2

PL-TR-93-2116

IRMA: A PROGRAM FOR THREE-DIMENSIONAL COLOR DISPLAY OF DATA

M. F. Tautz

Radex, Inc.
Three Preston Court
Bedford, MA 01730

May 30, 1993

Scientific Report No. 13

Approved for public release; distribution unlimited

PHILLIPS LABORATORY
Directorate of Geophysics
AIR FORCE MATERIEL COMMAND
HANSCOM AIR FORCE BASE, MA 01731-3010

4AO 385

"This technical report has been reviewed and is approved for publication"

Edward C Robinson

EDWARD C. ROBINSON
Contract Manager
Data Analysis Division

[illegible]

ROBERT E. McINERNEY, Director
Data Analysis Division

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 30 May 1993	3. REPORT TYPE AND DATES COVERED Scientific Report No. 13

4. TITLE AND SUBTITLE IRMA: A Program for Three-Dimensional Color Display of Data	5. FUNDING NUMBERS PE 62101F PR 7659 TA 05 WU AB Contract F19628-89-C-0068
6. AUTHOR(S) M. F. Tautz	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) RADEX, Inc. Three Preston Court Bedford, MA 01730	8. PERFORMING ORGANIZATION REPORT NUMBER RXR-93051
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Phillips Laboratory 29 Randolph Road Hanscom AFB, MA 01731-3010 Contract Manager: Edward C. Robinson/GPD	10. SPONSORING/MONITORING AGENCY REPORT NUMBER PL-TR-93-2116
11. SUPPLEMENTARY NOTES	
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for Public Release Distribution Unlimited	12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report describes the input, output, and interactive capabilities of the IRMA plotting program. Actual FORTRAN or C coding is not given. The IRMA program was designed for displaying data, from spacecraft charging and contamination codes, in three dimensions, and in color. The user interface features interactive control of viewing position and perspective by keyboard and mouse input. Text ports are available so that figure captions can be typed directly to the screen. The current graphics state can be saved in a parameter file for later restoration. The many data visualization options include drawing complex polygon objects, displaying particle trajectories, creating contour level surfaces, and building random dot plots to represent number density. Column density images can be obtained by integrating along an arbitrary line of sight. Image files can be written out to create a set of frames depicting the time sequence of events.

14. SUBJECT TERMS Data visualization, 3-D plotter, color plotter, column density, IRMA computer code	15. NUMBER OF PAGES 44
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT Unlimited

TABLE OF CONTENTS

List of Figures

ACKNOWLEDGEMENTS

The author wishes to thank Dr. David L. Cooke for his assistance and guidance during the course of the work. Many of the key ideas for structuring and implementing the IRMA code were provided by Dr. Cooke. The author would like to thank Mr. Thomas Proulx for programming assistance on the IRMA program during his stay as a RADEX summer appointment. The author also wishes to thank Dr. Alireza Setayesh for help with the SOCRATES to IRMA interface.

1.0 INTRODUCTION

The IRMA (IRis MAnipulator) program was developed for presentation and analysis of three dimensional distributions of physics data obtained from numerical simulations. The program has been used extensively to help understand and diagnose the complex output from large simulation codes (POLAR [*Lilley, et al.*, 1989], SOCRATES [*Elgin*, 1988] and MACH [*Tautz, et al.*, 1988]) operated by PHK. The IRMA code is currently implemented on a Silicon Graphics IRIS 3030 workstation which uses the UNIX operating system. A preliminary version has also been brought up on an IRIS INDIGO minicomputer.

The design considerations for IRMA have stressed ease of use, modularity and flexibility. The prototype code was configured by merging a number of Silicon Graphics demo programs and hence the graphics interface is inherently user friendly. The modularity of the program allows the large number of graphics options to be organized in a convenient and logical fashion. Although the program was written to handle data from specific codes, we have tried to keep to a general format so that IRMA can be easily accessed by other programs and the graphics options are flexible enough to satisfy most plotting requirements.

We list below the main features and capabilities of the IRMA program:

- can read and plot polygon objects
- can read and plot trajectories
- can read and plot 2D image planes (compatible with IRAF program)
- can read scalar functions f(x,y,z) of data defined on a discrete x,y,z grid
- can create a dot pattern representing the grid nodes at (x,y,z)
- can create contour level surfaces for f(x,y,z) using polygonal elements
- can strip the above contour surfaces along coordinate lines for emphasis
- can create planar slices through f(x,y,z) and color code them
- can create speckle plots representing the local magnitude of f(x,y,z)
- can set up an arbitrary observer viewpoint with arbitrary object orientation
- can calculate integrated column densities for an arbitrary line of sight
- accepts both ascii and binary node file input and MSIO files
- can save parameter files specifying the current graphics state for easy restoration and continuation at a later date

- has special window modes for viewing the clipping planes, for comparing objects side by side and a display mode suitable for hard copy output

- can do a screen dump to an image file which can be later pasted to the screen and manipulated with the window manager

- has a flexible color legend and a choice of several color and grey maps

- has text ports for insertion of captions and labeling of units and a primitive text editor for modifying captions

- has data scanning capabilities for inspecting data at each node

- has a multi-grid option allowing for nested grid data

- has a zbuffer toggle for shadowing of hidden surfaces

- has an extensive set of help files describing each graphics option

In the following sections we describe these features in more detail. Section 2.0 gives an overview of the basic user interface. The next sections (3.0 to 8.0) summarize the five program modules. Section 9.0 outlines the column density algorithm. A set of appendixes describes other features of separate interest.

2.0 THE USER INTERFACE

This section contains an overview of the basic IRMA interface: how to enter the program, how to navigate, how to set things and how to exit.

To execute IRMA you must first type 'mex' (not needed on the INDIGO) to turn on the SI window manager program and then type 'irma' (if you are not in the directory containing the executable file irma, you must set the path to the executable). The IRMA program then enters the input/output (IO) module. This is the place to read in any files that you wish to plot. Choose the menu option #1 to enter the graphics modules. The first time into graphics, you should be in module 3, in the set up mode. The various modules and modes available and the mechanics of mouse/keyboard input are described below.

MOUSE INPUT

The mouse allows continuous change of the graphics parameters which appear in the status window (upper left window in setup mode). Note that the mouse selection of a pop up menu option is a two step action: pressing down the mouse shows the menu, while letting it up selects the highlighted menu option.

Each graphics module has its own menu of options and functions. Some menu options generate sub-menus, but one should never have to go more than one layer deep to reach something.

Although the pop up menus are different for each module, there are a number of common entries, which are listed below (the * stands for any of the five modules):

/*/toggles	- contains toggles and switches that can be accessed from any module
/*/info	- used to generate information of various sorts. Print normally appears lower left window (in set up mode).
/*/reset_parameters	- enables one to reset all graphics parameters for the current module to the original (on entering graphics) settings.
/*/modules	- navigates between the graphics modules.
/*/modes	- switches between the various graphics modes.
/*/return	- allows one to return to the IO module or to quit program.

There are two ways to exit from the graphics modules. If you select /*/return/quit then the program exits completely. All graphics parameters are lost. If you select /*/return, however, the code returns back to the IO module (note that to activate the IO module screen on the IRIS 3030, you must first attach the window with the mouse). From there you can save your graphics parameters (option #11) or read in more data and go back in to graphics. You can also exit the program directly from the IO module by selecting option #-1.

