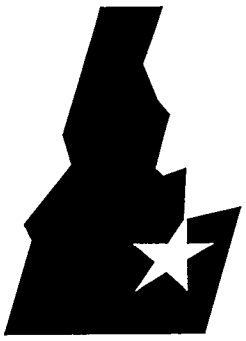
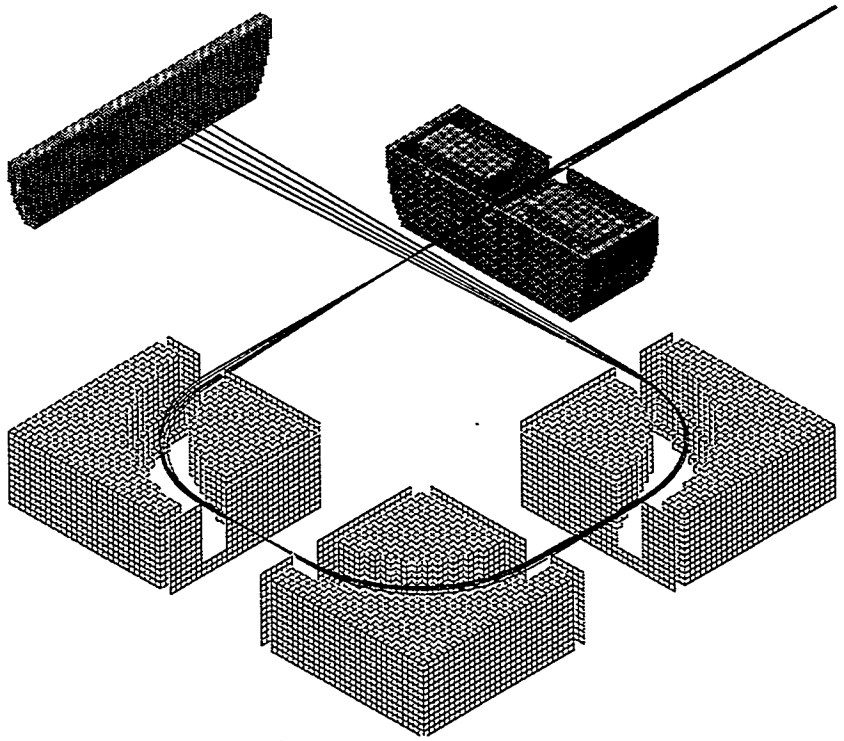


Idaho National Engineering Laboratory

INEL-95/0403
Formerly EGG-CS-7233
Rev. 4
August 1995

SIMION 3D VERSION 6.0 USER'S MANUAL

David A. Dahl



Lockheed Idaho Technologies Company

DISCLAIMER

INEL-95/0403
Formerly EGG-CS-7233
Rev. 4

SIMION 3D
VERSION 6.0
USER'S MANUAL

David A. Dahl

Published 1995

Idaho National Engineering Laboratory
Chemical Materials & Processes Department
Lockheed Idaho Technologies Company
Idaho Falls, ID 83415

Prepared for the
U. S. Department of Energy
Office of Energy Research
Under DOE Idaho Operations Office
Contract DE-AC07-94ID13223

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

The original SIMION was an electrostatic lens analysis and design program developed by D. C. McGilvery at Latrobe University, Bundoora Victoria, Australia, 1977. SIMION for the PC, developed at the Idaho National Engineering Laboratory, shares little more than its name with the original McGilvery version. INEL's fifth major SIMION release, version 6.0, represents a quantum improvement over previous versions. This C based program can model complex problems using an ion optics workbench that can hold up to 200 2D and/or 3D electrostatic/magnetic potential arrays. Arrays can have up to 10,000,000 points. SIMION 3D's 32 bit virtual Graphics User Interface provides a highly interactive advanced user environment. All potential arrays are visualized as 3D objects that the user can cut away to inspect ion trajectories and potential energy surfaces. User programs have been greatly extended in versatility and power. A new geometry file option supports the definition of highly complex array geometry. Extensive algorithm modifications have dramatically improved this version's computational speed and accuracy.

A NOTE TO NEW USERS

Welcome to the SIMION user family. Hopefully, SIMION 3D will prove useful for your needs. ***If you publish work that makes use of this program an acknowledgment would be appreciated.*** Note: In spite of the fact that you may have paid money for this program, remember that it wasn't paid to me. ***This is <u>not</u> the place to call for routine SIMION software support!*** However, if you have serious problems or suggestions feel free to contact me (*Warning: I may not choose to be grateful*).

David A. Dahl
MS 2208
Idaho National Engineering Laboratory
P.O. Box 1625
Idaho Falls, ID 83415

(***important issues only!***)
e-mail dhl@inel.gov
(208) 526-9915
Fax (208) 526-8541

***** WARNING *****

Acknowledgments

Projects of this sort generally get out of hand and become massive tasks. This is a good description of my experience with SIMION 3D Version 6.0. I would like to extend sincere appreciation to Anthony D. Appelhans for his valuable observations and careful review of this document. Second, I extend sincere thanks to all who Beta tested Version 6.0. I am particularly grateful for the Beta testing of: Bruno W. Schueler, Jon Denore, Dennis J. Chornay, Donald David, Stu Hansen, Richard Enck, Scott Bentley, and Nyle Utterback. You made a difference. Further, I would like to thank all the users of the INEL's previous SIMION versions in the U.S. and beyond. The fact that so many of you have found this program useful has in large measure provided me with the motivation to attack an epic project like SIMION 6.0.

Credit

This work was supported by the U.S. Department of Energy INEL internal research funds and by the Division of Chemical Sciences, Basic Energy Sciences, Office of Energy Research, Department of Energy under contract 3ED102.

Summary Table of Contents

Chapter One — *SIMION 6.0 Overview*

Chapter Two — *SIMION Basics*

Chapter Three *Useful Directions*

Chapter Four *Creating, Loading, and Saving Potential Arrays*

Appendix G — *Printing Options*

SIMION 3D Version 6.0 Overview

Introduction

SIMION is a PC based ion optics simulation program that models ion optics problems with 2D symmetrical and/or 3D asymmetrical electrostatic and/or magnetic potential arrays. It incorporates an ion optics workbench strategy that allows you to size, orient, and position up to 200 instances (*3D images - or mirages*) of these potential arrays within a workbench volume of up to 8 cubic km. Complex systems or even entire instruments can be modeled. Ions can be flown singly or in groups, displayed as lines or flying dots, and automatically be re-flown to provide movie effects when needed. Other features include data recording, charge repulsion, user programs, and geometry files. The result is a program that can model a wide range of problems including: Ion source and detector optics, time-of-flight instruments, ion traps, quadrupoles, ICR cells, or what have you.

No program can be all things to all people. SIMION 3D 6.0 is intended to provide direct and highly interactive methods for simulating a wide variety of general ion optics problems. The program balances of ease-of-use, speed, and accuracy to enable it to support many real-world applications. For example: It has successfully simulated the Phi-Evans TOF instrument using voltages that are within a few percent of the as-built instrument. Even if you just use it as a scoping tool (*saving the hard-to-use heavy artillery for later*), it can provide useful insights into your problems and perhaps help speed you toward to your final goal.

The Simulation Trap

Any simulation is only as good as the understanding that goes into it (*e.g. garbage in garbage out*). I have witnessed many distressing examples of SIMION being used in blind faith. Just because you manage to get a few ions through doesn't mean the design's OK. Are the ions truly representative? Are the fields modeled at all realistically? Do you really understand the physics of your problem? ***Just because a program is easy-to-use doesn't mean it thinks for you!*** You may get excited about a new concept, but wait until the design is built and tested successfully before declaring victory! ***Always be suspicious***.

Computer Requirements

The following summarizes the computer requirements for SIMION 3D 6.0. ***Appendix A gives more details and recommendations. Appendix B provides installation instructions.***

This version of SIMION requires at a minimum: A **386 class PC** ***with numerical coprocessor*** (*or above - e.g. **Pentium or P6 recommended***), **8 megabytes of RAM** (*16 or more recommended for large projects*), and at least **50 megabytes of free hard disk** drive (*or more*). The program makes use of a **32 bit DOS virtual memory GUI** (*Graphics User Interface - developed by the author*) that runs in, **DOS, Windows** (*3.xx, 95, NT - Intel machines*) **and OS/2** (*inc. Warp*). The GUI video drivers support: **VGA and SVGA** (*VESA BIOS - up to 1280x1024*). GUI printer/plotter drivers support: **PostScript** (*B/W and color*), **PCL5** (*B/W and color*), **HPGL2, and HPGL.**

Warning to Users of Prior SIMION Versions

Every attempt has been made to maintain logical continuity with earlier versions of SIMION. However, certain things have changed in the name of progress. ***It is important that you take the time to scan the manual.*** The following are only two examples of many significant differences:

1. In **Modify**, clicking the **right** mouse button *zooms* the view *instead* of *filling* the marked area. *The* **<Ctrl> right** *mouse button is now equivalent to the old* **right** *mouse button.*
2. No earlier version file formats are compatible with SIMION 6.0. *The only file formats that SIMION converts are two types of potential array files* (**.PA** *and* **.PA#**).

Remember: If All Else Fails, Read the Directions!

Structure of this Manual

Writing a manual to describe all the features of SIMION is a challenging task, because programs like SIMION tend to have a spatial organizational structure (*features are interrelated and cross-connected, much like the neurons in your brain*). ***The problem is: Where to start, what path to take, and what to describe along the way?***

The selected approach uses logical topic areas and the interrelationships between these topic areas to set the manual's structure. Each chapter and appendix deals with one or more related topics. Early chapters (*e.g. Chapter 2 - SIMION Basics*) attempt to give you a foundation for understanding SIMION. Later chapters (*e.g. Chapter 5 - Defining and Editing Array Geometry*) address specific tasks in detail. Appendices are reserved for infrequently referenced material (*e.g. Appendix B - Installing SIMION*) or advanced features of the program (*e.g. Appendix I - User Programming*). ***Take the time to thumb through the manual now to know where to look later.***

How to Proceed

Installing SIMION

Read Appendix A first to determine if your hardware is suitable. ***Next, read Appendix B***, select the version of SIMION to install (*there are two*), and follow the installation instructions for installing the selected SIMION version to run within the desired environment (*e.g. Windows*).

Learning SIMION

The best way to learn to use SIMION is to dive right in. First, read Appendix F - SIMION's GUI and scan Appendix G - Printing Options. Next, use Appendix C to guide you through a few SIMION sample runs. **Remember, if you have a question about something, point your cursor to it and click the F1 key** (*for specific help*).

At this point you should read Chapter 2 - SIMION Basics carefully to learn its structure and relationships. Chapter 3 acts as a signpost to the rest of the manual. Chapters 4-8 should now be scanned. They are organized as reference chapters for topics ranging from array creation to ion flying. ***Be sure to examine the many examples in the demo directories*** (*below* **\SIM6**).

Chapter 9 contains some advanced strategies and tactics to help get the most out of SIMION. This chapter also introduces SIMION's advanced features of user programming and geometry files (*each have their own reference appendix*). ***This is where the real fun begins.***

SIMION Basics

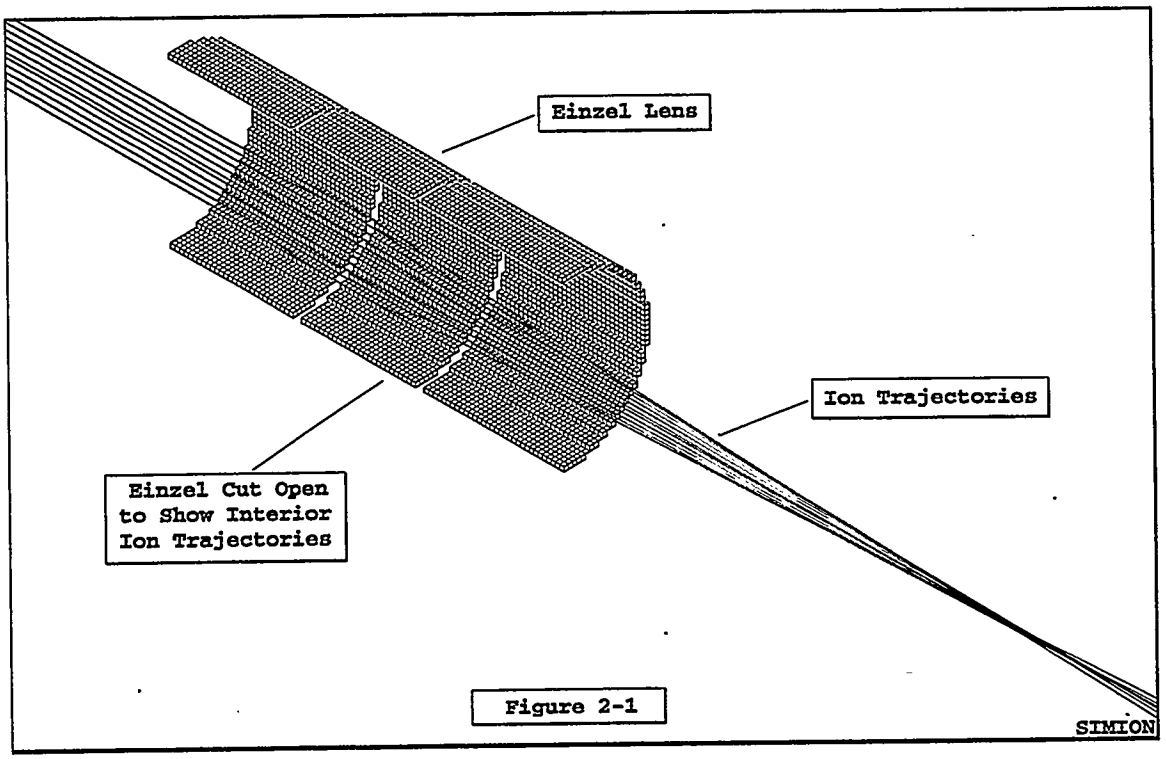


Figure 2-1

Introduction

SIMION makes use of potential arrays that define the geometry and potentials of electrodes and magnetic poles. The potentials of points outside electrodes and poles are determined by solving the Laplace equation by finite difference methods (*see Appendix E - Computational Methods*). In SIMION, this process is called ***refining*** the array. Refined arrays can then be projected as instances (*3D virtual images*) into an ion optics workbench volume. Ions can be flown within the workbench volume and their trajectories changed by the fields of the potential array instances they fly through.

The illustration above (*figure 2-1*) shows ions flying through a simple einzel lens. The einzel lens has been cut open (*a SIMION display trick*) to show the ion trajectories within the einzel lens. The einzel lens has been created using a simple 2D cylindrical array. Cylindrical 2D arrays form volumes of revolution about their x-axis when refined and displayed in the **View** function.

While this view of the ion trajectories is useful, it doesn't really explain why the ions are focusing. We can use SIMION's potential energy view (*PE View*) of the einzel lens to help us understand the electrostatic focusing process. Figure 2-2 below shows a potential energy view of the einzel lens trajectories above. Note that the potential energy surface is much like the surface of a golf course. Since ions react in much the same way to potential energy surfaces as golf balls react to hills and valleys it is quite easy to see why ions have the trajectories they do in this einzel lens. ***SIMION's capabilities are designed to develop intuition and promote understanding.***

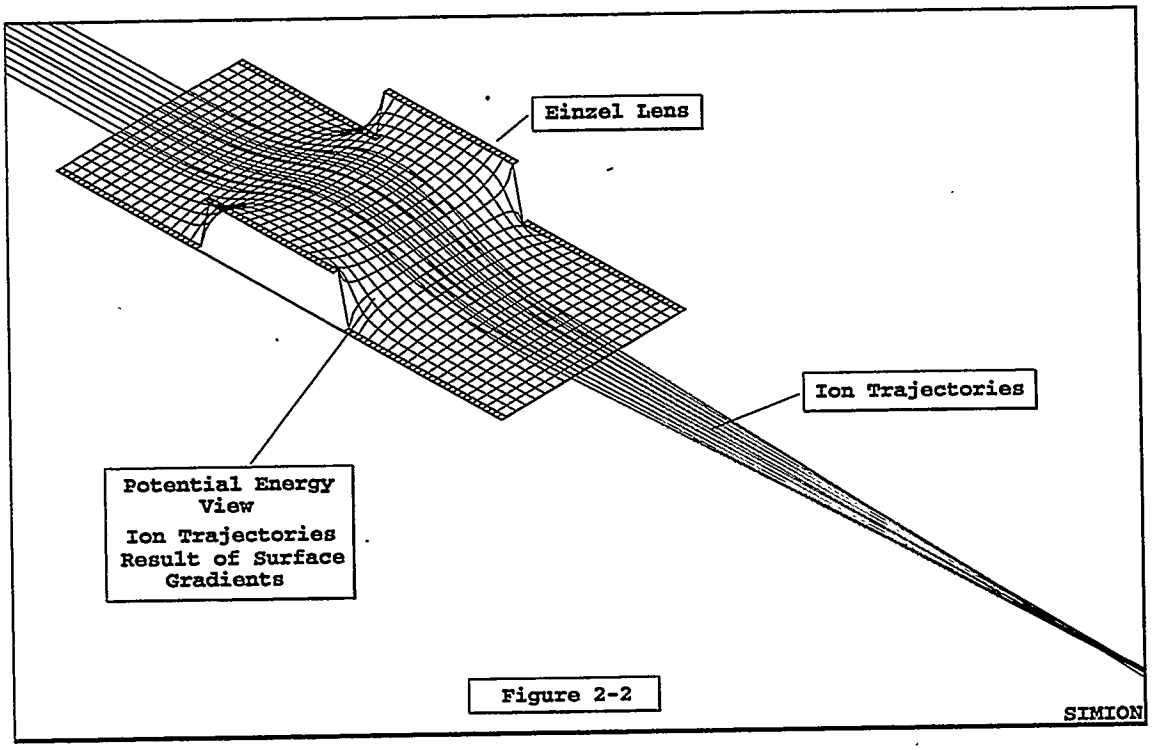


Figure 2-2

Some Basic Ion Optics Concepts

Ion optics utilize the electrostatic and/or the magnetic forces on charged particles to modify ion trajectories. The following is a short (*and simplified*) introduction to ion optics concepts. ***It is intended to introduce rather than educate.***

Equations From Basic Physics:

$$\text{Force} = \text{Mass} \times \text{Acceleration}$$

$$F = M A$$

$$\text{Work} = \text{Force} \times \text{Distance}$$

$$\text{Force}_{avg} = \text{Work/Distance}$$

$$F = dW/dr$$

Coulomb's Law:

$$F_e = k Q_i Q / r^2 \quad \{\text{Point charge}\}$$

$$F_e = Q_i \, \text{Sum}_n (k Q_n / r_n^2) \quad \{\text{Point charges}\}$$

$$Q_i = \text{The Ion's Charge}$$

$$E = F_e / Q_i = d(W/Q_i) / dr \quad \{\text{Electric field intensity}\}$$

$$E = \text{Volts / Meter}$$

$$\text{Volts} = \text{Joules / Coulomb}$$

$$F_e = -e\,E \quad \{e = \text{units of positive charge}\}$$

Electrostatic Equation of Motion:

$$A = F / M$$

$$A = dv / dt = -(e\,E) / m = -E / (m / e)$$

Magnetic Force Equation:

$$F_m = Q_i\,(U \times B) \quad \{\text{Vector Cross Product}\}$$

$$Q_i = \text{The Ion's Charge}$$

$$U = \text{Vector Velocity of Ion}$$

$$B = \text{Vector Magnetic Field Intensity}$$

Magnetic force is always normal to both the B magnetic field vector and the U velocity component normal to the B magnetic field vector (*following the right hand rule*):

$$U \times B = (U_y B_z - U_z B_y) i_x + (U_z B_x - U_x B_z) i_y + (U_x B_y - U_y B_x) i_z$$

A simpler equation results from using ***only*** the vector velocity component normal to the magnetic field:

$$F_m = Q_i\,U_n\,B \quad \{F_m \text{ normal to both } U_n \text{ and } B\}$$

$$U_n = \text{Velocity component normal to B}$$

Static Magnetic Equation of Motion:

$$A_n = F / M$$

$$A_n = (dv/dt)_n = (e\,U_n\,B) / m = U_n\,B / (m / e)$$

Note: Static magnetic fields change an ion's direction of motion but not its speed (*kinetic energy*).

Refraction in Ion Optics:

Refraction (*bending of ion trajectories*) results from electrostatic and/or magnetic forces normal (*at 90 degrees*) to the ion's velocity.

The Electrostatic Radius of Refraction

Normal Electrostatic Force = Centripetal Acceleration

$$-e E_n = m v^2 / r_n$$

$$r_n = m v^2 / (-e E_n) = -(m/e) v^2 / E_n$$

The Magnetic Radius of Refraction

Normal Magnetic Force = Centripetal Acceleration

$$B_n e v = m v^2 / r_n$$

$$r_n = m v / (B_n e) = (m/e) v / B_n$$

Interpretations of Radius of Refraction:

1. The electrostatic radius of refraction is proportional to the ion's kinetic energy per unit charge. Thus all ions with the same starting location, direction, and kinetic energy per unit charge will have identical trajectories in electrostatic (*only*) fields. ***Trajectories <u>are not</u> mass dependent in electrostatic fields.***

2. The magnetic radius of refraction is proportional to the ion's momentum per unit charge. Thus all ions with the same starting location, direction, and momentum per unit charge will have the identical trajectories in static magnetic (*only*) fields. ***Trajectories <u>are</u> mass dependent in static magnetic fields.***

3. Because of the v verses v^2 effects on the radius of refraction, ***magnetic ion lenses have superior refractive power at high ion velocities.***

Light Optics Verses Ion Optics

There are significant differences between light and ion optics:

Radius of Refraction

Light optics make use of sharp transitions of light velocity (*e.g. lens edges*) to refract light. These are very sharp and well defined (*by lens shape*) transitions. The radius of refraction is infinite everywhere (*straight lines*) except at transition boundaries where it approaches zero (*sharp bends*).

Ion optics makes use of electric field intensity and charged particle motion in magnetic fields to refract ion trajectories. This is a distributed effect resulting in gradual changes in the radius of refraction. ***Desired electrostatic/magnetic field shapes are much harder to determine and create since they result from complex interactions of electrode/pole shapes, spacing, potentials and can be modified significantly by space charge.***

Energy (Chromatic) Spreads

Visible light varies in energy by less than a factor of two.

Ions can vary in initial relative energies (*or momentum for magnetic*) by orders of magnitude. ***This is why strong initial accelerations are often applied to ions to reduce the relative energy spread.***

Physical Modeling

Light optics can be modeled using physical optics benches (*interior beam shapes can be seen with smoke, screens, or sensors*).

