX/</th>
<th>Number of spacewise steps(dx) along the propellant grain/ 20/</th>
<th>Maximum value = 29</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Format: Integer Units: None</td>
<td>Leading double-quote</td>
<td>Variable list index</td>
</tr>
<tr>
<td></td>
<td></td>
<td>FORTRAN variable name</td>
<td>Short variable description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Default value</td>
<td>Long variable description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

VII-15
The application-defaults file. The resources specified in the XPlot file located in the same directory with the xplot executable will be loaded in as fallback resources. Since user-specified resources take precedence over fallback resources specified by the application, resources set in either the user's .Xdefaults file or an XPlot file in the user's home account will override these settings.

Plot format files. Unlike the input files already described, there can be any number of plot format files, and they can have any name. The default format file, which is read upon program start-up, is named XPlot.def. This file is a simple text file enumerating the options selected in a plot format dialog. Each of the fields stored in the dialog is reflected in the file, in the same order as they appear in the dialog. Although there is no reason why the user could not edit this file using either the Input File dialog or a separate text processor, there is no reason to do so. These files are read and created by the Plot Format dialog directly. A sample listing for the XPlot.def file is given as Table VII.10.

<table>
<thead>
<tr>
<th>Plot #0</th>
<th>Plot title</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>Space data shown</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>Time data shown</td>
</tr>
<tr>
<td>0 0 0 0 0 0 0</td>
<td>Options selected</td>
</tr>
<tr>
<td>64 54 32 32</td>
<td>Border (L, T, R, B)</td>
</tr>
<tr>
<td>10.4 10.4</td>
<td>Axis label formats</td>
</tr>
<tr>
<td>10 10</td>
<td>Number of axis gridlines/labels</td>
</tr>
</tbody>
</table>

The example plots shown in this section were all produced by xplot with data generated by the Caveny program. All example plots used the default values for format attributes from the XPlot.clr and Xplot.def files. The setup required for the plots is described in the text following the plot. The data for these plots was generated by the Caveny code from the sample input file given as Appendix A.

Figures VII.9-VII.12 show the development of pressure in the motor at four different times (30, 60, 90, and 120 milliseconds, respectively). These plots trace the propagation of the initial igniter motor pressure wave down the length of the grain. Figures VII.11 and VII.12 are shown to the same scale to illustrate the effects of combustion on the chamber pressure. At 90 milliseconds, the grain has just begun to burn, and the spike towards the aft end of the section results from the igniter mass flow. At 120 milliseconds, much more of the grain has begun to burn, and combustion gases from the head end have made their way down the chamber.
Figure VII.9: Pressure versus Location at Time=30 ms

Figure VII.10: Pressure versus Location at Time=60 ms
Figure VII.11: Pressure versus Location at Time=90 ms

Figure VII.12: Pressure versus Location at Time=120 ms
These four plots are typical of the default type produced by xplot. All of the data are presented as a function of distance along the motor geometry. Figures VII.9 - VII.12 represent four time slices of the entire motor. Each plot contains a maximum of 30 points per curve, since this number is the maximum that the Caveny program allows for axial stations. Pressure was the only variable selected from the format dialog. The time was set using the Increment and Decrement buttons on the Plot Manager dialog. The scaling and number of gridlines were set manually. The number of decimal places for the x-axis were left at their default value of zero, but the y-axis labels were set to a 0.3 format.

Figures VII.13 and VII.14 show the data generated by the Caveny program in a different fashion. These plots illustrate the time history of a scalar quantity at two different axial distances along the motor. In this particular example, the chamber pressure is displayed as a function of time at the head end and aft end stations. Each of these plots contains 126 data points, since the Caveny input file was configured using its print interval and total simulation time to generate this number. This viewing option is set using the Space Plot/Time Plot toggle button on the Plot Manager dialog. With this option, the user can quickly switch back and forth in the manner chosen to view the data. The station is selected using the Increment and Decrement buttons on the same dialog. The scales, grids, and title were each set manually.
Figure VII.13: Pressure versus Time at Distance=3 cm

Figure VII.14: Pressure versus Time at Distance=1330 cm
Figures VII.15 and VII.16 show that multiple traces of data can be displayed on a single plot. In these two figures, the Mach number and grain burn rate are shown as functions of distance at two different times in the simulation (90 and 180 milliseconds). Of interest in the first plot is the spike in Mach number and the sharp drop-off in burn rate near the aft end of the motor. The spike again corresponds to the igniter discharge, and the burn rate reflects the fact that the aft end of the motor has not yet begun to burn. By 180 milliseconds, both curves have flattened out to more stable values all along the motor. In each plot, the legend reflects the colors associated with the traces for the two variables.

To configure these two plots, Mach number and burn rate were the only variables selected using the Format dialog. To make the traces print more distinctly, the colors and dashes for burn rate were changed from their default values. Using the Format Lines dialog, the burn rate color was set to black, and the dashes were set to 20 on 10 off (by entering "20 10" in the variable's appropriate field). In these two figures, the plots were manually titled and automatically scaled.

Figure VII.15: Mach Number & Burn Rate versus Distance at Time=90 ms
Figures VII.17 and VII.18 are presented as examples of variables that are functions only of time and not distance along the grain. In these two figures, the Reynolds number at the exit of the motor and the thrust generated by the motor are plotted. The curve for the thrust produced is in accordance with the plot of aft end pressure shown in Figure VII.14. The thrust is zero until the igniter gases begin to be discharged at approximately 65 milliseconds. The Reynolds number at exit is a predominantly linear function with the exception of the ignition spike from 60 to 100 milliseconds.

To create plots of variables that are time-dependent only, the Time Plot option must be selected from the Plot Manager dialog. If this option is not chosen, no data will be drawn in the plot frame. Note that the Increment and Decrement buttons in this dialog do not function if only time-dependent variables are selected. In all other aspects, these variables are configured and displayed in exactly the same way as the other variables when plotted as functions of time.
Figure VII.17: Reynolds Number at Exit versus Time

Figure VII.18: Motor Thrust versus Time
APPENDIX A

Sample Caveny Program Input File
MM-1 17DEC77 FLAME SPREAD CONTROL AT HEAD END, DYNAMIC THRUST CHECKOUT

&NAME

THMAX = 0.002,
TPRINT = 0.001,
DELTAT = 0.00016,
NDELX = 24,
TPI = 298.89,
PAM = 0.84717,
UNIT = -2.0,
TIGN = 2685.4,
TREF = 3353.0,
W = 28.18,
GAMA = 1.136,
AT = 2325.81,
XP = 2.67,
XE = 1329.972,
RUFSUR = 0.01,
FKPR = 0.0011,
ROPR = 1.758,
CPR = 0.3,
SIGF = 0.0009939,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
AT = 2325.81,
XP = 2.67,
XE = 1329.972,
RUFSUR = 0.01,
FKPR = 0.0011,
ROPR = 1.758,
CPR = 0.3,
SIGF = 0.0009939,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
AT = 2325.81,
XP = 2.67,
XE = 1329.972,
RUFSUR = 0.01,
FKPR = 0.0011,
ROPR = 1.758,
CPR = 0.3,
SIGF = 0.0009939,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
DE = 145.64,
CM = 0.97844,
AT = 2325.81,
XP = 2.67,
XE = 1329.972,
RUFSUR = 0.01,
FKPR = 0.0011,
ROPR = 1.758,
CPR = 0.3,
SIGF = 0.0009939,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
RREF = 1.054,
PREF = 68.08,
EBC = 0.0,
| 69 | 0.0000 | 0.0000 |
| 70 | 1.0000 | 1.0000 |
| 71 | 2       |        |
| 72 | 0.0000 | 0.0000 |
| 73 | 1.0000 | 1.0000 |
| 74 | 2       |        |
| 75 | 0.0000 | 0.0000 |
| 76 | 1.0000 | 1.0000 |
| 77 | 2       |        |
| 78 | 0.0000 | 0.0000 |
| 79 | 1.0000 | 1.0000 |
| 80 | 2       |        |
| 81 | 0.0000 | 0.0000 |
| 82 | 1.0000 | 1.0000 |
| 83 | 2       |        |
| 84 | 0.0000 | 0.0000 |
| 85 | 1.0000 | 1.0000 |
| 86 | 2       |        |
| 87 | 0.0000 | 0.0000 |
| 88 | 1.0000 | 1.0000 |
| 89 | 2       |        |
| 90 | 0.0000 | 0.0000 |
| 91 | 1.0000 | 1.0000 |
| 92 | 2       |        |
| 93 | 0.0000 | 0.0000 |
| 94 | 1.0000 | 1.0000 |
| 95 | 2       |        |
| 96 | 0.0000 | 0.0000 |
| 97 | 1.0000 | 1.0000 |
| 98 | 2       |        |
| 99 | 0.0000 | 0.0000 |
| 100| 1.0000 | 1.0000 |
| 101| 2       |        |
| 102| 0.0000 | 0.0000 |
| 103| 1.0000 | 1.0000 |
| 104| 2       |        |
| 105| 0.0000 | 0.0000 |
| 106| 1.0000 | 1.0000 |
| 107| 0.        |
| 108| 0.02   | 18.11  |
| 109| 0.028  | 46.95  |
| 110| 0.044  | 406.05 |
| 111| 0.052  | 484.35 |
| 112| 0.064  | 514.35 |
| 113| 0.07   | 516.69 |
| 114| 0.088  | 510.06 |
| 115| 0.18   | 427.47 |
| 116| 0.24   | 392.42 |
| 117| 0.288  | 371.00 |
| 118| 0.344  | 348.43 |
| 119| 0.356  | 339.86 |
| 120| 0.38   | 313.75 |
| 121| 0.424  | 275.18 |
| 122| 0.46   | 253.38 |
| 123| 0.508  | 232.73 |
| 124| 0.556  | 219.5  |
| 125| 0.566  | 0.0    |
| 126| 2.670  | 2538.440 | 1154.450 | 1154.45 | 31. | 22439. | 59741. | 2.21 |
| 127| 57.970 | 2543.620 | 1156.490 | 1156.49 |
| 128| 113.280 | 2561.700 | 1144.950 | 1144.95 |
| 129| 168.580 | 2706.870 | 1812.180 | 1812.18 |
| 130| 223.890 | 2612.100 | 1812.180 | 1812.18 |
| 131| 279.190 | 2645.000 | 1823.100 | 1823.10 |
| 132| 334.500 | 2658.850 | 1827.900 | 1827.90 |
| 133| 389.800 | 2705.320 | 1843.380 | 1843.38 |
| 134| 445.100 | 2774.530 | 1867.720 | 1867.72 |
| 135| 500.410 | 2844.420 | 1890.600 | 1890.60 |
| 136| 555.710 | 2915.190 | 1914.000 | 1914.00 |
| 137| 611.020 | 3029.790 | 1951.130 | 1951.13 | 649.52 | 2896.0 | 20003.0 | 1.5 |

A-3
<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>138</td>
<td>666.320</td>
<td>2757.930</td>
<td>186.160</td>
<td>186.16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>139</td>
<td>721.630</td>
<td>2720.090</td>
<td>184.880</td>
<td>184.88</td>
<td></td>
<td></td>
</tr>
<tr>
<td>140</td>
<td>776.930</td>
<td>2789.300</td>
<td>187.220</td>
<td>187.22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>141</td>
<td>832.230</td>
<td>2859.380</td>
<td>189.560</td>
<td>189.56</td>
<td></td>
<td></td>
</tr>
<tr>
<td>142</td>
<td>887.540</td>
<td>2930.330</td>
<td>191.890</td>
<td>191.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>143</td>
<td>942.840</td>
<td>3072.280</td>
<td>194.900</td>
<td>194.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>144</td>
<td>998.150</td>
<td>3025.310</td>
<td>197.400</td>
<td>197.40</td>
<td></td>
<td></td>
</tr>
<tr>
<td>145</td>
<td>1053.450</td>
<td>3224.030</td>
<td>201.280</td>
<td>201.28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>146</td>
<td>1108.760</td>
<td>3731.700</td>
<td>216.550</td>
<td>216.55</td>
<td></td>
<td></td>
</tr>
<tr>
<td>147</td>
<td>1164.060</td>
<td>4316.430</td>
<td>232.900</td>
<td>232.90</td>
<td></td>
<td></td>
</tr>
<tr>
<td>148</td>
<td>1219.360</td>
<td>4946.200</td>
<td>249.310</td>
<td>249.31</td>
<td></td>
<td></td>
</tr>
<tr>
<td>149</td>
<td>1274.670</td>
<td>5404.350</td>
<td>260.600</td>
<td>260.60</td>
<td></td>
<td></td>
</tr>
<tr>
<td>150</td>
<td>1329.972</td>
<td>5594.670</td>
<td>265.150</td>
<td>265.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>151</td>
<td>0.6808</td>
<td>0.1002</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>152</td>
<td>4.7651</td>
<td>0.4383</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>153</td>
<td>68.08</td>
<td>1.054</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| 154 | TMAX = 0.01,  
|     | DELFAC = 1.3  
| 155 | &NAME
| 156 | &END
| 157 |  
| 158 | &NAME
| 159 | TMAX = 0.30  
| 160 | &END
| 161 |  
| 162 | TMAX = 0.40,  
|     | TPRINT = 0.005  
| 163 | &END
| 164 |

A-4
APPENDIX B

Caveny Program Makefile
# File: makefile

# This file contains all dependencies necessary to make the caveny code

F77 = f77
FFLAGS= -g

OBJECTS =
    ans.o
    brcal.o
    fccal.o
    fct.o
    fct2.o
    filt.o
    hccal.o
    inhom.o
    iterpl.o
    ladin2.o
    lbc.o
    migdot.o
    pnch.o
    prepc.o
    prepp.o
    rbc.o
    redpun.o
    runge.o
    sdata.o
    sdatac.o
    slotk.o
    solutc.o
    solutp.o
    srnbug.o
    thursl.o
    timest.o
    trid.o
    decomp.o
    solv.o
    gsolv.o
    asolv.o
    vsolv.o
    repl2.o
    repl1.o
    sing.o
    tscal.o
    main.o

caveny: $(OBJECTS)
    $(F77) $(FFLAGS) -o caveny $(OBJECTS)
APPENDIX C

xplot Source Program
/*
xpCommands.c
*/

#include "xpIncs.h"
#include "xpDefs.h"
#include "xpwidgets.h"
#include "xpgeneral.h"
#include "xpmenus.h"
#include "xpformats.h"
#include "xpplots.h"

#define cSpace 32
#define cTab 9

/* $1: Global variables */
char cli[64]=('*');
char cliString[16][64]={ ................................
}, a i , t t e # p

int cliIndex=0,cliNum=0;

/* $2: Local variables - (none) */
char commandString[16][32], *commandStack[16];
char inputString[64];

/* $3: Global functions */
void Listcallback();
void SaveCommuknd();
void RecallCommand();
void Echo();

/* $4: Local functions */
void Push();
void BreakdownString();
void DoParse();

/* $5: Implementation */

void ListCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        XawListReturnStruct *r = (XawListReturnStruct*) call_data;
        char s[64];
        strcpy(s,(*r).string);
        XtVaSetValues(cliText,XtNstring,s,NULL);
        XawTextSetInsertionPoint(cliText,strlen(s));
    }

void SaveCommand (w, event, params, numParams)
    Widget w; XEvent *event; String *params; Cardinal *numParams;
    {
        XawTextPosition n;
        char s[64], *ss;
        int num;
        XtVaGetValues(cliText,XtNstring,&ss,NULL);
        n=strlen(ss);
        if (n>0) {
            if (n>63) n=63;
            strncpy(s,ss,n);
            s[n]='\0';
        }
Push(s);
cliNum++;
cliIndex=0;
XtVaSetValues(cliText,
   XtNstring,'',XtNinsertPosition,strlen(''),NULL);
printf("cli: %d %s\n",cliNum,s);
}
strcpy(inputString,s);
BreakdownString(inputString,commandString,&num);
DoParse(num,commandString);
}

void RecallCommand(w,event,params,numParams)

  Widget w; XEvent *event; String *params; Cardinal *numParams;

  { if(*numParams>0)
    if(params[0][0]=='1') {
      if((cliIndex<cliNum)&&(cliIndex<15)) {
        cliIndex++;
        XtVaSetValues(cliText,XtNstring,cliString[cliIndex],NULL);
        XawTextSetInsertionPoint(cliText,strlen(cliString[cliIndex]));
      }
    }
    if(params[0][0]=='2') {
      if(cliIndex>0) {
        cliIndex--;
        XtVaSetValues(cliText,XtNstring,cliString[cliIndex],NULL);
        XawTextSetInsertionPoint(cliText,strlen(cliString[cliIndex]));
      }
    }
  }

  void Echo(s)

    String s;

    { int sLength=strlen(s),code;
      XawTextBlock text;

      text.firstPos=0; text.length=sLength; text.ptr=s; text.format=FMT8BIT;

      code = XawTextReplace(logText,startPos,endPos,&text);
      startPos = startPos+sLength;
      endPos = startPos;
      XawTextSetInsertionPoint(logText,startPos);
    }

  void Push(s)