The procedure to set a graphics function parameter with the slider bar interface is given below:

1) To select a graphics function press the right mouse down to pop up a graphics menu, move the highlight bar to your choice, and then let the mouse up to toggle on the desired function. The function name and argument list should appear in the status window (upper left window in set up mode)

2) Use the left mouse to move the cursor and pick a variable from the argument list in the status window. This is done by placing the cursor over a number and pressing the mouse button down and up (the chosen variable will be highlighted)

3) Use the left mouse and cursor to pick and 'drag' the slider bar to the desired value. If the value is off scale, the slider bar will wrap around until it reaches the correct value.

KEYBOARD INPUT

The keyboard allows one to input discrete values. Do steps 1) and 2) above to select the variable that is to be changed. Instead of step 3) above type 'e' for enter. An enter prompt should appear in the slider bar window. Type in the desired number and press return to set the variable. The argument list and the slider bar will be updated. If you get a 'bad number' message, hit 'escape' to clear and enter the number again. Blank spaces are not allowed. 'E-format' is accepted.

Arguments can also be picked from the keyboard (step 2) above). First enter an 'o' to open a default argument (it will be highlighted). One can then step between args using the 'l' and 'r' keys (left and right). For those functions with two argument lines, one can jump to a new argument line with keys 'f' (forward or down) or 'b' (backward or up).

The keyboard is always ready to accept commands, except when in text input mode (see Appendix B) or during number entry mode (see above). Pressing 'escape' will always restore the keyboard to command mode.

Hit /*/info/keyboard to see a list what commands are available. There is some redundancy between the mouse menu options and the keyboard commands. For example, one can change modules from the keyboard using the number keys. Or one can change modes with 's', 'd', 'c', and 'x' keys (standing for set up, display, clipping and compare modes).

MODULES and MODES

The five modules implementing the graphics options are listed below:

1) module 1 - sets the observers viewpoint, using SI graphics functions ortho, window, persp, polarview and lookat.

2) module 2 - uses SI graphics routines scale, translate, and rotate to manipulate objects. Special objects can be singled out for independent transformation.

3) module 3 - creates graphical objects and modifies them. Manages the object list.

4) module 4 - creates images using the column density integration or loads in an IRAF image. The images can be manipulated and added to the object list.

5) module 5 - formatting control of various windows.

These graphics modules are described in more detail in Sections 4.0 to 8.0 of this report.

Graphics modes refer to user selectable window configurations. There are four modes currently available:

1) Set up mode - This is the 'working' mode. The status and slider bar windows appear at the upper left. The main display window is at the right. User controlled information appears in the box at the lower left. The box in the upper right corner indicates the current module and the state of the keyboard.

2) Display mode - This mode can be used when an acceptable view has been obtained and perhaps a hard copy is desired. Only the display window appears, with all the set up information stripped away. The zbuffer or depth cueing options only work in display mode (this restriction applies only on IRIS 3030).

3) Clipping mode - This view shows the placement of the clipping planes, corresponding to the current main window view. For convenience, a compressed view of the display window appears at the upper right. This mode can be used to clarify the clipping plane configuration.

4) Compare mode - Allows one to show selected (from the object list) objects side by side for direct comparison. Uses /m3/select_objects/window_1 or 2 to fill the two windows.

The current color map can be displayed (in set up mode) by going to module 3 and toggling /m3/color_map/show_map. If the color map is bad (e.g. after a system crash) it can be restored to the standard rainbow map by selecting /m3/color_map/normal_map.

There exists an irmafiles/help subdirectory with detailed information on most of the IRMA options and functions. This directory contains many small files. The names of the help files are chosen to be similar to the functions they describe, usually corresponding to the pop up menu entry string. Use Unix commands ls and/or grep to find the help you need.

3.0 INPUT/OUTPUT FILES (IO MODULE)

On entering the IRMA program the input/output (IO) module menu appears. This menu is used mainly to read and write files. There are presently six types of files supported:

1) surface polygon file - contains a list of color coded polygons describing some object e.g. the space shuttle. Format is compatible with POLAR/SHONTL/NPLDIS output.

2) ascii/binary node data files - A complete set of tools exist for ascii read/write and binary read/write. They are compatible with MACH and SOCRATES output.

3) MSIO node data files - these are 'indexed sequential binary' files, modeled on the CYBER mass storage routines.

4) trajectory files - contains particle trajectory step data. The format is compatible with POLAR/SHONTL/NPLDIS output.

5) IRAF image files - two dimensional images written by the IRAF program. These files can also be written by MACH.

6) graphics parameter files - allows one to save the graphics parameters from an IRMA session

Column density images can also be written out in ascii node file format (2 above). These files can be read by an auxiliary code and converted to SUATEK or MACHCON format for black and white contour plots.

When reading a file, IRMA will ask for the file name. If the file is not in the current directory, this name should include the path. After the read in is completed, IRMA always prints a little information to help verify that the file was read correctly. This is usually the minimum and maximum from a scan of the data that was read.

Polygon files:

The polygon file is ascii and has a very simple format. It consists of a list of polygons. Each polygon is specified by a card giving the number of vertices and the value at each vertex, followed by one card for each vertex with its x,y,z coordinates. For example, an equilateral triangle lying in the x-y plane, with node values -2.5, 0.0, 2.5 would be given by the cards:

```
 3 -2.5  0.  2.5  =  number of vertices and values at each one
-1.  0.  0.       =  x,y,z of vertex 1
 1.  0.  0.       =  x,y,z of vertex 2
 0. -1.  0.       =  x,y,z of vertex 3
```

Polygon vertices should always be drawn in the counter clockwise direction, as seen from positive z. If the /m3/modify_objects/solid toggle is on, the polygon will be drawn in one color using the value of the first vertex. If the /m3/modify_objects/gouraud option is on, the polygon will be gouraud shaded i.e. the colors will be linearly ramped between the vertices. Polygon files are read in by subroutine readob.f.

Node data files:

The node data files also have a simple format. It consists of some header lines and then columns of data as indicated below:

```
N  = number of input parameters (zero is expected)
M  = number of variables in this file
...followed by one caption string card for each of the M variables
NX  NY  NZ  =  number of grid nodes in x,y and z directions
...followed by x coordinates list
...followed by y coordinates list
...followed by z coordinates list
value1    value2   value3 ... =  up to M columns of data
...more columns of data
```

For the columns of data the x index runs fastest, then y, then z. The value of N can be set non-zero to maintain compatibility with old files, but currently no input parameters are passed to IRMA. Node files are read in by subroutine readab.f.

MSIO files:

MSIO files carry the same data as the node data files above. The advantages of using MSIO files are:

a) they are binary, hence smaller (by ~1/3)
b) they are quicker to read in
c) they have a 'key' structure allowing sub-file organization
d) they support multi-grid data
e) they are compatible across different machines

A disadvantage of the MSIO files is that there is considerable set up work needed to use them i.e. the routines to create these files must be constructed. A set of prototype routines to be used for developing this capability are located in a IRMA subdirectory called irmafiles/mstools. MSIO files are read in by subroutine readin.f.

Trajectory files:

The trajectory files contains a sequence of trajectory step data of the form:

```
x y z color = x,y,z coordinates and color index
...followed by more cards of same form
```

The color variable is a function of the track coordinates which can be used to describe local conditions, e.g., the electrostatic potential at each time step. The trajectory is plotted in increments of two. That is, a straight line is drawn from point 1 to point 2, from point 3 to 4, from point 5 to 6, etc. No line is drawn between point 2 and 3, 4 and 5, etc. This is the SCUBED format and is designed to follow multiple independent tracks. If a single continuous track is desired the connecting points must be equal i.e point 2 equals point 3, point 4 equals 5 and so on. When drawing the track, IRMA adds a small white line segment on the forward end to indicate the direction of the track. Trajectory files are read in by subroutine nrtraj.f.