Ion optics hardware is generally internally ***inaccessible*** and must normally be evaluated via end to end measurements. ***Numerical simulation programs like SIMION allow the user to create a virtual ion optics bench and look inside much like physical light optics benches.***

Determining Field Potentials

The electrostatic or magnetic field potential (*e.g. Volts or Mags - SIMION's magnetic potentials*) at any point within an electrostatic or static magnetic lens can be found solving the Laplace equation with the electrodes (*or poles*) acting as boundary conditions. The Laplace equation assumes that there are ***no*** space charge effects.

The Laplace Equation

$$DEL^2\ V = DEL \cdot DEL\ V = 0$$

$$DEL\ V = (dV / dx)_i + (dV / dy)_j + (dV / dz)_k = E$$

$$DEL^2\ V = DEL \cdot E = dE_x / dx + dE_y / dy + dE_z / dz = 0$$

The Laplace equation constrains all electrostatic and static magnetic potential fields to conform to a zero charge volume density assumption (*no space charge*). ***This is the equation that SIMION uses for computing electrostatic and static magnetic potential fields.***

Poisson's Equation Allows Space Charge

$$DEL^2\ V = DEL \cdot DEL\ V = -\ p / e$$

Poisson's equation allows a non-zero charge volume density (*space charge*). When the density of ions becomes great enough (*high beam currents*) they will (*by their presence*) significantly distort the electrostatic potential fields. In these conditions, Poisson's equation should be used (*instead of Laplace's*) to estimate potential fields. ***SIMION does not support Poisson solutions to field equations. It does however employ charge repulsion methods that can estimate certain types of space charge and particle repulsion effects.***

The Nature of Solutions to the Laplace Equation

The Laplace equation really defines the electrostatic or static magnetic potential of any point in space in terms of the potentials of surrounding points. For example, in a 2-dimensional electrostatic field represented by a very fine mesh of points the Laplace equation is satisfied (*to a good approximation*) when the electrostatic potential of any point is *estimated* as the average of its four nearest neighbor points:

$$V = (V_1 + V_2 + V_3 + V_4) / 4$$

Physical Models of the Laplace Equation Solutions

Physical models (*as opposed to numerical models*) have historically been used to solve many Laplace equation problems. These models have the advantage of giving physical insight into problems that can be difficult to understand.

Rubber-Sheet Models

Laplace equation solutions for electrostatic potential fields resemble surfaces of rubber-sheets stretched between electrodes where electrode height represents electrostatic potential. In fact, relatively flat rubber-sheet surfaces are good physical representations for electrostatic fields. Rubber-sheet models and marbles (*for ions*) have been used to model electrostatic fields (*e.g. early vacuum tube design*).

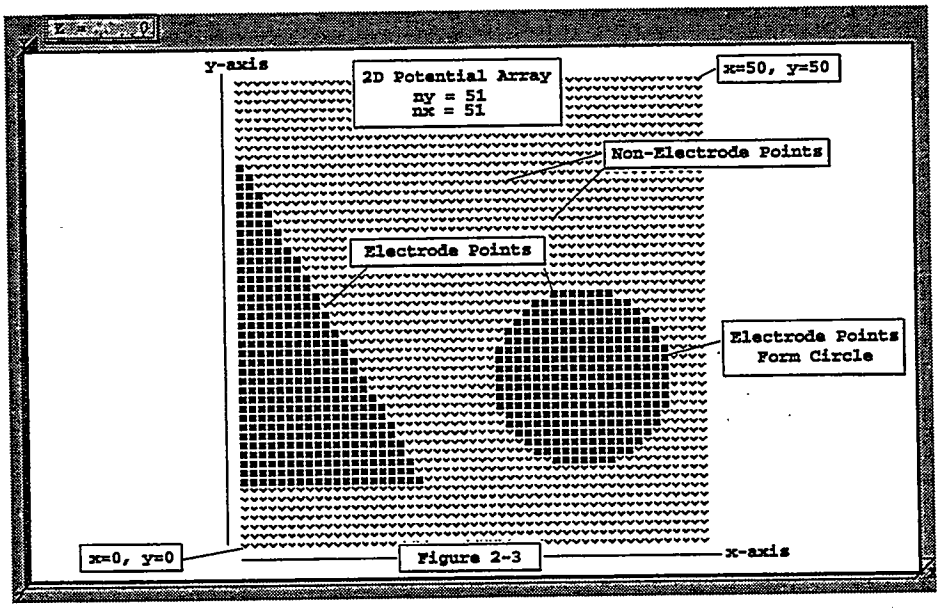
Resistance Paper Models

Resistance paper models have also been used. In this method electrodes of the designed shapes are pressed against paper having a uniform surface resistivity. Voltages are applied to the electrodes and electrostatic potential measurements are taken with a voltmeter. These measurements are then used to calculate ion trajectories.

Pros and Cons of Rubber-Sheet Models

Rubber-sheet models give the designer excellent insight into the behavior of ions in electrostatic fields. This is because we humans have each developed a certain physical intuition for the movement of balls on sloping surfaces (*e.g. miniature golf*). ***Note: SIMION's potential energy surfaces (figure 2-2) use the rubber-sheet style of display to assist the user in developing intuition and understanding.***

Unfortunately rubber-sheet models are difficult to build. Moreover, they are limited in their modeling accuracy because forces are proportional to the sine of the slope (*e.g. g dh/dr*) rather than the tangent of the slope (*e.g. dV/dx*) needed in electrostatic ion optics.



The Potential Array

SIMION utilizes potential arrays to define electrostatic and magnetic fields. ***A potential array can be either electrostatic or magnetic but not both***. If you require both electrostatic ***and*** magnetic fields in the same volume, instances of electrostatic ***and*** magnetic arrays must be ***superimposed*** in the workbench.

A potential array is an array of points organized so the points form equally-spaced square (*2D*) or cubic (*3D*) grids. ***Equally-spaced means that all points are equal distances from their nearest neighbor points.*** Figure 2-3 (*above*) shows a **Modify** function view of a 51 by 51 2D potential array.

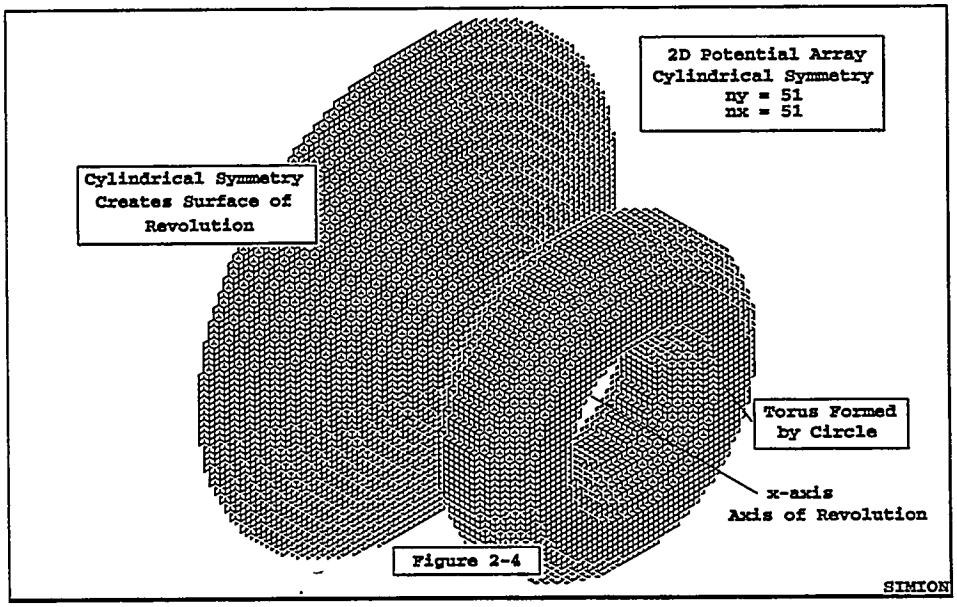
All points have a potential (*e.g. voltage*) and a type (*e.g. electrode or non-electrode*). Certain points are flagged as electrode or pole points. Points of type electrode/pole create the boundary conditions for the array. ***Groups of electrode points create the electrode and pole shapes (the finer the grid the smoother the shapes).*** Non-electrode and non-pole points within the array represent points outside electrodes and poles. We need to solve the Laplace equation to obtain the potentials of these points (*the non-electrode and non-pole points).*

Potential Array Dimensions

Potential arrays are dimensioned by the number of points in each dimension (*x, y, and z*). Therefore an array of nx = 51 would have 51 points in the positive x direction. All arrays have their lower left corner at the origin (*xmin = ymin = zmin = 0*). Thus if nx equals 51, xmin equals 0 and xmax equals 50 (*a width in x of 50 grid units*).

All 2D arrays have nz = 1 (*zmin = zmax = 0*). Thus 2D arrays are always located on the z = 0 xy-plane.

Arrays can use a lot of memory. A 100x by 100y by 100z 3D potential array requires one million points. Each point requires 10 bytes of RAM. ***Thus 10 megabytes are required for each million array points.***

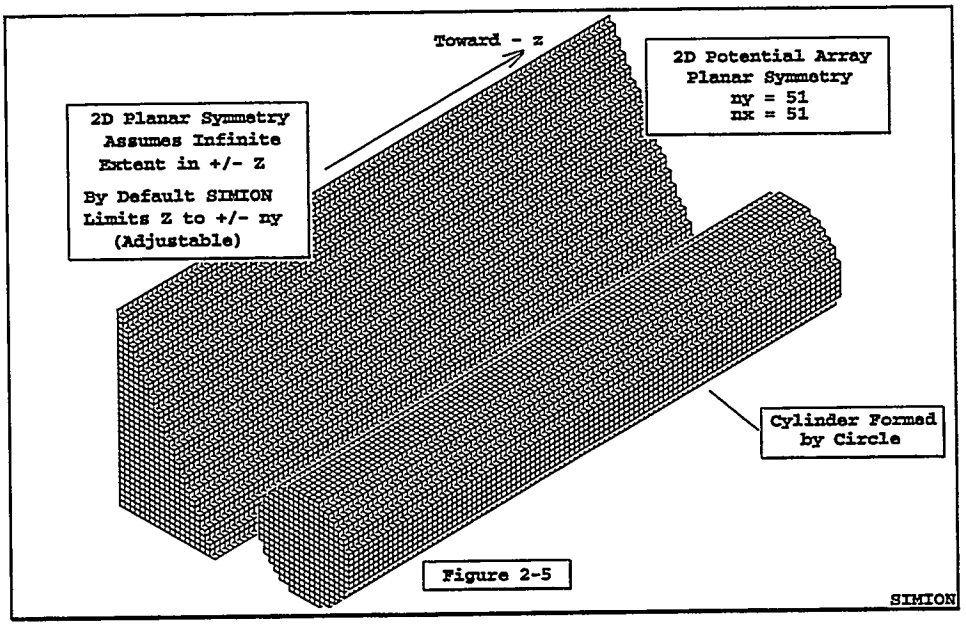


Potential Array Symmetries

Two potential array symmetries are supported: Planar (*2D and 3D arrays*) and cylindrical (*2D arrays only*). When SIMION refines (*solves the Laplace equation*) or projects an instance (*3D image*) of an array into its ion optics workbench it uses the array's symmetry. If the 2D array has a cylindrical symmetry it will be projected as a surface of revolution about the x-axis. Figure 2-4 (*above*) shows an isometric 3D view of the array in figure 2-3 assuming it has cylindrical symmetry. ***Note: The cylindrical 2D array is visualized as if it were 3D planar to take***

advantage of fast planar visualization methods. However, apertures are <u>really</u> round (despite how they appear) and the <u>cylindrical symmetry is fully retained for trajectory calculations</u>.

Figure 2-5 (*below*) shows an isometric 3D view of the array in figure 2-3 assuming it has planar symmetry:



Note that 2D planar symmetry assumes that the array ***itself*** is only one layer of an infinite number of layers in z (*for **refining** purposes*). When SIMION projects a 2D planar array as an instance within the workbench volume it assumes a z depth of +/- ny (*grid units*). ***You may edit the instance definition to <u>increase</u> this z depth.*** This option allows a 2D planar array to be used to represent the interior (*symmetrical*) portions of objects like quadrupole rods (*saves array space*).

The Use of Array Mirroring

Many electrode designs have a natural mirrored symmetry. Designs that mirror in y have a mirror image of the electrode/poles in the negative y. SIMION supports three mirroring symmetries: x, y, and z. Mirroring allows you to use a ***smaller*** array to model a larger area (*or volume*) when conditions permit. For example: A 3D planar array can be mirrored in x, y, and/or z. If the design symmetries permit, this would allow the modeling of a 3D volume with one eighth the number of points required if no mirroring were utilized. ***SIMION takes mirroring into account when refining arrays and projecting their instances into the workbench volume.***

X mirroring is ***allowed*** for all 2D and 3D arrays. Y mirroring is ***required*** of 2D cylindrical and ***allowed*** in 2D and 3D planar arrays. ***Z mirroring is <u>only</u> allowed in 3D planar arrays.***

Figure 2-6 (*below*) shows the potential array of figure 2-3 projected into the workbench assuming cylindrical symmetry with x and y mirroring.

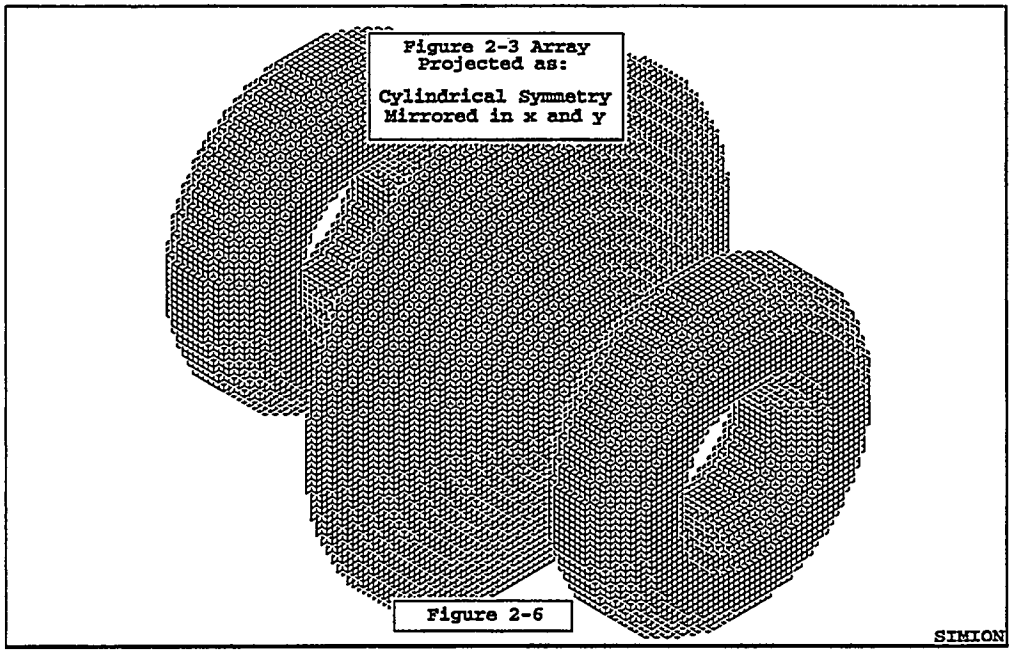
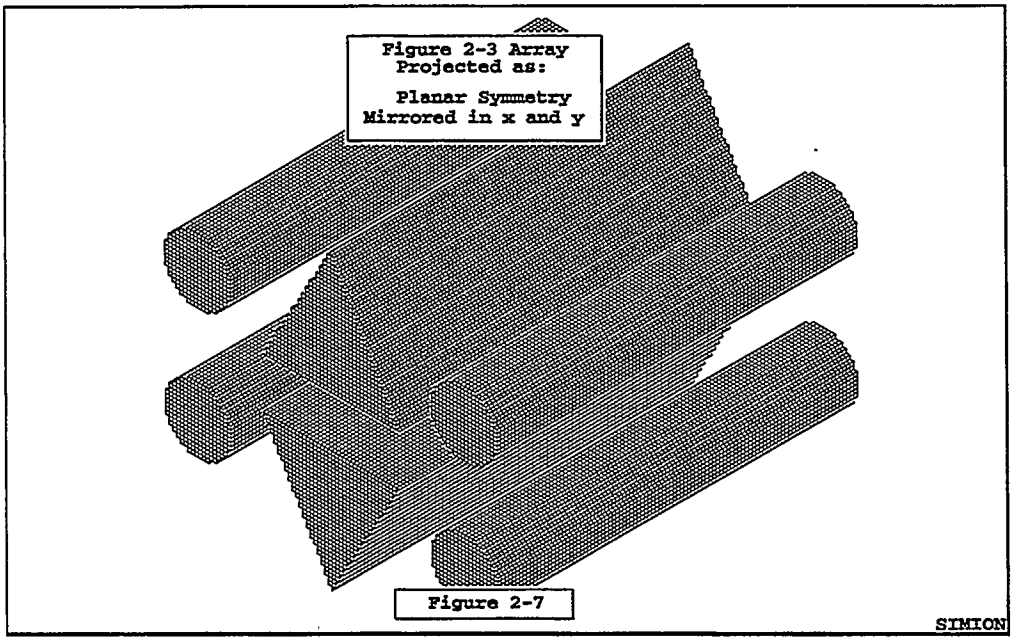


Figure 2-7 (*below*) shows the potential array of figure 2-3 projected into the workbench assuming planar symmetry with x and y mirroring.



Potentials and Gradients in Potential Arrays

SIMION uses the same methods for refining both electrostatic and magnetic potential arrays. However, the potentials and gradients used are different and it is important that you understand these differences.

Potentials and Gradients in Electrostatic Arrays

Potentials in electrostatic arrays are ***always*** in volts. SIMION uses the refined array potentials to determine field gradients (*voltage gradients*). Electrostatic field gradients are ***always*** in volts/mm. When a instance of a potential array is projected into the workbench volume it is scaled by a user specified number of millimeters per grid unit (*or defaulted to 1 mm/grid unit*). So although electrostatic gradients start out as volts/grid unit they are divided by the instance scaling factor to obtain volts/mm:

Gradient = PA Gradient / Instance Scaling

Volts / mm = (Volts / grid unit) / (mm / grid unit)

Thus instance scaling can have a dramatic impact on electrostatic gradients and therefore ion accelerations.

Potentials and Gradients in Magnetic Arrays

SIMION is not a magnetic circuit program. You have to supply the magnetic potentials for it to refine. Unlike electrostatics, in magnetics we normally think of ***and measure*** gradients or more precisely flux (*gauss*) as opposed to potentials. This presents a problem, because SIMION needs magnetic potentials to refine.

In order to deal with this dilemma SIMION defines magnetic potentials in Mags. *Mags* are defined to be ***gauss times grid units.*** The gradient of magnetic potential is gauss. ***Note: Mags are gauss times grid units instead of gauss times millimeters.*** If Mags were gauss times millimeters then instance scaling would confound us further. With this approach, magnetic fields remain the same whether the array is scaled to be a certain size or 10 times as large in the workbench volume. ***Thus instance scaling has no impact on the magnetic fields (flux in gauss) produced by magnetic potential arrays.***

SIMION also makes use of a magnetic scaling factor **ng** as a property of magnetic potential arrays. ***The* ng *scaling factor has been provided to further simplify your life.*** It would be very nice for Mags to be directly related to gauss. Let's say we have a simple two pole magnet: We would like to set one pole to 1000 Mags and the other to Zero Mags and have the field in between be approximately 1000 gauss. If the two poles are separated by let's say a 60 grid unit pole gap we would specify the value of 60 for **ng** to automatically scale the Mags potentials ***roughly*** into gauss.

PA Magnetic Flux = PA Magnetic Gradient (PA gauss)
Magnetic Flux = PA Magnetic Flux * ng

gauss = PA gauss * ng (pole gap scaling factor)

Note: The B field vector ***always*** points from greater magnetic potential (*e.g. 1000 Mags*) toward lesser magnetic potential (*e.g. 0 Mags*).

Beware! Magnetic potentials are not as simple as electrostatic potentials. While we can safely assume that all points of an electrode have the same electrostatic potential (*e.g. volts*), it is dangerous to assume that the same is generally true for magnetic poles. ***Magnetic poles do not as a rule have totally uniform magnetic potentials across their surfaces*** (*permeability*

not being infinite). Thus you must allow for this fact if the effects could be significant enough to impact your results. ***Remember: User beware.***

Three Ways to Adjust Array Potentials

We often need to change the electrode/pole potentials of an array to tune a lens or adjust a magnet's field. The following are the three different strategies supported by SIMION:

Modify the Potential(s) and Re-Refine

The first strategy is the brute force approach:

1. Use the **Modify** function to change the potentials of the points of one or more electrodes or poles in the potential array.
2. The use the **Refine** function to re-refine the potential array to obtain the resulting potentials of the non-electrode or non-pole points.

This is the **Modify-Refine** cycle of array potential adjustment. It is time consuming and invites errors (*e.g. accidentally not changing the potentials of all the points of an electrode or pole*).

Proportional Re-scaling of All Array Potentials (.PA arrays)

SIMION 6.0 supports proportional re-scaling of all **.PA** array potentials. This is useful in those cases when the potentials of all electrodes or poles can be changed proportionally to obtain the desired result. ***This approach works with any simple array (with .PA file extension) that has already been refined:***

1. Use the **Fast Adjust** function to change the potential of the highest potential electrode or pole in the potential array.
2. SIMION will automatically scale the potentials of all array points (*electrode/pole and non*) by the same proportion that you changed the potential of the highest potential electrode or pole.

This can be quite useful for a magnetic potential array with two poles and a gap. Proportional scaling provides a quick way to adjust the magnetic field. It even supports proportional scaling of non-uniform potentials on the pole surfaces (*fringe field effects*). ***This is a clever trick only if you can justify that these non-uniform pole surface potentials actually would scale proportionally in your problem.***

Using Fast Adjust Arrays (.PA# arrays)

The third approach supported by SIMION makes use of the additive solution property of the Laplace equation. This involves creating a separate array for each electrode we desire to adjust, setting the points of the desired electrode/pole to a fixed potential, and setting all other electrode/pole points to zero. Each of these separate electrode arrays is then refined separately and the results are saved on disk.