    String s;

    { int i;
      for(i=(NUMcliString-1);i>1;i--)
        strcpy(cliString[i],cliString[i-1]);
      strcpy(cliString[1],s);
      strcpy(cliString[0],"*");
      for (i=0;i<NUMcliString;i++)
        commandStack[NUMcliString-i-1]=cliString[i];
      XawListChange(commList,commandStack,15,0,False);
    }

  void BreakdownString(old,new,count)

    char old[32]; char new[32]; int *count;

    { int i=0,m=0,n=0;
      while (old[i]!=='\0') {
        while ((old[i]==cSpace)||(old[i]==cTab))
          i++;
        while ((old[i]==cSpace)&&(old[i]==cTab)&&(old[i]=='\0')&&(n<32))
          n++;
        old[i]='\0';
      }
      for (i=0;i<32;i++)
        new[i]=old[i];
    }
new[m][n] = old[i];
i++;
n++;
}
new[m][n]='\0';
m++;
n=0;
}
*count=m;
}

void DoParse(argc,argv)
int argc; char argv[](32);
{
   int l,m,n;
   if (argc>0) {
      if (strcmp(argv[0],'quit')==0) {
         exit(0);
      }
      else if (strcmp(argv[0],'change')==0) {
         printf("<change> unimplemented\n");
      }
      else if (strcmp(argv[0],'decrement')==0) {
         printf("<decrement> unimplemented\n");
      }
      else if (strcmp(argv[0],'edit')==0) {
         printf("<edit> unimplemented\n");
      }
      else if (strcmp(argv[0],'execute')==0) {
         printf("<execute> unimplemented\n");
      }
      else if (strcmp(argv[0],'increment')==0) {
         printf("<increment> unimplemented\n");
      }
      else if (strcmp(argv[0],'kill')==0) {
         if (argc>l) {
            if (strcmp(argv[l],'plot')==0) {
               n=atoi(argv[2]);
               if ((n>=0)&&(n<MAXPLOTS)&&(plotExists[n]))
                  KillPlot(n);
            }
         }
      }
      else if (strcmp(argv[0],'load')==0) {
         if (argc>l) {
            if (strcmp(argv[l],'data')==0) {
               LoadData(dsName);
            }
         }
      }
      else if (strcmp(argv[0],'make')==0) {
         if (argc>2) {
            n=atoi(argv[2]);
            if ((n>=0)&&(n<MAXPLOTS)&&(plotExists[n]))
               MakePlot(n);
         }
      }

}
else if (strcmp(argv[0],"print")==0) {
    printf("<print> unimplemented\n");
}
else if (strcmp(argv[0],"save")==0) {
    printf("<save> unimplemented\n");
}
else if (strcmp(argv[0],"set")==0) {
    if (argc>1) {
        if (strcmp(argv[1],"data")==0) {
            if (argc>2) {
                strcpy(dsName,argv[2]);
                SetStatusLabel();
            }
            else SelectString(1);
        }else if (strcmp(argv[1],"format")==0) {
            if (argc>2) {
                strcpy(pfName,argv[2]);
                SetStatusLabel();
            }
            else SelectString(2);
        }
    }else if (strcmp(argv[0],"show")==0) {
        if (argc>2) {
            if (strcmp(argv[1],"plot")==0) {
                n=atoi(argv[2]);
                if ((n>=0)&&(n<=7)&&(plotExists[n]))
                    ShowHide2(dummy,wPlot0+n,True,0);
            }
        }
    }else if (strcmp(argv[0],"view")==0) {
        printf("<view> unimplemented\n");
    }else {
        printf("unrecognized command [%s] [%d]...\n",argv[0],argc-1);
    }
}

C-5
float xMin,xMax,xRange;
float zMin,zMax,zRange;
float yMin[MAXPLOTS],yMax[MAXPLOTS],yRange[MAXPLOTS];

/* $2: Local variables */

float d[MAXPLANES][MAXCOLS+1][MAXROWS];

/* $3: Global functions */

void LoadData();
void PrintData();
void CopyData();

/* $4: Local functions */

void StatInd();
void StatData();
void PrintPlotData();

/* $5: Implementation ............................................................... */

void LoadData(infile)
char *infile;
{
    FILE *fi;
    int i,j,k,n=0;
    float dummy;
    char s[64];
    printf(">Now loading dataset from file: <<%s>>,infile);
    if (fi=fopen(infile,"r") != NULL)
        strcpy(s."Dataset: ");
        strcat(s,infile);
        XtVaSetValues(dataShell,XtNtitle,s,NULL);
        XtVaSetValues(dataText,XtNstring,infile,NULL);
        for (i=0;i<8;i++)
            XtVaSetValues(PMitem[l][i],XtNsensitive,True,NULL);
        dataLoaded=True;
        fscanf(fi,'%d%d%d%d
'.&numCols,&numRows,&numTimeDep);
        for (k=0;k<numCols;k++)
            fgets(columnNames[k],25,fi);
            for (i=0;i<strlen(columnNames[k]);i++)
                if (columnNames[k][i]=='\n') columnNames[k][i]='\0';
                XtVaSetValues(FEItem[1][k],XtNlabel,columnNames[k],
                XtNsensitive,True,NULL);
                XtVaSetValues(CFItem[0][MAXCOLS+k],XtNlabel,columnNames[k],
                XtNsensitive,True,NULL);
                XtVaSetValues(DAItem[0][8+k],XtNlabel,columnNames[k],NULL);
        if ((numRows>0) && (numCols>0))
        for (k=0;k<numTimeDep;k++)
            fgets(columnNames[MAXCOLS+k],25,fi);
            for (i=0;i<strlen(columnNames[MAXCOLS+k]);i++)
                if (columnNames[MAXCOLS+k][i]=='\n')
                    columnNames[MAXCOLS+k][i]='\0';
                    XtVaSetValues(FEItem[2][k],XtNlabel,columnNames[MAXCOLS+k],
                    XtNsensitive,True,NULL);
                    XtVaSetValues(CFItem[0][MAXCOLS+k],XtNlabel,columnNames[MAXCOLS+k],
                    XtNsensitive,True,NULL);
                    XtVaSetValues(DAItem[0][8+k],XtNlabel,columnNames[MAXCOLS+k],NULL);
    if ((numRows>0) && (numCols>0))
        for (k=0;
            do {
                n=fscanf(fi,"%e",&z[k]);
                if (n!=EOF) {
                    for (i=0;i<numRows;i++)
                        if (k==0)
fscanf(fi, "%e", &x[i]);

else
fscanf(fi, "%e", &dummy);

for (j = 0; j < numCols; j++)
fscanf(fi, "%e", &d[k][j][i]);

for (i = 0; i < numTimeDep; i++)
fscanf(fi, "%e", &d[k][MAXCOLS][i]);

k++;

} while ((n != EOF) && (k < (MAXPLANES - 1)));

numPlanes = k - 1;
close(fi);
StatInd();
else printf("Unable to load dataset (does not exist)\n");

}

void PrintData(outfile)
char *outfile;
{
FILE *fo;
int i, j, k;
printf(">Now printing dataset to file: <%s>\n", outfile);
fo = fopen(outfile, "w");
fprintf(fo, "%4d %4d %4d\n", numPlanes, numCols, numRows);
if ((numRows > 0) && (numCols > 0) && (numPlanes > 0))
for (k = 0; k < numPlanes; k++)
{
for (i = 0; i < numRows; i++)
{
for (j = 0; j < numCols; j++)
{
printf(fo, "%12.6f\n", z[k][j]);
}
}
}
fclose(fo);

void CopyData(plotNum)
int plotNum;
{
int i, j, k;
if (planar[plotNum])
for (j = 0; j < numCols; j++)
for (i = 0; i < numRows; i++)
plotData[plotNum][j][i] = d[currentPlane[plotNum]][j][i];
else {
for (j = 0; j < numCols; j++)
for (i = 0; i < numPlanes; i++)
plotData[plotNum][j][i] = d[i][j][currentRow[plotNum]];
for (j = 0; j < numTimeDep; j++)
for (i = 0; i < numPlanes; i++)
plotData[plotNum][MAXCOLS + j][i] = d[i][MAXCOLS][j];
}

void StatData(p)
int p;
{
int i, j, m, n, test;
float Max, Min;
int count;

count = (planar[p] ? numRows : numPlanes);
test = (planar[p] ? MAXCOLS : 2 * MAXCOLS);
for (j=0; j<test; j++)
    if (useCol[p][j]) {
        Min= 1000000;
        Max=-1000000;
        for (i=0; i<count; i++) {
            if (plotData[p][j][i]<Min) Min=plotData[p][j][i];
            if (plotData[p][j][i]>Max) Max=plotData[p][j][i];
        }
        dMin[p][j]=Min;
        dMax[p][j]=Max;
    }

Min= 1000000;
Max=-1000000;
for (j=0; j<test; j++)
    if (useCol[p][j]) {
        if (dMin[p][j]<Min) Min=dMin[p][j];
        if (dMax[p][j]>Max) Max=dMax[p][j];
    }

void StatInd()
{
    int i;
    xMin= 1000000;
    xMax=-1000000;
    for (i=0; i<numRows; i++) {
        if (x[i]<xMin) xMin=x[i];
        if (x[i]>xMax) xMax=x[i];
    }
    xRange=xMax-xMin;
    zMin= 1000000;
    zMax=-1000000;
    for (i=0; i<numPlanes; i++) {
        if (z[i]<zMin) zMin=z[i];
        if (z[i]>zMax) zMax=z[i];
    }
    zRange=zMax-zMin;
}

void PrintPlotData(p)
{
    int p;
    (int i, j, k);
    printf("%d %d
", currentPlane[p], numCols, currentRow[p]);
    printf("%d %d %d
", currentPlane[p], numCols, currentRow[p]);
    if (planar[p]) {
        printf("%d %d
", z[currentPlane[p]]);
        for (i=0; i<numRows; i++)
            printf("%10.4f
", x[i]);
        for (j=0; j<numCols; j++)
            printf("%10.4f,%10.4f
", xMin, xMax);
    }
    printf("%10.4f
", xMin);
    printf("%10.4f
", xMax);
    printf("%10.4f
", xRange);
}

else {

C-8
if ((numPlanes>0) && (numCols>0))
    printf("%10.4f\n",x[currentRow[p]]);
for(i=0;i<numPlanes;i++)
    printf("%10.4f",z[i]);
for(j=0;j<numCols;j++)
    printf("%10.4f",plotData[p][j][i]);
printf("\n");
printf("zMin: %10.4f\n",zMin);
printf("zMax: %10.4f\n",zMax);
printf("zRange: %10.4f\n",zRange);
printf("yMin: %10.4f\n",yMin[p]);
printf("yMax: %10.4f\n",yMax[p]);
printf("yRange: %10.4f\n",yRange[p]);
}

/*
   xpFormats.c
*/

#include 'xpInc's.h'
#include 'xpDefs.h'

#include 'xpgeneral.h'
#include 'xpwidgets.h'
#include 'xpmenus.h'
#include 'xpdata.h'
#include 'xpplots.h'
#define feLoad 1
#define feSave 2
#define feCancel 3
#define feAccept 4
#define feRevert 5

/* $1: Global variables */
PlotFormat f[MAXPLOTS];
int activeFormat=0;

/* $2: Local variables - (none) */

/* $3: Global functions */
void InitFormat();
void SetFEValues();
void GetFEValues();
void RecalcFormat();
void FEToggleCallback();
void FECommandCallback();
void SetFEValues();
void GetFEValues();

/* $4: Local functions - (none) */

/*
   void PrintFormat();
*/

/* $5: Implementation *****************************************************/
void InitFormat(p)
int p;
{
    int i,j;


C-9
sprintf(f[p].title, *"Plot \#d", p);
for (i=0; i<8; i++) {
    f[p].color[i] = i;
    f[p].color[MAXCOLS+i] = MAXCOLS+i;
    f[p].numDashes[i] = 0;
    for (j=0; j<8; j++)
        f[p].dashes[i][j] = 0;
    f[p].opt[i] = False;
}
}

void SetFEValues(p)
   int p;
   {
    char s[16][32], t[16];
    int i, j;
    strcpy(s[0], f[p].title);
    XtVaSetValues(FEItem[FEBTitle][0], XtNstring, s[0], NULL);
    for (i=0; i<(2*MAXCOLS); i++) {
        sprintf(s[i], *"%d", f[p].color[i]);
        XtVaSetValues(DAItem[1][i], XtNstring, s[i], NULL);
        XtVaSetValues(DAItem[0][i],
                      XtNbackground, lineColor[f[p].color[i]],
                      XtNforeground, rColor[Black].pixel, NULL);
        for (i=0; i<(2*MAXCOLS); i++)
            strcpy(s[i], *""");
        for (j=0; j<f[p].numDashes[i]; j++) {
            sprintf(t, *"%d", (int) f[p].dashes[i][j]);
            strcat(s[i], t);
            XtVaSetValues(DAItem[2][i], XtNstring, s[i], NULL);
        }
        sprintf(t, *"%%%sf", f[p].xFormat);
        sprintf(s[0], t, f[p].xMin);
        sprintf(s[1], t, f[p].xMax);
        sprintf(s[2], *"%d", f[p].xInc);
        sprintf(s[3], *"%d", f[p].xTick);
        strcpy(s[4], f[p].xFormat);
        sprintf(s[5], *"%d", f[p].borderL);
        for (i=0; i<7; i++)
            XtVaSetValues(FEItem[FEBX][i], XtNstring, s[i], NULL);
        sprintf(t, *"%%%sf", f[p].yFormat);
        sprintf(s[0], t, f[p].yMin);
        sprintf(s[1], t, f[p].yMax);
        sprintf(s[2], *"%d", f[p].yInc);
        sprintf(s[3], *"%d", f[p].yTick);
        strcpy(s[4], f[p].yFormat);
        sprintf(s[5], *"%d", f[p].borderT);
        for (i=0; i<7; i++)
            XtVaSetValues(FEBY[i], XtNstring, s[i], NULL);
    }
}
for (i=0;i<7;i++)
   XtVaSetValues(FEItem[FEBY][i], XtNstring, s[i], NULL);
for (i=0;i<MAXCOLS;i++)
   XtVaSetValues(FEItem[FEBV][i], XtNsensitive, True, NULL);
   if (i < numCols)
      XtVaSetValues(FEItem[FEBV][i], XtNsensitive, True, NULL);
else
   XtVaSetValues(FEItem[FEBV][i], XtNsensitive, False, NULL);
for (i=0;i<MAXCOLS;i++)
   if (i < numTimeDep)
      XtVaSetValues(FEItem[2][i], XtNsensitive, True, NULL);
   else
      XtVaSetValues(FEItem[2][i], XtNsensitive, False, NULL);
for (i=0;i<8;i++)
   XtVaSetValues(FEItem[FEBOpts][i], XtNstate, f[p].opt[i], NULL);

void GetFEValues(p)
{
   int p;