IRAF image files:

The IRAF image files contain f(x,y) data written in FITS format. There is a set of header cards and then a linear string of data values. Consult the IRAF documentation for details [*NOAO*, 1987].

While reading these files, IRMA will prompt for information. For IRAF files this is the number of columns of data and the format, which can be read off the header cards. For files written by MACH, one can also input information on the image boundaries. IRAF image files are read in by subroutine readim.f.

Graphics parameter files:

The graphics parameter files are written by IRMA from the IO module. The purpose of these files is to save the final graphics parameters from an IRMA session. When you return at a later date you can read this file and restore the graphics parameters to the original settings. Reading a parameter file over writes any existing parameter settings.

On the INDIGO version of IRMA, we have added an option to read the SOCRATES binary files directly. This feature bypasses the need to transfer data by means of intermediate files. By choosing the IO module option #11, one enters a sub-menu of IRMA and interactively selects the desired variables from the SOCRATES restart file. Plots can be made and then another variable selected. This provides a fast and convenient way to interrogate a SOCRATES run.

4.0 OBSERVER VIEWPOINTS (MODULE 1)

Module 1 sets the observers viewpoint, using SI graphics functions ortho, window, persp, polarview and lookat.

For a full description of the above SI graphics functions see their documentation. Briefly, the functions ortho, window and persp define the observers view by means of clipping planes. For ortho, which is the non-perspective case, there are six orthogonal clipping planes forming a parallelepiped which frames the view. For the perspective views, window and persp, the clipping planes form a viewing pyramid with the eye at the vertex. The window function is the same as persp, but allows more flexibility in setting the clipping planes.

To activate these transformations select menu option /m1/ortho/window/persp and choose the desired function. In order to facilitate usage of these graphics functions, we have defined some combination moves. These allow one to tie together arguments in the argument list in such a way as to efficiently produce effects that are needed frequently. These combinations are listed below:

l+r =	move left and right clipping planes together (with constant difference) to produce a horizontal shift
b+t =	move bottom and top clipping planes together to produce a vertical shift
n+f =	move near and far clipping planes together to shift the clipping span
n-f =	move near and far clipping planes in opposition to expand or narrow the clipping span
zoom =	move the clipping planes in such a way as to produce a zoom. This is done differently for each function: ortho - all six clipping planes are moved in opposition window - the front window sides are moved in opposition persp - not needed (the fov variable does a zoom)

When one chooses one of these combinations, a message appears to the right in the status window indicating which variable to pick in order to move the arguments together.

There are also some conversion options to translate one view to another e.g. use ortho/window to go from window to ortho. These will only give an approximate correspondence, since exact matching is not always possible.

The polarview and lookat functions position the observers eye at a specified location. The polarview function puts the observer on the surface of a sphere, looking towards the origin. The lookat function puts the eye at an arbitrary position and with an arbitrary line of sight. It is thus more general than polarview. However, polarview has been found adequate for most purposes and lookat has only been infrequently used. To modify these functions choose menu option /m1/polarview/lookat.

There are menu options to look directly down the x, y and z axis. These work by changing the angle arguments to polarview. They can be used to quickly look at an object from all sides. These options over write the current polarview parameters.

The view clipping menu option brings up a sub-menu that can be used to modify the two views of the clipping mode. One can toggle between perspective and orthogonal views, zoom, rotate and change the observer distance.

Typically one would like to frame the grid data as closely as possible, for example when calculating column densities. This can be done by first creating the grid nodes object and then using the clipping mode and the module 1 graphics functions to set the planes at the grid boundaries.

5.0 OBJECT TRANSFORMATIONS (MODULE 2)

Module 2 uses SI graphics routines scale, translate, and rotate to manipulate objects. One can think of these transformations as active (observer fixed, object moves) as opposed to the module 1 viewing options which can be thought of as passive (observer moves, object fixed). If one just wants to inspect an object, this module can be a little easier to use e.g. the rotations are done directly around each of the three coordinate axis rather than in terms of the more complicated azimuth, inclination and twist angles. The total viewing matrix will be the product of the observer transformations of module 1 and the object transformations (if any) of module 2.

The translate, rotate and scale transformations work as described by SI:

/m2/translate: moves objects parallel to the x, y or z axis

/m2/rotate: rotates objects around the x, y or z axis (units are degrees/10)

/m2/scale: shrinks, expands or mirrors objects along each of the x, y or z axis

In IRMA we have defined an additional function, zoom, which scales objects equally in the x,y,z directions. One can combine these transformations to create more complex transformations. The order is important. Rotates are done before translates in the order x, then y, then z.

The main complication in module 2 is the definition of special objects. This allows one to treat certain objects differently than others with respect to rotate, translate and scaling. This feature can be used to make adjustments for input coming from different programs. For example, if a polygon model of the space shuttle was created by one program, say POLAR, for use with the data from another program, say SOCRATES, it might be necessary to introduce a scale factor, translation, and/or a rotation to properly place and orient the POLAR shuttle in the SOCRATES grid space. We can do this by defining the shuttle to be special; it can then be transformed independently of other objects.

The two IRMA functions that handle special objects are:

/m2/select_special	- This is a switch that toggles on/off the special object transformations for selected objects. It has no effect if a special transformation has not previously been defined for the object. The list option summarizes the current status.
/m2/modify_special	- This switch must be turned on before an a selected object can be modified. The rotate, translate and/or scaling transformations can then be set. The status window should say which object is ready for change, and give a summary of all the argument values. The arguments cannot be changed at this point. To do this one must pick out the appropriate transformation from the menu options and set it in the normal way.

By default, the transformations of module 2 apply to all graphical objects in the object list and to make a certain object special one would do the following:

a) use modify_special to enable the object as special
c) choose a transformation, say rotate, and set the angles

If one uses /m2/select_special_objects to toggle off the special status, the object will revert to normal status. Select /m2/modify_special/none to reset module 2 so that any subsequent transformations will apply to all objects.

The action of the module 2 transformations can be observed by using the IRMA clipping mode. View any object (e.g. the x,y,z axis) and vary the translate, rotate, or scale parameters. The object will respond but the clipping planes will remain fixed. This is in contrast to the viewing transformations of module 1, which change the clipping planes, with the object held fixed.

The special object settings are written to the IRMA parameter file so they can be recovered in subsequent runs.

Note that the /toggles/socrates switch uses the special objects option. This switch sets the graphics parameters up for a reasonable first look at SOCRATES data. It assumes that the shuttle object has been read in from the IO module and it makes this object special. This can lead to some confusion if you toggle socrates on, without reading in the shuttle model.

6.0 CREATE AND MANIPULATE OBJECTS (MODULE 3)

Module 3 is used to create graphical objects, to modify them, and to manage the object list. It also can be used to manage the grid list if more than one grid is read in. The color map options are also in this module.

The module builds graphical objects of various types, based on grid node data that has been previously read in. The module 3 pop up menu options that actually create objects are marked with an asterisk. Each object that is created is automatically placed in the object list. A list of the currently available objects is given below.

CONTOURS	- these are 2D surfaces of constant value. Use the /m3/set_levels function to choose up to three surface levels per object. The algorithm builds a polynomial object by finding the intersections of the specified level value with the volume cell boundary lines and then constructing triangles connecting to the intersections and to a common cell midpoint.
SLICES	- these are 2D surfaces on a planar cut through the data. Use the /m3/set_nodes function to choose the i,j,k grid nodes. For example, selecting $j = 8$ would specify the plane defined by $y(8) =$ constant. The default slice format is the wire frame. Use /m3/modify_objects/gouraud to produce a color coded slice.
CLOUDS	- these are speckle plots with random pixel dot density representing the data magnitude (assumed > 0). For example, this has been used to depict particle number density distributions. The number of dots put in volume cell i is $n_i = c*f(x,y,z)*dV$ where c is a normalization constant set so that $\Sigma n_i = n$, where n is user input (default = 10000). Use /m3/set_cloud to modify the cloud parameters.
GRIDS	- this is a dot pattern representing the grid nodes. The options are a single white dot giving the node location or a colored dot (actually a cluster of 7 closely spaced dots) coded according to the data value at the node. Use /m3/modify_objects/color_or_white to switch between these two options.