Composite fields are obtained by scaling each electrode's array to its desired voltage and adding the individual field contributions together to obtain the desired result. This is all quite fast (*avoids re-refining*) if you can keep track of all the book-work. ***Fortunately, SIMION is designed to do all the hard work for you!***

1. Use the **Modify** function to define the geometry of your electrodes. ***All points of the first adjustable electrode/pole must be set to exactly 1 volt/Mag.*** Likewise, all points of the ***second*** must be set to 2, and so on. ***Adjustable electrode/pole potentials must not be skipped (e.g. 1,2,4,5).*** Up to 35 adjustable electrodes can be defined in this manner

within a potential array. Non-adjustable electrodes or poles may also be defined providing they do not have the ***exact*** potentials from 1 to 35 (*e.g. 10.001 volts is OK*).

2. Save this potential array with a **.PA#** file extension to signal to SIMION that this is a Fast Adjust Definition File (*e.g. save as* **TEST.PA#**).

3. Refine the **.PA#** file. SIMION will ***recognize*** that this is a Fast Adjust Definition File, examine it, create each required electrode solution array, refine it, and save it to your disk automatically. SIMION first creates a base solution array **.PA0** (*e.g.* **TEST.PA0**). This array contains the potentials of all non-adjustable electrodes/poles. The **.PA0** array is called the Fast Adjust Array because this is the array you ***actually*** load and fast adjust. The individual electrode solution arrays have the extension **.PA1 - .PA9** and **.PAA - PAZ** as required by the number of adjustable electrodes defined (*electrode **one** is stored in a* **.PA1** *file, e.g.* **TEST.PA1**).

4. Use the **Fast Adjust** function on the **.PA0** file to set potentials . If you try to fast adjust a **.PA#** file SIMION will automatically load its **.PA0** file for adjustment. ***If you want to save the current potential settings of a* .PA0 *file between SIMION sessions, simply save the* .PA0 *file to disk.***

The **Fast Adjust** function is accessible from the Main Menu Screen or from within the **View** function (*even while ions are flying*).

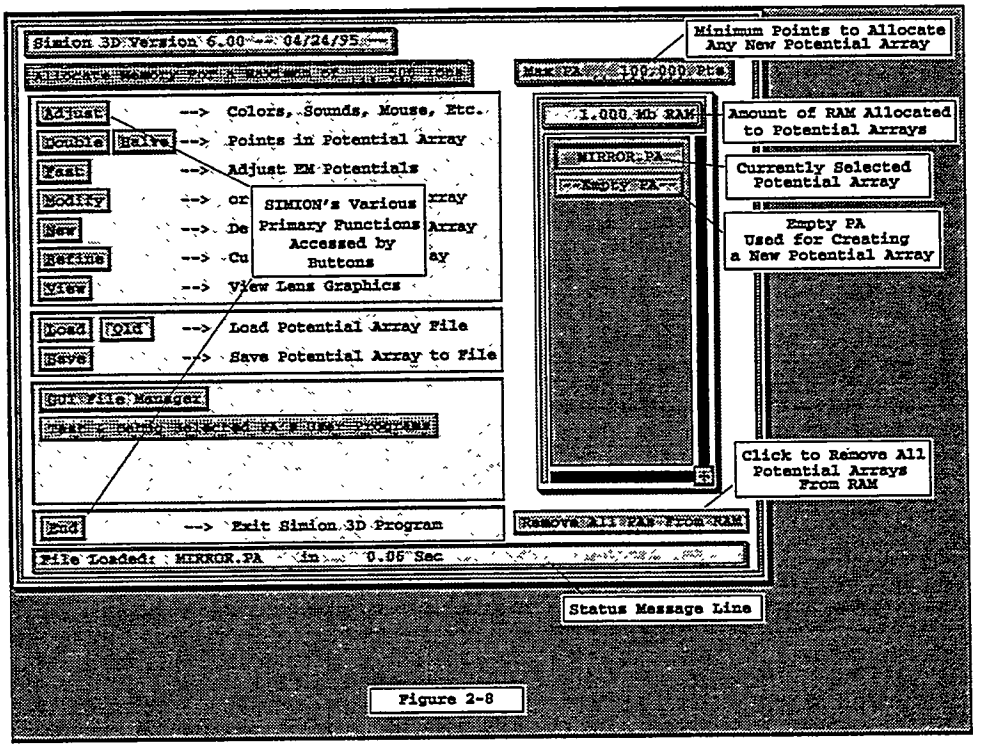


Figure 2-8

The SIMION Main Menu

Figure 2-8 (*above*) shows the SIMION Main Menu Screen. This is the first screen you will see in SIMION. It serves as the point of departure for all SIMION adventures. The buttons on the left allow you to access the various primary functions (*e.g.* **Modify**). ***Notice that the first letter in each button's label is underlined.*** This means that you can access a particular button by ***either*** clicking on it with the mouse ***or*** by entering the underlined key from the keyboard (*e.g.* **m** *for* **Modify**).

The window object on the right contains the list of potential arrays currently in RAM (*in this case, one:* **MIRROR.PA**). Other objects (*buttons and etc.*) support display and control potential array data, parameters, and fate (*more of this later*).

Adjusting User Preferences

The **Adjust** button allows you to adjust various SIMION and GUI characteristics. Clicking the **Adjust** button brings up a screen that allows you to change colors, sounds, blinking, other features (*e.g. mouse speed*), and video resolution. ***If you haven't explored* Adjust, *do it now.*** Descriptions of all options can be found in Appendix F.

The **Video Adjust** button allows you to select/change the displayed screen resolution. If your video card supports a VESA BIOS (*most new video cards do*) *and* the VESA BIOS is currently active (*loaded*), SIMION will scan your video card for what resolutions it supports and provide buttons for any screen resolution that its video drivers will support. ***If your VESA BIOS is not loaded or your video card doesn't support VESA BIOS you will be limited to VGA screen resolution.***

If you are running SIMION from within Windows or OS/2 you may have problems context switching between SIMION and the OS (*e.g.* **<Alt Tab>**) if your screen resolution is ***higher*** than VGA. This is because Windows and OS/2 video drivers are notoriously buggy. If you context switch back into SIMION and the screen is trashed, hold down the **Crtl** or **Alt** key and hit the **V** key (**<Ctrl V>** *or* **<Alt V>**) and SIMION will regenerate its current screen (*e.g. restore video mode, color palette, and current view*). This compensates for all but the most poorly written Windows and OS/2 video drivers.

The Role of the Subdirectory in SIMION Projects

SIMION requires that all files relating to a project (e.g. *.PA, *.IOB, *.FLY, and etc.) are contained in the same subdirectory of your hard disk. This is useful because it keeps things together and in so doing forces a touch of organization that many of us need so badly. Note: There can be more than one project in a subdirectory. However, this can often create a lot of clutter.

Each of the demos provided with SIMION is in its own directory below **\SIM6**. These demos help show how various projects might be approached.

The GUI File Manager can be used to quickly create subdirectories. ***Click on the parent directory, click the* Other *Button, enter the name of the new subdirectory, and press* Enter.** The subdirectory is created. Now click on this subdirectory to make it the currently active directory. ***Note: The GUI File Manager can also be used to copy or move files between directories and drives (Appendix F).***

Potential Arrays and RAM

SIMION maintains a ***working copy*** of each active potential array (*up to 200*) in memory (*RAM*). ***When you change something in a potential array you are only changing the memory copy.*** Likewise when you fly ions through instances of potential arrays in the workbench you are using the ***memory*** copies.

Potential arrays are normally saved as **.PA** or **.PA#** (*extension*) files in your project directory. When you create a new potential array ***only the in-memory copy is created.*** It is your responsibility to save

any new or changed potential arrays to your project directory as required. ***Saving potential arrays preserves your work between sessions.***

Allocating Memory for Potential Arrays

Potential arrays can use up a lot of memory. Each point of a potential array requires 10 bytes of RAM storage. Thus a 100 x by 100 y by 100 z 3D array has 1,000,000 points and requires 10 megabytes of RAM. ***SIMION allocates memory only once for each PA memory region it creates. Once the PA memory has been allocated it is not returned or changed until the* Remove All PAs from RAM *button is used to remove all memory allocated PAs (de-fragment the heap).*** This is done to prevent heap fragmentation lockups due to the large size of typical potential arrays.

SIMION normally allocates 100,000 points of memory for each new potential array. This is usually large enough to allow you to increase the array size (*e.g. via* **Modify** *or* **Double**) without exceeding the memory allocated for a PA. ***You have the option of adjusting this default PA memory allocation size up or down to suit your needs (with the* Max PA *panel object) before you create or load a potential array in an empty PA memory region.***

Deallocating Potential Array Memory

The main menu screen has a **Remove All PAs From RAM** button. This button is used to remove all working copies of PAs from RAM. ***Its primary function is to restore a clean slate when you are about to start a new project or the clutter of PAs has become unmanageable.***

The List of Potential Arrays

The Main Menu Screen (*Figure 2-8*) contains a window with a button for each currently allocated PA memory region and the name of the potential arrays in each. One of these buttons will be ***depressed.*** This button selects the ***currently active potential array.*** This is the array that will be acted upon by functions like: **Load, Save, Modify, Fast Adjust, Refine, Double,** and etc.

A particular potential array is selected by clicking (depressing) its button.

The last button in the list is always marked **empty PA.** ***This* empty PA *is available for allocating memory for a new potential array.*** Up to 200 potential arrays can be loaded in RAM at one time.

Creating, Refining, and Flying Ions in Your First Potential Array

The following material is an illustrated step-by-step example of how to create an electrostatic potential array and use it to fly ions. It is recommended that you use SIMION to follow along with the example below. ***The best way to learn SIMION is to use it.***

Creating a Project Directory

The first step in starting any new SIMION project is to create a project directory to hold all the files you'll create. Remember, SIMION expects that all project files reside in the *same* directory. ***You ignore this rule at your peril!***

Let's create a new directory called **MYSTUFF** that is directly below the **\SIM6** installation directory. Start SIMION. Now, starting at the Main Menu Screen, use the following steps to create the new project directory (*Figure 2-9 below*):

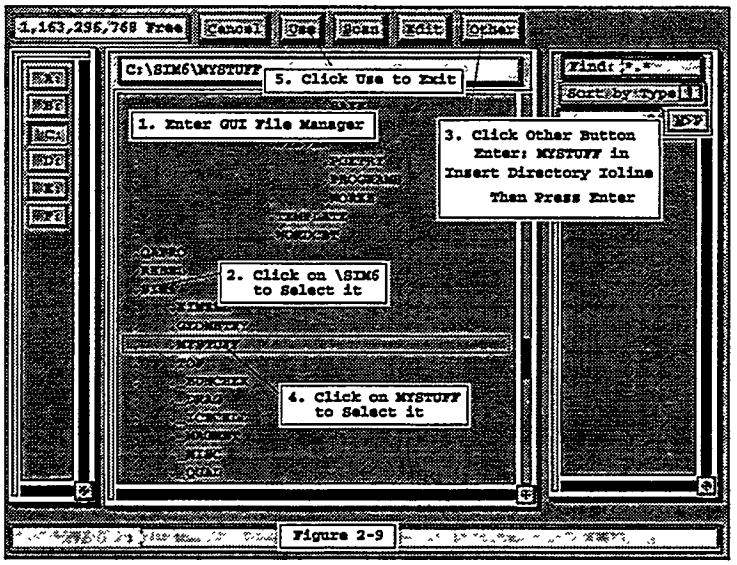
1. Click on the **GUI File Manager** button.

2. Click on the **\SIM6** directory (*middle window*) to make it the current directory if it is not already selected (*marked*).

3. Click the **Other** button and enter **MYSTUFF** in the **Insert Directory** ioline and press **Enter**. SIMION will create the **MYSTUFF** directory beneath the **\SIM6** directory.

4. Click on the **MYSTUFF** directory to make it the ***current*** directory.

5. Click the **USE** button to exit the GUI File Manager.



Controlling Automatic Directory Scanning

SIMION stores images of each drive's directory tree in files so it won't have to rescan the drive for directories each time you access it with the GUI File Manager. To insure that the directory is complete and correct SIMION ***automatically*** scans each drive for directories the ***first time it is accessed*** in any program session.

This approach assumes that you have added or deleted directories between SIMION sessions. If this is not the case, the automatic scan feature wastes time (particularly *on large drives*), because nothing has changed.

You have the option of turning automatic directory scanning off. However, *you* are then ***responsible*** for clicking the **Scan** button in the GUI File Manager to update a directory tree that may be incorrect (*Figure 2-9*). The following procedure turns auto-scanning off (*or back on*):

1. Click the **Adjust** button on the Main Menu Screen.
2. Click the **Other Adjust** button to access the Other Preferences Screen.
3. Click the **Scan On** button. It should now display **Scan OFF**.
4. Click the **OK** button.

Creating a New Potential Array in Memory

The next step is to create a new potential array in memory. Use the following steps for this example (*Figure 2-10 below*):

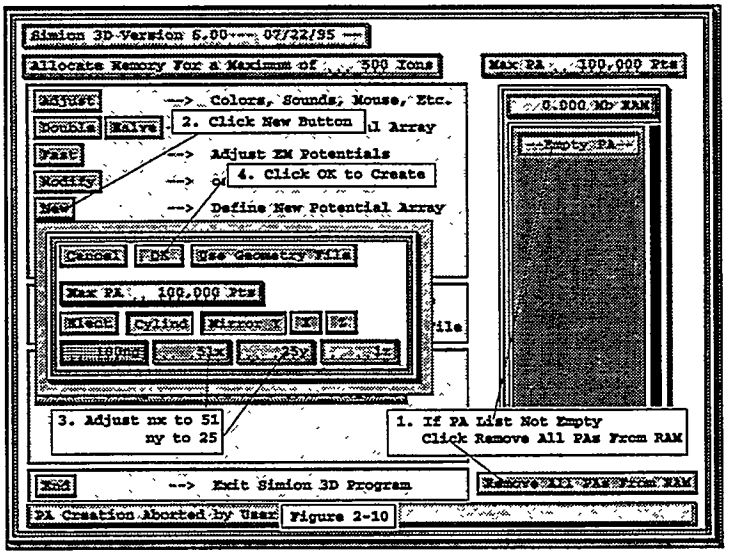
1. If the potential array list is not empty (*it should be*), click the **Remove All PAs From RAM** button.

2. Click the **New** button to access the Potential Array Creation Screen.

3. Adjust the **X** dimension panel to 51 and the **Y** dimension panel to 25. ***We will use the defaults for all the other array parameters.***

4. Click **OK** to create a new potential array.

Inserting Electrode Geometry



For this exercise, we are going to make a Fast Adjust Definition array (**.PA#**) of a simple three element lens. It is a useful example, because Fast Adjust Definition arrays are the most commonly created array type. The choice of a simple three element lens provides something to play with that will give you insight into how electrostatic ion optics really work.

The Method

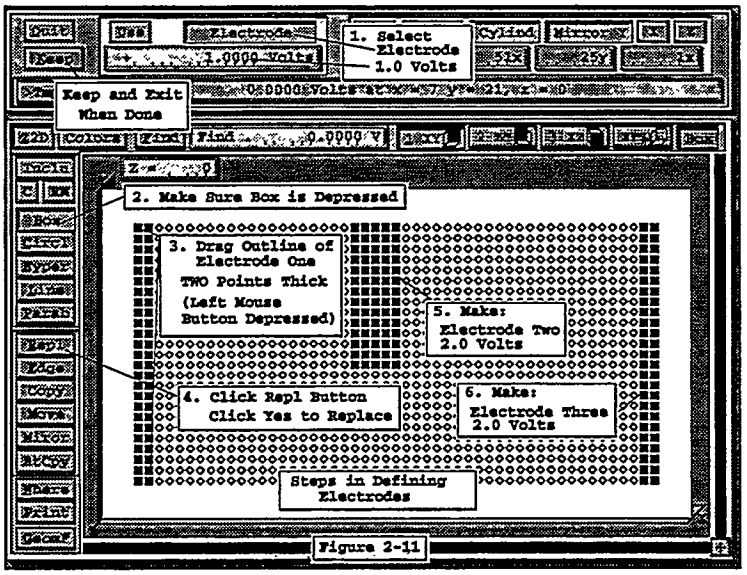
The **Modify** function will be used to define the electrode geometry. **Modify** also can be used to change array dimensions, symmetry, mirroring as well as edit the geometry definitions of any existing potential array.

The Plan

We are going to create three adjustable electrodes. Electrode number one will be a circular plate on the left edge of the array. Electrode number two will be a disk with a hole in it in the center of the array (*remember this is a 2D cylindrical array*). Electrode number three will be a circular plate on the right edge of the array.

Electrodes will be ***at least two array points thick*** so that SIMION will treat them as ***solid*** objects rather than as grids for ion flying.

Array points within electrode number one will be 1.0 volt, those in electrode two will be 2.0 volts, and those in electrode three will be 3.0 volts. ***This is required for SIMION to recognize these points as adjustable electrode points.***



Let's Do It!

Click the **Modify** button on the Main Menu Screen to access the **Modify** function. You should now be looking at the Modify Screen (*Figure 2-11*). Each electrode is defined by setting its point type (*e.g. electrode*) and voltage, marking its area of points, and then clicking the **Repl** button to replace the marked points with the defined values. The followings steps should be used:

1. For electrode number one, make sure the **Electrode** button is ***depressed*** and set the **Voltage** panel to 1.0 volt.
2. Check to see that the **Box** button is ***depressed***. It selects box area marking.
3. Now move your cursor to the upper **left** corner point of the array. Hold down the ***left*** mouse button, and drag the cursor (*keeping the left mouse button depressed*) to the bottom of the array one point to the right (*to mark an area two points thick*). Now release the ***left*** mouse button and the area is marked. ***If you make a mistake marking, just re-enter the correct mark.***
4. Click the **Repl** button and click the **YES** button to replace the points in the marked area with 1 volt electrode points. If the wrong points are marked, you can erase them by changing the point definition to Non-Electrode of 0.0 volts, marking the error's area, and replacing the marked points with non-electrode 0.0 volt points.
5. Change the **Voltage** panel to 2.0 volts and insert electrode number two (*as in Figure 2-11*) in the same manner as electrode one.
6. Change the **Voltage** panel to 3.0 volts and insert electrode number three on the ***right edge*** (*as in Figure 2-11*) in the same manner as electrode one.

When you're done creating the electrode definitions, click the **Keep** button to exit **Modify** and keep the array changes (*in-memory copy*).

Saving The Array as TEST.PA#

At this point the only copy of this array is the in-memory copy. SIMION has assigned it a temporary name of **NONAME01.PA**. We next need to save this file as **TEST.PA#** in the **MYSTUFF** directory. The **.PA#** file extension tells SIMION that this is a Fast Adjust Definition file (*important*). Use the following steps to save the array:

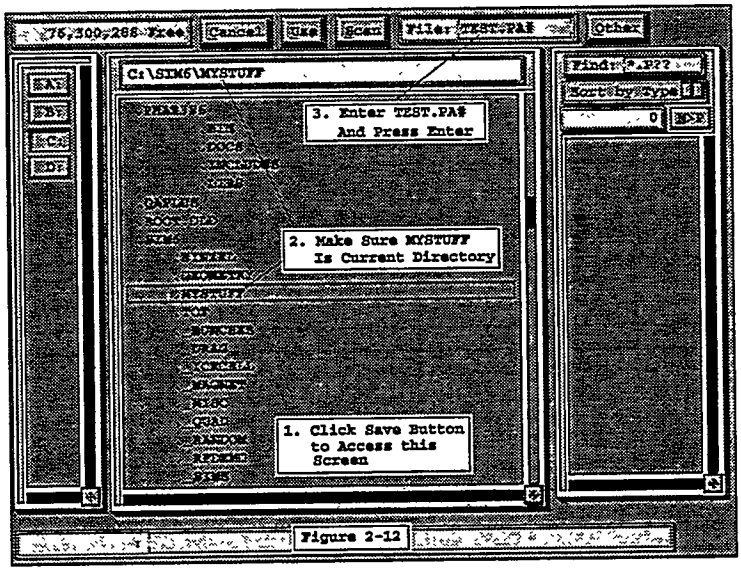


Figure 2-12

1. Click the **Save** button on the Main Menu Screen.

2. Make sure the current directory is **MYSTUFF** (*if not highlighted - click on the directory*).

3. Enter **TEST.PA#** in the **File** ioline and press **Enter.**

4. SIMION will now save the array and display a Memo Screen. You have the choice of entering a short note (*memo*). Just hit the **Enter** key (*to skip the memo*), and the Main Menu Screen returns.

Refining the Fast Adjust Potential Array

The next task is to refine the potential array. When you refine a **.PA#** potential array SIMION doesn't actually refine the **.PA#** array itself. It uses the **.PA#** array as a definition for the collection of arrays that **Refine** actually creates, refines, and saves in the current directory (**MYSTUFF**). Use the follow steps to refine the array (*Figure 2-13 below*):

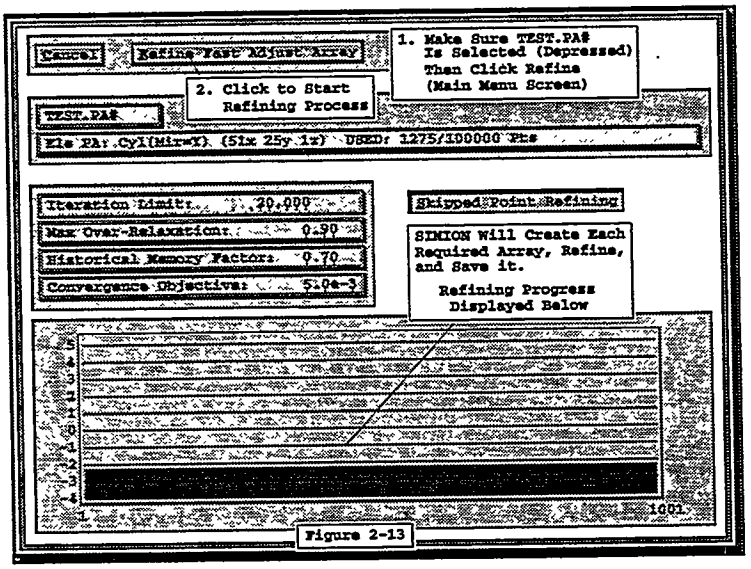


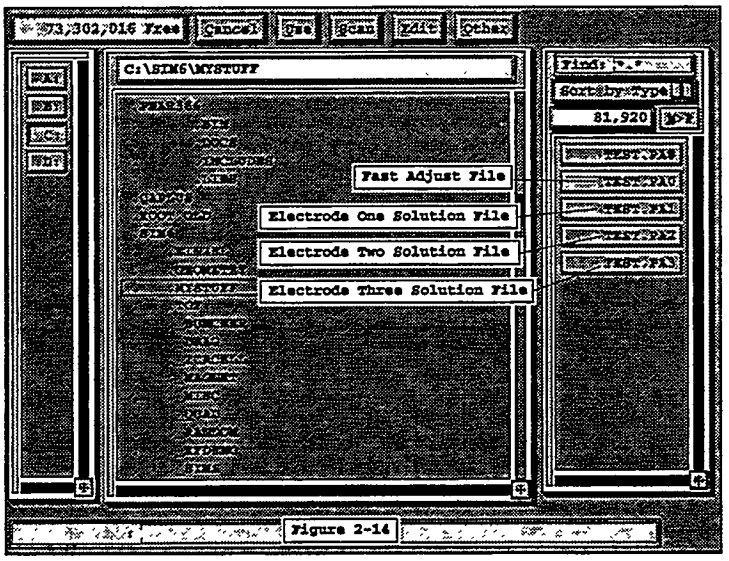
Figure 2-13

1. Check to see that **TEST.PA#** is the currently selected array (*its button is depressed on the Main Menu Screen*). Click the **Refine** button. You should now be looking at the Refine Screen (*Figure 2-13*).