   String t;
   char s[16][16];
   int i,j,count,u[8];
   XtVaGetValues(FEItem[0][0], XtNstring, &t, NULL);
   strcpy(f[p].title, t);
   for (i=0;i<(2*MAXCOLS);i++) {
      XtVaGetValues(DAItem[l][i], XtNstring, &t, NULL);
      strcpy(s[i],t);
      j = atoi(s[i]);
      f[p].color[i] = ( (j>=0) && (j<=31) ? j : i );
   }
   for (i=0;i<(2*MAXCOLS);i++) {
      XtVaGetValues(DAItem[2][i], XtNstring, &t, NULL);
      count = sscanf(s[i],'%d %d %d %d %d %d %d %d',
                     &u[0],&u[l],&u[2],&u[3],&u[0],&u[l],&u[2],&u[3]);
      f[p].numDashes[i] = count;
      f[p].dashes[i][j] = (char) u[j];
   }
   for (i=0;i<7;i++)
      XtVaGetValues(FEItem[FEBX][i], XtNstring, &t, NULL);
      strcpy(s[i],t);
   f[p].xMin = atoi(s[0]);
   f[p].xMax = atoi(s[1]);
   f[p].xNum = atoi(s[2]);
   f[p].xTick = atoi(s[3]);
   strncpy(f[p].xFormat, s[4], 15);
   f[p].borderL = atoi(s[5]);
   f[p].borderR = atoi(s[6]);
   for (i=0;i<7;i++)
      XtVaGetValues(FEItem[FEBY][i], XtNstring, &t, NULL);
      strcpy(s[i],t);
   f[p].yMin = atoi(s[0]);
   f[p].yMax = atoi(s[1]);
   f[p].yNum = atoi(s[2]);
   f[p].yTick = atoi(s[3]);
   strncpy(f[p].yFormat, s[4], 15);
f[p].borderT=atoi(s[5]);
f[p].borderB=atoi(s[6]);

for(i=0;i<numCols;i++)
  XtVaGetValues(FEItem[FEBVars][i],XtNstate,&useCol[p][i],NULL);
for(i=0;i<numTimeDep;i++)
  XtVaGetValues(FEItem[2][i],XtNstate,&useCol[p][MAXCOLS+i],NULL);
  for(i=0;i<8;i++)
    XtVaGetValues(FEItem[FEBOpts][i],XtNstate,&f[p].opt[i],NULL);

void RecalcFormat(p)
int p;
{
  f[p].xInc=(f[p].width-f[p].borderL-f[p].borderR)/f[p].xNum;
  f[p].yInc=(f[p].height-f[p].borderT-f[p].borderB)/f[p].yNum;
  f[p].nGrid=f[p].xNum+f[p].yNum+2;
  f[p].xLength=f[p].xNum*f[p].xInc;
  f[p].yLength=f[p].yNum*f[p].yInc;
  /*PrintFormat(p); */
}
*/

void PrintFormat(p)
int p;
{
  printf("\n");
  printf(" width: \d\n",f[p].width);
  printf(" height: \d\n",f[p].height);
  printf(" xInc: \d\n",f[p].xInc);
  printf(" yInc: \d\n",f[p].yInc);
  printf(" xLength: \d\n",f[p].xLength);
  printf(" yLength: \d\n",f[p].yLength);
}
*/

void LoadFormat()
{
  FILE *fi;
  char s[64],*t;
  int i,j,a=activeFormat;
  XtVaGetValues(FEItem[9][0],XtNstring,&t,NULL);
  strcpy(s,t);
  fi = fopen(s,"r");
  if (fi!=NULL) {
    fgets(t,63,fi);
    strcpy(s,t);
    if(strlen(s)>1)
      strncpy(f[a].title,s,strlen(s)-1);
    fgets(t,63,fi);
    strcpy(s,t);
    if(strlen(s)>1)
      strncpy(f[a].xFormat,s,strlen(s)-1);
    fgets(t,63,fi);
    strcpy(s,t);
    if(strlen(s)>1)
      strncpy(f[a].yFormat,s,strlen(s)-1);
    for (i=0;i<8;i++) {
      fscanf(fi,"%d",&f[a].color[i]);
      printf("%d ",f[a].color[i]);
    }
    printf("\n");
    for (i=0;i<8;i++) {
      fscanf(fi,"%d",j);
      f[a].opt[i] = ( (j==0) ? False : True );
    }
    fscanf(fi,"%d %d",&f[a].xNum,&f[a].yNum);
    fscanf(fi,"%e %e %e %e", &f[a].xMin,&f[a].xMax,&f[a].yMin,&f[a].yMax);
The code snippet provided is a C function that reads from a file and saves the format settings to another file. The functions include `fscanf` for reading data from the file, `fclose` for closing the file, and other functions related to setting and managing format values. The code is designed to save the current format settings into a file for later retrieval.

Here is a breakdown of the code:

```c
fscanf(fi, "%d %d", &f[a].xTick, &f[a].yTick);
fscanf(fi, "%d %d %d %d",
       &f[a].borderL, &f[a].borderR, &f[a].borderT, &f[a].borderB);
close(fi);
SetFEValues(a);

void SaveFormat()
{
    FILE *fo;
    char s[64], *t;
    int i, a = activeFormat;
    XtVaGetValues(FEItem[9] [0], XtNstring, &t, NULL);
    strcpy(s, t);
    if (s[0] != NULL) {
        fo = fopen(s, "w");
        fprintf(fo, "%s\n", f[a].title);
        fprintf(fo, "%s\n", f[a].xFormat, f[a].yFormat);
        for (i = 0; i < 8; i++)
            fprintf(fo, "%d\n", f[a].color[i]);
        fprintf(fo, "\n");
        for (i = 0; i < 8; i++)
            fprintf(fo, "%d\n", f[a].opt[i]);
        fprintf(fo, "\n");
        fprintf(fo, "%d %d\n", f[a].xNum, f[a].yNum);
        fprintf(fo, "%d %d\n", f[a].xMin, f[a].xMax, f[a].yMin, f[a].yMax);
        fprintf(fo, "%d %d %d %d\n", f[a].xTick, f[a].yTick);
        fprintf(fo, "%d %d %d %d\n", f[a].borderL, f[a].borderR, f[a].borderT, f[a].borderB);
        fclose(fo);
    }
}

void FEPageCallback(w, client_data, call_data)
{
    Widget w; XtPointer client_data, call_data;
    int i = (int) client_data;
    char s[64];
    switch (i) {
    case 0:
        if (activeFormat < (MAXPLOTS - 1)) {
            activeFormat++;
            SetFEValues(activeFormat);
            sprintf(s, "Format #%d", activeFormat);
            XtVaSetValues(FETop[FEBPage], XtNlabel, s, NULL);
        }
        break;
    case 1:
        if (activeFormat > 0) {
            activeFormat--;
            SetFEValues(activeFormat);
            sprintf(s, "Format #%d", activeFormat);
            XtVaSetValues(FETop[FEBPage], XtNlabel, s, NULL);
        }
        break;
    }
}

void FECommandCallback(w, client_data, call_data)
{
    Widget w; XtPointer client_data, call_data;
}
```

The code is part of a larger application, possibly related to a graphics or data visualization application, given the references to format settings and graphical elements like `xTick`, `yTick`, `xNum`, `yNum`, and graphical borders. The functions `fscanf` and `fclose` are used to read from and close files, while `SetFEValues` and `XtVaGetValues` are used to manage and retrieve format settings.
int i = (int) client_data;
switch (i) {
    case feLoad:
        LoadFormat();
        break;
    case feSave:
        SaveFormat();
        break;
    case feCancel:
        ShowHide2(dummy, wFormat, True, 0);
        SetFEValues(activeFormat);
        break;
    case feAccept:
        GetFEValues(activeFormat);
        if (plotExists[activeFormat]) {
            RecalcFormat(activeFormat);
            StatData(activeFormat);
            SetupGrid(activeFormat);
            SetupAxisLabels(activeFormat);
            SetupPoints(activeFormat);
            RedrawPlot2(activeFormat);
        }
        break;
    case feRevert:
        SetFEValues(activePlot);
        break;
}
/*
 xpGeneral.c
-----------------------------------------------
*/
#include "xpIncs.h"
#include "xpDefs.h"
#include "xpwidgets.h"
#include "xpcommands.h"
#include "xpdata.h"
#include "xplots.h"

/* $1: Global variables */
Screen *screen;
Display *display;
Boolean windowUp[48];
GC gc[MAXGCS];
XColor exact, color[16], rColor[16];
Font f6x10, f6x12, f6x13, fHelv18b;
XFontStruct *fs6x10, *fs6x12, *fs6x13, *fHelv18b;
XawTextPosition startPos = 0, endPos = 0;
char inName[64] = "", dsName[64] = "", pfName[64] = "", plName[4] = "";

/* $2: Local variables - (none) */

/* $3: Global functions */
void LoadFonts();
void LoadColors();
void InitData();
void ProcessKey();
void Idle();

/* $4: Local functions - (none) */

C-14
void LoadFonts()
{
    f6x10  = XLoadFont(display,"6x10");
    f6x12  = XLoadFont(display,"6x12");
    f6x13  = XLoadFont(display,"6x13");
    fHelv18b = XLoadFont(display,"-helvetica-bold-r-*-18-180-***");
    fs6x10 = XQueryFont(display,f6x10);
    fs6x12 = XQueryFont(display,f6x12);
    fs6x13 = XQueryFont(display,f6x13);
    fsHelv18b = XQueryFont(display,fHelv18b);
}

void LoadColors()
{
    colormap cmap;
    status r;
    file *fi;
    char s[64], t[64], *u, *v;
    int i, count=0;

    cmap = XDefaultColormap(display, DefaultScreen(display));

    fi = fopen("XPlot.clr", "r");
    if (fi==NULL) {
        while (fscanf(fi,"%d %s", &i, s)!=EOF) {
            printf("--%4d %s
", i, s);
            if ((i>=0) && (i<=31)) {
                r= XAllocNamedColor(display, cmap, s, &exact, &color[i]);
                fclose(fi);
            }
        }
    }

    r= XAllocNamedColor(display, cmap, "Black", &exact, &rColor[Black]);
    r= XAllocNamedColor(display, cmap, "White", &exact, &rColor[White]);
    r= XAllocNamedColor(display, cmap, "Red", &exact, &rColor[Red]);
    r= XAllocNamedColor(display, cmap, "Green", &exact, &rColor[Green]);
    r= XAllocNamedColor(display, cmap, "Blue", &exact, &rColor[Blue]);
    r= XAllocNamedColor(display, cmap, "Yellow", &exact, &rColor[Yellow]);

    for (i=0;i<(2*MAXCOLS);i++)
        lineColor[i]=color[i].pixel;
}

void InitData()
{
    int i, j, k;

    for(k=0;k<MAXPLOTS;k++)
        for(j=0;j<(2*MAXCOLS);j++)
            useCol[k][j]=false;

    for(i=0;i<NUMcliString;i++)
        strcpy(cliString[i], "");

    for(i=0;i<MAXPLOTS;i++)
        plotExists[i]=false;
}

void InitGCs()
{
    Window wind = XtWindow(topLevel);
}

/* $5: Implementation =============================== */
for (i=0; i<4; i++)
gc[i]=XCreateGC(display, wind, 0, 0);
XSetForeground(display, gc[0], rColor[Black].pixel);
XSetFont(display, gc[0], f6x10);
XSetForeground(display, gc[1], rColor[Black].pixel);
XSetFont(display, gc[1], f6x12);
XSetForeground(display, gc[2], rColor[Black].pixel);
XSetFont(display, gc[2], fHelv18b);
}

int i;

void DoKey(w, event, params, numParams)
  Widget w; XEvent *event; String *params; Cardinal *numParams;
  { XKeyEvent *e = (XKeyEvent*) event; }

void Idle(w, event, params, numParams)
  Widget w; XEvent *event; String *params; Cardinal *numParams;
  {
  /* xpGeometry.c
   */
   
   #include "xpIncs.h"
   #include "xpDefs.h"
   
   #include "xpwidgets.h"
   #include "xpgeneral.h"
   #include "xpcommands.h"
   #include "xpinput.h"
   
   #define pi 3.1415926535
   
   /* $1: Global variables - (none) */
   
   /* $2: Local variables */
   int activeGeometry=-1;
   float geomVars[8];
   
   static String MGInputLabels[6][8] = {
       { "Web Thickness", "Internal Radius", "Half-Angle", "Epsilon", "Fillet Radius", "", "", "" }
   };
   
   static String MGOutputLabels[6][8] = {
       { "Initial Area", "Final Area", "", "", "", "", "", "", "", "" }
   };
   
   /* $3: Global functions */
   void MGCallback();
   void ChangeMGLabels();
   void ResizeGeometry();
   void ExposeGeometry();
   
   /* $4: Local functions */
   void RecalcGeometry();
   void RedrawGeometry();
/* §5: Implementation ============================================== */

void MGCallback(w, client_data, call_data)
Widget w; XtPointer client_data, call_data;
{
    int n = (int) client_data;
    int m;
    char s[64], *t;

    switch (n) {
    case 0:
        m = (int) XawToggleGetCurrent(MGItem[2][0]);
        if (m != NULL) {
            activeGeometry = m - 1;
            RecalcGeometry();
        }
        break;
    case 1:
        ShowHide2(dummy, wMotorConfiguration, True, 0);
        break;
    case 2:
        ShowHide2(dummy, wMotorGeometry, False, 0);
        break;
    }
}

void ChangeMGLabels(n)
int n;
{
    int i;
    n = 0;
    for (i = 0; i < 8; i ++)
        XtVaSetValues(MGItem[0][i], XtNlabel, MGInputLabels[n][i], NULL);
    for (i = 0; i < 2; i ++)
        XtVaSetValues(MGItem[3][i], XtNlabel, MGOutputLabels[n][i], NULL);
}

void RecalcGeometry()
{
    int i, j, k;
    char s[64], *t;
    float re, ri, rr, w, bp;

    for (i = 0; i < 8; i ++)
        XtVaGetValues(MGItem[1][i], XtNstring, &t, NULL);  
        strcpy(s, t);  
        sscanf(s, "%e", &geomVars[i]);
    }

    switch (activeGeometry) {
    case 0:
        re = geomVars[0];  
        ri = geomVars[1];  
        w = re - ri;
        bp = 2 * ri * pi;
        break;
    case 1:
        re = geomVars[0];  
        ri = geomVars[1];  
        rr = geomVars[2];  
        w = re - ri;
        bp = 2 * ri * pi + 2 * rr * pi;
        break;
    case 2:
        break;
    }
    sprintf(s, "%.4f", bp);
}

C-17
XtVaSetValues(MGItem[4][0], XtNstring, s, NULL);

void ExposeGeometry(w, event, params, numParams)
    Widget w; XEvent *event; String *params; Cardinal *numParams;
    {
        XExposeEvent *myevent = (XExposeEvent*) event;
        static int i = 0;
        if ((*myevent).count == 0) {
            i++;
            printf(":ExposeGeometry (%d)...\n", i);
            RedrawGeometry();
        }
    }

void ResizeGeometry(widget, event, params, numParams)
    Widget widget; XEvent *event; String *params; Cardinal *numParams;
    {
        static Dimension oldWidth = 360, oldHeight = 360;
        Dimension w, h;
        XtVaGetValues(MGPict, XtNwidth, &w, XtNheight, &h, NULL);
        if ((w <= oldWidth) && (h <= oldHeight)) {
            printf(":ResizeGeometry calling RedrawGeometry...\n");
            RedrawGeometry();
            oldWidth = w;
            oldHeight = h;
        }
    }

void RedrawGeometry()
    {
        Dimension w, h;
        Window wind = XtWindow(MGPict);
        printf(":\n");
        XtVaGetValues(MGPict, XtNwidth, &w, XtNheight, &h, NULL);
        XClearWindow(display, wind);
        XSetLineAttributes(display, gc[0], 4, LineSolid, CapButt, JoinRound);
        XDrawArc(display, wind, gc[0], 4, w-8, h-8, 0, 360*64);
        XSetLineAttributes(display, gc[0], 0, LineSolid, CapButt, JoinRound);
        switch (activeGeometry) {
            case 0:
                break;
        }
    }
#define NDELX 0
#define TPI 1
#define PAM 2
#define NINERT 3
#define UNIT 4
#define AT 5
#define XP 6
#define XE 7
#define XG 8
#define GAMA 9
#define W 10
#define TIGN 11
#define TFREF 12
#define DSRG 14
#define DDC 15
#define PKPR 16
#define ROPE 17
#define TOREF 19
#define SIGP 20
#define TPSRI 21
#define REF 22
#define TREF 23
#define BREXP 24
#define EBC 25
#define EBEX 26
#define DE 27
#define CM 28
#define ALFAD 29
#define EEOAT 30
#define EROEXP 31
#define TMAX 32
#define DELTAT 33
#define TPRINT 34
#define PZONE 35
#define LAMBDA 36
#define NIGTAB 37
#define NAPDVX 38
#define NDATA 39
#define NPPR 40
#define ITABLE 41
#define GTABLE 42
#define BTABLE 43

/* $1: Global Variables */
char varName[MAXVARS][32], varDesc[MAXVARS][80];
char varNumb[MAXVARS][12], varInfo[MAXVARS][800];
float varValue[MAXVARS];
float iTable[30][2], gTable[30][8], bTable[30][2];

/* $2: Local Variables */
int numVars, inputIndex=0;
char *IEStrings[100], IEEEntries[100][20];

/* $3: Global Functions */
void LoadInputData();
void IECallback();
void IEListCallback();
void WriteInputFile();
void AcceptInputEntry();

/* $4: Local Functions */
void GetInputValues();
void setInputValues();
void IncDecInput();

/* $5: Implementation *********************************/

void LoadInputData()
{
    FILE *fi;
    int i,j=0;
    char s[64];

    fi=fopen("XPlot.var","r");
    do {
        i=fscanf(fi,"%*[ 	
"]);
        if (i!=EOF) {
            i=fscanf(fi,"%*[ 	
"]);
            i=fscanf(fi,"%[^/]/\n",varName[j]);
            i=fscanf(fi,"%[^/]/\n",varDesc[j]);
            i=fscanf(fi,"%[^/]/\n",varNumb[j]);
            i=fscanf(fi,"%[^]/\\n",varInfo[j]);
            j++;
        }
    } while ( i!=EOF)&&(i!=0)&&(j<AXVARS) );
    numVars=j-1;

    for (j=0;j<numVars;j++)
    i=sscanf(varNumb[j],"%e",&varValue[j]);
    printf("\n");

    for (i=0;i<=BTABLE;i++) {
        if (i<ITABLE)
            sprintf(IEEntries[i],"%-6s = %10.4f",varName[i],varValue[i]);
        else
            sprintf(IEEntries[i],"..... %c%s ......",varName[i][0],"TABLE");
        IEStrings[i] = IEEntries[i];
        printf("%2d %s\n",i,IEEntries[i]);
    }
    printf("\n");
    XawListChange(IEList,IEStrings,BTABLE+1,0,False);
}

void DoInput()
{
    int i,j,k;
    char s[64],*t,*u,*v;