Once an object has been placed in the object list, it may be toggled on and off using /m3/select_objects. One can remove the last object (only) from the list using /m3/delete_last_object. There are a number of other options that may be used to manipulate objects selectively i.e. governed by select_objects, which are described below.

/m3/modify_objects	- change from wire frame to solid, gouraud shade and other options. An automatic redraw is done.
/m3/keep/cut	- allows one to remove a selected volume region from the polygon draw loop. One can remove all polygons outside a specified region (a keep) or inside a region (a cut).

/m3/strip	- allows one to redraw a polygon surface contour using only the planar projection in one of the coordinate directions. These stripped polygons are drawn with double width lines to enhance visibility. The stripped objects do not appear in the object list, they have their own sub-menu for selecting on and off.
/m3/redraw	- Sometimes it is desirable to redraw a polygon object. This might be done after a color mapping change, after turning on a keep or cut or for some other reason.
/m3/center	- Sometimes it is easier to manipulate an object if it is centered with respect to the origin. This switch enables one to center and un-center selected objects. The centering transformation is based on the first object that was read in by the IO module.

The /m3/select_grid option allows one to pick out individual grids for display and analysis. Each node data set that is read in, goes into a separate grid (they may be identical). So switching grids enables one to choose between different data sets. The /m3/select_grid option can also be used in combination with the nesting function to treat multi-grid input (see Appendix D). The nesting function applies to all grids that are selected on. It partitions the volume space in such a way that any volume element is covered only by the highest imbedded grid. If toggled on, the nesting function is used in any subsequent object creations.

The /m3/lin_log switch allows one to convert the node data to log10. This may still have some glitches.

The /m3/color_map option allows one to control the color map and the color legend (see Appendix A).

7.0 CREATE AND MANIPULATE IMAGES (MODULE 4)

Module 4 enables one to handle images. Images differ from objects in two main ways.

a) they are two dimensional planes holding projected image data
b) they are drawn on the back clipping planes, using the current view parameters

Images can be promoted to objects in the object list (using the /m4/image_to_object option). But images are generally more transitory than objects i.e. if the observer view is changed, then the image will no longer be lined up on the back clipping plane, and a new image should probably be created.

The two main image building options are:

/m4/create_columns	- This enables one to create column density images by integrating through the input grid node data. See Section 9.0 for column density details.
/m4/load_IRAF	- Any IRAF image file that was read in from the IO module can be loaded in here.

When a new image is created, it overwrites any previous one, so that there is only one current image.

The resolution lines defining a column image may be viewed using the /m4/image_lines option. Each image line is then surrounded by a colored coded box. The color index is based on the column density value for a ray going through the center of the box. There are also options for displaying solid or gouraud shaded boxes.

The IRAF images can be modified using the options in set_IRAF. The default resolution is that of the input data file, but one can also change it (with /m4/set_IRAF) so that the column density resolution setting is used instead, in which case one gets interpolated data values. One can also position the image with respect to arbitrary boundary limits (one must specify the true image boundaries at read in time). For example, the image files written by MACH for the velocity space distribution function, have differing angle boundaries, which can be lined up with respect to a common range say, 0 to 180 degrees. Tics and numbers can be printed to make the boundaries clearer.

The color legend for images is separate from that for objects. It will change automatically from the object legend to the image legend if one builds column densities or loads in an IRAF image. One can use /m4/image_legend to choose one of two scale modes which are:

1) CALC	- The natural scale. Here the min and max are calculated by a scan of the image data values.
2) SET	- An imposed scale. Here one sets the min and max with the mouse.

One would typically use CALC to first generate the limits. Then SET might be used to round off the end points or to set a common scale for subsequent images. Any data that falls off the end of the scale retains the color of the end point.

The units for the image legend are arbitrary. They can be modified for column density images (see Section 9.0). Press key 'u' for a display of the current units above the legend.

If you have more than one IRAF image to view, read in the first one and go into graphics, create it and put it into the object list using /m4/image_to_object. Then return to the IO module, read in the next image and go back to graphics. Any number of images can be assembled this way.

Two images may be viewed side by side using the compare mode. The procedure for setting up this view is given below:

1) pick the compare mode to view the two windows

2) create the first image and use /m4/image_to_objects to place it in the object list. Use /m4/select_objects/window_1 to specify that this object is to appear in the left window. Then select this object off.

3) create the second image. You don't have to make this image into an object and specify window 2 (although that would work) because that is where it will go by default.

Note that if an object and an image are both sent to the display window, the object will have precedence over the image. Select all objects off to reveal the image. The current image (the last one created) can be toggled separately using /m4/image_on/off.

8.0 SCREEN FORMATS (MODULE 5) AND SCREEN DUMPS

Module 5 allows one to change the format of the display window and of the text ports. The main options are:

/m5/axis	- This is the set of orthogonal x,y,z line segments that are normally drawn at the origin. One can vary the length of the lines and displace them.
/m5/display	- This option enables one to change the size and location of the main display window. (This must be toggled on with /toggles/modify_display on the INDIGO.)
/m5/text_port	- Various options for formatting text in the text port inside of the main display window.
/m5/info_port	- Various options for formatting text in the information port.
/m5/info_box	- Various options for drawing an outline box within the information port.
/m5/title	- This displays the caption line that is read in off the node data file. One can change the position, color and borders.
/m5/grid_units	- This allows one to print out the axis length in units of KM. One can modify the position and color of this caption.
/m5/legend	- Allows one to change the position and size of the color legend.

See Appendix B for a description of the available text ports. These must be switched on to see the effects of modifications.

Once one has an acceptable screen view it is possible to dump it onto an image file which can be picked up later (post-IRMA) and manipulated with the window manager. There are three screen dump options:

1) /toggles/**screen_dump - this dumps the entire screen (not available on the INDIGO)

2) /toggles/**display_dump - this dumps only the image in the display window

3) /toggles/**zbuffer_and_dump - this is the same as the display dump, but must be used with zbuffered images (see Appendix E)

When a screen dump option is toggled on, there will be a pause of a few seconds, and then some noise lines will cross the screen on the IRIS 3030, which signals that the dump is complete. A line of print is written to confirm the dump. The output files will be called frame1, frame2 etc. Any existing files of the same name will be overwritten.

To put an image frame onto the screen, outside of IRMA, use the SI image tool routine 'ipaste'. The image can then be pushed, popped and moved (but not reshaped) like any other window. Other image tool routines that are useful are:

clip	- allows one to extract a subimage from an existing image
snap	- allows one to save an existing image onto an image file
cedit	- edit the color map
mag	- allows one to scale up a image
movie	- shows a sequence of frames

These routines are found in the IRMA subdirectory /irmafiles/mextools.

We have used IRMA and the image tools on the IRIS 3030 to create a movie to represent the time sequence of a SOCRATES plume simulation. The steps required to do this are outlined below:

1) run SOCRATES at successive time intervals and write the IRMA input node data files
2) use IRMA to create image frames for integrated column density at sequential time increments and dump the image frames
3) use the IRMA text port capability to create suitable header frames
4) connect the RGB output of the IRIS to an encoder and the encoder output to a tape recorder.
5) switch the IRIS signal to NTSC mode (using the SI setmonitor command) and run and record the movie

Using this procedure, we have created a short movie consisting of about 50 image frames that depict the time development of a shuttle plume for ram, perpendicular, and wake engine burns.