2. Click the **Refine Fast Adjust Array** button to start the refining process.

SIMION will scan the array, determine the number of adjustable electrodes, create each of the four required arrays (**.PA0, .PA1, .PA2, .PA3**), refine each array and save them in the **MYSTUFF** directory.

When all this is completed, you will be returned to the Main Menu Screen. You can verify that these four fast adjust support files have been created by clicking on the **GUI File Manager** button to look at the contents of **MYSTUFF** (*Figure 2-14*). Click either the **Cancel** or **Use** button to return to the Main Menu Screen.



Fast Adjusting the Voltages of the TEST.PA0 File

The actual Fast Adjust File is the **TEST.PA0** file. This file contains the solutions for any non-adjustable electrodes as well as the current potential settings for all adjustable electrodes (*all adjustable electrode voltages are initially set to zero when the array was created by* **Refine**).

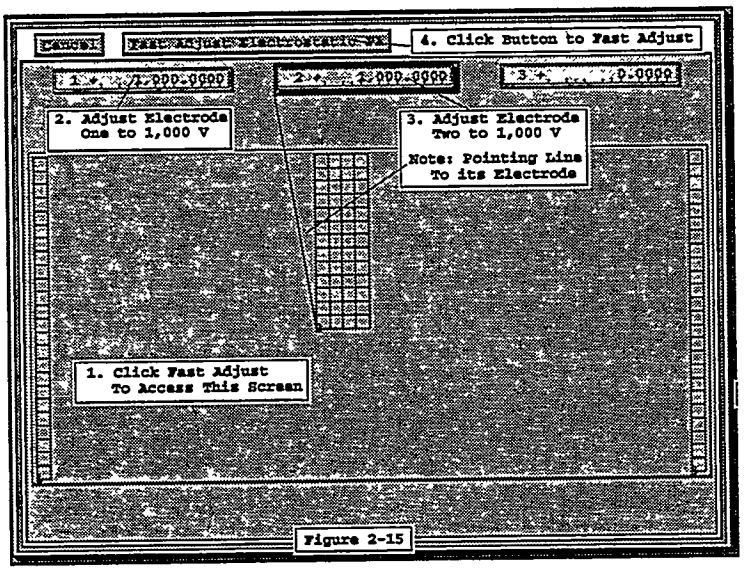
The **TEST.PA0** file is the file we must actually fast adjust. We could load it in place of the **TEST.PA#** file by clicking the **Load** button, pointing to the **TEST.PA0** file button, and clicking ***both*** mouse buttons to load the file. However, we're lazy. Let's let SIMION do this automatically for us by clicking the **Fast Adjust** button (*assuming that the* **TEST.PA#** *file is currently selected*). SIMION looks at the file, sees it is a **.PA#** file, and ***automatically*** loads the **.PA0** file in its place and fast adjusts it (*Figure 2-15*).

We want to set electrode number one to 1,000 volts, electrode number two to 1,000 volts, and electrode number three to 0 volts (*its current value*). Use the following steps:

1. Adjust electrode number one's voltage to 1,000 volts using its panel object. Note: SIMION automatically draws a red line connecting an electrode's voltage control panel with one of its electrode points when the cursor is in the panel object.

2. Adjust electrode number two's voltage to 1,000 volts using its panel object.

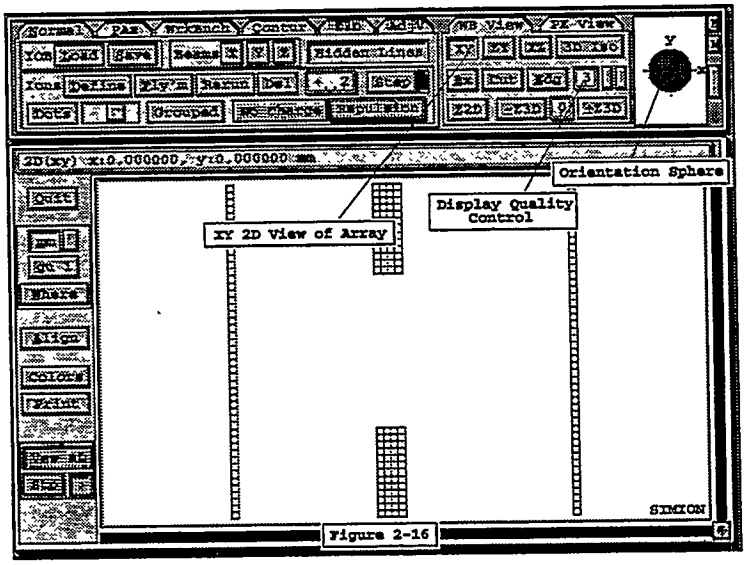
3. Click the **Fast Adjust** button and the array is fast adjusted.



Note: SIMION fast adjusts the ***in-memory*** copy of the **.PA0** file. If you want to retain these values between sessions you should save the updated in-memory copy back to disk. For Example: Click the **Save** button, hit the **Enter** key (*to save*), click **YES** to replace, and hit the **Enter** key to skip the memo.

Viewing the Potential Array

To view the **TEST.PA0** potential array, make sure its button is depressed (*on the Main Menu Screen*) and click the **View** button. Your screen should look like Figure 2-16 below:



You can click the **ZY**, **XZ**, and **3D Iso** buttons to see other standard views. Adjust the **Display Quality** panel from 0 (*lowest quality*) to 9 (*highest quality*) in **3D Iso** view to see what it does. Also use the **Orientation Sphere** object to change views (*point the cursor to it and drag the sphere about with either mouse button depressed*). There are 12 2D and 8 3D standard views.

See if you can use the **Orientation Sphere** to see all of them. When you're through playing, click the **XY** button to return to your starting view.

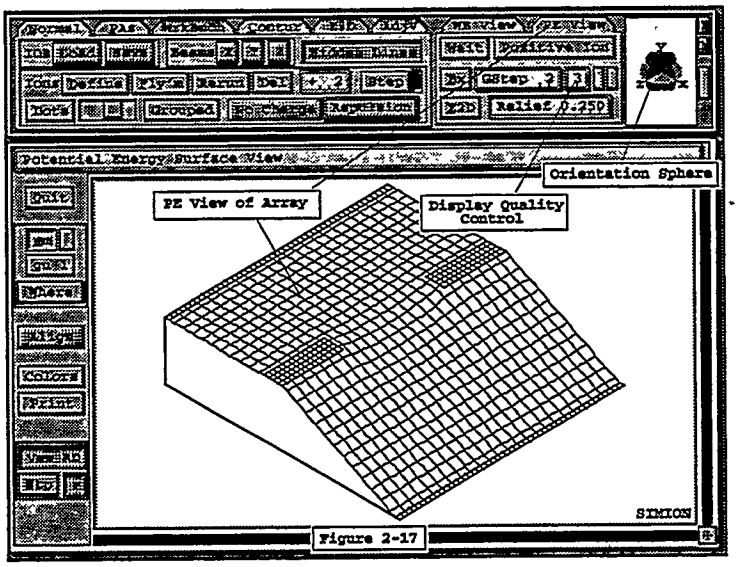
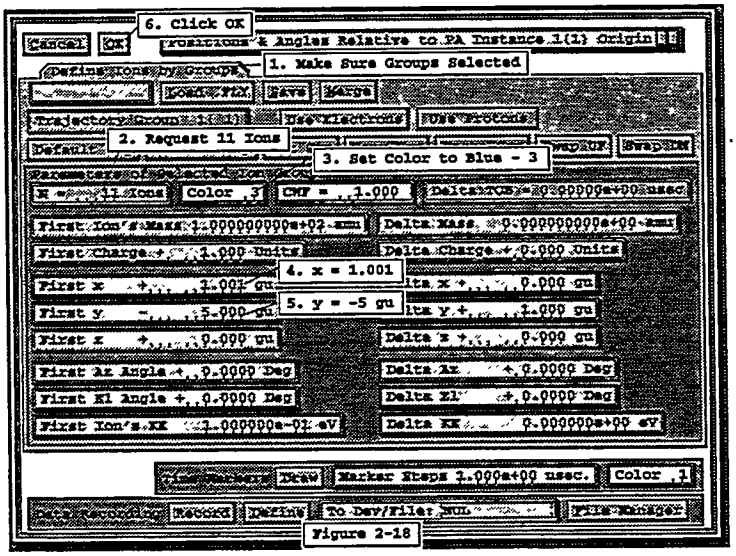


Figure 2-17 shows a potential energy view of the potential array. To duplicate this view, click the **XY** button (*or make sure it is depressed*) and then click the **PE View** tab to switch to a potential energy view. The PE view has its own display quality and Orientation Sphere controls. You can toggle back and forth between WB View and PE view and these settings will be remembered.

Defining Some Ions to Fly

The next task is to define some ions to fly. This is done by clicking the **Define** button on the Normal Controls Screen (*the* **Normal** *tab is selected*). You should now be looking at the Ion Definition Screen (*Figure 2-18 below*).



SIMION allows us to define ions by two methods: In groups or individually. For this example we will define ions by groups:

1. Make sure that the **Define Ions by Groups** tab is selected.
2. Set the number of ions in group 1 to 11 using the **N Ions** panel.
3. Set the color of the ions to blue (*3*) using the **Color** panel.
4. Set the **First x** panel to 1.001 (*to start just to the right of electrode number one*).
5. Set the **First y** panel to -5.0. and make sure the rest of your settings match those in Figure 7-18.
6. Now click **OK** to keep the definitions and return to the View Screen.

Flying Ions

Click the **Fly'm** button to fly the defined ions. Your screen should be something like Figure 2-19 (*below*) if you're still in a PE View.

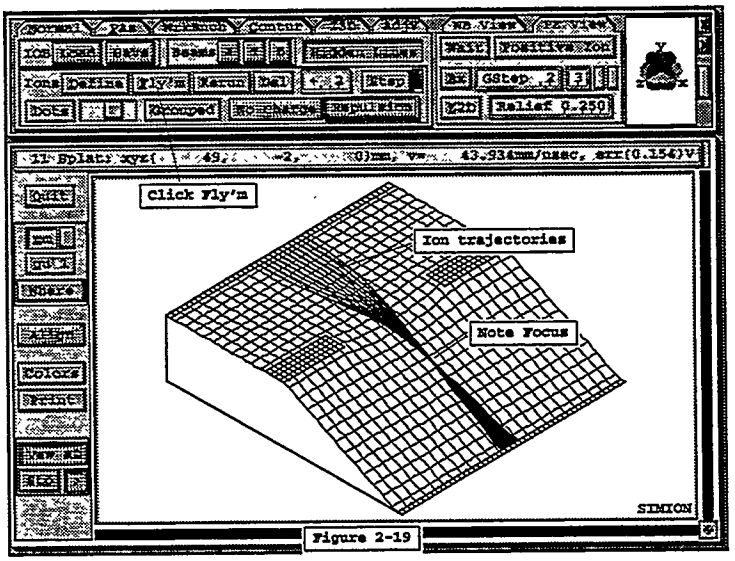


Figure 2-19

You can fly the ions together by ***depressing*** the **Grouped** button before you click **Fly'm**. Further, you can keep the ions flying by depressing the **Rerun** button too. It gives a movie effect (*click* **Fly'm** *again or hit the* **Esc** *key to stop*). Trajectory computations will speed up if you change the **Trajectory Computational Quality** panel from 2 (*its default*) to 0. ***There are all sorts of things to learn and try for yourself!***

Fast Adjusting Electrodes From Within View

SIMION allows you to fast adjust electrodes from within the **View** function. This is accomplished by clicking the **PAs** tab to access the PAs Control Screen, use the **Instance Selection** panel to select the desired instance (*there is only one instance in this example*), and then click the **Fast Adj** button.

For this example adjust the voltage of electrode number two to zero volts and fast adjust it. Now click the **Normal** tab and then click **Fly'm**. Notice the shape of the potential energy surface has changed dramatically. The ions ***diverge*** rather than focus (*Figure 2-20 below*).

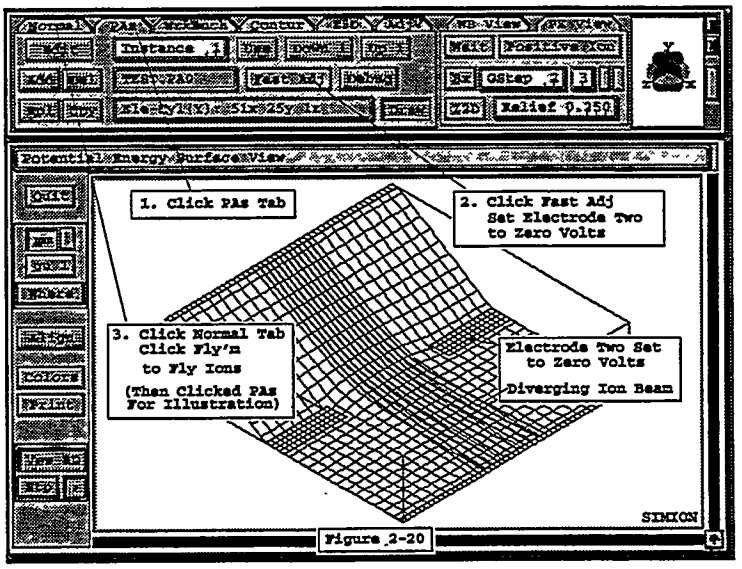


Figure 2-20

Let's assume we want to adjust the voltages so that the ions *just* focus when they hit electrode number three. The first step is to *depress* the **Rerun** and **Grouped** buttons. Now click the **Fly'm** button. Notice that the ions keep flying like in a movie. Click the **PAs** tab and the **Fast Adj** button. Adjust electrode number two to 900 volts, and check the focus. If it's not quite right click the **Fast Adj** button and try again. This is known as interactive tuning.

Saving Your Work

It is important that you save your work. This allows you to quickly resume your efforts in a subsequent SIMION session. The following material shows you how to save your **.IOB** file (*ion optics bench definition file*) as well as a **.FLY** (*ion group definition file*).

Saving an Ion Optics Bench File .IOB

To save the current workbench definition (*including the voltages used in all instances*), click the **Normal** tab to access the Normal Controls Screen. If ions are currently flying from the above adventure, click the **Fly'm** button (*or hit the* Esc *key*) to stop the flight. Now click the **Save** button on the Normal Controls Screen, enter the word **TEST** for a file name and hit the **Enter** key. SIMION will automatically append the **.IOB** extension to the file, and save your current workbench definitions as **TEST.IOB** in the **MYSTUFF** directory (*assuming it's the current directory*).

You can save more than one **.IOB** file. Let's say you wanted to save an **.IOB** with different voltage settings. You would adjust the electrodes to the desired voltages and ***then*** save an additional **.IOB** file (*perhaps* **TEST1.IOB**) in the same manner you saved **TEST.IOB** above.

The Value of File Memos and How to Create Them

This example brings up the issue of file memos. With names like **TEST.IOB** and **TEST1.IOB** you will probably not know which one to choose a month (*or day*) from now. You should attach a file memo to each file when it is saved to describe its features.

If you forget to create a memo, all is not lost. Click the **GUI File Manager** button on the Main Menu Screen. Point to the file's button that you want to create a memo for (*e.g.* **TEST.IOB**). Now move the cursor to the ***left*** off the button. The memo line (*at the bottom of the screen*) should have the file's name on it (*if you moved the cursor to the left properly*).

Move the cursor into the **Memo** ioline and enter the desired memo text. ***That's all there is to it.***

You can verify that the file has your memo by moving the cursor up and down across the file name buttons. When the cursor touches the file with a memo, the memo will be automatically and instantly displayed. Very handy.

Saving an Ion Group Definition File .FLY

To save your ion definitions click the **Define** button on the Normal Controls Screen. Now click the **Save** button, enter the word **TEST** for a file name and hit the **Enter** key. SIMION will automatically append the **.FLY** extension to the file, and save the current ion group definitions as **TEST.FLY** in the **MYSTUFF** directory (*assuming it's the current directory*).

Reloading Your Work

Let's pretend that this is a new SIMION session and that we want to reload that landmark simulation that we performed above. Click the **Quit** button and **YES** button to exit from **View** back to the Main Menu Screen. Now click the **Remove All PAs From RAM** button and **YES** to zip PA memory.

Reloading the Workbench

Click the **View** button. SIMION doesn't have a potential array to view (*the* **empty PA** *button is depressed*), so it automatically calls the GUI File Manager with instructions to load an **.IOB** file. Point the cursor at the **TEST.IOB** file button (*saved above*) and click ***both*** mouse buttons. SIMION will load the **.IOB** file and all the potential arrays it references. You will be asked if you want to restore all potentials to what they were when the **.IOB** file was last saved. Click **YES** and all voltages will be automatically restored (*by fast adjust* - **.PA0 files** *or fast scaling methods* - **.PA files**).

Reloading a .FLY File

Click the **Define** button on the Normal Controls Screen to access the Ion Definition Screen. ***Note: SIMION didn't forget our ion definitions*** (*unless this is a new session*). Let's pretend it did. Click the **Load** button and point the cursor to the **TEST.FLY** file button and click ***both*** mouse buttons. The **.FLY** file is loaded. Now click the **OK** button to return to **View** and fly the ions.

Summary

We have covered a lot of ground in this chapter. While this information should serve to get you started with SIMION, there is a whole lot more to learn. You have the option of either using the **F1** (*help*) key to ***crash***, ***burn***, and ***learn***; or you can keep on reading.

It is recommended that you read a bit and use SIMION a bit. This is probably the quickest way to learn how to use SIMION effectively. As you learn, try to use SIMION for some real problems, but be careful not to be too ambitious with what you tackle until you have learned the techniques to attack it properly.

The Adventure Continues . . .

Useful Directions

Introduction

This chapter acts as a signpost for the rest of the manual. ***It discusses what to know and where to go when learning to operate SIMION.*** It is assumed that you have run the demos in Appendix C as well as read Chapter 2 and successfully created and flown ions in your first potential array using the step-by-step example at the end of the chapter. If not, ***stop*** and do these things ***before*** proceeding with this chapter.

Always Use Project Directories

SIMION *assumes* that all your project files are in the same directory. Further, it ***requires*** that the project directory be the currently selected directory. This was done to ***enforce*** one directory for each project. The example in Chapter 2 shows how to create and use a project directory. If you try to cheat (*use files from other directories instead of copying them to the active project directory*), SIMION will probably complain about not being able to save or load this or that (*you will get caught*).

SIMION Uses RAM For Almost Everything

The working copies of potential arrays, ion definitions, and almost everything else is kept in RAM. This approach provides the maximum speed *if* you have enough RAM. If not, the program will try to virtual. If you get an out of memory error, read the material in Appendix B on how to virtual your particular extender version successfully. ***There is no substitute for RAM if you want speed.***

Memory Allocation and Heap Fragmentation

SIMION grabs all the memory it uses from heap memory. The trick is to avoid heap fragmentation. This occurs when chunks of memory are allocated helter-skelter all over the heap to the point that no large contiguous chunks of memory (*e.g. for potential arrays*) can be found anywhere. This is why SIMION allocates particular memory regions for potential arrays once (*Chapter 4*). These memory regions cannot be re-sized after they are created to help prevent heap fragmentation problems.

The **Remove All PAs From RAM** button (*Chapters 2 and 4*) is SIMION's method of de-fragmenting the heap. If things get too messy, save the potential arrays, click the **Remove All PAs From RAM** button, and reload the arrays into the desired size of memory regions. Chapter 4 provides useful methods for allocating and reallocating array memory regions.

Be Temperate in Your Array Sizing

The bottom line here is not to be too greedy. Keep the size of your arrays reasonable. Do you really need that large an array? The best policy is to start small and increase size only when it becomes obvious that increased size appears necessary for a successful simulation. You can always make use of array doubling later (*Chapter 5*) or if you make use of geometry files (*Appendix J*), project the definitions into a larger array.

File Saving

If you want to preserve it - save it. Because SIMION keeps everything in RAM during the program session, everything goes away when the program session ends. It is your responsibility to save things you want to keep between sessions: Potential arrays, workbench definitions, ion definitions, data recording definitions, and contouring definitions.

Potential Arrays Not Associated with an .IOB File

If you want the current potentials of a **.PA** or **.PA0** array (*not associated with an* **.IOB** *file*) to be retained for future use between sessions you must save them. This assumes that you plan to load the array via an **Old** or **Load** command and expect to see the potentials retained.

Potential Arrays Associated with an .IOB file

The workbench definition **.IOB** files of the **View** function *automatically* remember the potentials (*when the* **.IOB** *was saved*) of all **.PA** and **.PA0** files (**.P?0** *too*) they reference. When an **.IOB** is reloaded, SIMION asks if you want it to re-adjust the arrays to these potentials (*via the appropriate* **Fast Adjust** *methods*).

Thus, if you are making use of **.IOB** files (*recommended practice*) you really do not need to save the current potentials of **.PA** or **.PA0** files. ***Note: A up-to-date refined image of each .PA and .PA# (e.g. its .PA0 and possibly .P?0) file must always be in the project directory for this to work properly.***

File Compatibility With Earlier SIMION Versions

In general, *no* file formats from earlier SIMION versions are compatible with SIMION 6.0. SIMION will however convert potential arrays with either **.PA** or **.PA#** extensions to 6.0 format. Moreover, you can convert certain potential arrays back to 3.0-5.0 format (*via a trick*). See Chapter 4 for details.

Array Creation

Arrays can be created for an **Empty PA** region with either the **New** (*Chapter 4*) or **Modify** (*Chapter 5*) functions. Use the approach that is most convenient for you. Be sure to allocate enough memory for any planned array size increases. Otherwise, you may be forced save the arrays, zip PA memory, and reload them into larger memory regions (*Chapter 4*).