    ShowHide2(dummy,1000,True,0);
}

void AcceptInputEntry(w,event,params,numParams)
{
    Widget w; XEvent *event; String *params; Cardinal *numParams;
    if(numParams>0) {
        if(params[0][0]=='+')
            IncDecInput(True);
        else
            IncDecInput(False);
    }
}

void GetInputValues(i)
{
    int j,k;
    char s[2500],*t;

    if ( (i>0)&&(i<ITABLE) ) {
void SetInputValues(int i)
{
    int j, k;
    char s[2500], t[80];
    if (i==ITABLE) {
        XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
        XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
        XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
        sprintf(s, "%.2f", varValue[i]);
        XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
        XawListHighlight(IEList, i);
    }
    else if (i==GTABLE) {
        XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
        XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
        XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
        strcpy(s, "");
        k = varValue[NAPDXV];
        for (j=0; j<k; j++) {
            sprintf(t, "%10.4f%10.4f%10.4f%10.4f %10.4f%10.4f%10.4f%10.4f",
                gTable[j][0], gTable[j][1], gTable[j][2], gTable[j][3],
                gTable[j][4], gTable[j][5], gTable[j][6], gTable[j][7]);
            strcat(s, t);
        }
        XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
        XawListHighlight(IEList, i);
    }
    else if (i==BTABLE) {
        XtVaSetValues(IEItem[0][0], XtNstring, varName[i], NULL);
        XtVaSetValues(IEItem[1][0], XtNstring, varDesc[i], NULL);
        XtVaSetValues(IEItem[2][0], XtNstring, varInfo[i], NULL);
        strcpy(s, """);
        k = varValue[NIGTAB];
        for (j=0; j<k; j++) {
            sprintf(t, "%10.4f%10.4f%10.4f%10.4f %10.4f%10.4f%10.4f%10.4f",
                tVaGetValues(IEItem[3][0], XtNstring, &t, NULL);
            strcpy(s, t);
            sscanf(t, "%e", &varValue[i]);
            sprintf(IEEntries[i], "%-6s = %10.4f", varName[i], varValue[i]);
        }
        XawListUnhighlight(IEList);
    }
}

XtVaGetValues(IEItem[3][0], XtNstring, &t, NULL);
strcpy(s, t);
sscanf(t, "%e", &varValue[i]);
sprintf(IEEntries[i], "%-6s = %10.4f", varName[i], varValue[i]);
XawListUnhighlight(IEList);

C-21
for (j=0; j<k; j++) {
    sprintf(t, "%10.4f%10.4f\n", bTable[j][0], bTable[j][1]);
    strcat(s, t);
}
XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
XtVaSetValues(IEItem[3][0], XtNstring, s, NULL);
XawListHighlight(IEList.1);
}

void IECallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        int n = (int) client_data;
        char s[128], *t;
        switch (n) {
            case 0: /* Start | Restart */
                inputIndex = 0;
                SetInputValues(inputIndex);
                break;
            case 1: /* Insert */
                break;
            case 2: /* Prev */
                IncDecInput(False);
                break;
            case 3: /* Next */
                IncDecInput(True);
                break;
            case 4: /* Write */
                XtVaGetValues(IEItem[4][0], XtNstring, &t, NULL);
                strcpy(s, t);
                if (strcmp(s, "") != 0) WriteInputFile(s);
                break;
            case 5: /* Cancel */
                ShowHide2(dummy, wInput, False, 0);
                inputIndex = 0;
                SetInputValues(inputIndex);
                break;
        }
    }

void IEListCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        XawListReturnStruct *r = (XawListReturnStruct*) call_data;
        char s[64];
        int n,
        n = (*r).list_index;
        printf(\"Element [\%02d] selected from IEList\n\", n);
        GetInputValues(inputIndex);
        inputIndex = n;
        SetInputValues(inputIndex);
    }

void IncDecInput(inc)
    Boolean inc;
    {
        char s[1024], *t;
        if ( (inc) && (inputIndex<(BTABLE)) ) {
            GetInputValues(inputIndex);
            inputIndex++;
            SetInputValues(inputIndex);
        }
        if ( (!inc) && (inputIndex>0)) {
            }
GetInputValues(inputIndex);
inputIndex--;
SetInputValues(inputIndex);

void WriteInputFile(s)
char *s;
{
FILE *f;
int i,j,k;
f = fopen(s, "w");
fprintf(f, " &NAME
' );
fprintf(f," TMAX = \$10.4f\n*, varValue[ 0]);
fprintf(f," LAMBDAD = \$10.4f\n*, varValue[ 1]);
fprintf(f," TPRINT = \$10.4f\n*, varValue[ 2]);
fprintf(f," NPUNCH = \$10.4f\n*, varValue[ 3]);
fprintf(f," NBL = \$10.4f\n*, varValue[ 4]);
fprintf(f," NDELF = \$10.4f\n*, varValue[ 5]);
fprintf(f," TPI = \$10.4f\n*, varValue[ 6]);
fprintf(f," PAM = \$10.4f\n*, varValue[ 7]);
fprintf(f," UNIT = \$10.4f\n*, varValue[ 8]);
fprintf(f," NINERT = \$10.4f\n*, varValue[ 9]);
fprintf(f," NPNPXT = \$10.4f\n*, varValue[10]);
fprintf(f," AT = \$10.4f\n*, varValue[11]);
fprintf(f," XP = \$10.4f\n*, varValue[12]);
fprintf(f," XIP = \$10.4f\n*, varValue[13]);
fprintf(f," XG = \$10.4f\n*, varValue[14]);
fprintf(f," TPR = \$10.4f\n*, varValue[15]);
fprintf(f," TIG = \$10.4f\n*, varValue[16]);
fprintf(f," TFREF = \$10.4f\n*, varValue[17]);
fprintf(f," TIGN = \$10.4f\n*, varValue[18]);
fprintf(f," RUP = \$10.4f\n*, varValue[19]);
fprintf(f," DDRC = \$10.4f\n*, varValue[20]);
fprintf(f," DDRC = \$10.4f\n*, varValue[21]);
fprintf(f," APIAT = \$10.4f\n*, varValue[22]);
fprintf(f," AFTOS = \$10.4f\n*, varValue[23]);
fprintf(f," FKPR = \$10.4f\n*, varValue[24]);
fprintf(f," ROPR = \$10.4f\n*, varValue[25]);
fprintf(f," CPR = \$10.4f\n*, varValue[26]);
fprintf(f," TOREF = \$10.4f\n*, varValue[27]);
fprintf(f," SIGH = \$10.4f\n*, varValue[28]);
fprintf(f," TPSCRE = \$10.4f\n*, varValue[29]);
fprintf(f," RREF = \$10.4f\n*, varValue[30]);
fprintf(f," PPRE = \$10.4f\n*, varValue[31]);
fprintf(f," BEXP = \$10.4f\n*, varValue[32]);
fprintf(f," EBC = \$10.4f\n*, varValue[33]);
fprintf(f," EBSK = \$10.4f\n*, varValue[34]);
fprintf(f," DE = \$10.4f\n*, varValue[35]);
fprintf(f," CM = \$10.4f\n*, varValue[36]);
fprintf(f," ALFAD = \$10.4f\n*, varValue[37]);
fprintf(f," EROAT = \$10.4f\n*, varValue[38]);
fprintf(f," EROEXP = \$10.4f\n*, varValue[39]);
fprintf(f," NIGTAB = \$10.4f\n*, varValue[40]);
fprintf(f," NAPDTX = \$10.4f\n*, varValue[41]);
fprintf(f," NDATA = \$10.4f\n*, varValue[42]);
fprintf(f," NPDP = \$10.4f\n*, varValue[43]);
fprintf(f," PZONE = \$10.4f\n*, varValue[44]);
fprintf(f," NRESRT = \$10.4f\n*, varValue[45]);
fprintf(f," TDELTA = \$10.4f\n*, varValue[46]);
fprintf(f," PCI = \$10.4f\n*, varValue[47]);
fprintf(f," GAMAN = \$10.4f\n*, varValue[48]);
fprintf(f," TPSUBK = \$10.4f\n*, varValue[49]);
fprintf(f," DELTTF = \$10.4f\n*, varValue[50]);
fprintf(f," DFSST = \$10.4f\n*, varValue[51]);
fprintf(f," TIN = \$10.4f\n*, varValue[52]);
fprintf(f," FCRIT = \$10.4f\n*, varValue[53]);
```c
fprintf(f,"OPEN = %10.4f\n",varValue[54]);
fprintf(f,"DELFAC = %10.4f\n",varValue[55]);
fprintf(f,"CHC = %10.4f\n",varValue[56]);
fprintf(f,"BF1 = %10.4f\n",varValue[57]);
fprintf(f,"DELTFF = %10.4f\n",varValue[58]);
fprintf(f," &END\n ");
for (i=0;i<varValue[NIGTAB];i++)
fprintf(f,"%10.4f%10.4f\n",iTable[j][0],iTable[j][1]);
for (i=0;i<varValue[NAPDVX];i++) {
    fprintf(f,"%10.4f%10.4f%10.4f%10.4f",
        gTable[j][0],gTable[j][1],gTable[j][2],gTable[j][3]);
    if (gTable[j][4]!=0) fprintf(f,"%10.4f",gTable[j][4]);
    if (gTable[j][5]!=0) fprintf(f,"%10.4f",gTable[j][5]);
    fprintf(f,"%10.4f\n",gTable[j][6]);
}
for (i=0;i<varValue[NDATA];i++)
fprintf(f,"%10.4f%10.4f\n",bTable[j][0],bTable[j][1]);
fclose(f);

/*
 */
xplot.c

#include "xpInc.h"
#include "xpDefs.h"
#include "xpwidgets.h"
#include "xpgenerals.h"
#include "xpmenus.h"
#include "xpcommands.h"
#include "xpinput.h"
#include "xpdata.h"
#include "xpformats.h"
#include "xpplots.h"
#include "xpgeometry.h"

void main(argc,argv)
int argc; char **argv;
{
    FILE *fi;
    static XtActionsRec xplotActions[] = {
        ("idle", Idle),
        ("doKey", DoKey),
        ("redrawPlot", RedrawPlot),
        ("saveCommand", SaveCommand),
        ("recallCommand", RecallCommand),
        ("resizePlot", ResizePlot),
        ("resizeGeometry", ResizeGeometry),
        ("exposeGeometry", ExposeGeometry),
        ("acceptInputEntry", AcceptInputEntry),
    };;
    XtAppContext app_context;
    int i;

    fi = fopen("XPlot.dat","w");
    topLevel = XtVaAppInitialize(&app_context,"XPlot",NULL,0,
        argc,argv,NULL,NULL);
    XtAppAddActions(app_context,xplotActions,XtNumber(xplotActions));
    CreateWidgets();
    XtRealizeWidget(topLevel);
}
```
screen = XtScreen(topLevel);
display = XtDisplay(topLevel);
LoadFonts();
LoadColors();
InitData();
InitGCs();
for (i = 0; i < MAXPLOTS; i++)
  InitFormat(i);
LoadInputData();
SetFEValues(0);
SetStatusLabel();
ChangeMGLabels(0);
printf("\n");
XtInstallAccelerators(rootPane, cliText);
XtInstallAccelerators(cliLabel, cliText);
XtAppMainLoop(app_context);

/*
 * xpMenus.c
 *
 */
#include "xpincs.h"
#include "xpDefs.h"
#include "xpwidgets.h"
#include "xpgeneral.h"
#include "xpcommands.h"
#include "xpdata.h"
#include "xpformats.h"
#include "xpplots.h"

/* $1: Global variables - (none) */
/* $2: Local variables - (none) */
/* $3: Global functions */
void MenuSelect();
void CommandSelect();
void VerbCallback();
void SFCallback();
void SFGetFile();
void SelectString();
void ShowHide2();
void Loadlnput();
void FileCallback();
void ExecCallback();
/* $4: Local functions */
void SetStatusLabel();
void RunExec();
/* $5: Implementation ***************************************************************************/
void MenuSelect(w, client_data, call_data)
  Widget w; XtPointer client_data, call_data;
  int menuData = (int) client_data;
  int menuNum, itemNum;
char s[64], *t;
FILE *f;

menuNum = menuData/HiWORD;
itemNum = menuData&HiWORD;

switch(menuNum) {
case IDMenuFile:
    switch(itemNum) {
    case 0:
        ShowHide2(dummy, wFile, True, 0);
        break;
    case 1:
        XtVaGetValues(menuText, XtNstring, &t, NULL);
        strcpy(s, t);
        f = fopen(s, "r");
        if (f != NULL) { 
            XtVaSetValues(fileText, XtNstring, s, XtNtype, XawAsciiFile, NULL);
        }
        break;
    case 2:
        ShowHide2(dummy, wFile, True, 0);
        break;
    case 3:
        break;
    case 4:
        XtVaGetValues(menuText, XtNstring, &t, NULL);
        strcpy(s, t);
        f = fopen(s, "w");
        XtVaGetValues(fileText, XtNstring, s, XtNtype, XawAsciiFile, NULL);
        fprintf(f, "%s", t);
        fclose(f);
        break;
    case IDMenuQuit:
        exit(0);
        break;
    }
    break;
    case IDMenuInpu:
    switch(itemNum) {
    case IDmInpuSet:
        SelectString(0);
        break;
    case IDmInpuLoad:
        LoadInput();
        break;
    case IDmInpuLoadSF:
        SelectString(0);
        LoadInput();
        break;
    case 4:
        ShowHide2(dummy, wInput, True, 0);
        break;
    }
    break;
    case IDMenuExec:
    switch(itemNum) {
    case IDmExecSetup:
        ShowHide2(dummy, wExecute, True, 0);
        break;
    case IDmExecRun:
        RunExec();
        break;
    }
    break;
    case IDMenuData:
    switch(itemNum) {
    case IDmDataSet:
        SelectString(1);
        break;
    case IDmDataLoad:
        LoadData(dsName);
    }
    break;
}
break;
case IDMDataLoadSF:
    SelectString(1);
    LoadData(dsName);
    break;
case IDMDataView:
    ShowHide2(dummy,wData,True,0);
    break;
case IDMDataPrint:
    PrintData("ofile.dat");
    break;
}
break;
case IDMenuForm:
switch(itemNum) {
case IDMFormSet:
    SelectString(2);
    break;
case IDMFormLoadSF:
    SelectString(2);
    break;
}
break;
case IDMenuPlot:
switch (itemNum) {
case IDMPlotSet:
    SelectString(3);
    break;
case IDMPlotInc:
    IncDecPlot(True);
    break;
case IDMPlotDec:
    IncDecPlot(False);
    break;
}
break;
case IDMenuWind:
switch (itemNum) {
case 0:
    ShowHide2(dummy,wWindowManager,True,0);
    break;
case 1:
    ShowHide2(dummy,wFile,True,0);
    break;
case 2:
    ShowHide2(dummy,wInput,True,0);
    break;
case 3:
    ShowHide2(dummy,wExecute,True,0);
    break;
case 4:
    ShowHide2(dummy,wData,True,0);
    break;
case 5:
    ShowHide2(dummy,wFormat,True,0);
    break;
case 6:
    ShowHide2(dummy,wFormatAxes,True,0);
    break;
case 7:
    ShowHide2(dummy,wFormatLines,True,0);
    break;
case 8:
    ShowHide2(dummy,wFormatCurveFits,True,0);
    break;
case 9:
    ShowHide2(dummy,wPlotManager,True,0);
    break;
case 10:
void CommandSelect(w, client_data, call_data)
{
    int commNum = (int) client_data;
    char s[64];
    static Boolean up[1] = {False};

    switch (commNum)
    {
    case 0:
        ShowHide2(dummy, wWindowManager, True, 0);
        break;
    case 1:
        ShowHide2(dummy, wFile, True, 0);
        break;
    case 2:
        ShowHide2(dummy, wInput, True, 0);
        break;
    case 3:
        ShowHide2(dummy, wExecute, True, 0);
        break;
    case 4:
        ShowHide2(dummy, wData, True, 0);
        break;
    case 5:
        ShowHide2(dummy, wFormat, True, 0);
        break;
    case 6:
        ShowHide2(dummy, wFormatAxes, True, 0);
        break;
    case 7:
        ShowHide2(dummy, wFormatLines, True, 0);
        break;
    case 8:
        ShowHide2(dummy, wFormatCurveFits, True, 0);
        break;
    case 9:
        ShowHide2(dummy, wPlotManager, True, 0);
        break;
    case 10:
        ShowHide2(dummy, wMotorGeometry, True, 0);
        break;
    case 11:
        ShowHide2(dummy, wMotorConfiguration, True, 0);
        break;
    }
}

void ShowHide2(w, id, show, code)
{
    Widget w; int id; Boolean show; int code;
    char s[64];

    if ((id==0) && (id<8) && (plotExists[id])) {
        if (show) {
            windowUp[id] = True;
            XtPopup(plotShell[id], XtGrabNone);
        } else {