On the INDIGO, we have used the IRIS SHOWCASE program to generate the header frames.

9.0 COLUMN INTEGRATION

In order to calculate emissions from a distribution of molecules in space it is necessary to integrate along the line of sight to obtain the associated column densities. IRMA has an option to do this calculation for an arbitrary observer viewpoint. This option has been used in the analysis of SOCRATES simulation results for space shuttle plumes. The algorithm proceeds as follows:

1) given the current view parameters (either window or ortho), the front clipping plane is gridded uniformly to form transverse area elements.

2) A ray from the eye to the center of each area element is pushed from the front to back clipping plane and the integrated sum is accumulated and saved.

3) When all the rays have been integrated, the column density for each area element is gouraud shaded and then projected onto the back clipping plane.

To activate this calculation, use the /m3/*create_columns pop up menu option. One should line up the observers viewpoint so that the region of data is closely framed by the clipping planes (one can use the clipping mode to determine this).

Some aspects of the calculation are given below. Let f(x,y,z) represent the source distribution. Let a,b denote the transverse coordinates in the front clipping plane and the variable c stand for the distance along a line of sight through point (a,b), between the front and back clipping planes. Then the column density at (a,b) will be given by

$$d(a,b) = \Sigma\, f(x,y,z)\, dc$$

where the coordinates x,y,z are determined by the mapping of (a,b,c) onto the grid coordinates (x,y,z):

$$x = x(a,b,c)$$
$$y = y(a,b,c)$$
$$z = z(a,b,c)$$

As c is stepped, the points (x,y,z) do not usually fall right on a grid node, and the algorithm uses trilinear interpolation in each grid box to assign a data value. If the point (x,y,z) lies outside the grid, zero is assigned.

In this calculation, the resolution can be varied by setting integers nab and nc such that

$$da = (a2-a1)/nab$$
$$db = (b2-b1)/nab$$
$$dc = (c2-c1)/nc$$

where 1,2 denote the boundaries set by the clipping planes. The column resolution may be set with the /m3/set_columns menu option (upper limit of 50). For 32 by 32 by 32 resolution, the calculation typically takes a little over a minute on the IRIS 3030.

As indicated in Section 7.0, the column rays through the front clipping planes can be displayed so one can obtain a feeling for whether more resolution is needed. Some numerical details of the calculation are available through the /*/info/image_calc menu option. For example, this prints out the total intensity in the viewing window

$$t = \Sigma\ d(a,b)\ da\ db = \Sigma\ f(x,y,z)\ da\ db\ dc\ = \text{volume sum}$$

The units for the column density calculation can be modified with the /m3/image_units menu option. The units options assume the input grid node data f(x,y,z) is in units of # events/cm^3/sec. These are the units for reaction rate which are output from SOCRATES. In this case, the natural units for column density are # events/cm^2/sec, since the integration over c cancels out one power of length. Other units are available. For example, in order to convert to units of watts/cm^2/sr one needs to multiply by the energy emitted per event, de, and divide by 4π for solid angle. Assuming one photon is released per event de = h(c/l) where l is the photon wavelength, which can be set with the mouse.

If the units are different than the above, there is an arbitrary scale factor available to convert the data to any set of units.

REFERENCES

Elgin, J. B., "User's Manual for the SOCRATES Monte Carlo Contamination Model", GL-TR-88-0303, 1988. **ADA205181**

Lilley, J. L., Cooke, D., Jongeward, G., and Katz, I., "POLAR User's Manual", GL-TR-89-0307, 1989. **ADA232103**

National Optical Astronomy Observatories (NOAO), "IRAF User Handbook, Vol. 1B, Version 2.5, 1987.

Tautz, M. F., Cooke, D. C., Rubin, A. G., and Yates, G. K., "Preliminary Documentation of the MACH Code", GL-TR-88-0035, 1988. **ADA198956**

APPENDIX A. COLOR MAPS AND THE COLOR LEGEND

The colors for the IRIS can be indexed to a color map in the range 0 to 1023. The programmer can build his own map and that will become the system map until someone rebuilds it. If the system crashes, the color map is often corrupted and the color map has to be rebuilt. To display the current color map, toggle '/m3/color_map/show_map'. The color maps available are:

1) the normal map - this map mixes blue with green and green with red to produce a 'rainbow' of colors

2) a shade map - this map has a rainbow with bands of color mixed from black towards white. It can be used with the depth cueing option.

3) a grey map - this map runs from black towards white. Used for black and white hard copy.

4) a sawtooth grey map - this map runs between black and white in an approximate sawtooth pattern. Used for special lighting effects.

5) a color band map. Test map with different bands of solid color.

The color map works with the legend to relate color to value. IRMA contains options to modify this relation. The white rectangular box drawn on the IRMA color map depicts the boundaries of the mapping function for the legend i.e. data values are mapped into successive rows starting at the lower left corner and ending with the upper right corner of the box. The shape of the box can be changed with the /m3/color_map/color_range and /m3/color_map//shade_range options. The former changes the vertical boundaries of the box and the latter the horizontal boundaries. One can also modify the legend lower and upper data limits and the number of intervals using the /m3/color_map/data_range option.

If the legend is changed, previously colored objects may need to be selectively redrawn (using /m3/*re-draw).

If one toggles the log/linear switch, the legend should reflect this change. This switch does not currently work with nested multi-grids.

On the INDIGO, the legend can be displayed vertically using /toggles/legend_hor/ver.

APPENDIX B. TEXT PORTS

Suppose one has set up an acceptable view and is ready to produce a hard copy. At this point it may be desirable to add an informative caption of some kind. In IRMA one can do this by opening the text port. There are two main types of text port:

1) the primary text port - this port is opened within the main display window, at the top. To enter it, type 't' on the keyboard. The textport box should then appear on the screen at the top of the display window. The keyboard is automatically set to text input mode. This should be indicated in the upper right corner box (in set up mode).

2) the secondary (information) port - this port opens a new window. Thus, it can be located anywhere on the screen and be of arbitrary size. Otherwise it is similar to the primary port. There is some added flexibility in that an interior box for outlining text can be defined. Use his port when there is a lot of elaborate information to be displayed. To enter it, type 'i'. The port window opens up over the display window and is in text input mode.

When in the text input mode, a small blue cursor will be visible within the text port window. Any keys that are struck will type directly into the text port to the left of the cursor. There is a primitive editor in IRMA that allows one to modify the text port. The cursor can be moved by means of the arrow keys or with the backspace key and the space bar. To exit the text input mode, type 'escape'.

The text ports can be modified using the formatting module (5). Things that can be modified are listed below:

a) The text port can be repositioned on the screen

b) The size of the port can be reduced or expanded either horizontally or vertically.

c) The text, border and background colors can be changed.

d) The number of lines within the port can be modified. Up to 30 lines are allowed. Line markers can be inserted.

e) Text may be shifted horizontally within the port.

Text port information is saved on the IRMA parameter file and so can be recovered later.

There also exists a one line text port which allows one to insert captions above the color legend. This can be used to specify special sets of units. Type 'u' to enter this port. The cursor works here and the line can be edited.

The text ports can be turned on and off from the toggles menu.

APPENDIX C. DATA SCANS

It is possible in IRMA to inspect selected node data points from within the graphics modules. This option can be used to help clarify the situation, if the graphics output doesn't look quite right. There are currently three ways to do this:

1) toggle /*/info/coordinates - this will list the current grid coordinates and the data values at the listed points.