Defining Electrode/Pole Geometry

Electrode and pole geometry can be defined by using either the **Modify** function (*Chapter 5*) or geometry files (*Appendix J*). It is suggested that you take the time to ***really*** learn how to use **Modify** effectively.

However, it is also important that you remember that geometry files are probably the ***best approach*** for defining complex 3D geometry. Geometry files can be easily scaled so that a geometry file can be made to work with any size array. This means you can get better electrode/pole surface definitions by inserting geometry file definitions into a larger array (*handy*).

Geometry can be defined for two different types of potential arrays: **.PA** and **.PA#**. It is important that you recognize when and how to use these array types.

Tips on the .PA Potential Array

The **.PA** array is the basic potential array. Its electrode/pole potentials are formally defined in **Modify** or by geometry files. The only way its potentials can be changed quickly is via ***proportional re-scaling*** of ***all*** potentials via **Fast Adjust** (*Chapter 6*). If you want to change the potential of a single electrode or pole you must change the potentials of ***all*** of the electrode's or pole's ***points*** with either **Modify** or by changing a geometry file, and then re-refine the array (*Chapter 6*).

The **.PA** array is most useful for magnetic pole definitions with non-uniform pole potentials or electrostatic elements like simple single-stage reflectrons. Here proportional scaling can be quite useful. Another good use for **.PA** arrays would be for simple solid electrode beam stops.

Tips on the .PA# Potential Array

You should normally use **.PA#** arrays for your geometry definitions. The **.PA#** arrays are Fast Adjust Definition arrays. Their individual electrodes can be fast adjusted either by you (*Chapter 6*) or by user programs as the ions fly (*Appendix I*). Moreover, these arrays are easy to define (*e.g. all points in electrode number one are set to 1.0 volts*).

It is important to remember that while **.PA** arrays are refined, fast adjusted, and used for ion flying, **.PA#** arrays are ***only*** refined. They are ***never*** fast adjusted or used for ion flying. The refining process (*Chapter 6*) creates a Fast Adjust Array with the **.PA0** extension. ***This*** is the array that you fast adjust and fly ions through.

You can also create more than one Fast Adjust Array for the same **.PA#** array. This allows you to have ***multiple*** instances of the ***same*** potential array that have ***different*** potentials assigned to their adjustable electrodes/poles. These arrays have the **.P?0** extension where **?** is **A - Z**. Chapters 4, 5, 6, and 7 discuss the creation and use for these arrays.

Refining Potential Arrays

Arrays must be refined if they (*or their derivatives*) are to be successfully used for ion flying (*Chapter 6*). The most important thing to keep in mind is that if you ever change the definitions of electrode/pole geometry or potentials within a **.PA** or **.PA#** file, be sure to **Refine** it. If this is a **.PA** array, you ***must*** also ***save*** a copy of it to the project directory ***after*** it has been refined to preserve the results between sessions and to support the **.IOB** file's capability to re-scale **.PA** potentials when **.IOB** files are reloaded.

Fast Adjusting Potential Arrays

SIMION supports fast adjustment of potential for two types of potential arrays (*Chapter 6*). The arrays must have been refined before fast adjustment has any meaning.

Fast Adjusting .PA Arrays

SIMION fast adjusts **.PA** arrays by ***scaling all*** points in the array by a ***common scale factor***. This is useful if all potentials can be scaled proportionally to obtain the desired result. If this is not the case, you will need to use **Modify** (*Chapter 4*) or changes in geometry file changes (*Appendix J*) to modify the potentials of the desired electrode/pole points and then **Refine** the resulting array.

Fast Adjusting .PA0 Arrays

SIMION creates a **PA0** Fast Adjust Array when you **Refine** a **.PA#** array. The **.PA0** array is automatically saved in the project directory. Thus you do not need to save anything. The advantage of Fast Adjust Arrays is that the potentials of individual electrodes/poles can be adjusted separately. ***You should strive to use Fast Adjust Arrays in your simulations whenever possible.***

Instances and the Workbench

It is very important that you gain a solid understanding of the ion optics workbench used by the **View** function and how instances of arrays are projected into it (*Chapter 7*).

The Workbench

Think of the workbench as a volume that you can size to fit your problem. A workbench can be enormous. SIMION supports workbench volumes of up to 8 cubic kilometers.

The Instance

An instance is a 3D virtual image of a selected potential array that has been positioned, sized, and oriented at a certain location within the workbench volume. The image is 3D because SIMION uses the array's symmetry and mirroring when projecting it. For example a 2D cylindrical array is projected as a cylinder (*surface of revolution*).

Up to 200 instances (*images*) of potential arrays can be projected into a workbench volume. More than one instance can project an image of the same potential array. Instance images can overlap. Conflict resolution: The instance with the highest number (*e.g. 6 is higher than 1*) will be used to fly the ions in the common volume region where instances overlap.

Each instance has no knowledge of any other instance (*all are blind*). This means that the proximity of other instances have ***absolutely no impact*** on the ***fields*** within an instance. It is up to you to set the boundary conditions properly so that the fields make sense (*Chapters 5, 7, and 9*). There is a way to copy the 3D images of electrodes or poles from one instance into the equivalent locations in another instance (*Chapters 7 and 9*). This allows you to project the impact of one instance into another because you can **Refine** the resulting array.

The .IOB File

SIMION can save a workbench definition including all its instance definitions and the potentials of referenced potential arrays (**.PA** *and* **.PA0**) in an **.IOB** file (*Chapter 7*). When an **.IOB** file is loaded by the **View** function, the workbench volume is recreated, all instances are restored, all referenced potential arrays are loaded, and SIMION will ***optionally*** restore the potentials of all referenced **.PA** and **.PA0** files.

It is recommended that you make extensive use of **.IOB** files. It is the most effective way to define, save, and restore a simulation project.

WB and PE Views

SIMION allows you to view simulations using either WB or PE views (*Chapter 7*). Each view type has it strengths.

WB View

The workbench or WB View allows you to see 2D and 3D views of the workbench volume or some inner volume of the workbench. SIMION allows you to zoom into a small volume and watch the ions fly through it. You will use WB Views the most often. There are lots of viewing options (*e.g. 3D pointing, cutaway clipping and etc.*). Take the time to learn them all. ***The better you can command the visualization tools, the more likely you are to see something properly and understand it.***

PE View

The Potential Energy or PE View allows you to see a potential energy surface of a 2D plane within the current viewing volume. This transform, displays ***2D field gradients*** as surface slopes. PE Views are particularly useful for electrostatic fields because they graphically illustrate the electrostatic forces acting on the ions. ***Note: PE Views have less value in 3D arrays, because displaying 3D gradients is a 4D problem*** (*the use of 3D contouring surfaces can provide insights about 3D field gradients - Chapter 7*). ***The PE View is probably the single best way for you to really learn to understand electrostatic optics on an intuitive level.***

Contouring

SIMION supports the drawing of contours and 3D contouring surfaces for both potentials and gradients (*Chapter 7*). Many people find the automatic contouring features helpful, because they can easily create topographic maps of potentials or gradients. Contours can be very helpful in PE Views as well.

3D contouring surfaces give you the ability to see the direction of forces in 3-dimensional electrostatic arrays (*a 4D display problem*). It takes a bit of insight to interpret these surfaces. However, they can be visually quite impressive.

Defining Ions

Ions can be defined in groups or individually (*Chapter 8*). Each method has its uses. Ions are normally defined by groups when they are used to create ion beams. Then the ions only differ by one or two parameters (*e.g. y starting position*). If ions can be defined by incrementing a starting parameter by some factor the ions should be defined by group definition methods. Ions defined this way can be saved in **.FLY** files (*used extensively by the demos*). ***You should use this ion definition method whenever possible.***

In some cases each ion is totally unrelated to any other ion. These ions are best defined by individual definition methods. SIMION provides an ASCII file format **.ION** file to save individual ion definitions. This format allows you to create your own ion definitions outside SIMION (*via editor or other program*) and use them within SIMION (*Appendix D - file format*). ***This is the most flexible ion definition method.***

Flying Ions

Once the ions are defined, they can be flown in many ways (*Chapter 8*). They can be flown: Separately or together, as rays or dots, with or without charge repulsion, at various computational qualities, and even kept re-flying in a movie style format.

SIMION allows you to change most anything while the ions are flying (*Chapters 7, 8, and 9*). This makes the program highly interactive. If you would like to change something while the ions are flying, try changing it (*a view, an instance, or whatever*). Chances are that SIMION will cooperate.

Data Recording

SIMION supports an extensive data recording capability (*Chapter 8*). You have the option of selecting which parameters are included in each data record, what event(*s*) trigger a data record, and the format used for the data record. You also can control the output of header records too.

Data records can be displayed on your screen and/or saved to a data file. This allows you to export simulation data for analysis and process by other programs (*e.g. a spreadsheet*).

User Programs

The most powerful feature of SIMION is user programs (*Chapter 9 and Appendix I*). User programming allows you to write your own routines and have SIMION automatically compile and use these routines while flying ions.

User programs can randomize your ion definitions, simulate collisional or viscous damping, change any adjustable voltage as the ion flies, change an ion's definitions while in flight (*e.g. color*), update potential energy surfaces, selectively kill ions, and control the re-flying of ions. This is but a glimpse of the enormous power you have available with user programming.

The better you know how to use SIMION, the more potential user programs will have for you. ***If you want to push it to the limits with user programming, learn all the tricks.***

Geometry Files

Geometry files provide and general method for defining complex electrode/pole geometry (*Chapter 9 and Appendix J*). ***This is an advanced SIMION feature.*** You should learn to use the **Modify** function well before attempting to use geometry files.

Geometry files contain geometry language instructions. SIMION compiles these instructions and inserts the geometry defined into the target potential array. You have the option of positioning, scaling, and orienting this geometry anywhere within (*or around*) the potential array.

The greatest value of geometry files is the ability to define complex 3D array geometry.

Geometry files are also quite useful when you want to increase the array size (*to improve definition*). If the geometry is defined in a geometry file, all you need do is change the insertion scaling factor when inserting the geometry definitions in a larger (*or smaller*) array.

Creating, Loading, and Saving Potential Arrays

Introduction

It is assumed that you have read the discussion of potential arrays in Chapter 2. If not, read the material before proceeding further. This chapter covers the creation of new potential arrays as well as how to save and load disk copies of potential arrays.

The Role of the Subdirectory in SIMION Projects

SIMION requires that all files relating to a project (e.g. *.PA, *.IOB, *.FLY and etc.) are contained in the same subdirectory of your hard disk. This is useful because it keeps things together and in so doing forces a touch of organization that many of us need so badly. Note: There can be more than one project in a subdirectory. However, this can often contribute to a lot of clutter.

Each of the demos provided with SIMION is in its own directory below **\SIM6**. These demos help show how various projects might be approached.

The GUI File Manager can be used to quickly create subdirectories. ***Click on the parent directory, click the*** **Other** ***Button, enter the name of the new subdirectory, and press*** **Enter**. The subdirectory is created. Now click on this subdirectory to make it the currently active directory. ***Note: The GUI File Manager can also be used to quickly copy or move files between directories and drives (see Appendix F).***

Working Copies of Potential Arrays are in RAM

SIMION maintains a working copy of each active potential array (*up to 200*) in memory (*RAM*). ***When you change something in a potential array you are only changing the memory copy.*** Likewise when you fly ions through instances of potential arrays in the workbench you are using the memory copies.

Working potential arrays are normally saved as **.PA, .PA#,** or **.PA0** (*extension*) files in your project directory. When you create a new potential array ***only the in-memory copy is created.*** It is your responsibility to save any new or changed potential arrays into your project directory as required. ***Saving potential arrays preserves your work between sessions.***

The List of Potential Arrays

The Main Menu Screen contains a window with a button for each currently allocated PA memory region and the name of its potential array. One of these buttons will be depressed. This button selects the ***currently selected potential array***. This is the array that will be acted upon by functions like: **Load, Save,** and etc.

A particular potential array is selected by clicking its button.

There is a display panel at the top of the file list window that displays the number of megabytes of RAM currently allocated to potential arrays. This allocation includes the potential arrays in the list as well as any specific electrode solution arrays loaded to directly support fast adjust electrodes changed by any active **Fast_Adjust** user program segments (*Appendix I*).

The last button in the list is always marked empty (*e.g.* **Empty PA**). The **Empty PA** is available for allocating memory for a new potential array. ***Up to 200 potential arrays can be loaded in RAM at one time (memory permitting).***

De-allocating Potential Array Memory

The Main Menu Screen has a **Remove All PAs From RAM** button. This button is used to remove all working copies of PAs from RAM. Its primary function is to restore a clean slate when you are about to start a new project or the clutter of PAs has become unmanageable.

Allocating Memory for Potential Arrays

Potential arrays can use up a lot of memory. Each point of a potential array requires 10 bytes of RAM storage. Thus a 100 x by 100 y by 100 z 3D array has 1,000,000 points and requires 10 megabytes of RAM. ***SIMION allocates memory only once for each PA memory region it creates. Once a PA memory region has been allocated it is not returned or changed until the* Remove All PAs from RAM *button is used to remove all memory allocated to PAs (de fragment the heap).*** This is done to prevent heap fragmentation lockups due to the large size of typical potential arrays.

Changing the Default PA Memory Allocation

SIMION normally allocates a minimum of 100,000 points of memory (*1 MB of RAM*) for each new potential array. This is usually large enough to allow you to increase the array size (*e.g. via* **Modify** *or* **Double**) without exceeding the memory allocated for a PA. ***You have the option of adjusting this default PA memory allocation size up or down to suit your needs (with the* Max PA *panel object) before you create or load a potential array in an empty PA memory region (memory region size range: 20,000 to 10,000,000 points).***

Specifying Desired Default Allocation at Start-up

You also have the option of specifying the initial default memory to allocate for each new potential array at start-up. This is done by including the desired initial default allocation size in points on the SIMION command line:

SIMION 0	Sets default to 20,000 points (*min. allowed*)
SIMION 1000000	Sets default to 1,000,000 points
SIMION 100000000	Sets default to 10,000,000 points (*max. allowed*)

How to Minimize RAM Usage

Use the minimum of 20,000 points for default PA memory allocation. This minimizes the amount of RAM required for ion optics bench definitions that involve many small potential arrays.

How to Increase a PA's Memory Allocation

You may on occasion need to expand a potential array's memory allocation beyond its current value (*e.g. to double it*). The following procedure can be used to accomplish this:

1. Save the potential array (*and any others you wish to keep*) to the current project directory.
2. Click the **Remove All PAs From RAM** button to de-allocate all potential arrays.
3. Adjust the **Max PA** panel to the desired memory to allocate.
4. Re-load the potential array into the **Empty PA**.

Creating New Potential Arrays

New potential arrays can be created either via the **New** function or by loading an existing potential array and saving it with a new name. In either case, the **Modify** function can *then* be used to redefine the new potential array and its electrode/pole geometry. ***Remember to save (replace) potential arrays after modifying them if you want changes to persist between SIMION sessions.***

Creating Potential Arrays From Existing Potential Arrays

You can always use an existing potential array as the start for a new potential array. The following procedure is recommended:

1. Click the **Empty PA** button on the Main Menu Screen if it is not already depressed.
2. Adjust the **Max PA** panel for a memory region of sufficient size for any planned array expansion.
3. Use the **Load** function on the Main Menu Screen to load an existing potential array into the **Empty PA** region.
4. Use the **Modify** function to re-define the potential array as required (*re-size and/or re-define electrode/pole geometry*).
5. Use the **Save** function to save a copy of the new potential array with its new name in the desired project directory.

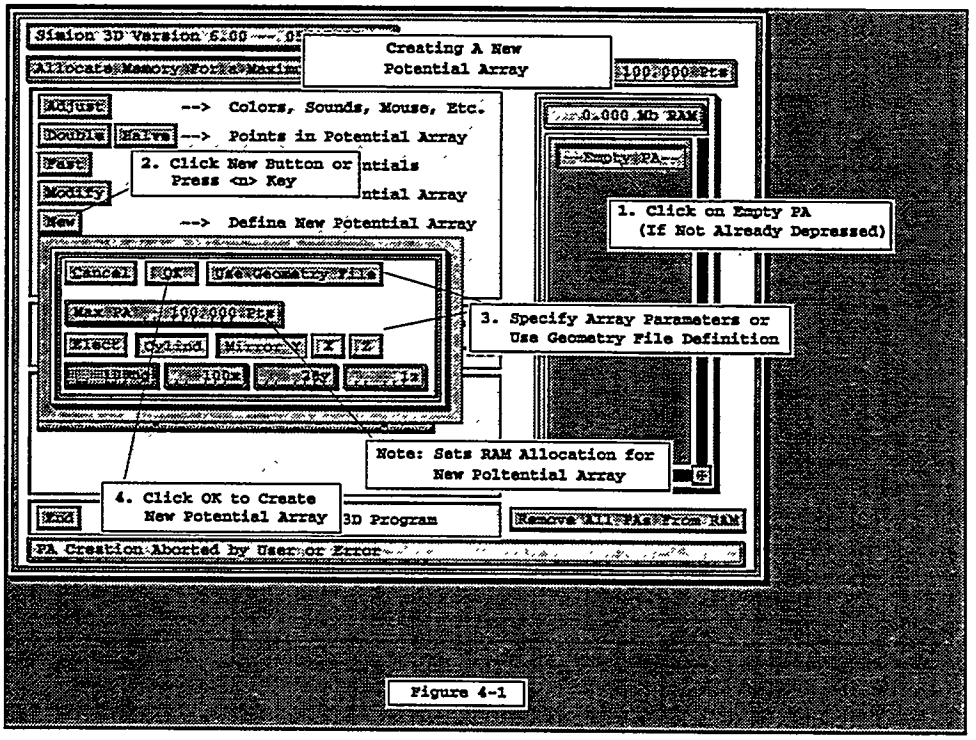


Figure 4-1

Creating Potential Arrays Using the New Function

The **New** function can be used to create a potential array. The following is the recommended procedure (*see figure 4-1 above*):

1. Click the **Empty PA** button on the Main Menu Screen if it is not already depressed.
2. Click the **New** button or press the **<n>** key to access the **New** function.
3. Specify the desired array parameters *or* click the **Use Geometry File** button (*advanced feature*). Note: The only important parameter is the array's RAM allocation (**Max PA** *panel*). ***All the other parameters can be changed later in the* Modify *Function.***
4. Click the **OK** button to create the new potential array.

Adjusting the Default Allocation Size within New

The **Max PA** panel (*in the New Function Screen*) defines the amount of RAM to allocate for the new potential array. The initial value will be the currently active default allocation value on the **Max PA** panel on the Main Menu Screen (*normally 100,000 points or 1 MB of RAM*). ***Any changes you make in allocation size will only impact RAM allocation for the new array***. The current default PA allocation will remain unchanged.

Be sure to allocate enough memory for any planned array expansion (*e.g. doubling*).

Adjusting Array Dimensions

The x, y, and z panels are used to specify array dimensions. Note: 2D arrays are specified when z = 1. 3D arrays are specified when z > 1.

The x, y, and z array dimension panels will not allow any array dimensions that would exceed the current point allocation. Thus, if you need a larger array, adjust the **Max PA** panel *first*, then enter the desired array dimensions.

Selecting Electrostatic or Magnetic Potential Arrays

SIMION *always* assumes that you want an electrostatic potential array. If the array is to be magnetic, click the **Elect** button (*the title will switch to Magnt*).

When a magnetic potential array is selected access to the **ng** panel will be unblocked. The default value is 100. You should set ng to the number of grid units of pole gap so that the magnetic potentials (*Mags*) are reasonably direct indications of the magnetic field in gauss between the pole pieces. ***The* ng *parameter can be changed later in the* Modify *function.***

Cylindrical or Planar Symmetry

SIMION allows 2D arrays to have either planar or cylindrical (*surface of revolution*) symmetry. ***All 3D arrays must be planar (SIMION enforced).*** The **New** function assumes cylindrical by default. If you want planar click the **Cylind** button.

Mirroring Options

You have the option of mirroring the potential array for negative x, y, and z values. This can be used to reduce array sizes if the design symmetries allow. The **New** function assumes y mirroring by default. You can select what you desire. If the combination is illegal, SIMION will beep and restore the button to its former status. The possible array mirroring combinations are: **None, X, Y, Z, XY, YZ, XZ, XZY**. The legal mirroring combinations by array type are:

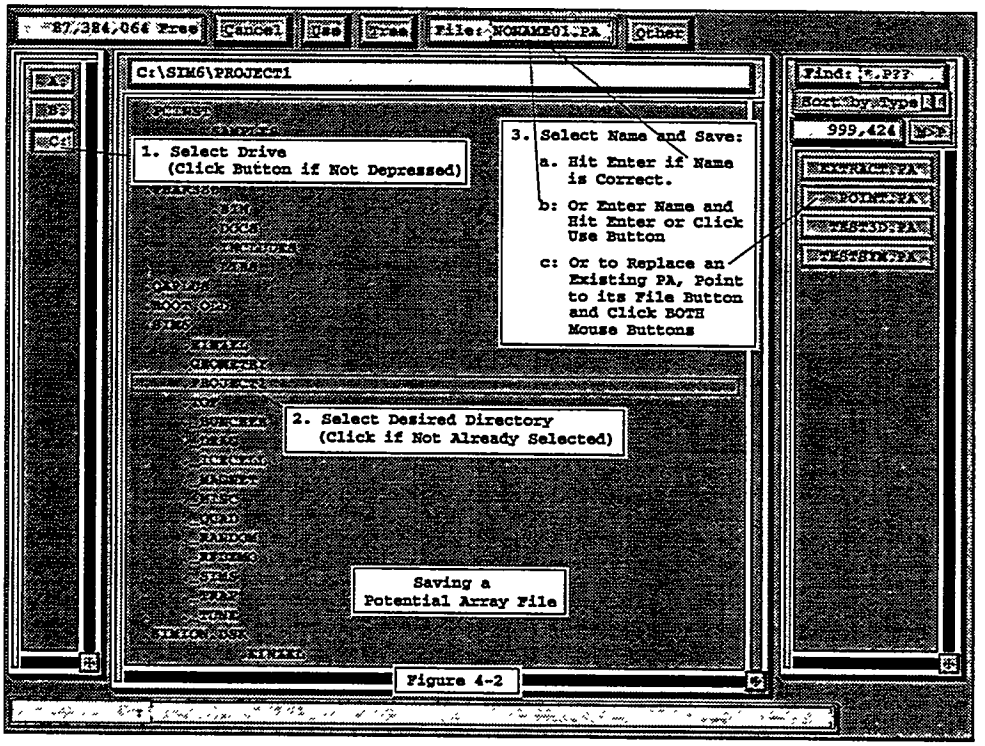
All 3D arrays:	All mirroring options are legal
Planar 2D arrays:	All mirroring except z are legal
Cylindrical 2D arrays:	y mirroring is required, x is legal, z is illegal

Using Geometry Files

You can also use geometry files to create new potential arrays. Geometry files are an advanced SIMION feature (*see Appendix J*). If the geometry file's first instruction is **PA_Define,** SIMION will use it to define the new potential array. Otherwise, SIMION will ask you if you want to use the array parameters defined on the New Menu Screen to create the potential array.