        }
    }
}
switch (id) {
    case wWindowManager:
        if (!windowUp[id]) {
            strcpy(s, "Hide Window Manager");
            XtPopup(commandShell, XtGrabNone);
        } else {
            strcpy(s, "Show Window Manager");
            XtPopdown(commandShell);
        }
        XtVaSetValues(c[0], XtNlabel, s, NULL);
        XtVaSetValues(mEntry[6][0], XtNlabel, s, NULL);
        windowUp[id] = !windowUp[id];
        break;
    case wFile:
        if (!windowUp[id]) {
            strcpy(s, "Hide File Editor");
            XtPopup(fileShell, XtGrabNone);
        } else {
            strcpy(s, "Show File Editor");
            XtPopdown(fileShell);
        }
        XtVaSetValues(c[1], XtNlabel, s, NULL);
        XtVaSetValues(mEntry[6][1], XtNlabel, s, NULL);
        windowUp[id] = !windowUp[id];
        break;
    case wInput:
        if (!windowUp[id]) {
            strcpy(s, "Hide Input File Editor");
            XtPopup(IEShell, XtGrabNone);
        } else {
            strcpy(s, "Show Input File Editor");
            XtPopdown(IEShell);
        }
        XtVaSetValues(c[2], XtNlabel, s, NULL);
        XtVaSetValues(mEntry[6][2], XtNlabel, s, NULL);
        windowUp[id] = !windowUp[id];
        break;
    case wExecute:
        if (!windowUp[id]) {
            strcpy(s, "Hide Execution Setup");
            XtPopup(execShell, XtGrabNone);
        } else {
            strcpy(s, "Show Execution Setup");
            XtPopdown(execShell);
        }
        XtVaSetValues(c[3], XtNlabel, s, NULL);
        XtVaSetValues(mEntry[6][3], XtNlabel, s, NULL);
        windowUp[id] = !windowUp[id];
        break;
    case wData:
        if (!windowUp[id]) {
            strcpy(s, "Hide Current Dataset");
            XtPopup(dataShell, XtGrabNone);
        } else {
            strcpy(s, "Show Current Dataset");
            XtPopdown(dataShell);
        }
        XtVaSetValues(c[4], XtNlabel, s, NULL);
        XtVaSetValues(mEntry[6][4], XtNlabel, s, NULL);
        windowUp[id] = !windowUp[id];
    }
}

windowUp[id] = False;
XtPopdown(plotShell[id]);
break;

case wFormat:
  if (!windowUp[id]) {
    strcpy(s,'Hide Format');
    XtPopup(FEShell,XtGrabNone);
  }
  else {
    strcpy(s,'Show Format');
    XtPopdown(FEShell);
  }
  XtVaSetValues(c[5],XtNlabel,s,NULL);
  XtVaSetValues(mEntry[6][5],XtNlabel,s,NULL);
  windowUp[id]=!windowUp[id];
  break;

case wFormatAxes:
  break;

case wFormatLines:
  if (!windowUp[id]) {
    strcpy(s,'Hide Format Lines');
    XtPopup(DAShell,XtGrabNone);
  }
  else {
    strcpy(s,'Show Format Lines');
    XtPopdown(DAShell);
  }
  XtVaSetValues(c[7],XtNlabel,s,NULL);
  XtVaSetValues(mEntry[6][7],XtNlabel,s,NULL);
  windowUp[id]=!windowUp[id];
  break;

case wFormatCurveFits:
  if (!windowUp[id]) {
    strcpy(s,'Hide Format Curve Fits');
    XtPopup(CFShell,XtGrabNone);
  }
  else {
    strcpy(s,'Show Format Curve Fits');
    XtPopdown(CFShell);
  }
  XtVaSetValues(c[8],XtNlabel,s,NULL);
  XtVaSetValues(mEntry[6][8],XtNlabel,s,NULL);
  windowUp[id]=!windowUp[id];
  break;

case wPlotManager:
  if (!windowUp[id]) {
    strcpy(s,'Hide Plot Manager');
    XtPopup(PMShell,XtGrabNone);
  }
  else {
    strcpy(s,'Show Plot Manager');
    XtPopdown(PMShell);
  }
  XtVaSetValues(c[9],XtNlabel,s,NULL);
  XtVaSetValues(mEntry[6][9],XtNlabel,s,NULL);
  windowUp[id]=!windowUp[id];
  break;

case wMotorGeometry:
  if (!windowUp[id]) {
    strcpy(s,'Hide Motor Geometry');
    XtPopup(MGShell,XtGrabNone);
  }
  else {
    strcpy(s,'Show Motor Geometry');
    XtPopdown(MGShell);
  }
  XtVaSetValues(c[10],XtNlabel,s,NULL);
  XtVaSetValues(mEntry[6][10],XtNlabel,s,NULL);
  windowUp[id]=!windowUp[id];
  break;

case wMotorConfiguration:
if (!windowUp[id]) {
    strcpy(s, "Hide Motor Configuration");
    XtPopup(MGPictShell, XtGrabNone);
} else {
    strcpy(s, "Show Motor Configuration");
    XtPopdown(MGPictShell);
    XtVaSetValues(c[11], XtNlabel, s, NULL);
    XtVaSetValues(mEntry[6][11], XtNlabel, s, NULL);
    windowUp[id] = !windowUp[id];
    break;
}

void VerbCallback(w, client_data, call_data) {
    Widget w; XtPointer client_data, call_data;
    XawListReturnStruct *r = (XawListReturnStruct*) call_data;
    strcpy(cli, (*r).string);
    XtVaSetValues(cliText, XtNstring, cli, NULL);
    XawTextSetInsertionPoint(cliText, strlen(cli));
}

/*

void SFCallback(w, client_data, call_data) {
    Widget w; XtPointer client_data, call_data;
    int action = (int) client_data;
    char filename[32], s[64];
    String f;
    f = XawDialogGetValueString(sfDialog);
    strcpy(filename, f);
    XtPopdown(sfGetFileShell);
    switch (action) {
        case SFCommCancel:
            break;
        case SFCommLoad:
            sprintf(s, "The filename selected was: %s
", filename);
            break;
        case SFCommSave:
            break;
    }
}

void SFGetFile() {
    XtPopup(sfGetFileShell, XtGrabNone);
}
*/

void SelectString(code) {
    int code;
    char s[64], *ss;
    int i;
    XtVaGetValues(menuText, XtNstring, &ss, NULL);
    strncpy(s, ss, 63);
    s[63] = '\0';
    if (s[0] != '\0') {
        switch (code) {
            case 0:
strcpy(inName,s);
break;
case 1:
strcpy(dsName,s);
break;
case 2:
strcpy(pfName,s);
break;
case 3:
i=atoi(s);
if((i>=0)&&(i<=7)&&(plotExists[i])) {
    activePlot=i;
sprintf(plName,"%d",i);
    break;
}
}
SetStatusLabel();
}
void SetStatusLabel()
{
char s[256];
strcpy(s,"I: <>");
strcat(s,inName);
strcat(s,"D: <>");
strcat(s,dsName);
strcat(s,"F: <>");
strcat(s,pfName);
strcat(s,"P: <>");
strcat(s,plName);
strcat(s,"*");
XtVaSetValues(statusLabel,XtNlabel,s=NULL);
}
void FileCallback(w,client_data,call_data)
{
int n = (int) client_data;
FILE *f;
char s[128],*t;
XtVaGetValues(fileEntry[3],XtNstring,&t,NULL);
strcpy(s,t);
XtVaGetValues(fileText,XtNstring,&t,NULL);
switch (n) {
case 0:
    ShowHide2(dummy,wFile,False,0);
break;
case 1:
    f = fopen(s,"w");
    fprintf(f,"%s",t);
    fclose(f);
break;
case 2:
    f = fopen(s,"r");
    if (f!=NULL)
        XtVaSetValues(fileText,XtNstring,s,XtNtype,XawAsciiFile,NULL);
break;
}
void LoadInput()
{
FILE *fi;
char s[128], *t;
XtVaGetValues(fileEntry[3], XtNstring, &t, NULL);
strcpy(s, t);
fi = fopen(s, "r");
if (fi != NULL) {
    XtVaSetValues(fileText, XtNstring, s, XtNtype, XawAsciiFile, NULL);
}

void ExecCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    int n = (int) client_data;
    switch (n) {
    case 0:
        ShowHide2(dummy, wExecute, False, 0);
        break;
    case 1:
        break;
    case 2:
        RunExec();
        break;
    }

void RunExec()
    char s[128], t[128], *u;
    XtVaGetValues(execItem[1][0], XtNstring, &u, NULL);
    strcpy(s, u);
    XtVaGetValues(execItem[1][1], XtNstring, &u, NULL);
    strcpy(t, u);
    strcat(s, t);
    strcat(s, " ");
    XtVaGetValues(execItem[1][2], XtNstring, &u, NULL);
    strcpy(t, u);
    if (strlen(t) != 0) {
        strcat(s, " <");
        strcat(s, t);
    }
    XtVaGetValues(execItem[1][3], XtNstring, &u, NULL);
    strcpy(t, u);
    if (strlen(t) != 0) {
        strcat(s, " >");
        strcat(s, t);
    }
    system(s);
*/
/*
 xpPlots.c
*/

#include "xpIncs.h"
#include "xpDefs.h"
#include "xpwidgets.h"
#include "xpgeneral.h"
#include "xpdata.h"
#include "xpformats.h"
/* $1: Global Variables */
unsigned long lineColor[64];
unsigned long curvColor[32];
int activePlot=-1;
Boolean plotExists[MAXPLOTS];
Boolean useCol[MAXPLOTS][2*MAXCOLS];
Boolean planar[MAXPLOTS]=(True,True,True,True,True,True,True,True);

/* $2: Local variables */
XSegment Grid[MAXPLOTS][50];
XPoint pt[MAXPLOTS][2*MAXCOLS][MAXPLANES];
XPoint pt2[MAXPLOTS][2*MAXCOLS][128];
char xLabels[MAXPLOTS][24][16];
char yLabels[MAXPLOTS][24][16];
char zLabels[MAXPLOTS][24][16];
Dimension oldWidth[MAXPLOTS] = {640,640,640,640,640,640,640,640};
Dimension oldHeight[MAXPLOTS] = {480,480,480,480,480,480,480,480};

/* $3: Global functions */
void MakePlot();
void KillPlot();
void ActivatePlot();
void RedrawPlot();
void RedrawPlot2();
void ResizePlot();
void PMActiveCallback();
void PMCreateCallback();
void PMShowHideCallback();
void PMPlanarCallback();
void PMActionCallback();
void IncDecPlot();

/* $4: Local functions */
void RecreatePlot();
void SetupGrid();
void SetupAxisLabels();
void SetupPoints();
void DrawGrid();
void DrawAxisLabels();
void DrawPlotLabels();
void DrawLegend();
void DrawPoints();
void SmoothAxisValues();
void GESCP();
void poly();
void spline();

/* $5: Implementation */
void PMActiveCallback(w,client_data,call_data)
{ 
  int n = (int) XawToggleGetCurrent(PMitem[0][0]);
  if ((n>=1)&(n<=8))
    activePlot=n-1;
}

C-34
void PMCreateCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        int p = (int) client_data;
        if ((plotExists[p] && (dataLoaded)) {
            XtVaSetValues(PMitem[0][p], XtNsensitive, True, NULL);
            XtVaSetValues(PMitem[1][p], XtNlabel, "Destroy", NULL);
            XtVaSetValues(PMitem[2][p], XtNsensitive, True, XtNlabel, "Hide", NULL);
            MakePlot(p);
            ShowHide2(plotShell[p], p, True, 0);
        } else if (plotExists[p]) {
            KillPlot(p);
            XtVaSetValues(PMitem[0][p], XtNsensitive, False, NULL);
            XtVaSetValues(PMitem[1][p], XtNlabel, "Create", NULL);
            XtVaSetValues(PMitem[2][p], XtNsensitive, False, XtNlabel, "Show", NULL);
        }
    }

void PMShowHideCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        int p = (int) client_data;
        if (windowUp[p]) {
            XtVaSetValues(PMitem[2][p], XtNlabel, "Show", NULL);
            ShowHide2(plotShell[p], p, False, 0);
        } else {
            XtVaSetValues(PMitem[2][p], XtNlabel, "Hide", NULL);
            ShowHide2(plotShell[p], p, True, 0);
        }
    }

void PMPlanarCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        int which = (int) client_data;
        if ((activePlot >= 0) && (activePlot < MAXPLOTS)) {
            if (planar[activePlot]) {
                planar[activePlot] = False;
                XtVaSetValues(PMitem[3][0], XtNlabel, "Plane Plot", NULL);
            } else {
                planar[activePlot] = True;
                XtVaSetValues(PMitem[3][0], XtNlabel, "Space Plot", NULL);
            }
            CopyData(activePlot);
            StatData(activePlot);
            SetupGrid(activePlot);
            SetupAxesLabels(activePlot);
            SetupPoints(activePlot);
            RedrawPlot2(activePlot);
        }
    }

void PMActionCallback(w, client_data, call_data)
    Widget w; XtPointer client_data, call_data;
    {
        int commNum = (int) client_data;
        switch (commNum) {
            case 1:
                IncDecPlot(True);
                break;
            case 2:
                
                C-35
IncDecPlot(False);
break;
case 3:
   if ((activePlot>=0)&&(activePlot<=7)) {
      if (planar[activePlot])
         while (currentPlane[activePlot]<(numPlanes-1))
            IncDecPlot(True);
      else
         while (currentRow[activePlot]<(numRows-1))
            IncDecPlot(True);
   } break;
case 4:
   if ((activePlot>=0)&&(activePlot<=7)) {
      if (planar[activePlot])
         while (currentPlane[activePlot]>0)
            IncDecPlot(False);
      else
         while (currentRow[activePlot]>0)
            IncDecPlot(False);
   } break;
case 5:
   ShowHide2(dummy,wFormat,True,0);
break;
case 6:
   ShowHide2(dummy,wPlotManager,True,0);
break;
}

void MakePlot(p)
int p;
{
   plotExists[p]=True;
   if(dataLoaded) {
      CopyData(p);
      StatInd();
      StatData(p);
      /* PrintPlotData(p); */
      SetupGrid(p);
      SetupAxisLabels(p);
      SetupPoints(p);
      ActivatePlot(p);
   } else
      printf("No dataset currently loaded\n");
}

void KillPlot(p)
int p;
{
   ShowHide2(dummy,wPlot+p,False,0);
   windowUp[wPlot+p]=False;
   plotExists[p]=False;
   if (p==activePlot) {
      XawToggleUnsetCurrent(PMitem[0][0]);
      activePlot=-1;
   }
}

void ActivatePlot(p)
int p;
{
   if((p>=0)&&(p<MAXPLOTS)&&(plotExists[p])) {
      activePlot = p;
      XawToggleSetCurrent(PMitem[0][0],(caddr_t) p+1);
   }
if (planar[activePlot])
    XtVaSetValues(PMitem[0][0], XtNlabel, "Space Plot", NULL);
else
    XtVaSetValues(PMitem[0][0], XtNlabel, "Plane Plot", NULL);
}

void RedrawPlot(Widget w, XEvent *event, String *params, Cardinal *numParams)
{
    Widget w; XEvent *event; String *params; Cardinal *numParams;
    XExposeEvent *myevent = (XExposeEvent*) event;
    Window wind=XtWindow(w);
    static int i=0;
    int p = atoi(params[0]);
    if ((*myevent).count==0) {
        i++;
        XClearWindow(display,wind);
        DrawGrid(p);
        DrawAxisLabels(p);
        DrawPlotLabels(p);
        DrawLegend(p);
        DrawPoints(p);
        printf("RedrawPlot(%d) - #%d
", p, i);
    }
}

void RedrawPlot2(p)
{
    Window wind=XtWindow(plot[p]);
    XClearWindow(display,wind);
    DrawGrid(p);
    DrawAxisLabels(p);
    DrawPlotLabels(p);
    DrawLegend(p);
    DrawPoints(p);
}

void ResizePlot(Widget widget, XEvent *event, String *params, Cardinal *numParams)
{
    Widget widget; XEvent *event; String *params; Cardinal *numParams;
    Dimension w,h;
    int p = atoi(params[0]);
    XtVaGetValues(plotShell[p], XtNwidth,&w,XtNheight,&h,NULL);
    /* printf("ResizePlot(%d) - (%d,%d)
", p, w, h); */
    f[p].width = w;
    f[p].height = h;
    if ((w!=oldWidth[p])|| (h!=oldHeight[p])) {
        RecalcFormat(p);
        SetupGrid(p);
        SetupPoints(p);
    }
    if ((w<=oldWidth[p])&&(h<=oldHeight[p]))
        RedrawPlot2(p);
    oldWidth[p] =w;
    oldHeight[p]=h;
}

void IncDecPlot(inc)
{
    Boolean inc;
    ( Boolean changed=False;
}
if (activePlot>=0)&&(activePlot<MAXPLOTS))
    if (planar[activePlot]) {
        if ((inc)&&(currentPlane[activePlot]<numPlanes-1)) {
            currentPlane[activePlot]++;
            changed=True;
        } else if ((inc)&&(currentPlane[activePlot]>0)) {
            currentPlane[activePlot]--;
            changed=True;
        }
    } else if (inc)&&(currentRow[activePlot]<numRows-1)) {
        currentRow[activePlot]++;
        changed=True;
    } else if ((inc)&&(currentRow[activePlot]>0)) {
        currentRow[activePlot]--;
        changed=True;
    }
}
if (changed) {
    CopyData(activePlot);
    StatData(activePlot);
    PrintPlotData(activePlot);
    SetupGrid(activePlot);
    SetupAxisLabels(activePlot);
    SetupPoints(activePlot);
    RedrawPlot2(activePlot);
}
void RecreatePlot(p)
    int p;
    { CopyData(p);
        StatData(p);
        PrintPlotData(p);
        SetupGrid(p);
        SetupAxisLabels(p);
        SetupPoints(p);
        RedrawPlot2(p);
    }
void SetupGrid(p)
    int p;
    { int dx=f[p].xInc, dy=f[p].yInc;
        int numx=f[p].xNum, numy=f[p].yNum;
        int lengthx=f[p].xLength, lengthy=f[p].yLength;
        int BL=f[p].borderL, BR=f[p].borderR;
        int BT=f[p].borderT, BB=f[p].borderB;
        int i;
        for (i=0;i<=numx;i++) {
            Grid[p][i].xl = BL-i*dy;
            Grid[p][i].x2 = BL+i*dy;
        } for (i=0;i<=numy;i++) {
            Grid[p][i].xl = BL-i*dx;
            Grid[p][i+numx].x2 = BL+i*dx;
        } Grid[p][numx][numy].x1 = BL-1*dy;
        Grid[p][numx][numy].y2 = BT+lengthy+f[p].xTick;
    }
}
void DrawGrid(p)

    int p;