2) toggle /*/info/probe - This function uses '/m3/set_nodes' to vary the position of a small probe (a parallelepiped) that appears on the screen. For example, set grid node indexes i,j,k to values within the grid range, to specify a point at location x(i), y(j), z(k). The color of the probe will correspond to the data value at the node. The size of the probe is that of a half-width cell, centered on the node. The data value and the corresponding color index at the node is printed in the information box at the left (in set up mode).

3) toggle /*/info/grid_scan - This brings up a one dimensional line profile through the data. Use '/m3/set_nodes' to pick the desired scan line. This is done be setting to zero the node index of the 'floating' dimension to be scanned. For example, to see the data as a function of y, with x and z held fixed, set the y coordinate index j = 0 and the x and z indexes i,k to fixed values so that the coordinates x(i), z(k) lie within the grid boundaries.

By means of these scan options it is possible to look at any point in the grid.

A quick way to obtain a 'global' view of the data is to build the grid as an colored object using /m3/*_create_grids. Each node will then be represented by a color coded dot with the color corresponding to the local node value.

APPENDIX D. MULTI-GRIDS

Multi-grid data can be of two types.

1) Data for two different variables existing on two independent grids.

2) Data for the same variable existing on one or more nested grids.

In order to handle both types of data we have implemented a flexible system, where one can display the grids separately or one can nest them.

a) For independent grids (case 1 above) the /m3/select_grid option can be used to select one grid as the current grid. All subsequent object creation will then be based on that grid variable.

b) For true multi-grids (case 2 above) that require nesting, use the /m3/select_grid option to choose any combination of grids. Then toggle on the /m3/define_nest menu entry. This will create a nesting function so that any objects that are subsequently created are nested with respect to all grids selected on. One can enable or disable the nesting function with the '/m3/select_grid/nest on/off' toggle. Nested grids can be viewed by using the '/m3/*create_grid_nodes' option. If the input nested grids do not align on the boundaries, it is possible to pad out (with linear interpolates) the inner grids so that they fit together with no gaps.

The nesting function works as follows. The selected grids are looped through, starting with the lowest grid. At each node a test is made to determine: what is the highest grid that contains this node. This number defines a nesting function for each grid. In subsequent graphics creation options (for clouds,slices,levels, etc.), the grids are again looped over and the nesting function is used at each node - if the nesting function is not equal to the current grid, then the drawing action is skipped. In this way the volume of the computational grid is partitioned in such a way that in any cell the highest level grid is always used. Note that this algorithm will only work the nested grids are read in sequence from course (lowest grid number) to fine (highest number). If two nested grids have the same resolution and they overlap, the one with the higher grid number will predominate in the shared region.

This section is still in development.

APPENDIX E. ZBUFFER

When IRMA draws polygonal objects it normally starts with the first polygon in the list and draws them in order. If the first polygon is in the background (i.e. furthest from the observer) this causes no problem. But if the first polygon in the list is in the foreground, then it can subsequently get drawn over by polygons behind it, leading to a confusing picture where the true orientation and structure of the object may be obscured. This problem can be fixed by using the Silicon Graphics ZBUFFER option, which enables one to shadow out hidden surfaces. It works by testing the z coordinate of each pixel and only drawing the pixel with the z value nearest to the observer.

On the 3030 workstation the zbuffer process is slow and is not done continuously. In IRMA there is a /toggles/zbuffer switch that can be used to zbuffer any fixed view. The screen remains in the zbuffered state until the next event in the queue is encountered (This could be a mouse click or any keyboard event). The screen then reverts back to normal. Note that one must be in display mode in order for the zbuffer option to work (only one window can be open). The zbuffer switch can also be activated from the keyboard by typing a 'd' to go to display mode and a 'z' to turn on the zbuffer. On the INDIGO, zbuffer can be on continuously.

The zbuffer option does not represent lines embedded in a polygon surface properly (the 'decal' problem) because it cannot distinguish between two pixels with the same z value. Thus any lines lying right on a surface can tend to look fragmented. To work around this problem we have added an option to IRMA (accessed from /m3/modify/split_wire_frame) to split the wire frame outline of each polygon surface so that two frames are drawn, each displaced plus and minus epsilon (user specified) along the normal to the polygon. This separates slightly the z coordinate of the wire frame lines from that of the surface polygon and makes the lines more visible during zbuffering.

The depth cueing option switch in IRMA is subject to the same constraints as zbuffer. It maps color with z depth and can be used with the shade map to make near lines appear brighter and far lines to have decreased intensity. This option has not been used much.

APPENDIX F. PROGRAMMING NOTES

The IRMA program consists of a mixture of C and fortran coding. There are approximately 25K lines of source code, of which 60% is in FORTRAN. Assembly language wrappers provide entry point interfaces for FORTRAN-C calls. The main program is in FORTRAN and most of the graphics code is in C. The code uses the Silicon graphics multiple windowing capability (mex on the IRIS 303, 4Dwm on the INDIGO) to manage up to 10 windows. The executable file 'irma' is located in directory /irma along with many sample input files. The directory /irma has a number of subdirectories. This directory tree contains all the code needed to compile and link IRMA.

The source routines and object files are located in subdirectory /src. To update the executable, type 'make' in this directory. This will automatically remake the wrappers and files with modified includes. The linker references two archive libraries:

1) /npltools - this contains routines needed to use the MSIO files, borrowed from NPLDIS program.

2) /mextools - this contains SI image tools needed to do screen dumps and to manipulate the image files that are produced.

The include files are in subdirectory /inc. The file irmas.h contains important size parameters. The grap.h and grap_ext.h files hold the C commons variables. Some of the other .h are SI system files, put here so that IRMA is self-contained (except for one proprietary file get.h).

The FORTRAN to C (f2c) and C to FORTRAN (c2f) wrapper scripts are in subdirectory /wrapper. To remake a wrapper by hand type:

```
../wrapper/wrapfc name (for FORTRAN TO C)
../wrapper/wrapcf name (for C to FORTRAN)
```

Here 'name' is the source file name without extension. Normally you don't have to do this (make does it all). However, the construction of the wrapper files is described by these scripts. There are examples for both c2f and f2c wrappers in /src. A 'f2c' routine will be recognized by its having three files: name.c, name.o, and name.fc. A 'c2f' routine will also have three files: name.f, name.o and name.cf.

The help files are in subdirectory /help.

The source code for constructing utilities for handling MSIO files are in subdirectory /mstools.

The preliminary INDIGO version of IRMA is functionally the same as the IRIS 3030 version. The main differences are summarized below.

1) The wrapper routines are done differently. The FORTRAN to C routines are handled the same, but the C to FORTRAN wrappers are changed. To call a FORTRAN routine from C, you append an underscore to the routine name and all arguments are passed by reference (by placing an ampersand in front of each argument)

2) You no longer have to be in display mode to turn on the z buffer. And the z buffer works while an object is being moved, instead of toggling off immediately with the detection of any queue event.

3) The windows are much more flexible, since they can be moved and resized using the 4Dwm control bar menus. The display mode is still active, but it simply clears away the status and slider bar windows and does not modify the display window.

4) There is a test 'zrgb' option in the toggles menu which enables one to enter a graphics state that is double buffered, z buffered and is in rgb color mode. This option will not be supported on all machines. To exit this mode, hit the 'escape' key.

5) There is an enhanced Makefile in directory /irma. Type 'make' in this directory and all required source files will be compiled and linked, including the external libraries.

6) The SOCRATES binary files (.RST) can be read from within irma. The source code for this option is in directory /socplot.

APPENDIX G. SAMPLE PLOTS

The following plots give a sample of the IRMA capabilities.