When you make use of geometry files within the **New** function SIMION will automatically insert the electrode/pole geometry defined into the newly created potential array. For a quick painless example of geometry files:

1. Click the **New** button on the Main Menu Screen (*Empty PA button must be depressed*).
2. Click the **Use Geometry File** button.
3. You are now in the GUI File Manager. Click the **GEOMETRY** directory (*just below the* **\SIM6** *main directory*). Now point to the **TRAP1.GEM** button and click ***both*** mouse buttons to load and execute this file.
4. Click the **View** button to take a look. This geometry file makes a simple 2D hyperbolic trap.



Saving Potential Array Files

Potential Arrays in memory are saved to disk in the following manner (*Figure 4-2 above*):

1. Select the desired potential array to save by clicking its PA button (*if it is not already depressed*).
2. Click the **Save** button or hit the <s> key from the Main Menu Screen.
3. Select the destination drive and directory (*by clicking on them - if not already selected*).
4. Enter the desired file name and press **<Enter>** or click both mouse buttons on an existing file to replace it.
5. If the name selected is that of an existing file, SIMION will ask if you want to replace it.
6. After the potential array is saved/replaced SIMION will display the file's current memo string for entry/editing. Press **<Enter>** if you want to skip memo entry/editing.

Legal File Extensions

SIMION only allows **.PA** (*normal potential arrays*) and **.PA#** (*fast adjust definition potential arrays*) to be saved by the user to disk. All other types of potential arrays can only be saved by SIMION itself. This helps to prevent you from mucking things up accidentally. ***A couple of fast adjust file (.PA0) exceptions listed below***:

Creating Secondary Fast Adjust Files (*e.g. .PB0*)

There are times when you may want two or more instances of the same fast adjust array (**.PA0**) in the workbench with each instance ***having different voltages***. To avoid having to create and refine two or more identical **.PA#** arrays SIMION allows saving multiple **.PA0** files. This is accomplished by loading the desired **.PA0** file and then saving it as a **.PB0** file (**.PC0 - .PZ0** *are also legal to allow several such instances*). As long as the filename (*as opposed to its extension*) remains the same, SIMION will automatically use the same specific electrode solution files when fast adjusting each **.P?0** (*where ? is* **A-Z**) file.

Fast Adjust Files (*.PA0, .PB0, and etc.*)

SIMION will allow you to ***replace*** (*via saving*) ***existing*** fast adjust **.P?0** files (*where ? is* **A-Z**) to retain the current adjustable electrode/pole potential settings between sessions.

File Memos

SIMION allows you to attach a memo string to any file via its file manager. This is handy for including descriptions with your files. When you are in the GUI File Manager, a file's memo (*if any*) will automatically be displayed when you point the cursor to the file's button.

A file's memo can be edited from within the GUI File Manager by pointing to the file's button and then moving the cursor to the left off the button. This will retain the file's memo on the memo ioline object (*file's name appears in memo ioline to confirm selection*). You may now edit/create the file's memo. If you want help, point to the memo ioline object and hit the **<F1>** key for help.

File memos are kept in the **MEMOINFO.GUI** file in the same directory. Erasing this file erases all memos for files in the directory. ***The GUI File Manager will automatically copy/move the appropriate file memos when selected files are copied/moved via the file manager.***

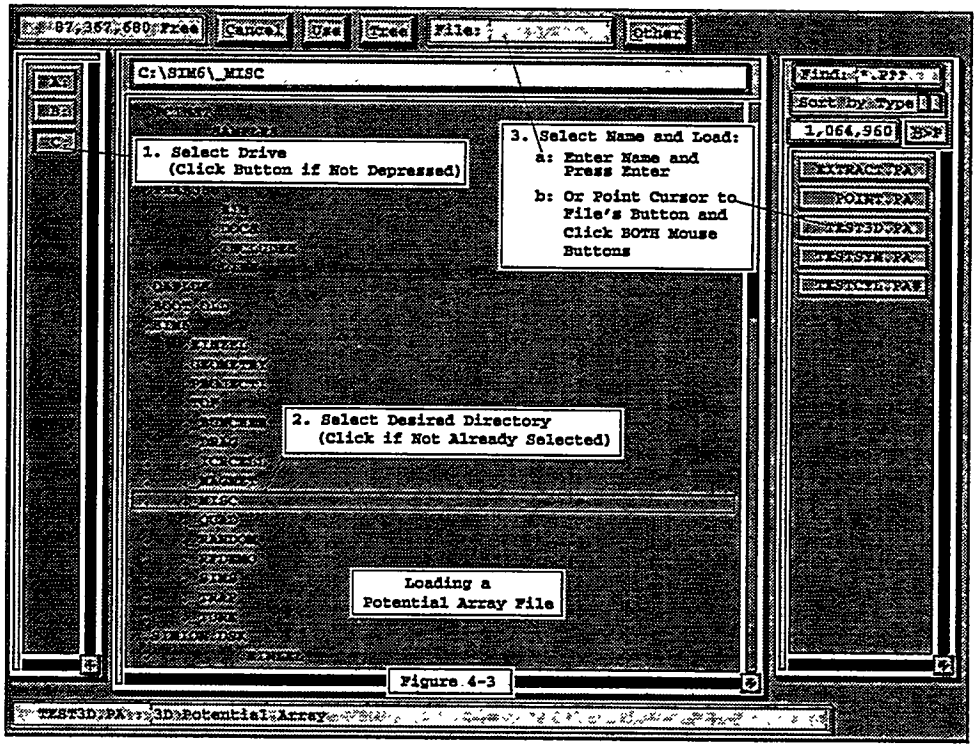
Saving SIMION PAs in SIMION 3-5 Version File Format

SIMION 6.0 potential array file formats are different from earlier SIMION file formats. You will probably never need to convert back to an earlier format. However, SIMION provides a method for conversion to 3.0-5.0 version PA file format should you need it. This option is elected at program start-up by entering a **/4** on the SIMION command line:

SIMION /4 *Turns on backward file conversion option*

When SIMION is started in this manner it will automatically ask if you want a 4.0 version file format whenever you save a potential array that ***could*** be converted to a 3-5 version format. ***Note: The file must be 2D without x or z mirroring to be recognized as convertible.***

This is handy if you want to do some cross comparisons with earlier versions of SIMION with potential arrays created and refined by SIMION 6.0. ***This procedure should only be done with files with .PA or .PA# extensions.***



Loading Potential Array Files

Potential arrays in memory are loaded from disk in the following manner (*Figure 4-3 above*):

1. Select an existing memory region (*in-memory PA*) or **Empty PA** by clicking on its button.
2. Click the **Load** or **Old** button or hit the **<l>** or **<o>** key from the Main Menu Screen.
3. Select the source drive and directory (*if not already selected*).

4. Enter the desired file name and press **<Enter>** or click ***both*** mouse buttons on an existing potential array file to load it.

Memory Allocation and Potential Array File Loading

If you select a non-empty PA (an existing PA), the PA file to be loaded cannot be larger than the size of its memory region. If you point the cursor at the PA's button, the status line (*bottom object*) will display the PA's description along with the memory allocation for the region in points (*16356/100000 means: Current PA uses 16,356 points of 100,000 allocated to memory region*). ***SIMION will refuse to load any PA into a memory region that is not large enough to contain it.***

If you select the **Empty PA** button, SIMION will allocate the ***largest*** of either the current default region size (*the value of the* **Max PA** *panel*) or the actual region size required to hold the file. Thus if you load a million point PA file into the **Empty PA** when the **Max PA** panel is set to 100,000, the memory region allocated will be 1,000,000 points (*10 MB*), exactly enough for the PA file. ***If you plan to expand the size of the potential array later be sure the*** **Max PA** ***value is set high enough to allow for this expansion.***

Converting SIMION 2.0-5.0 Version Potential Arrays to 6.0 Format

SIMION 6.0 PA files have a different format than previous in versions of SIMION. If you attempt to load a SIMION version 2.0 - 5.0 PA file, SIMION will ask you if you want it converted. If you say yes the existing file will be loaded ***and immediately re-saved*** as a SIMION 6.0 file. ***Thus you should hide from SIMION 6.0, any PA files that you don't want accidentally converted.***

Out of Heap Space Errors When Loading a Potential Array

SIMION will try to use existing RAM when loading potential arrays. However, when this is exhausted, SIMION will try to virtual (*use disk for RAM*). If you are using the Intel extender version, an out of heap error indicates you need to expand your region size (*see Appendix B*). If you are using the Rational extender version in DOS you need to activate the virtual support or increase the virtual size (*see Appendix B*). If you are running the Rational extender version in some version of Windows or OS/2 see Appendix B and your Windows or OS/2 documentation for information on **DPMI** memory limits and how to change them.

Potential Array File Structure

See the end of Appendix D for information on SIMION's potential array file structure. ***You may use this information to create or read SIMION 6.0 potential arrays with your own C programs.***

Defining and Editing Array Geometry

Introduction

It is assumed that you have read the discussion of potential arrays in Chapter 2 and have scanned Chapter 4. If not, read the material before proceeding further. This chapter covers the defining and editing of potential array geometry. The methods discussed in this chapter involve the **Double**, **Halve**, and **Modify** functions as well as a few comments concerning geometry files (*an advanced SIMION feature - Appendix J*).

For the sake of this discussion, array geometry means the methods employed to define potential arrays to simulate real-world electrostatic or magnetic fields. ***One potential array can only simulate either electrostatic or magnetic fields.*** Thus combined electrostatic and magnetic fields require two superimposed potential array instances (*one electrostatic and one magnetic*). The actual positioning of arrays as instances in the workbench is discussed in Chapter 7 (*Positioning and Viewing Arrays in the Workbench*).

Changed Arrays are Flagged

When you change the type, size, symmetry, mirroring, and/or geometry of a potential array SIMION automatically flags the array as modified. ***This is done by placing a red outline around the PA's button in the PA list window on the Main Menu Screen.*** When you save the changes to disk the red outline will be removed. If PA changes are unsaved at program exit SIMION will allow you to selectively save the desired potential arrays before exiting the program.

Array Geometry Involves Many Things

The use of arrays of square (*2D*) or cubic (*3D*) mesh grid points to simulate real-world electrostatic or magnetic field devices is at best an approximation of reality. ***The extent that SIMION simulates the real-world accurately is the result of the care you take in understanding and defining array geometry properly.***

Potential Arrays are Like Pixel Displays

SIMION's potential arrays contain points arranged to form square or cubic meshes (*grids*). ***The separation distance between two adjacent grid points is called a grid unit.*** Individual points can be flagged as electrode/pole or non-electrode/non-pole. ***Groups of adjacent electrode/pole points having the same potential serve to define electrode or pole shapes.***

Potential arrays are thus very similar to pixel displays. The higher the pixel density (*grid density*) the better the view (*e.g. simulation*). However, just like pixel displays, a course potential array can successfully model reality very well if reality shares the integral spacing of the potential array (*all edges actually occur at array points*) and the shapes to be simulated are planes (*or cylinders and planes in cylindrical symmetry*) aligned with the potential array points. ***The better reality fits the modeling method, the better the modeling method will simulate reality.*** This is a very important concept to remember when you are choosing the spacing and shapes of your electrodes/poles.

The Array Density Verses Accuracy Problem

If you are modeling complex shapes with non-integral alignment and spacing (*relative to the potential array*) the problem of accuracy becomes more complex. Using points to model something like a hyperbolic shaped electrode introduces a collection of issues that can be mitigated by increasing the potential array size (*e.g. its density*).

Surface Jags Near Electrodes/Poles

The accuracy of the fields near electrode/pole slanted or curved surfaces can suffer from the jags (*steps in electrode/pole points to simulate surface shapes*). Ions flying very close to (*or originating from*) these jagged electrodes/poles will definitely ***not*** have realistic trajectories.

Increasing the grid density may allow you to fly these ions further from the surfaces (*in terms of grid units*). In this case, the averaging nature of the Laplace equation comes to your rescue to smooth out the fields better in the far field region (*3 or more grid units off the surface*).

This, however, doesn't help ions originating from slanted or curved surfaces. The best trick here is to start the ions out a bit (*around one grid unit away from the surface*) with the energies and directions appropriate to their adjusted starting location.

The Issue of Field Curvature

Another frequently unrecognized issue involves the accuracy with which a given grid density models a particular field's curvature. Think of potential array grids as square plates that can be warped a bit to match field shapes. ***The less each grid needs to be warped the better the chance that the potential array accurately models reality. Simply refining an array to a very low error term does not guarantee any particular level of accuracy***. If your array density is too course for your field's curvature you will encounter significant systematic errors in field estimates that high accuracy refining simply will not remove.

Laplace to the Rescue

There are several important issues to keep in mind despite the above gloom and doom. First, the Laplace equation is fundamentally an averaging process. This means that jags and the like tend to be averaged out quite well in the far field. ***It is surprising how crude the surface of a spherical ESA can be (in terms of jags) and still have acceptable far field ion trajectories.***

Moreover, field imperfections are only important to the extent that they adversely impact ion trajectories. There are two relatively painless methods to establish whether you have significant problems: First, double the array size and see what impact this has on ion trajectories. If things don't change much your grid density is probably OK. Second, try to back-fly your ions from their termination points (*using reversed directions and termination energies*). The ability to duplicate ion trajectories bi-directionally (*at least approximately*) means that the modeling is reasonably self-consistent. ***However, neither test guarantees that the results are really correct. There is no substitute for testing the as-built.***

Defining Solid or Grid Boundaries

Electrodes/poles are represented in potential arrays as groups of electrode/pole points with the same potential. SIMION can treat a collection of these electrode/pole points as either a solid object or a grid. The difference is that ions ***will*** pass through a grid but ***will not*** pass through a solid object.

Defining Solid Electrodes/Poles

SIMION kills (*splats*) an ion whenever the ion finds itself within a 2D grid square with electrode/pole points at ***all four corner points*** of the grid or within a 3D grid cube with electrode/pole points at ***all eight corner points*** of the grid cube. ***Thus an electrode/pole must be defined in terms of grid squares (2D) or grid cubes (3D) for SIMION to consider it solid in any particular region.***

Minimum Flying Separation Between Electrodes/Poles

Note: ***The potentials of the grid square or cube corner electrode/pole points do not have to be the same for the ion to splat.*** Thus, SIMION requires that at least one non-electrode/non-pole point separates electrodes or poles of differing potentials for ions to be allowed to fly between them (*more is better for field simulation purposes*).

Splats When Electrostatics and Magnetics Overlap

When an ion finds itself within the volume of overlap of both an electrostatic and magnetic instance, ***SIMION checks*** the ion's location against ***both*** instances for splat conditions. ***If the ion splats in either, it is considered to have died.***

Defining Electrode Grids

SIMION supports the notion of grids for electrostatic arrays (*works for magnetic instances too - though probably not as useful*). ***A grid is any collection of electrostatic grid points that do not fit the solid electrode/pole definition above.*** The important point is that ions ***do not*** splat (*die*) when they pass through grids.

The Definition of Simple Grids

The simplest (*and most accurately modeled*) grids are created by electrode points that create planes (*or planes and cylinders in cylindrical symmetry*) in instances that are integrally aligned relative to the potential array ($x = c$ *or* $y = c$ *lines in 2D or* $x = c$, $y = c$, *or* $z = c$ *planes in 3D, where c is an integer*).

Issues with Curved or Slanted Surface Grids

Curved surfaces (*e.g. a hyperbolic grid*) or slanted surfaces (*e.g.* $y = mx + b$) that model grids will introduce the jags. Since ions are probably expected to fly through these grids it is important to realize what to expect. ***The worst case involves decelerating curved/slanted grids that pass ions at a very low energies.*** If the ion's energy as it passes through the grid is not a couple of orders of magnitude greater than the energy drop in the jags region (*within plus or minus three grid units of the grid's rough surface*) the ion's trajectory will be highly suspect. Increasing the grid density can reduce this problem. ***If at all possible, try to transverse curved/slanted grids with reasonably energetic ions.***

The Notion of Ideal Grids

The typical grid defined within SIMION involves a connected straight or jagged (*sloped or curved*) line of electrode/pole points. This type grid is treated as an ideal grid. That is, SIMION assumes that the grid holes are infinitely small (*no field penetration through the holes*) and that the grid passes 100% of the ions. ***Would that we could actually build such grids!*** The reason for this is simply that each point on the boundary of the grid is an electrode/pole point (*with fixed potential*). Thus there is no path for fields to penetrate through the grid.

Constructing Non-Ideal Grids

Non-ideal grids are modeled by replacing a percentage of electrode/pole points along the grid boundary with non-electrode/non-pole points. This allows fields to penetrate through the grid. A relatively open grid might have only every 5th point as an electrode/pole point. In the case of 2D arrays these points actually model wires (*wire circles in cylindrical*). ***Non-ideal crossed wire grids must be simulated in 3D arrays.***

Each point that acts as a wire can be thought of as having a diameter of about 0.4 grid units in terms of its field effects. However, single points do not stop ions. ***This means all ions will pass through grids with single point wires.*** To create a non-ideal grid that captures ions requires that the wires be solid (*formed by one or more connected four point grid meshes in the manner solid electrodes/poles are created*).

Doubling or Halving Potential Arrays

SIMION provides quick methods to double or halve the size (*and thus density*) of the currently selected potential array (*its button depressed on Main Menu Screen*). These functions ***attempt*** to keep the array's existing geometry intact.

If a workbench instanced (*referenced*) array is doubled or halved, SIMION will automatically detect that the array has been doubled or halved when you re-enter the **View** function. SIMION will ask if you want to have the instance's definition sized and adjusted to fit the doubled or halved PA instance exactly where the previous PA instance fit (*handy*). ***You should then save the adjusted* .IOB *file.***

The Double Function

The **Double** function can be used to double the size of any potential array. ***The main use for doubling is to increase the grid density of the array to improve its ability to model the potential fields created by its electrodes/poles.*** Array doubling eats up more RAM. Thus you should avoid overdoing it.

Procedure to Double an Array

The procedure to double a potential array is simple (*starting at Main Menu Screen*):

1. Select the PA to double (*depress its button if not already depressed*).
2. Click the **Double** button or hit the **<d>** key.
3. SIMION will ask you if you're sure you want to double the potential array.
4. You also will be asked if you want to interpolate electrode/pole potentials. Select the default (**NO**) if you don't know what this means (*discussed below*).

Remember to Refine Doubled Arrays

The doubling process inserts new points into the array and the potentials of existing non-electrode/non-pole points become unreliable. ***Thus you must always refine any doubled potential array before you attempt to use it for ion flying.***

Memory Requirements for Doubled Arrays

Each time you double a 2D array you increase its size by approximately a factor of four (*3D arrays increase by about a factor of eight*). The exact size required is:

New Size = (2nx -1)(2ny -1)(2nz -1)
Where nx, ny, and nz are the current array dimensions

SIMION will refuse to double an array in a PA memory region with insufficient memory allocated for the doubled potential array size. Your recourse when this happens is to save the PA, remove all PAs from RAM, and reload the PA into a larger allocated memory region (**Max PA** *panel set to needed memory value or above before reloading the PA*).

How SIMION Conserves Geometry

When you double an array, SIMION places rows and columns of non-electrode/non-pole points between existing points in 2D arrays (*planes of points in 3D arrays*). It then tries to decide which of these newly inserted points to convert to electrode/pole points. ***The basic rule is if adjacent points in the x, y, or z directions are electrodes/poles of the same potential then the point in question is converted to an electrode/pole of that potential too.*** The process requires multiple passes through the array to fully resolve the fate of all the new points.

How Successful is this Approach?

The advantage of this process is that grids are conserved (*not made solid by doubling*). Electrode/pole surfaces that are ***aligned*** with the potential array are also re-sized faithfully. ***Unfortunately, diagonal and curved surfaces become more jagged.***

One Way to Fix Jagged Surfaces in Doubled Arrays

The **Modify** function can be used to examine the results of doubling. If there are jagged surfaces that need a bit of smoothing, use the various **Modify** filling tools (*lines, circles, and boxes*) to smooth out your array.

Reducing Jagged Surfaces Via Geometry Files

The best way to painlessly reduce jagged surfaces when doubling is to make use of geometry files. If all the geometry for the potential array is defined in a geometry file (*see Appendix J*), smooth doubling becomes a snap! The following procedure shows you how:

1. **Double** the potential array.
2. Enter the **Modify** Function and Click the **GeomF** button to access geometry files.
3. Select the appropriate geometry file (*if not already selected*).
4. Erase the geometry definitions in the potential array (*using the* **erase** *button*).
5. Set the scaling factor to 2.0 to adjust for doubling (*e.g. 4.0 if doubled twice*).
6. Insert geometry into potential array (*using the* **insert** *button*).

The beauty of using geometry files is that SIMION automatically creates the optimal geometry for the grid density of the array (***providing you scaled properly***). All surfaces converge on their ideal shape and locations as the grid density is increased (*through successive doubling*). The only limit is RAM or patience with virtual speeds (*as always*).

Interpolating Electrode/Pole Potentials

The **Double** function also supports interpolation of electrode/pole potentials as an option. When this option is active, SIMION converts any newly inserted point (*due to the doubling process*) to an ***interpolated*** electrode/pole point if it is adjacent to two electrode points of ***differing*** potentials. The actual potential given to this new point is the average (*mean*) of the

potentials of the two adjacent electrode/pole points. This option is useful in certain special cases:

Doubling Surfaces with Non-Uniform Potentials

Interpolation is useful when doubling a magnetic array with non-uniform surface potentials (*surface potential gradients*). Doubling with the interpolation option active preserves the existing gradients (*at least in a piece-wise linear sense*) in the doubled array.

Interfacing Fields Between Potential Arrays

The interpolating option is also useful in creating an electrode gradient boundary around a potential array to allow it to interface properly into a larger potential array (*e.g. an emission point inside a larger cavity*).

For example, let's say we want a high point density model of a field emission point inside some other potential array. One approach that will work is to create a model of the outer cavity including the point itself (*somewhat crude*). Now refine this array and save it.