    { Window wind=XtWindow(plot[p]);
        XSetForeground(display,gc[0],rColor[Black].pixel);
        XDrawSegments(display,wind,gc[0],Grid[p].f[p].nGrid);
    }

void SetupAxisLabels(p)

    int p;

    { float temp, minx, miny, maxx, maxy, minz, maxz, rangex, rangey, rangez;

        int numx=f[p].xNum, numy=f[p].yNum;
        int i;
        char s[16], t[16];

        sprintf(s, "%%%sf", f[p].xFormat);
        sprintf(t, "%%%sf", f[p].yFormat);

        if (!f[p].opt[oAutoScale]) {
            minx=f[p].xMin;
            maxx=f[p].xMax;
            rangex=f[p].xMax-f[p].xMin;
            miny=f[p].yMin;
            maxy=f[p].yMax;
            rangey=f[p].yMax-f[p].yMin;
        }
        else {
            if (planar[p]) {
                minx=xMin;
                maxx=xMax;
                rangex=xRange;
            }
            else {
                minz=zMin;
                maxz=zMax;
                rangez=zRange;
            }
            if (planar[p]) {
                for (i=0; i<=numx; i++) {
                    temp=minx+i*rangex/numx;
                    sprintf(xLabels[p][i], s, temp);
                }
                for (i=0; i<=numy; i++) {
                    temp=miny+i*rangey/numy;
                    sprintf(yLabels[p][i], t, temp);
                }
            }
            else {
                for (i=0; i<=numx; i++) {
                    temp=minz+i*rangez/numx;
                    sprintf(xLabels[p][i], s, temp);
                }
                for (i=0; i<=numy; i++) {
                    temp=miny+i*rangey/numy;
                    sprintf(yLabels[p][i], t, temp);
                }
            }
        }
    }
void DrawAxisLabels(p)
{
    Window wind=XtWindow(plot[p]);
    int dx=f[p].xInc, dy=f[p].yInc;
    int maxx=f[p].xNum, maxy=f[p].yNum;
    int i, w, x, y;
    x=f[p].borderL;
    y=f[p].borderT+f[p].yLength+f[p].xTick+12;
    XSetForeground(display, gc[0], rColor[Black].pixel);
    for (i=0; i<=maxx; i++)
    {
        w=XTextWidth(fs6xl0, xLabels[p][i], strlen(xLabels[p][i]))/2;
        XDrawString(display, wind, gc[0], x+i*dx-w, y,
            xLabels[p][i], strlen(xLabels[p][i]));
    }
    x=f[p].borderL-f[p].yTick-6;
    y=f[p].borderT+f[p].yLength;
    for (i=0; i<=maxy; i++)
    {
        w=XTextWidth(fs6xl0, yLabels[p][i], strlen(yLabels[p][i]));
        XDrawString(display, wind, gc[0], x-w, y-i*dy,
            yLabels[p][i], strlen(yLabels[p][i]));
    }
}

void DrawPlotLabels(p)
{
    int p;
    Window wind=XtWindow(plot[p]);
    char t[64];
    int w, n, h, v;
    XSetForeground(display, gc[0], rColor[Black].pixel);
    strcpy(t, f[p].title);
    w=XTextWidth(fsHelv18b, t, strlen(t));
    h=f[p].borderL-f[p].xLength-w)/2;
    v=f[p].borderT-8;
    XDrawString(display, wind, gc[2], h, v, t, strlen(t));
    if (planar[p])
    {
        sprintf(t, "Distance (inch) [Time = \%6.0f : \%3d/%3d]",
            z[currentPlane[p]], currentPlane[p]+1, numPlanes);
    }
    else
    {
        sprintf(t, "Time (msec) [Dist = \%6.0f : \%3d/%3d]",
            x[currentRow[p]], currentRow[p]+1, numRows);
    }
    w = XTextWidth(fsHelv10, t, strlen(t));
    h=f[p].borderL-f[p].xLength-w)/2;
    v=f[p].borderT+f[p].yLength+30;
    XDrawString(display, wind, gc[0], h, v, t, strlen(t));
}

void DrawLegend(p)
{
    int p;
    Window wind=XtWindow(plot[p]);
    int j, k=-1;
    XSegment Legend[l];
    int len=15, xDist=100, yDist=14, x, y;
    test = ( planar[p] ? MAXCOLS : 2*MAXCOLS );
    XSetLineAttributes(display, gc[0], 8, LineSolid, CapButt, JoinRound);
    for (j=0; j<test; j++)
    {
        if (useCol[p][j])
        {
            k++;
            x=f[p].borderL+(k%4)*xDist;
            y=f[p].borderT+f[p].yLength+(k/4)*yDist+45;
            Legend[0].xl=x;
        }
    }
}

C-40
Legend[0].y1=y-4;
Legend[0].x2=x+len;
Legend[0].y2=y-4;
XSetForeground(display, gc[0], rColor[Black].pixel);
XDrawString(display, wind, gc[0], x+len+5, y,
columnNames[j], strlen(columnNames[j]));
XSetForeground(display, gc[0], lineColor[f[p].color[j]]);
XDrawSegments(display, wind, gc[0], Legend, l);
XSetForeground(display, gc[0], rColor[Black].pixel);
XSetLineAttributes(display, gc[0], 0, LineSolid, CapButt, JoinRound);
}
void SetupPoints(p)
int p;
{
int bL=f[p].borderL, bR=f[p].borderR;
int bT=f[p].borderT, bB=f[p].borderB;
float minx=xMin, miny=yMin[p], minz=zMin;
float rangeX=xRange, rangeY=yRange[p], rangeZ=zRange;
int lengthX=f[p].xLength, lengthY=f[p].yLength;
int i, j;
if (!f[p].opt[AutoScale]) {
  minx=f[p].xMin;
  rangeX=f[p].xMax-f[p].xMin;
  minz=f[p].xMin;
  rangeZ=f[p].xMax-f[p].xMin;
  miny=f[p].yMin;
  rangeY=f[p].yMax-f[p].yMin;
}
else {
  if (planar[p]) {
    minx=xMin;
    rangeX=xRange;
  }
  else {
    minz=zMin;
    rangeZ=zRange;
  }
  miny=yMin[p];
  rangeY=yRange[p];
}
if (planar[p]) {
  for(j=0; j<numCols; j++) {
    if (!useCol[p][j])
      for(i=0; i<numRows; i++) {
        pt[p][j][i].x=bL÷((x[i]-minx)/rangeX)*lengthX;
        pt[p][j][i].y=bT÷lengthY-((plotData[p][j][i]-miny)/rangeY)*lengthY;
      }
  }
} else {
  for(j=0; j<2*MAXCOLS; j++) {
    if (!useCol[p][j])
      for(i=0; i<numPlanes; i++) {
        pt[p][j][i].x=bL÷((z[i]-minz)/rangeZ)*lengthX;
        pt[p][j][i].y=bT+lengthY-((plotData[p][j][i]-miny)/rangeY)*lengthY;
      }
  }
}
void DrawPoints(p)
int p;
{
}
Window wind=XtWindow(plot[p]);
static char dashPattern[] = {16,8};
int j,test;
XRectangle r;
int bL=f[p].borderL,bR=f[p].borderR;
int bT=f[p].borderT,bB=f[p].borderB;
int lengthx=f[p].xLength.lengthy=f[p].yLength;
XSetLineAttributes(display,gc[0],4,LineOnOffDash,CapRound,JoinRound);
XSetDashes(display,gc[0],0,dashPattern,2);
r.x=bL-2; r.y=bT-2; r.width=lengthx+4; r.height=lengthy+4;
XSetClipRectangles(display,gc[0],0,0,r,Unsorted);
test=(planar[p] ? MAXCOLS : 2*MAXCOLS);
for(j=0;j<test;j++)
  if (useCol[p][j]) {
    if (f[p].numDashes[j]>0) {
      XSetLineAttributes(display, gc[0],4,LineOnOffDash,CapRound,JoinRound);
      XSetDashes(display, gc[0],0,f[p].dashes[j],f[p].numDashes[j]);
    }
    else
      XSetLineAttributes(display, gc[0],4,LineSolid,CapRound,JoinRound);
    XSetForeground(display,gc[0],Black.pixel);
    if (planar[p])
      XDrawLines(display,wind,gc[0],pt[p][j],numRows,CoordModeOrigin);
    else
      XDrawLines(display,wind,gc[0],pt[p][j],numPlanes,CoordModeOrigin);
  }
XSetForeground(display,gc[0],rColor[Black].pixel);
XSetLineAttributes(display,gc[0],0,LineSolid,CapButt,JoinRound);
XSetClipMask(display,gc[0],None);
}

void SmoothAxisValues(vMin,vMax)
float vMin,vMax;
{
  float vRange;
  int a,a1,a2,b,b1,b2;
  vRange = vMax-vMin;
  a = log10(vRange);
  a1= a-1;
  a2= a-2;
  b = vRange/a;
  printf("%d",b);
}

/*
 * The following routines apply only to curve-fitting (not implemented)
 */

void GESCP(n,a,b)
int n; float a[10][10],b[10];
{
  int i,j,k,p,nrow[10],ncopy;
  float max,n,sum,s[10];
  bL=p;
  for (i=0;i<n;i++)
    if (fabs(a[i][j]) >max) max = fabs(a[i][j]);
  return max;
s[i]=max;
 nrow[i]=i;
}

for (i=0;i<(n-1);i++) {

max=-1.0E+10;

for (j=i+1;j<n++)

if ( fabs(a[nrow[j]][i])/s[nrow[j]] ) > max ) {

max = fabs(a[nrow[j]][i])/s[nrow[j]];

p=j;
}

if (a[nrow[p]][i]==0.0) printf("No unique solution exists\n");

if (nrow[i] == nrow[p]) {

ncopy=nrow[i];

nrow[i]=nrow[p];

nrow[p]=ncopy;
}

if (j=i+1;j<n;j++) {

m = a[nrow[j]][i]/a[nrow[i]][i];

for (k=0;k<(n+1);k++)

a[nrow[j]][k]=a[nrow[j]][k]-m*a[nrow[i]][k];

}

if (a[nrow[n-l]][n-l]==0.0) printf("No unique solution exists\n");

b[n-l]=a[nrow[n-l]][n]/a[nrow[n-l]][n-l];

for (i=(n-2);i>=0;i--) {

sum=0;

for (j=i+1;j<n;j++)

sum=sum+a[nrow[i]][j]*b[j];

b[i] = (a[nrow[i]][n]-sum)/a[nrow[i]][i];
}

void poly(n,m,x,y)

int n,m; float x[],y[];

{

int i,j,k;

float a[10][10],b[10],p[256],e[256],sum;

for (i=0;i<n;i++) {

for (j=0;j<m;j++)

sum=0;

for (k=0;k<m;k++)

sum=sum+(i+j)==0 ? 1.0 : (float) pow((double) x[k],(double) i+j)

}

if ( ((i+j)==0) sum=m;

a[i][j]=sum;
}

for (i=0;i<n;i++) {

sum=0;

for (k=0;k<m;k++)

sum=sum+y[k]*(i==0 ? 1.0 : (float) pow((double) x[k],(double) i));

a[i][n]=sum;
}

GESCP(n,a,b);

for (i=0;i<n;i++)

printf("x%d = %12.6f\n",i,b[i]);

for (i=0;i<m;i++) {

sum=0;

for (k=0;k<n;k++)

sum=sum+b[k]*(k==0 ? 1.0 : (float) pow((double) x[i],(double) k));

p[i]=sum;

e[i]=y[i]-p[i];

printf("%12.6f %12.6f%12.6f%12.6f\n",x[i],y[i],p[i],e[i]);
}

sum=0;

for (i=0;i<m;i++)

sum+=(float) pow((double) e[i],(double) 2);

printf("\n\nTotal error = %12.6f\n",sum);

}

C-43
void spline(n,x,y)
  int n; float x[ ],y[ ];
  {
    int i,j;
    float a[256],b[256],c[256],d[256],
          h[256],alpha[256],l[256],u[256],z[256];

    for (i=0;i<=n;i++)
      a[i]=y[i],
      for (i=0;i<n;i++)
        h[i]=x[i+1]-x[i];
    for (i=1;i<n;i++)
      alpha [i]=3*(a[i+1]*h[i-1]-a[i] *(x[i+1]-x[i-1])+a[i-1]*h[i])/
        (h[i-1]*h[i]);
    l[0]=1;
    u[0]=0;
    z[0]=0;
    for (i=1;i<n;i++)
      l[i]=2*(x[i+1]-x[i-1])h[i-1]*u[i-1];
    u[i]=h[i]/l[i];
    z[i]=(alpha [i]-h[i-1]*z[i-1])/l[i];
  }

  l[n]=l[0];
  z[n]=0.0;
  c[n]=0.0;
  for (j=(n-1);j>=0;j--) {
    c[j]=z[j]-u[j]*c[j+1];
    b[j]=(a[j+1]-a[j])h[j]*(c[j+1]+2*c[j])/3;
    d[j]=(c[j+1]-c[j])/(3*h[j]);
  }

/*
 ------------------------------
 xpWidgets.c
 ------------------------------
 */
#include "xpIncs.h"
#include "xpDefs.h"

/* The following files have to be included for callback routines */
#include "xpmenus.h"
#include "xpcommands.h"
#include "xpinput.h"
#include "xpfmforms.h"
#include "xpfmplots.h"
#include "xpgeometry.h"

/* $1: Global variables */

Widget topLevel;
Widget menuPane,menuText;
Widget statusLabel;
Widget verbViewport,verbList,commViewport,commList;
Widget cliPane,cliLabel,cliText;
Widget mMenuButton[NUMMenus],mSimpleMenu[NUMMenus];
Widget mEntry[NUMMenus][MAXENTRIES];
Widget commandShell,commandBox,c[NUMCommands];
Widget PMShell,PMForm,PMBox[4],PMTop[4],PMitem[4][8];
Widget infoShell,infoCore;
Widget logShell, logText;

Widget fileShell, filePane, fileMenuPane, fileText, fileEntry[4];

Widget EShell, EForm, EBox[8], ETop[8], EItem[8][2],
    IEViewPort, IEList;

Widget DAShell, DAForm, DABox[5], DATop[5], DAItem[5][16],
    DLabel[2];

Widget CFShell, CFForm, CFBox[4], CFTop[4], CFItem[4][16];

Widget MGPictShell, MGPict;

Widget FAShell, FAForm, FABox[5], FATop[5], FAItem[5][16];

Widget execShell, execForm, execBox[4], execTop[4], execItem[4][8];

Widget plotShell[MAXPLOTS], plot[MAXPLOTS];

Widget dataShell, dataText;

Widget formatShell, formatText;

Widget FEShell, FEForm, FEBox[16], FETop[16], FEItem[16][16];

Entry, dummy;

Widget sfGetFileShell, sfDialog, sfCancel, sfLoad, sfSave;