Plot 1 (Figure 1):	The IRMA set up mode configuration.
Plot 2 (Figure 2):	The POLAR code model of the DMSP satellite.
Plot 3 (Figure 3):	The integrated column density for a SOCRATES simulation of a perpendicular burn of the RCS engines.
Plot 4 (Figure 4):	The POLAR code depiction of the plasma sheath around the SPEAR rocket. The curved line represents an electron trajectory.

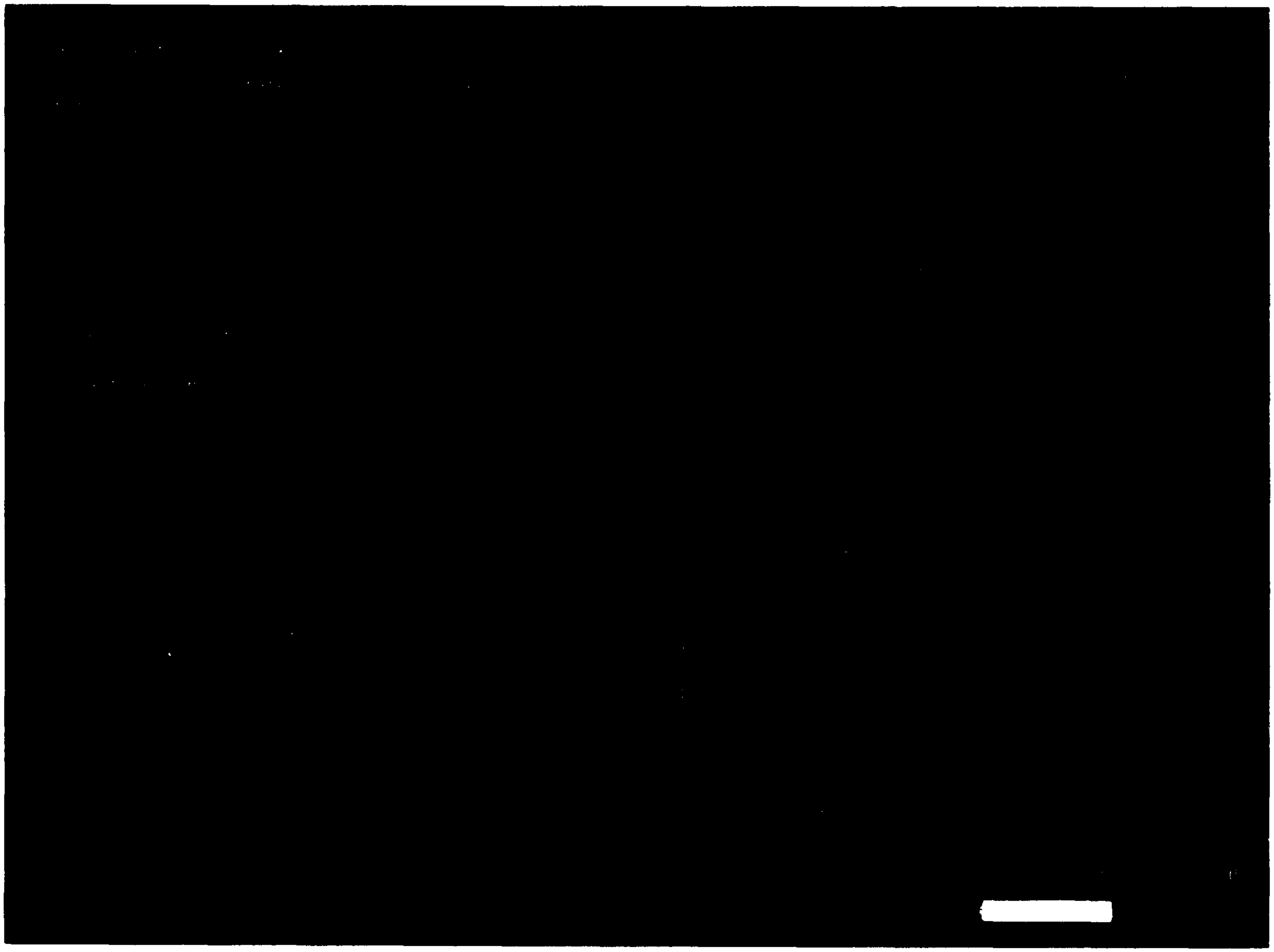

Figure 1. The IRMA set up mode configuration. The POLAR micro shuttle model is in the display window.

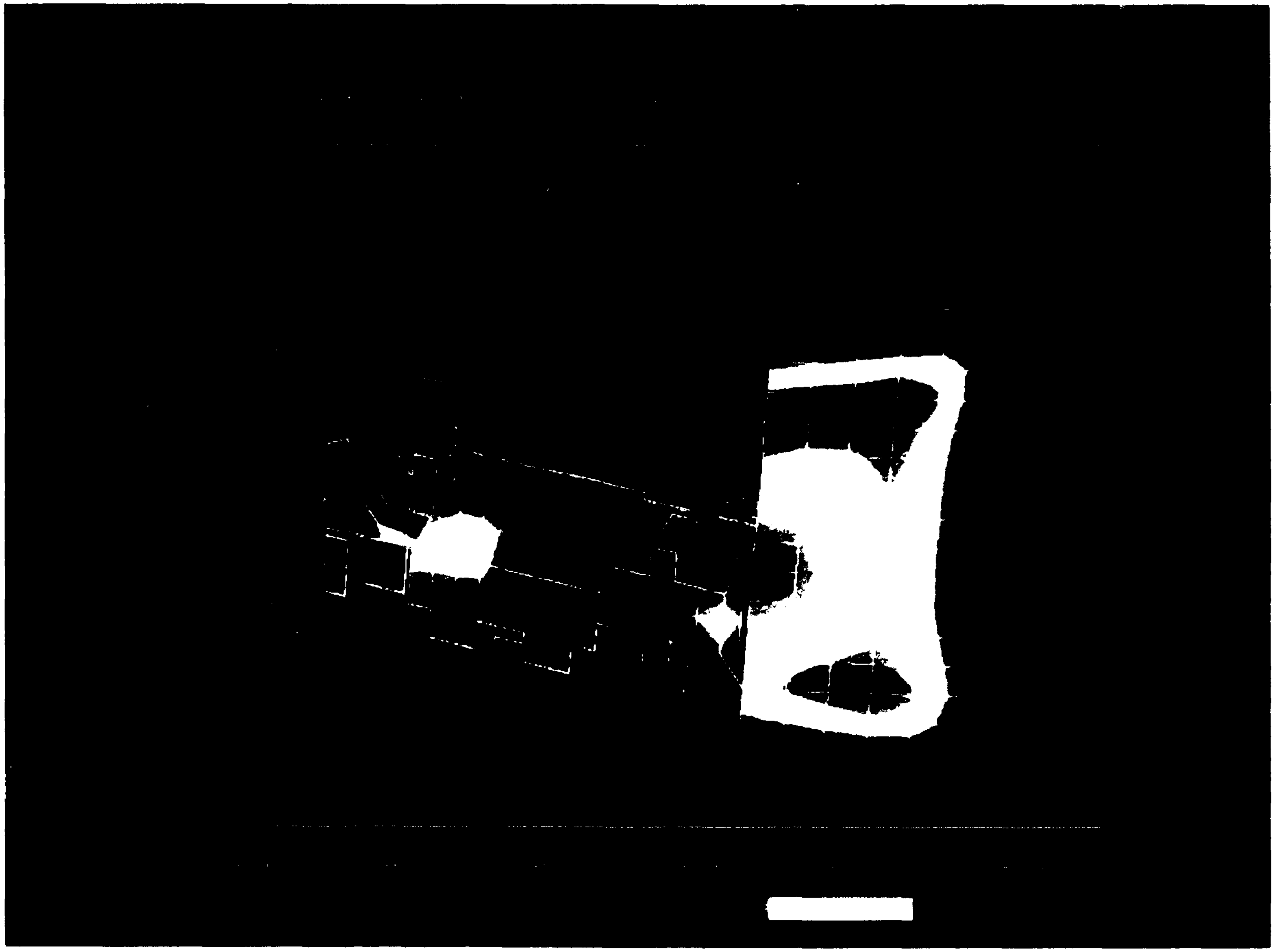

Figure 2. The POLAR code model of the DMSP satellite. The colored coded surfaces represent surface charging.