The next step is to create the high detail inner potential array. This could be done by starting with a refined copy of the outer potential array and modifying it. The **Modify** function would be used to convert a small boundary around the point into electrode points and then re-size the array to just the region around the point. The **Replace** function with **Find** active (*with appropriate boundary marking*) would be used to convert array boundary points to electrodes without changing their potentials. **Double** would then be used with the electrode interpolation option active (*one or more times*) to obtain a higher density point. **Modify** would then be used to better define the point's shape, followed by **Refine**, and finally by removing the outer boundary of electrode points with **Modify** (*optional*).

At this point, an instance of the point's detailed array can be superimposed on the instance of the outer cavity. The use of this approach insures field consistency across the instance boundaries (*though not necessarily correctness*).

The Halve Function

The **Halve** function can be used to halve the size of any potential array. Halving is the exact reverse of doubling. ***Its main use is for reducing large 2D arrays to a reasonable size so they can be converted to manageable 3D arrays with*** **Modify**.

Procedure to Halve an Array

The procedure to halve a potential array is simple (*starting at Main Menu Screen*):

1. Select the PA to halve (*depress its button if not already depressed*).
2. Click the **Halve** button or hit the **<h>** key.
3. SIMION will ask you if you're sure you want to halve the potential array.
4. You also will be asked which halving method to use. Select the default method if you don't know what this means (*discussed below*).

Remember to Refine Halved Arrays

The halving process deletes points from the array and the potentials of existing non-electrode/non-pole points become unreliable. ***Thus you must always refine any halved potential array before you attempt to use it for ion flying.***

How SIMION Conserves Geometry

When you halve an array, SIMION uses one of two methods (*selected by you*) to *attempt* to conserve array geometry (*emphasizing conserving electrodes/poles or minimizing distortions*).

Copying Alternate Points (Default Method)

The default array reduction method just copies alternate points into the new reduced size array. This approach introduces the minimum amount of geometry distortion. ***However, grids and other thin details can be missed.***

Conserving Electrode/Pole Points

The second option tries to conserve electrode points by looking at the four (*2D*) or eight (*3D*) points that are to be converted into a single point in the halved array. If any of these points is an electrode point the equivalent point in the halved array will be made an electrode point too. ***While this approach conserves grids and thin electrodes it also can introduce more distortion.***

Examine an Array After Halving

If you **halve** an array be sure to use **Modify** to examine/correct for distortions. ***Remember an array can be doubled and then halved without distortion.*** However, the converse can introduce geometry distortions.

Reducing Distortions Via Geometry Files

The best way to painlessly reduce distortions when halving is make use of geometry files. If all the geometry for the potential array is defined in a geometry file (*see Appendix J*), smoother halving becomes a snap! The following procedure shows you how:

1. **Halve** the potential array.
2. Enter the **Modify** Function and Click the **GeomF** button to access geometry files.
3. Select the appropriate geometry file (*if not already selected*).
4. Erase the geometry definitions in the potential array (*using the erase button*).
5. Set the scaling factor to 0.5 to adjust for halving (*e.g. 0.25 if halved twice*).
6. Insert geometry into potential array (*using the insert button*).

The beauty of using geometry files is that SIMION automatically creates the optimal geometry for the grid density of the array (providing you scaled properly). All surfaces retain the best approximation of their shape and location as the grid density is decreased (*through successive halving*). ***Note: When the array gets too small for its geometry, even the optimal can get pretty bad.***

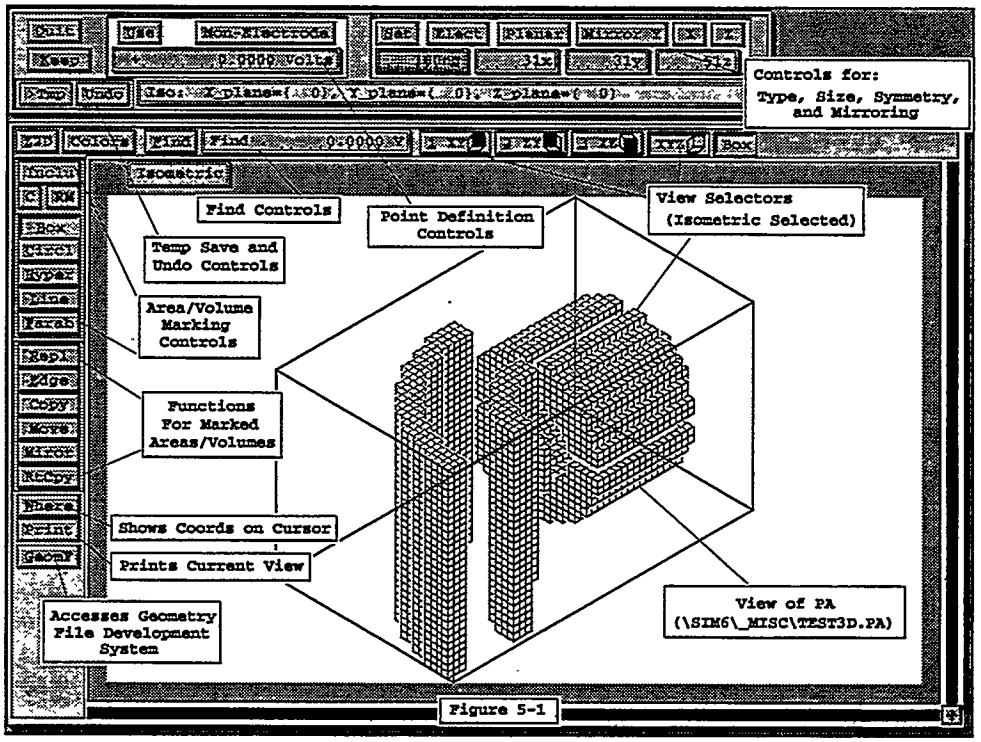
The Modify Function

The **Modify** function is used to change the type, size, symmetry, mirroring, and geometry definition of potential arrays. It also provides access to the interactive geometry file development system (*an advanced SIMION feature - Appendix J*).

The **Modify** function is accessed from the Main Menu Screen by selecting a PA (*depressing its button*) and then clicking the **Modify** button or hitting the **<m>** key. A sample of the Modify Screen appears below (*Figure 5-1 below*).

The Temp Save and Undo Buttons

SIMION supports what might be thought of as an ***anticipated*** undo function to help you protect yourself from self-inflicted disasters. The **>Tmp** button is used to quick save the current PA image to the **TEMP_0.PA** temporary file. The **Undo** button reloads the most recently saved copy of the PA: **TEMP_0.PA** (*>Tmp used*) or the currently saved version of PA (*if it exists*). ***Be sure to click the >Tmp button before you do anything really dangerous.*** The undo capability is implemented in this manner to provide protection for the cautious without incurring the performance/storage penalties that saving huge amounts of data for an automatic undo would require.



Changing PA Type, Size, Symmetry, and Mirroring

Controls for viewing and changing potential array type, symmetry, mirroring, and size appear in the upper right corner of the Modify Screen. These controls normally display the characteristics of the current PA. However, you can change any of these parameters and then click the **Set** button to actually change the potential array.

If you change any of these parameters (*without clicking the Set button afterwards*), their original values will automatically be restored when you move the cursor out of the region. This can accidentally happen when you are trying to point to the **Set** button. ***Thus it is recommended that you change the desired parameters and then hit the <s> key to activate the*** **Set** ***function to avoid the chance of accidental resetting.***

Changing Array Type

Modify can be used to switch potential arrays between electrostatic and magnetic types. For example, if an electrostatic the array is to be made magnetic, click the **Elect** button (*the title will switch to* **Magnt**).

When a magnetic potential array is selected access to the **ng** panel will be unblocked. The default value is 100. You should set ng to the number of grid units of pole gap so that the magnetic potentials (*Mags*) are reasonably direct indications of the magnetic field in gauss between the pole pieces (*discussed in Chapter 2*).

Changing Array Symmetry

SIMION allows 2D arrays to have either planar or cylindrical (*surface of revolution*) symmetry. ***All 3D arrays must be planar (SIMION enforced).*** For example, if you want planar click the **Cylind** button (*title will switch to* **Planar**).

Changing Array Mirroring

You have the option of mirroring the potential array for negative x, y, and z values. This can be used to reduce array sizes if the modeled real-world device has the appropriate mirroring symmetries. You can select the desired mirroring. If the combination is illegal, SIMION will beep and enforce a legal mirroring button status. The possible array mirroring combinations are: ***None, X, Y, Z, XY, YZ, XZ, XZY.*** The legal mirroring combinations by array category are:

All 3D arrays:	All mirroring combinations are legal
Planar 2D arrays:	All mirroring except z are legal
Cylindrical 2D arrays:	y mirroring is required, x is legal, z is illegal

Changing Array Size

The x, y, and z panels are used to specify array dimensions. Note: 2D arrays are specified when z = 1. 3D arrays are specified when z > 1.

Array Size Bounded by PA's Memory Region

The x, y, and z array dimension panels will not permit array dimensions that would exceed the points allocated to the PA's memory region. Your recourse when this happens is to save the PA, remove all PAs from RAM, and reload the PA into a larger allocated memory region (**Max PA** *panel set to needed memory value or above before reloading the PA*).

How SIMION Changes Array Size in Modify

When array dimensions are reduced (*x, y, or z*) within **Modify**, SIMION simply removes the excess points ***and*** whatever geometry they contained from the new reduced size array.

When an array is expanded SIMION ***normally*** inserts zero potential non-electrode/non-pole points for all new points in the expanded array. The exception involves converting 2D arrays to 3D arrays (*discussed immediately below*).

Converting a 2D Planar Array to a 3D Array

A 2D planar array can be converted into a 3D array (*assuming enough memory is allocated to the PA's memory region*) by setting the z dimension to a value greater than one. SIMION will try to help you by asking if you want to copy points from the z = 0 plane into the newly created z planes. ***This duplicates the z = 0 geometry in all z planes.***

Converting a 2D Cylindrical Array to a 3D Array

A 2D cylindrical array can be converted into a 3D array as with 2D planar above. SIMION will try to help you by asking if you want to ***rotate copy*** points from the z = 0

plane into the newly created z planes. The points are ***normally*** copied into the new z planes assuming a ***surface of revolution*** about the x-axis.

SIMION will ask if you want r boundary electrode/poles extended into the corner regions. ***If you elect this option, r boundary electrode/pole points will be extended to the 3D boundary limits (forming rectangles instead of circles).*** This is normally desirable (*if not visually*) to assure that refining retains more of the infinite r extent assumption for r boundary electrodes/poles that the cylindrical symmetry assumption implied.

Trick: If we have a cylindrical symmetry 2D array of x = 100, y = 20, z = 1 and change the size to x = 100, y = 39, z = 39. SIMION will recognize that it has exactly enough room to convert the 2D array into its 3D equivalent and will ask you if that is what you want (SIMION will also recognize half cylinders too e.g. x = 100, y = 20, z = 39 or x = 100, y = 39, z = 20). This is a very quick way to convert cylindrically symmetrical elements into a 3D array.

Converting a 3D Array to a 2D Array

A 3D array can be converted to a 2D array (*with obvious loss of data*) by setting the z dimension to a value of one. Note: Only the z = 0 layer will remain. ***If something important exists on some other z layer be sure to move or copy it to the z = 0 layer before converting from 3D to 2D.***

Selecting From the Available Views

There are four available views: **XY**, **ZY**, **XZ**, and **XYZ** (*isometric*). These views are selected via view selection buttons (*see Figure 5-2 below*). The first three (**XY**, **ZY**, *and* **XZ**) are 2D views. The fourth (**XYZ**) is an isometric 3D view. ***Only the* XY *view is allowed with 2D arrays.*** To select a view, click on its button (*if not blocked*).

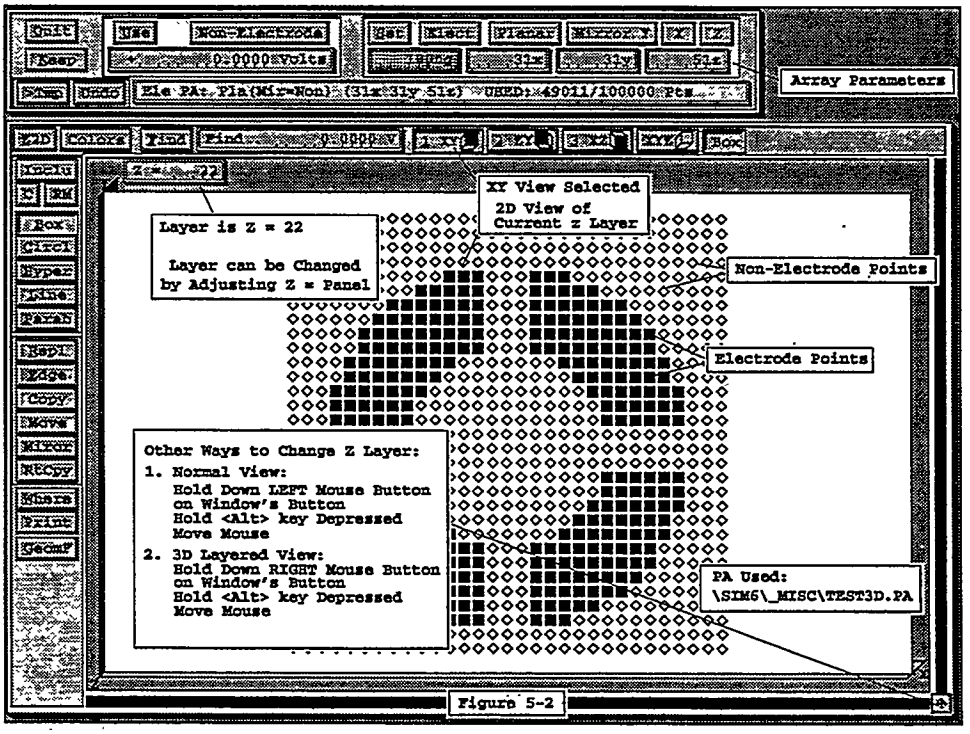


Figure 5-2

Layer Displays in 2D Views

The three 2D views (**XY**, **ZY**, *and* **XZ**) display a layer (*or plane*) of array points (*Figure 5-2 above*). ***In the case of 2D arrays the view is always the z = 0 layer of the XY view.*** This means when you are viewing a 3D array in the **XY** view you are looking at a specific z layer or plane of points (**x** *layer in* **ZY** *view and* **y** *layer in* **XZ** *view*). In Figure 5-2 above, the view is of the **XY** view of the z = 22 layer. This example uses the **\SIM6_MISC\TEST3D.PA** potential array. ***You might want to load this 3D array yourself and follow along with the manual's examples.***

Changing the Displayed Layer in a 2D View

There are many occasions you will want to look at the layers of a 3D array with one or more 2D views. SIMION provides several methods for selecting the layer to display. ***The recommended way is to use 3D layered view layer switching*** (*discussed below*):

Using the Layer Panel

The most direct method involves using the layer panel itself (*upper left corner of 2D view screen*). The value of the desired panel can be entered directly into the panel or the mouse buttons can be used to click the desired digit up or down (***right mouse button for up, left mouse button for down***).

Using the Window Button

The window button can (*lower right corner*) be used in two ways to change the displayed layer:

Normal View Layer Switching

This method changes the layers of the normal 2D view as you move the mouse.

1. Hold down the ***left*** mouse button while on the window button (*lower right corner of window*).
2. Also hold down the **<Alt>** key.
3. Move the mouse about and the window's layer will change. ***Holding down the right mouse button <u>too</u> will speed up re-displays.***

3D Layered View Layer Switching

This method changes the layers by using a stacked 3D isometric view as you move the mouse. Each layer appears in its location (*much like in a deck of cards*). The screen drawing is quite fast and you get a better feeling about where you are. ***This is the recommended way to change layers.***

1. Hold down the ***right*** mouse button while on the window button (*lower right corner of window*).
2. Also hold down the **<Alt>** key.
3. Move the mouse about and the window's layer will change. ***Holding down the left mouse button <u>too</u> will speed up re-displays.***
4. Release the mouse button(s).

When you adjust layers and switch between views SIMION tries to keep the lower left corner point (*min. x, y, and z*) as the min. in all 2D layer views. This can result in apparent 2D zooming (*reduced view size*). ***You can always restore the full view of the layer by clicking the <u>right</u> mouse button while the cursor is in the view area*** (*or clicking the Z2D button or hitting the <z> key*).

Changing Layers From an Isometric View

SIMION also supports changing all three layers (*x, y, and z*) from within an isometric view via the notion of 3D isometric pointing (*Figure 5-3 below*). ***This is also useful for cutaway views of 3D arrays:***

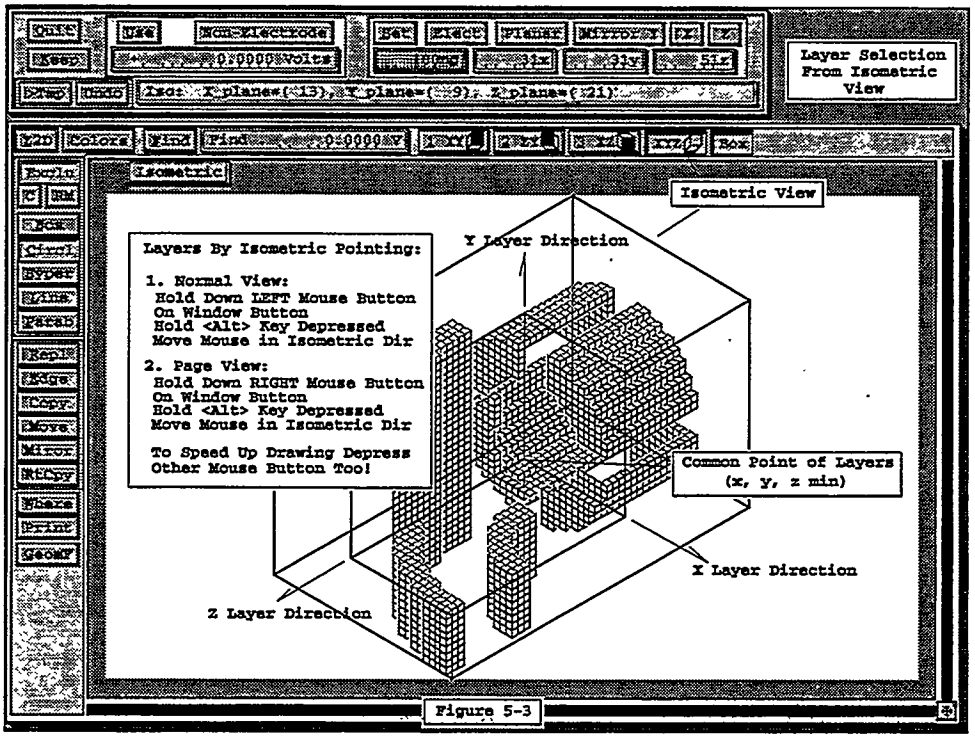


Figure 5-3

Normal View Layer Switching

This method changes the layers of an isometric view as you move the mouse (**XYZ** *button depressed*).

1. Hold down the ***left*** mouse button while on the window button (*lower right corner of window*).
2. Also hold down the **<Alt>** key too.
3. Move the mouse in the desired isometric direction(*s*) and the x, y or z layer(*s*) will change. ***Holding down the right mouse button <u>too</u> will speed up re-displays.***

Page View Layer Switching

This method changes the layers of an isometric view as you move the mouse (**XYZ** *button depressed*). This method makes use of page view drawing (*faster*).

1. Hold down the ***right*** mouse button while on the window button (*lower right corner of window*).
2. Also hold down the **<Alt>** key.
3. Move the mouse in the desired isometric direction(*s*) and the x, y or z layer(*s*) will change. ***Holding down the left mouse button <u>too</u> will speed up re-displays.***
4. Release mouse button(*s*).

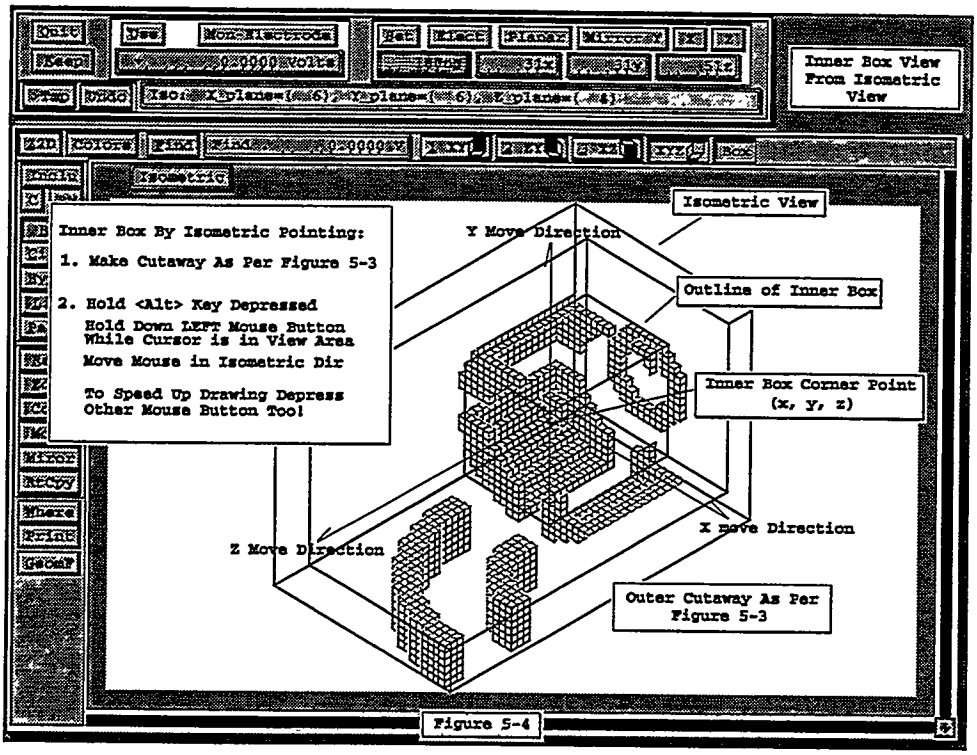


Figure 5-4

Showing Interior Volumes From an Isometric View

The ability to create cutaway views as shown in Figure 5-3 above can be very useful. SIMION also adds the ability to define an inner box to be viewed within this cutaway. This can be very helpful to see geometry within the inner box that would have been blocked by the outer cutaway region.

1. Create cutaway view using methods described above.
2. Move cursor into view area.
3. Hold **<Alt>** key depressed.
4. Depress ***either*** mouse button and drag mouse in isometric directions to define inner volume (*box*). ***Holding down the other mouse button too will speed up re-displays.***
5. Release mouse button(*s*).