/* $2: Local variables */

static char mMenuButtonNames[][12] = {
    "mBarFile", "mBarInpu", "mBarExec", "mBarData",
    "mBarForm", "mBarPlot", "mBarWind", 
};

static char mSimpleMenuNames[][12] = {
    "menuFile", "menuInpu", "menuExec", "menuData",
    "menuForm", "menuPlot", "menuWind", 
};

static char mEntryNames[NUMMenus][12][24] = {
    [ "mFile_00", "mFile_01", "mFile_02", "mFile_03",
    "mFile_04", "mFile_05", "mFile_06", "mFile_07",
    [ "mFile_08" ],
    [ "mInpu_00", "mInpu_01", "mInpu_02", "mInpu_03", "mInpu_04",
    "mInpu_05", "mInpu_06", "mInpu_07", "mInpu_08",
    [ "mInpu_09" ],
    [ "mExec_00", "mExec_01", "mExec_02", "mExec_03",
    "mData_00", "mData_01", "mData_02", "mData_03",
    "mData_04", "mData_05", "mData_06", "mData_07",
    "mData_08", "mData_09", "mData_10", "mData_11",
    [ "mData_12", "mData_13", "mData_14", "mData_15",
    [ "mData_16", "mData_17", "mData_18", "mData_19",
    [ "mData_20", "mData_21", "mData_22", "mData_23",
    [ "mData_24" ],
    [ "mForm_00", "mForm_01", "mForm_02", "mForm_03",
    "mForm_04", "mForm_05", "mForm_06", "mForm_07",
    "mForm_08", "mForm_09", "mForm_10", "mForm_11",
    [ "mForm_12", "mForm_13", "mForm_14", "mForm_15",
    [ "mForm_16", "mForm_17", "mForm_18", "mForm_19",
    [ "mForm_20", "mForm_21", "mForm_22", "mForm_23",
    [ "mForm_24" ],
    [ "mPlot_00", "mPlot_01", "mPlot_02", "mPlot_03",
    "mPlot_04", "mPlot_05", "mPlot_06", "mPlot_07",
    "mPlot_08", "mPlot_09", "mPlot_10", "mPlot_11",
    [ "mPlot_12", "mPlot_13", "mPlot_14", "mPlot_15",
    [ "mPlot_16", "mPlot_17", "mPlot_18", "mPlot_19",
    [ "mPlot_20", "mPlot_21", "mPlot_22", "mPlot_23",
    [ "mPlot_24" ],
    [ "mWind_00", "mWind_01", "mWind_02", "mWind_03",
    "mWind_04", "mWind_05", "mWind_06", "mWind_07",
    "mWind_08", "mWind_09", "mWind_10", "mWind_11",
    [ "mWind_12", "mWind_13", "mWind_14", "mWind_15",
    [ "mWind_16", "mWind_17", "mWind_18", "mWind_19",
    [ "mWind_20", "mWind_21", "mWind_22", "mWind_23",
    [ "mWind_24" ] ],

static int numMenuEntries[NUMMenus] = {
    NUMFileEntries, NUMInpuEntries, NUMExecEntries, NUMDataEntries,
    NUMFormEntries, NUMPlotEntries, NUMWindEntries 
};

static String commElements[] = {
    "active", "change", "color", "data", "decrement", "edit", "execute", "fit",
    "format", "hide", "increment", "input", "kill", "list", "load", "make",
    "NULL"};

static String verbs[] = {
    "active", "change", "color", "data", "decrement", "edit", "execute", "fit",
    "format", "hide", "increment", "input", "kill", "list", "load", "make",
static String IEentries[] = {
    "plane", "plot", "print", "quit", "save",
    "set", "show", "space", "view", NULL};

#define NUMFEOptions 2
#define NUMFECommandString 4
#define NUMFEAxesString 4

static char s[64], t[64];

$3: Global functions */

void CreateWidgets();

$4: Local functions

$5: Implementation ===============================================

void CreateWidgets()
{
    static char mMenuButtonNames[][10] = {
        "mBarFile", "mBarInput", "mBarExec", "mBarData",
        "mBarForm", "mBarPlot", "mBarWind", 
    };
    static char mSimpleMenuNames[][10] = {
        "menuFile", "menuInput", "menuExec", "menuData",
        "menuForm", "menuPlot", "menuWind", 
    };
    static char mEntryNames[NUMMenus][12][12] = {
        { "mFile_00", "mFile_01", "mFile_02", "mFile_03", "mFile_04", "mFile_05", "mFile_06", "mFile_07", "mFile_08", "mFile_09", "mFile_10", "mFile_11" },
        { "mData_00", "mData_01", "mData_02", "mData_03", "mData_04", "mData_05", "mData_06", "mData_07", "mData_08", "mData_09", "mData_10", "mData_11" },
        { "mForm_00", "mForm_01", "mForm_02", "mForm_03", "mForm_04", "mForm_05", "mForm_06", "mForm_07", "mForm_08", "mForm_09", "mForm_10", "mForm_11" },
        { "mPlot_00", "mPlot_01", "mPlot_02", "mPlot_03", "mPlot_04", "mPlot_05", "mPlot_06", "mPlot_07", "mPlot_08", "mPlot_09", "mPlot_10", "mPlot_11" },
        { "mWind_00", "mWind_01", "mWind_02", "mWind_03", "mWind_04", "mWind_05", "mWind_06", "mWind_07", "mWind_08", "mWind_09", "mWind_10", "mWind_11" },
    };
    static int numMenuEntries[NUMMenus] = {
        NUMFileEntries, NUMInputEntries, NUMExecEntries, NUMDataEntries,
        NUMFormEntries, NUMPlotEntries, NUMWindEntries 
    };
    static String commElements[] = {
        "active", "change", "color", "data", "decrement", "edit", "execute", "fit",
        "format", "hide", "increment", "input", "kill", "list", "load", "make",
        "plane", "plot", "print", "quit", "save", "set",
        "show", "space", "view", NULL);
    static String IEentries[] = {
        "", NULL};
    int i, j, total;

    /* Generate the pane that will serve as the root pane for the window */
    rootPane = XtVaCreateManagedWidget("rootPane", panedWidgetClass, topLevel, 
           XtNorientation, XtorientsVertical, 
           NULL);

    C-46
/* A pane to hold all of the menu bar entries */
menuPane = XtVaCreateManagedWidget("menuPane",pinnedWidgetClass,rootPane,
    XtOrientation,XtorientHorizontal,
    XtNshowGrip,False,
    NULL);

/* Generate the menu buttons, placing them one after another */
for(i=0;i<NUMMenus;i++)
mMenuButton[i] = XtVaCreateManagedWidget(
    mMenuButtonNames[i],menuButtonWidgetClass,menuPane,
    XtNborderWidth,0,
    XtNshowGrip,False,
    XtNmenuName,mSimpleMenuNames[i],
    NULL);

for(j=0;j<NUMMenus;j++)
    mSimpleMenu[j] = XtVaCreatePopupShell(
        mSimpleMenuNames[j],simpleMenuWidgetClass,mMenuButton[j],
        NULL);

for(i=0;i<numMenuEntries[j];i++)
    if (mEntryNames[j][i][0]=='/')
        mEntry[j][i] = XtVaCreateManagedWidget(
            mEntryNames[j][i],smeLineObjectClass,mSimpleMenu[j],
            NULL);
    else {
        mEntry[j][i] = XtVaCreateManagedWidget(
            mEntryNames[j][i],smeBSBObjectClass,mSimpleMenu[j],
            NULL);
        XtAddCallback(mEntry[j][i],XtNcallback,MenuSelect,
            (XtPointer) (j*H_WORD+i));
    }

menuText = XtVaCreateManagedWidget(
    "menuText",asciiTextWidgetClass,menuPane,
    XtNborderWidth,0,
    XtNshowGrip,False,
    XtNeditType,XawtextEdit,
    NULL);

statusLabel = XtVaCreateManagedWidget(
    "statusLabel",labelWidgetClass,rootPane,
    XtNwidth,600,
    XtNshowGrip,False,
    XtNskipAdjust,True,
    NULL);

commPane = XtVaCreateManagedWidget(
    "commPane",pinnedWidgetClass,rootPane,
    XtOrientation,XtorientHorizontal,
    XtNshowGrip,False,
    NULL);

verbViewport = XtVaCreateManagedWidget(
    "verbViewport",viewportWidgetClass,commPane,
    XtNmin,100,
    XtNallowVert,True,
    XtNforceBars,True,
    XtNshowGrip,False,
    NULL);

verbList = XtVaCreateManagedWidget(
    "verbList",listWidgetClass,verbViewport,
    XtNverticalList,True,
    XtNcolumnSpacing,0,
    XtNrowSpacing,2,
    XtNdefaultColumns,1,
    XtNforceColumns,True,
    XtNlist,verbs,
    NULL);

XtAddCallback(verbList,XtNcallback,VerbCallback,0);

commViewport = XtVaCreateManagedWidget(}
/* Create the widgets used to make the command palette work */
commandShell = XtVaCreatePopupShell("commandShell", transientShellWidgetClass, topLevel,
XNtitle, "Window Manager", NULL);
for (i=0; i<12; i++) {
    sprintf(s, "c%02d", i);
c[i] = XtVaCreateManagedWidget("c[i]", commandWidgetClass, commandBox,
XNborderWidth, 0,
XNwidth, 200,
XNresize, False,
XNjustify, XtJustifyLeft,
NULL);
    XtAddCallback(c[i], XtNcallback, CommandSelect, (XtPointer) i);
}

/* Create the widgets used to make the command line log work */
logShell = XtVaCreatePopupShell("logShell", transientShellWidgetClass, topLevel,
XNtitle, "Command Line Log", NULL);
logText = XtVaCreateManagedWidget("logText", asciiTextWidgetClass, logShell,
XNwidth, 480,
XNheight, 240,
XNeditType, XawtextAppend,
XNscrollHorizontal, XawtextScrollAlways,
XNshowGrip, False,
NULL);
*/
/* Create the widgets used to view plots (up to MAXPLOTS=8) */

for(i=0;i<MAXPLOTS;i++) {
    char name[64], label[64];
    sprintf(name, "plotShell_%d", i);
    sprintf(label, "xplo# %d", i);
    plotShell[i] = XtVaCreatePopupShell(  
        name, transientShellWidgetClass, topLevel,  
        XtNminWidth, 320,  
        XtNminHeight, 240,  
        XtNsaveUnder, false,  
        XtNtitle, label,  
        NULL);
    sprintf(name, "plot%d", i);
    plot[i] = XtVaCreateManagedWidget(  
        name, widgetClass, plotShell[i],  
        XtNwidth, 640,  
        XtNheight, 480,  
        NULL);
}

/* Create the widgets used to view the current dataset */
dataShell = XtVaCreatePopupShell(  
    "dataShell", transientShellWidgetClass, topLevel,  
    XtNtitle, "Dataset",  
    NULL);
dataText = XtVaCreateManagedWidget(  
    "dataText", asciiTextWidgetClass, dataShell,  
    XtNwidth, 480,  
    XtNheight, 240,  
    XtNeditType, XawTextRead,  
    XtNtype, XawAsciiFile,  
    XtNscrollVertical, XawTextScrollAlways,  
    XtNstring, "XPlot.dat",  
    NULL);

/* Create the widgets used for a standard get|put file dialog */
sfGetFileShell = XtVaCreatePopupShell(  
    "sfGetFileShell", transientShellWidgetClass, topLevel,  
    XtNtitle, "Filename Dialog",  
    NULL);
sfDialog = XtVaCreateManagedWidget(  
    "sfDialog", dialogWidgetClass, sfGetFileShell,  
    XtNlabel, "Load Dataset (Named):",  
    XtNvalue, "",  
    NULL);
sfCancel = XtVaCreateManagedWidget(  
    "sfCancel", commandWidgetClass, sfDialog,  
    XtNlabel, "Cancel",  
    NULL);
sfLoad = XtVaCreateManagedWidget(  
    "sfLoad", commandWidgetClass, sfDialog,  
    XtNlabel, "Load",  
    NULL);
sfSave = XtVaCreateManagedWidget(  
    "sfSave", commandWidgetClass, sfDialog,  
    XtNlabel, "Save",  
    NULL);
XtAddCallback(sfCancel, XtNcallback, SFCallback, (XtPointer) 0);
XtAddCallback(sfLoad, XtNcallback, SFCallback, (XtPointer) 1);
XtAddCallback(sfSave, XtNcallback, SFCallback, (XtPointer) 2);

/* Create the widgets used in the dialog to edit inputs */
IEShell = XtVaCreatePopupShell(  
    "IEShell", transientShellWidgetClass, topLevel,  
    XtNtitle, "Input File Editor",  
    NULL);

C-49
IEForm = XtVaCreateManagedWidget("IEForm", formWidgetClass, IEShell, NULL);
for (i=0; i<8; i++) {
    sprintf(s, "IEBox_%d", i);
    IEBox[i] = XtVaCreateManagedWidget(s, boxWidgetClass, IEForm,
        XtNhSpace, 1,
        XtNvSpace, 1,
        XtNorientation, XtorientHorizontal,
        NULL);
    sprintf(s, "IETop_%d", i);
    IETop[i] = XtVaCreateManagedWidget(s, labelWidgetClass, IEBox[i],
        XtNresize, False,
        NULL);
}
for (i=0; i<6; i++) {
    sprintf(s, "IEItem_%d_0", i);
    if ((i<3) || (i==5))
        IEItem[i][0] = XtVaCreateManagedWidget(s, asciiTextWidgetClass, IEBox[i],
            XtNdisplayCaret, False,
            NULL);
    else
        IEItem[i][0] = XtVaCreateManagedWidget(s, asciiTextWidgetClass, IEBox[i],
            XtNeditType, XawtextEdit,
            NULL);
}
for (i=0; i<6; i++) {
    sprintf(s, "IEItem_6_%d", i);
    IEItem[6][i] = XtVaCreateManagedWidget(s, commandWidgetClass, IEBox[6],
        NULL);
    XtAddCallback(IEItem[6][i], XtNcallback, IECallback,(XtPointer) i);
}
IEViewport = XtVaCreateManagedWidget("IEViewport", viewportWidgetClass, IEBox[7],
    XtNallowVert, True,
    XtNforceBars, True,
    XtNtop, XtChainTop, XtNfromHoriz, IEBox[0],
    NULL);
IEList = XtVaCreateManagedWidget("IEList", listWidgetClass, IEViewport,
    XtNverticalList, True,
    XtNcolumnSpacing, 0,
    XtNrowSpacing, 2,
    XtNdefaultColumns, 1,
    XtNforceColumns, True,
    XtNlist, IEentries,
    NULL);
    XtAddCallback(IEList, XtNcallback, IEListCallback, 0);
    XtVaSetValues(IEBox[0], XtNleft, XtChainLeft, XtNtop, XtChainTop, NULL);
    for (i=1; i<7; i++)
        XtVaSetValues(IEBox[i], XtNfromVert, IEBox[i-1],
            XtNleft, XtChainLeft, NULL);
    XtVaSetValues(IEBox[7], XtNfromHoriz, IEBox[0], XtNtop, XtChainTop,
        XtNorientation, XtorientVertical, NULL);
    /* _Create the widgets used in the dialog for the colors and hashes */
    DAShell = XtVaCreatePopupShell("DAShell", transientShellWidgetClass, topLevel,
/* Create the widgets used in the dialog to edit formats */

FEShell = XvCreatePopupShell("
  "FEShell", transientShellWidgetClass, topLevel,
  "XtNtitle", "Format Editor",
  NULL);  
FEForm = XvCreateManagedWidget("FEForm", formWidgetClass, FEShell,
  NULL);  
for (i=0;i<10;i++) {  
sprintf(s, "FEBox_%d", i);  
FEBox[i] = XvCreateManagedWidget(s, boxWidgetClass, FEForm,
  XtNhSpace, 1,
  XtNvSpace, 1,
  XtNorientation, XtorientVertical,
  XtNtop, XchainTop,
  NULL);  
sprintf(s, "FETop_%d", i);  
FETop[i] = XvCreateManagedWidget(s, labelWidgetClass, FEBox[i],
  XtNresize, False,
  NULL);  
}  
/* Create the widgets used in the dialog to edit formats */