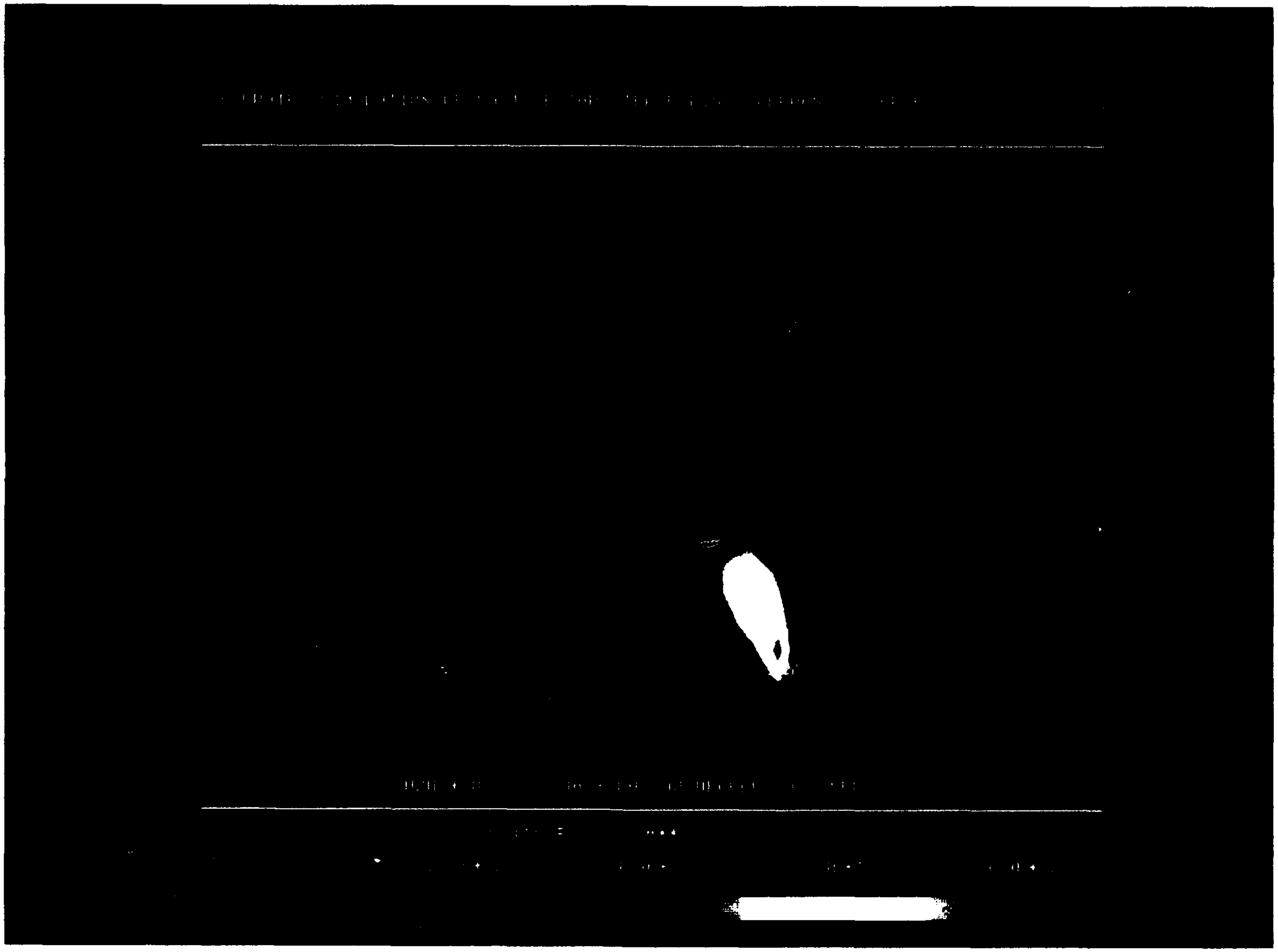

Figure 3. The integrated column density for a SOCRATES simulation of a perpendicular burn of the RCS engines.

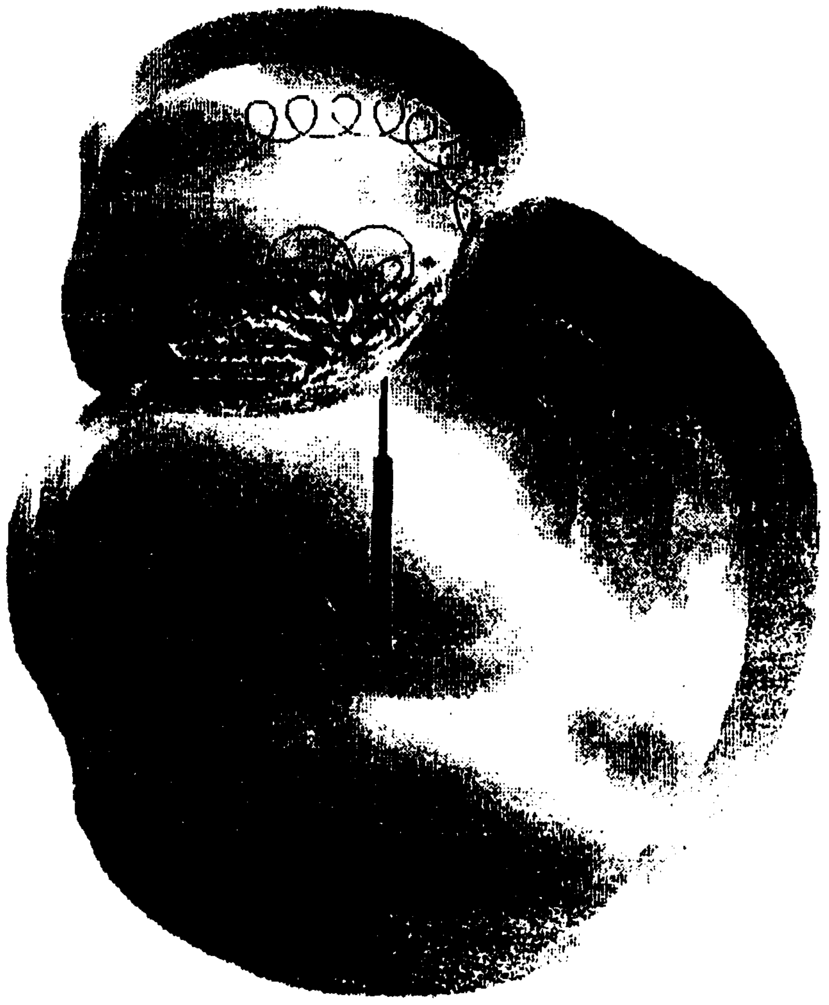

Figure 4. The POLAR code depiction of the plasma sheath around the SPEAR rocket. The 'sawtooth' black and white color map was used to produce the forward light shading.