XvCreateManagedWidget("DAForm", formWidgetClass, DAShell,
  NULL);  
DAForm = XvCreateManagedWidget("DAForm", formWidgetClass, DAShell,
  NULL);  
/* DAForm */  
DALabel[0] = XvCreateManagedWidget("DALabel_0", labelWidgetClass, DAForm,
  XtNfromHoriz, DABox[0],
  NULL);  
DALabel[1] = XvCreateManagedWidget("DALabel_1", labelWidgetClass, DAForm,
  XtNfromHoriz, DALabel[0],
  NULL);  
for (i=0;i<5;i++) {  
sprintf(s, "DABox_%d", i);  
DABox[i] = XvCreateManagedWidget(s, boxWidgetClass, DAForm,
  XtNhSpace, 1,
  XtNvSpace, 1,
  XtNorientation, XtorientVertical,
  XtNtop, XchainTop,
  NULL);  
sprintf(s, "DATop_%d", i);  
DATop[i] = XvCreateManagedWidget(s, labelWidgetClass, DABox[i],
  XtNresize, False,
  NULL);  
}  
for (i=1;i<5;i++)  
  XvSetValues(DABox[i], XtNfromHoriz, DABox[i-1], NULL);  
for (i=0;i<16;i++) {  
sprintf(s, "DAItem_0_%02d", i);  
DAItem[0][i] = XvCreateManagedWidget(s, labelWidgetClass, DABox[0],
  XtNresize, False,
  NULL);  
}  
for (j=1;j<5;j++)  
  for (i=0;i<16;i++) {  
sprintf(s, "DAItem_%d_%02d", j, i);  
  DAItem[j][i] = XvCreateManagedWidget(s, asciiTextWidgetClass, DABox[j],
  XtNeditType, XawtextEdit,
  NULL);  
  }  
*/
XtVaSetValues(FEBox[9], XtNleft, XtChainLeft, XtNtop, XtChainTop,
XtNorientation, XtorientHorizontal, NULL);
XtVaSetValues(FEBox[0], XtNleft, XtChainLeft, XtNfromVert, FEBox[9],
XtNorientation, XtorientHorizontal, NULL);
XtVaSetValues(FEBox[1], XtNleft, XtChainLeft, NULL);
XtVaSetValues(FEBox[5], XtNleft, XtChainLeft, NULL);
for (i=1;i<5;i++)
  XtVaSetValues(FEBox[i], XtNfromVert, FEBox[0],
  XtNfromHoriz, FEBox[i-1], NULL);
for (i=5;i<9;i++)
  XtVaSetValues(FEBox[i], XtNfromVert, FEBox[1],
  XtNfromHoriz, FEBox[i-1], NULL);
XtVaSetValues(FEBox[1], XtNfromHoriz, NULL, NULL);
XtVaSetValues(FEBox[5], XtNfromHoriz, NULL, NULL);

FEItem[9][0] = XtVaCreateManagedWidget(
  "FEItem_9_0", asciiTextWidgetClass, FEBox[9],
  XtNeditType, XawtextEdit,
  NULL);
FEItem[0][0] = XtVaCreateManagedWidget(
  "FEItem_0_0", asciiTextWidgetClass, FEBox[0],
  XtNeditType, XawtextEdit,
  NULL);
for (i=0;i<8;i++)
  sprintf(s, "FEItem_1_%d", i);
  FEItem[1][i] = XtVaCreateManagedWidget(
    s, toggleWidgetClass, FEBox[1],
    XtNresize, False,
    XtNsensitive, False,
    XtNlabel, "",
    NULL);
for (i=0;i<8;i++)
  sprintf(s, "FEItem_2_%d", i);
  FEItem[2][i] = XtVaCreateManagedWidget(
    s, toggleWidgetClass, FEBox[2],
    XtNresize, False,
    XtNsensitive, False,
    XtNlabel, "",
    NULL);
for (i=0;i<8;i++)
  sprintf(s, "FEItem_3_%d", i);
  FEItem[3][i] = XtVaCreateManagedWidget(
    s, toggleWidgetClass, FEBox[3],
    XtNresize, False,
    NULL);
for (i=0;i<3;i++)
  sprintf(s, "FEItem_4_%d", i);
  FEItem[4][i] = XtVaCreateManagedWidget(
    s, commandWidgetClass, FEBox[4],
    NULL);
  XtAddCallback(FEItem[4][i], XtNcallback, FEPageCallback, (XtPointer) i);
for (i=0;i<7;i++)
  sprintf(s, "FEItem_5_%d", i);
  FEItem[5][i] = XtVaCreateManagedWidget(
    s, labelWidgetClass, FEBox[5],
    XtNjustify, XtJustifyRight,
    NULL);
for (i=0;i<7;i++)
  
C-52
sprintf(s,"FEItem_6_%d",i);

FEItem[6][i] = XtVaCreateManagedWidget(s,asciiTextWidgetClass,FBox[6],
XtNeditType,XawtextEdit,
NULL);

} for (i=0;i<7;i++) {
    sprintf(s,"FEItem_7_%d",i);
    FEItem[7][i] = XtVaCreateManagedWidget(s,asciiTextWidgetClass,FBox[7],
XtNeditType,XawtextEdit,
NULL);

} for (i=1;i<6;i++) {
    sprintf(s,"FEItem_8_%d",i);
    FEItem[8][i] = XtVaCreateManagedWidget(s,asciiTextWidgetClass,FBox[8],
XtNeditType,XawtextEdit,
NULL);

} for (i=0;i<8;i++) {
    sprintf(s,"PMBox_%d",i);
    PMBox[i] = XtVaCreateManagedWidget(s,boxWidgetClass,PMForm,
XtNhSpace,1,
XtNvSpace,1,
XtNtop,XtChainTop,
NULL);
    sprintf(s,"PMTop_%d",i);
    PMTop[i] = XtVaCreateManagedWidget(s,labelWidgetClass,PMBox[i],
NULL);
    XtVaSetValues(PMBox[0],XtNleft,XtChainLeft,NULL);
    for (i=1;i<4;i++)
        XtVaSetValues(PMBox[i],XtNfromHoriz,PMBox[i-1],NULL);
    for (i=0;i<8;i++) {
        sprintf(s,"PMToggle%d",i);
        PMitem[0][i] = XtVaCreateManagedWidget(s,toggleWidgetClass,PMBox[0],
XtNsensitive,False,
XtNradioData,(caddr_t)i+1,
XtNradioGroup,PMitem[0][0],
XtNlabel,s,
XtNcallback,PMActiveCallback,(XtPointer)i);
        if (i!=0)
            XawToggleChangeRadioGroup(PMitem[0][i],PMitem[0][0]);
    }
*/

PMShell = XtVaCreatePopupShell("PMShell",transientShellWidgetClass,topLevel,
XtNtitle,"Plot Manager",
NULL);
PMForm = XtVaCreateManagedWidget("PMForm",formWidgetClass,PMShell,
NULL);
for (i=0;i<4;i++) {
    sprintf(s,"PMBox_%d",i);
    PMBox[i] = XtVaCreateManagedWidget(s,boxWidgetClass,PMForm,
XtNhSpace,1,
XtNvSpace,1,
XtNtop,XtChainTop,
NULL);
    sprintf(s,"PMTop_%d",i);
    PMTop[i] = XtVaCreateManagedWidget(s,labelWidgetClass,PMBox[i],
NULL);
    XtVaSetValues(PMBox[0],XtNleft,XtChainLeft,NULL);
    for (i=1;i<4;i++)
        XtVaSetValues(PMBox[i],XtNfromHoriz,PMBox[i-1],NULL);
    for (i=0;i<8;i++) {
        sprintf(s,"PMToggle%d",i);
        PMitem[0][i] = XtVaCreateManagedWidget(s,toggleWidgetClass,PMBox[0],
XtNsensitive,False,
XtNradioData,(caddr_t)i+1,
XtNradioGroup,PMitem[0][0],
XtNlabel,t,
XtNcallback,PMActiveCallback,(XtPolnter)i);
        if (i!=0)
            XawToggleChangeRadioGroup(PMitem[0][i],PMitem[0][0]);
    }
*/

C-53
for (i=0; i<8; i++) {
    PMitem[1][i] = XtVaCreateManagedWidget("entry", commandWidgetClass, PMBox[1],
        XtNresize, False,
        XtNsensitive, False,
        XtNlabel, "Create",
        NULL);
    XtAddCallback(PMitem[1][i], XtNcallback, PMCreateCallback, (XtPointer) i);
}
for (i=0; i<8; i++) {
    PMitem[2][i] = XtVaCreateManagedWidget("entry", commandWidgetClass, PMBox[2],
        XtNsensitive, False,
        XtNresize, False,
        XtNlabel, "Show",
        NULL);
    XtAddCallback(PMitem[2][i], XtNcallback, PMSHOWHiDECallback, (XtPointer) i);
}
for (i=0; i<l; i++) {
    sprintf(s, "PMitem_3_%d", i);
    PMitem[3][i] = XtVaCreateManagedWidget(s, commandWidgetClass, PMBox[3],
        XtNresize, False,
        NULL);
    XtAddCallback(PMitem[3][i], XtNcallback, PMPlanarCallback, 0);
}
for (i=l; i<7; i++) {
    sprintf(s, "PMitem_3_%d", i);
    PMitem[3][i] = XtVaCreateManagedWidget(s, commandWidgetClass, PMBox[3],
        XtNsensitive, False,
        NULL);
    XtAddCallback(PMitem[3][i], XtNcallback, PMActionCallback, (XtPointer) i);
}
execShell = XtVaCreatePopupShell("execShell", transientShellWidgetClass, topLevel,
    XtNtitle, "Execution Summary",
    NULL);
execForm = XtVaCreateManagedWidget("execForm", formWidgetClass, execShell,
    NULL);
for (i=0; i<3; i++) {
    sprintf(s, "execBox_%d", i);
    execBox[i] = XtVaCreateManagedWidget(s, boxWidgetClass, execForm,
        XtNhSpace, l,
        XtNvSpace, l,
        XtNtop, XtChainTop,
        NULL);
    sprintf(s, "execTop_%d", i);
    execTop[i] = XtVaCreateManagedWidget(s, labelWidgetClass, execBox[i],
        NULL);
    XtVaSetValues(execBox[i], XtNfromHoriz, execBox[i-l], NULL);
}
for (i=1; i<3; i++)
    XtVaSetValues(execBox[i], XtNfromHoriz, execBox[i-1], NULL);
for (i=0; i<4; i++) {
    sprintf(s, "execItem_0_%d", i);
    execItem[0][i] = XtVaCreateManagedWidget(s, labelWidgetClass, execBox[0],
        XtNsensitive, False,
        XtNjustify, XtJustifyRight,
        NULL);
for (i=0;i<4;i++) {
    sprintf(s,'execItem__1_%d',i);
    execItem[1][i] = XtVaCreateManagedWidget(
        s, asciiTextWidgetClass, execBox[1],
        XtNeditType, XawTextEdit,
        NULL);
}

for (i=0;i<3;i++) {
    sprintf(s,'execItem_2_%d',i);
    execItem[2][i] = XtVaCreateManagedWidget(
        s, commandWidgetClass, execBox[2],
        XtNresize, False,
        XtNjustify, XtJustifyCenter,
        NULL);
    XtAddCallback(execItem[2][i], XtNcallback, ExecCallback, (XtPointer) i);
}

fileShell = XtVaCreatePopupShell(
    "fileShell", transientShellWidgetClass, topLevel,
    XtNtitle, "Input Data",
    NULL);

filePane = XtVaCreateManagedWidget(
    "filePane", panedWidgetClass, fileShell,
    NULL);

fileMenuPane = XtVaCreateManagedWidget(
    "fileMenuPane", panedWidgetClass, filePane,
    XtNorientation, XtorientHorizontal,
    XtNshowGrip, False,
    XtNskipAdjust, True,
    NULL);

fileEntry[0] = XtVaCreateManagedWidget(
    "fileEntry_0", commandWidgetClass, fileMenuPane,
    XtNlabel, "Done",
    XtNshowGrip, False,
    NULL);

fileEntry[1] = XtVaCreateManagedWidget(
    "fileEntry_1", commandWidgetClass, fileMenuPane,
    XtNlabel, "Save",
    XtNshowGrip, False,
    NULL);

fileEntry[2] = XtVaCreateManagedWidget(
    "fileEntry_2", commandWidgetClass, fileMenuPane,
    XtNlabel, "Load",
    XtNshowGrip, False,
    NULL);

for (i=0;i<3;i++)
    XtAddCallback(fileEntry[i], XtNcallback, FileCallback, (XtPointer) i);

 MGShell = XtVaCreatePopupShell(
    "MGShell", transientShellWidgetClass, topLevel,
    XtNtitle, "Solid Rocket Motor Geometry",
    NULL);

MGForm = XtVaCreateManagedWidget(
    "MGForm", panedWidgetClass, MGShell,
    XtNlabel, "Motor Geometry",
    NULL);
for (i=0;i<6;i++) {
    sprintf(s, "MGBox_%d", i);
    MGBox[i] = XtVaCreateManagedWidget(
        s, boxWidgetClass, MGForm,
        XtNhSpace, 1,
        XtNvSpace, 1,
        NULL);
    sprintf(s, "MGTop_%d", i);
    MGTop[i] = XtVaCreateManagedWidget(
        s, labelWidgetClass, MGBox[i],
        NULL);
}

XtVaSetValues(MGBox[0], XtNleft, XtChainLeft, XtNtop, XtChainTop, NULL);
for (i=1; i<3; i++)
    XtVaSetValues(MGBox[i], XtNfromHoriz, MGBox[i-1], NULL);
XtVaSetValues(MGBox[3], XtNfromVert, MGBox[0], XtNleft, XtChainLeft, NULL);
for (i=4; i<6; i++)
    XtVaSetValues(MGBox[i], XtNfromHoriz, MGBox[i-1],
                  XtNfromVert, MGBox[0], NULL);

MGPictShell = XtVaCreatePopupShell(
    "MGPictShell", transientShellWidgetClass, topLevel,
    XtNminWidth, 240,
    XtNminHeight, 240,
    XtNmaxAspectX, 1,
    XtNmaxAspectY, 1,
    XtNsSaveUnder, False,
    XtNtitle, "XXX",
    NULL);
MGPict = XtVaCreateManagedWidget(
    "MGPict", widgetClass, MGPictShell,
    XtNwidth, 360,
    XtNheight, 360,
    NULL);

for (i=0; i<8; i++) {
    sprintf(s, "MGItem_0_%d", i);
    MGItem[0][i] = XtVaCreateManagedWidget(
        s, labelWidgetClass, MGBox[0],
        XtNresize, False, NULL);
}

for (i=0; i<8; i++) {
    sprintf(s, "MGItem_1_%d", i);
    MGItem[1][i] = XtVaCreateManagedWidget(
        s, asciiTextWidgetClass, MGBox[1],
        XtNeditType, XawTextEdit, NULL);
}

for (i=0; i<6; i++) {
    sprintf(s, "MGItem_2_%d", i);
    MGItem[2][i] = XtVaCreateManagedWidget(
        s, toggleWidgetClass, MGBox[2],
        XtNradioGroup, MGItem[2][0],
        XtNradioData, (caddr_t) i+1,
        XtNresize, False, NULL);
}

XtVaSetValues(MGItem[2][0], XtNstate, True, NULL);
for (i=0; i<2; i++) {
    sprintf(s, "MGItem_3_%d", i);
    MGItem[3][i] = XtVaCreateManagedWidget(
        s, labelWidgetClass, MGBox[3],
        XtNresize, False, NULL);
}

for (i=0; i<2; i++) {
    sprintf(s, "MGItem_4_%d", i);
    MGItem[4][i] = XtVaCreateManagedWidget(
        s, asciiTextWidgetClass, MGBox[4],
        XtNeditType, XawTextEdit, NULL);
```c
for (i=0; i<4; i++) {
    sprintf(s, "MGItem_5_%d", i);
    MGItem[5][i] = XtVaCreateManagedWidget(
        s, commandWidgetClass, MGBox[5], NULL);
    XtAddCallback(MGItem[5][i], XtNcallback, MGCallback, (XtPointer) i);
}

/* Create the widgets for the curve fitting dialog */

CFShell = XtVaCreatePopupShell(
    "CFShell", transientShellWidgetClass, topLevel,
    XtNtitle, "Data Curve Fitting", NULL);
CFForm = XtVaCreateManagedWidget(
    "CFForm", formWidgetClass, CFShell, NULL);
for (i=0; i<3; i++) {
    sprintf(s, "CFBox_%d", i);
    CFBox[i] = XtVaCreateManagedWidget(
        s, boxWidgetClass, CFForm,
        XtnhSpace, 1, XtnvSpace, 1, XtnTop, XtChainTop,
        NULL);
    sprintf(s, "CFTop_%d", i);
    CFTop[i] = XtVaCreateManagedWidget(
        s, labelWidgetClass, CFBox[i], NULL);
    for (j=1; j<3; j++)
        XtVaSetValues(CFBox[j], XtNfromHoriz, CFBox[j-1], NULL);
}
for (i=0; i<16; i++) {
    sprintf(s, "CFItem_0_%d", i);
    CFItem[0][i] = XtVaCreateManagedWidget(
        s, toggleWidgetClass, CFBox[0],
        XtnLabel, "", XtnResize, False, XtnSensitive, False, NULL);
}
for (i=0; i<4; i++) {
    sprintf(s, "CFItem_1%d", i);
    CFItem[1][i] = XtVaCreateManagedWidget(
        s, asciiTextWidgetClass, CFBox[1],
        XtnEditType, XawTextEdit, NULL);
}
for (i=0; i<4; i++) {
    sprintf(s, "CFItem_2 %d", i);
    CFItem[2][i] = XtVaCreateManagedWidget(
        s, commandWidgetClass, CFBox[2], NULL);
}
```