Zooming

There are several ways to zoom in and out in the current view (*whether 2D or isometric*):

Zooming Via Area Marking

The easiest way to zoom in is to mark the desired area (*by box, circle, or line mark below*), and click the ***right*** mouse button (*any prior marks will be restored after zooming*). Ten levels of zoom are remembered. To zoom out one level just click the ***right*** mouse button ***without*** making a mark first. To zoom back in a level hold a **shift** key depressed and click the ***right*** mouse button without making a mark first (*prior marks are conserved*).

Zooming in Page View

You can also zoom using the window button in page view. If you hold down the ***right*** mouse button while the cursor is on the window's button (*page view*) you will see a blue rectangular outline around the currently displayed area. If you hold down the **Ctrl** key while keeping the ***right*** mouse button depressed and move the mouse you can change the size of this rectangular outline (*zooming is fast*).

Zooming in Normal View

You can zoom using the window button in normal view. Hold down the ***left*** mouse button while the cursor is on the window's button (*normal view*). If you hold down the **Ctrl** key while keeping the ***left*** mouse button depressed and move the mouse you can change the area viewed (*zooming is slower than page view*).

Scrolling

To scroll a zoomed view move the cursor to the window's button. Hold down the ***left*** (*normal view*) or ***right*** (*page view*) mouse button. Now move the mouse in the desired direction (*keeping the mouse button depressed*) and the view will scroll (*2D scrolling supported*).

You can also demand scroll when the cursor is in the view area. Hold the **Ctrl** key depressed and move the cursor. When you try to push the cursor outside the view area, the view will automatically scroll in that direction.

The Where Button - Displaying Position Numbers

If you depress the **Where** button (*Figure 5-1*). The cursor will display the position numbers on all 2D views near the crosshairs cursor's intersection point. ***This is handy for making sure that you are pointing at the desired location when marking volumes below***:

An Introduction to Marking Areas and Volumes in Modify

Generally you must first mark an area or volume in **Modify** before you can access functions like **Replace** or **Move**. Thus, the ability to mark a region of points properly is crucial to using **Modify**. SIMION treats marks made in 3D isometric views differently from marks made in 2D views. Marks in isometric views are ***only*** used for zooming. While marks in 2D views are used ***either*** for zooming ***or*** area/volume marking.

Marking a Region in an Isometric 3D View

SIMION uses the simplest notion of marking for 3D isometric views. Marks on an isometric view only support the zooming function. To mark a region for zooming move the cursor to one corner of the desired region and then drag the cursor to the diagonally opposite corner with the ***left*** mouse button depressed. SIMION will draw a rectangle around the marked screen area. Now click the ***right*** mouse button to zoom into the marked region (*or click* **Z2D** *button or hit the* <z> *key*).

Marking an Area or Volume in 2D View(s)

SIMION's notion of marking changes when you mark from within a 2D view (**XY**, **ZY**, *or* **XZ**). Marks in 2D views are used to define areas/volumes as well as zoom areas. We will focus our discussion on how areas/volumes are marked.

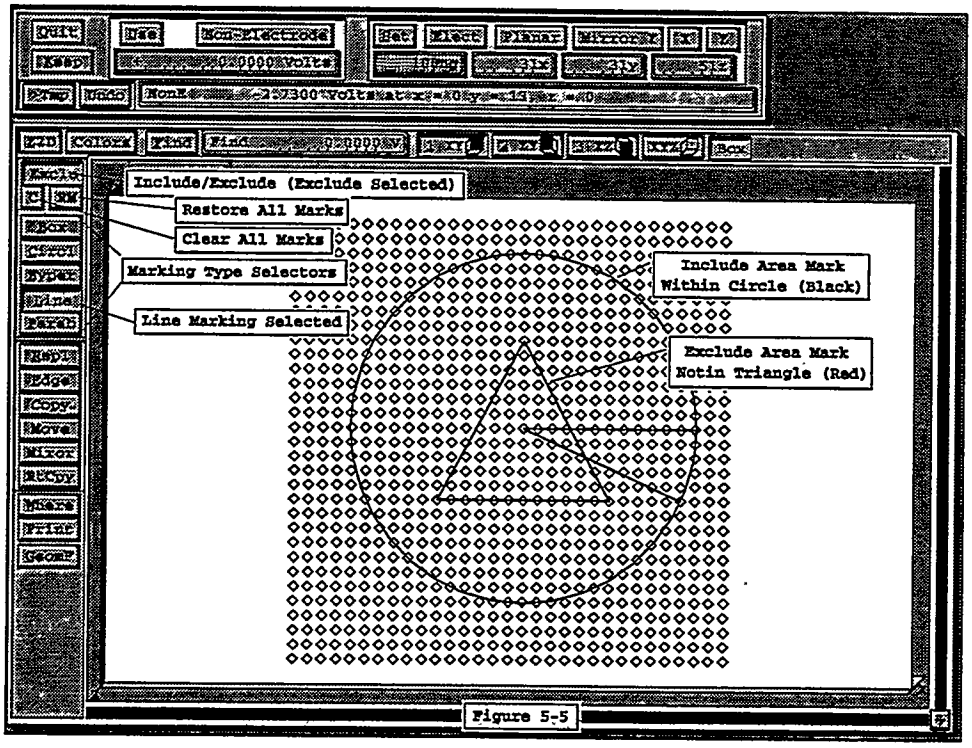


Figure 5-5

The Strategy for Marking Volumes

Whenever you mark an area within any layer in a 2D view, you are actually marking all layers with the same mark. ***Thus an area mark is really a volume mark.*** For example, if a circle is marked in any layer in an **XY** view, SIMION assumes a cylinder has been marked (*for a 3D array*).

Constraining the Marked Volume Further

In most cases we do not want to mark volumes that span all layers of the view. SIMION solves this problem by allowing you to mark areas (*volumes*) in the other 2D views to further constrain the marked volume. Thus if we switched to the **ZY** view and marked a rectangle that intersected the cylinder (*above*), SIMION would automatically limit the marked volume to the intersection volume of the two marked volumes (*a cylindrical disk*).

This means that the marked volume is always the volume of intersection of area marks in one to three of the 2D views (XY, ZY, and/or XZ). If you define area marked volumes that don't have a volume of intersection SIMION will object when you try to do something (*e.g.* **Move** *points*).

Types of Marks Supported

Modify allows you to mark areas with: Rectangles, circles (*or ellipses*), connected line segments, parabolas, or hyperbolas. You have the option of using a different type of mark for each area marked. This provides considerable flexibility in defining the shape of marked volumes (*Figure 5-5*).

The Notion of Include and Exclude Volumes

To further complicate your life, SIMION allows both include (*within*) and exclude (*notin*) volume marking. Thus you have the option of defining up to six area marks (*an include and exclude mark in each of the three 2D views*).

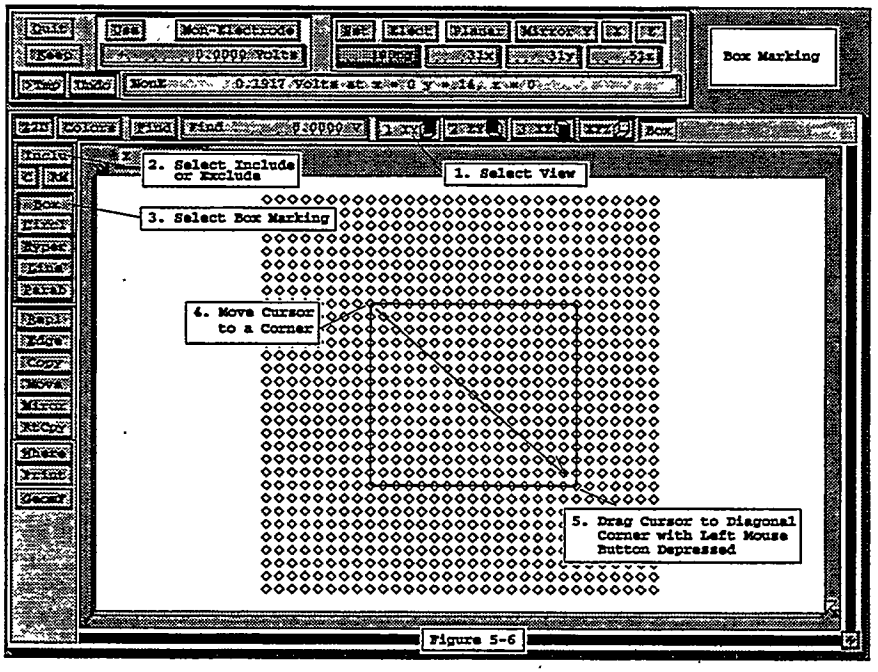
SIMION will consider a point to be within the marked volume if it is within the intersection volume of the include marks and outside the intersection volume of the exclude marks. Note: If you define include area marks that do not have a volume of intersection or exclude marks that do not have a volume of intersection, SIMION will object when you try to do something (*e.g.* **Replace** *points*).

The Modify Screen has a button in the marking control area to control whether a mark is considered to be include (*within*) or exclude (*notin*). ***Click the button to the desired marking type before starting the marking process*** (*Figure 5-5*).

In the example above, we could have also marked a triangular ***exclude*** mark within the circle mark in the **XY** view to create a circular disk with a triangular hole in it (*Figure 5-5 above*).

How to Select and Use the Various Area Marks

The recommended steps when creating a 2D view mark are to switch to the desired view, select include or exclude, select the type of mark (*e.g.* **Box**), and draw the mark on the view area. The following gives specific instructions about each marking type:



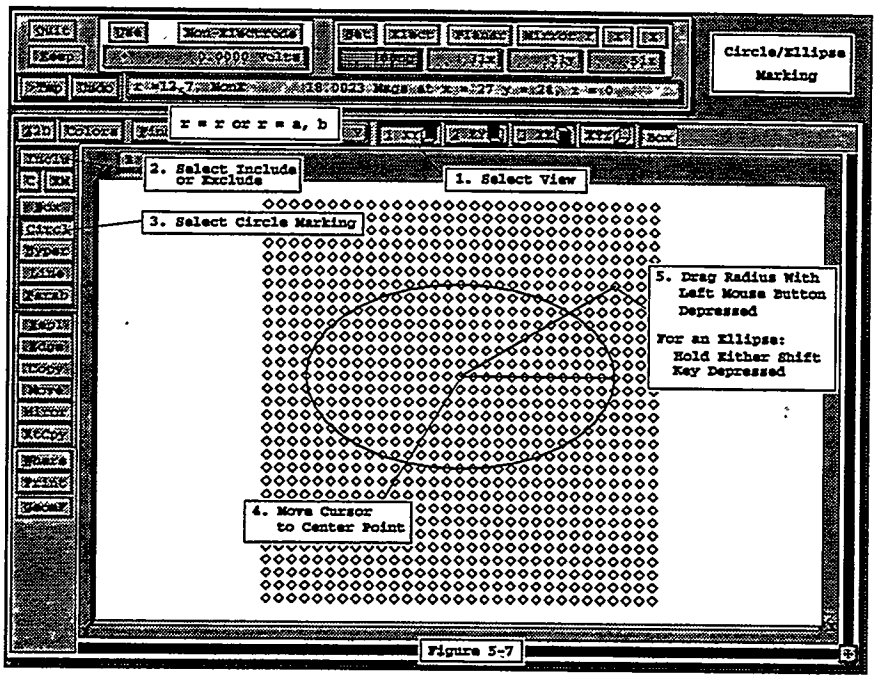
How to Use Box Marking

To mark a rectangle make sure the* Box *button is depressed (*Figure 5-6*). Now move the cursor to a corner of the box. Hold the ***left*** mouse button depressed and drag the cursor to the diagonally opposite box corner. Now release the ***left*** mouse button.

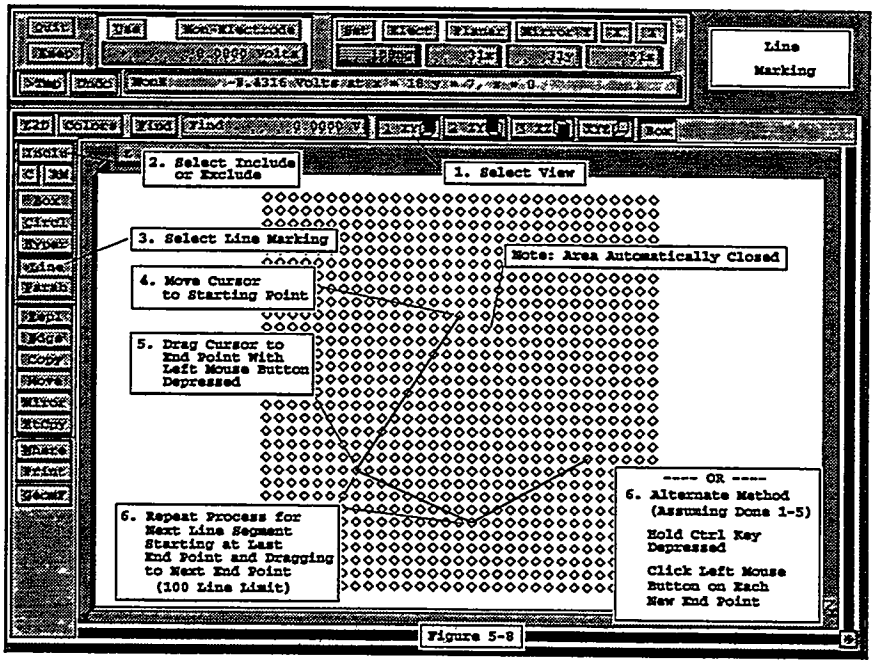
How to Use Circle (or Ellipse) Marking

To mark a circle make sure the* Circle *button is depressed (*Figure 5-7*). Now move the cursor to the center of the desire circle. Hold the ***left*** mouse button depressed and drag the cursor to the desired circle radius (*the status line will display the radius value when you pause*). Now release the ***left*** mouse button.

To mark an ellipse make sure the* Circle *button is depressed (*Figure 5-7*). Now move the cursor to the center of the desire ellipse. Hold the ***either* Shift** key ***and*** the ***left*** mouse button depressed and drag the cursor to the desired ellipse shape (*the status line will display the r = a, b radius values when you pause*). Now release the **Shift** key and ***left***



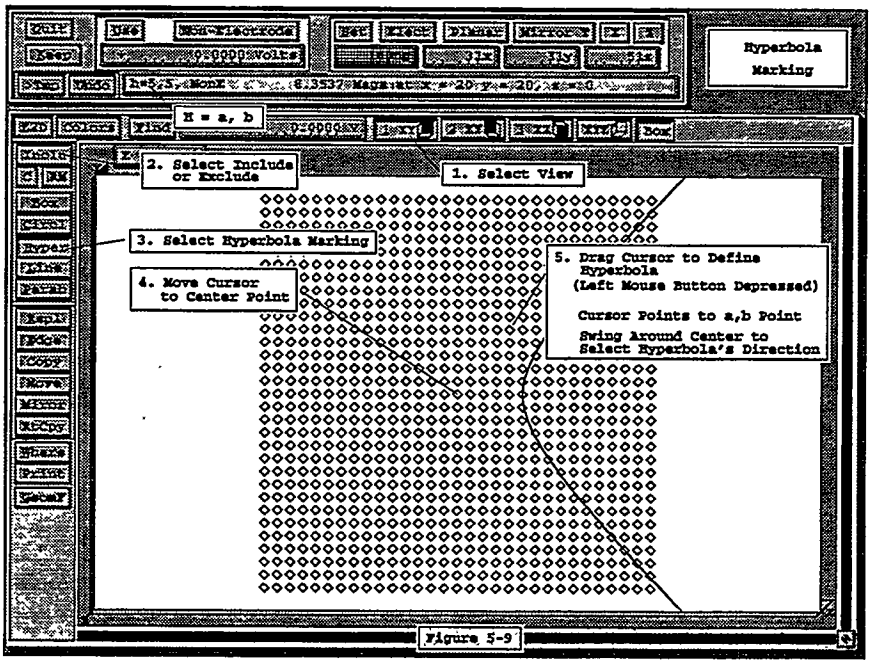
mouse button.



How to Use Line Segment Marking

To mark by line segments make sure the **Line** ***button is depressed*** (*Figure 5-8*). Now move the cursor to the beginning of the first line segment. Hold the ***left*** mouse button depressed and drag the cursor to the end of the first line segment. Now release the ***left*** mouse button. To add a line segment move the cursor to the end of the most recently defined line segment and drag an additional line segment as above. ***Up to 100 connected line segments can be used.*** SIMION will automatically add a closing line segment to areas marked by line segments (*last point same as first point*) if required. ***Hollow regions can be made by drawing them linked with the outer line boundary.***

The shortcut to drawing lines is to move the cursor to the beginning of the first line and click the *left* mouse button. Now hold the **Ctrl** key depressed. Move to each successive line connected point and click the *left* mouse button. Release the **Ctrl** key when you're done.



How to Use Hyperbola Marking

To mark with a hyperbola make sure the **Hyper** ***button is depressed*** (*Figure 5-9*). Now position the cursor at the hyperbola center point and drag the desired hyperbola (***left*** *mouse button depressed*). ***Note: Swing the cursor around the center point to select the desired hyperbola direction.*** The following equations are used:

$$y^2/b^2 - x^2/a^2 = 1$$
$$x^2/b^2 - y^2/a^2 = 1$$

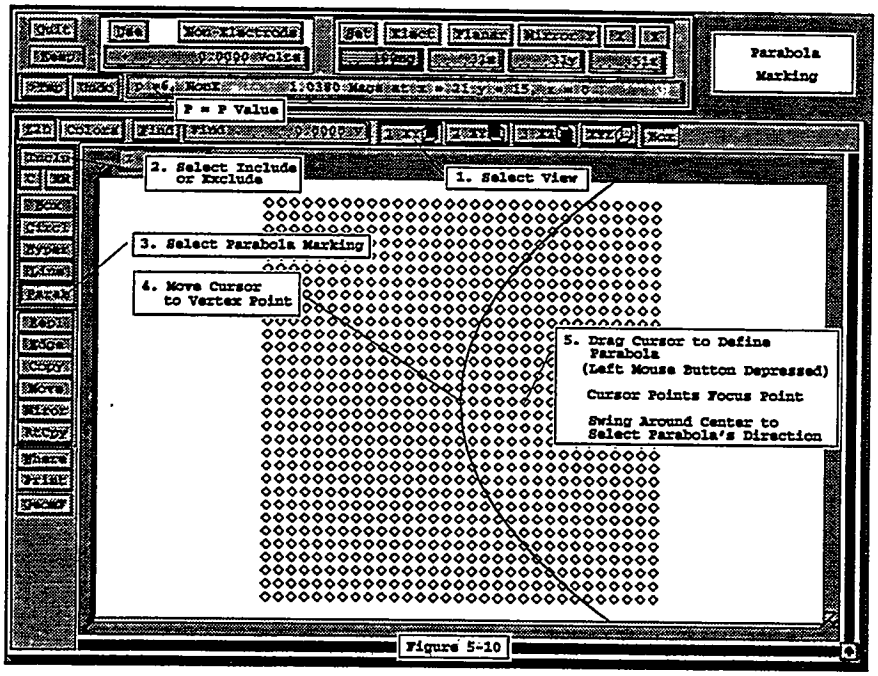
Moving the cursor about changes the values of a and b. The status line displays these values (*e.g. h = a,b*).

How to Use Parabola Marking

To mark with a parabola make sure the **Parab** ***button is depressed*** (*Figure 5-10*). Now position the cursor at the parabola vertex point and drag to the focus of the desired parabola (**left** *mouse button depressed*). ***Note: Swing the cursor around the center point to select the desired parabola direction.*** The following equations are used:

$$y = x^2/(4p)$$
$$x = y^2/(4p)$$

Moving the cursor about changes the values of p (*focus offset*). The status line displays this value (*e.g. p = p*).



Clearing and Restoring Marks

Clearing All Marks

To clear ***all*** marks click the **C** button (*just below the* **Include/Exclude** *button*). A copy of the current marks (*if any*) will be saved in mark memory (*Figure 5-5*).

Clearing a Specific Mark

A specific mark (*include or exclude*) in any 2D view can be deleted by switching to the desired 2D view, selecting the desired include/exclude status (*with the* **Include/Exclude** *button*), moving the cursor into the view object, and pressing the **Delete** key.

Restoring Marks From Mark Memory

Functions like **Replace, Move, Copy,** and the **C** button automatically save any current marks in the mark memory and then clear all marks. If you want to restore these cleared marks click the **RM** (*Restore Marks*) button (*just below the* **Include/Exclude** *button*).

Point Definition Area

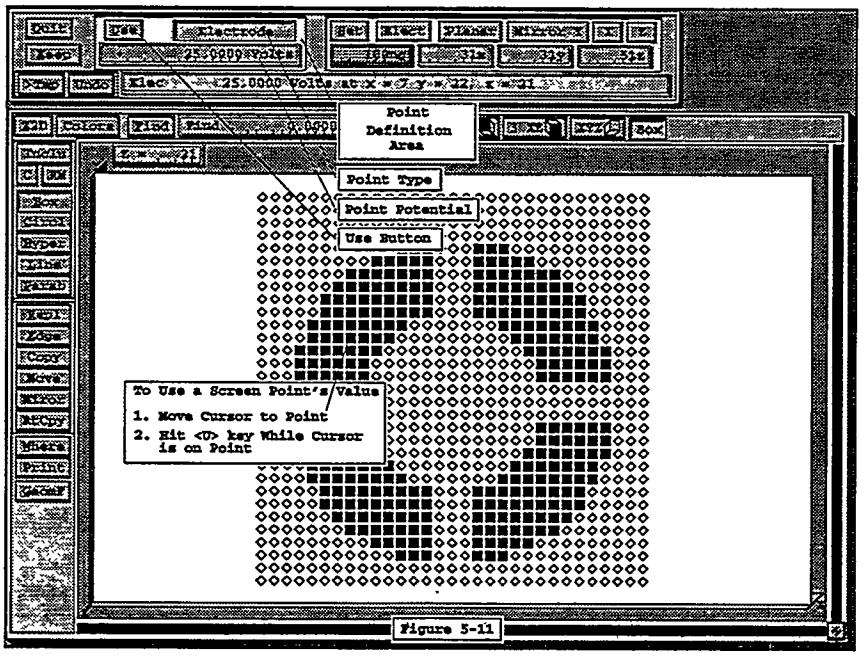
The area immediately to the left of the array definition area is the point definition area (*Figure 5-11*). The point values in the point definition area are used as the replacement values for marked points that are acted on by the **Replace** or **Edge** functions (*when* **Find** *is inactive*) as well as defining the points to search for when the **Find** function is active.

Viewing and Setting the Currently Defined point.

Control objects allow you to view/set the type (*e.g. electrode or non-electrode*) and potential (*e.g. Volts*) of the currently defined point.

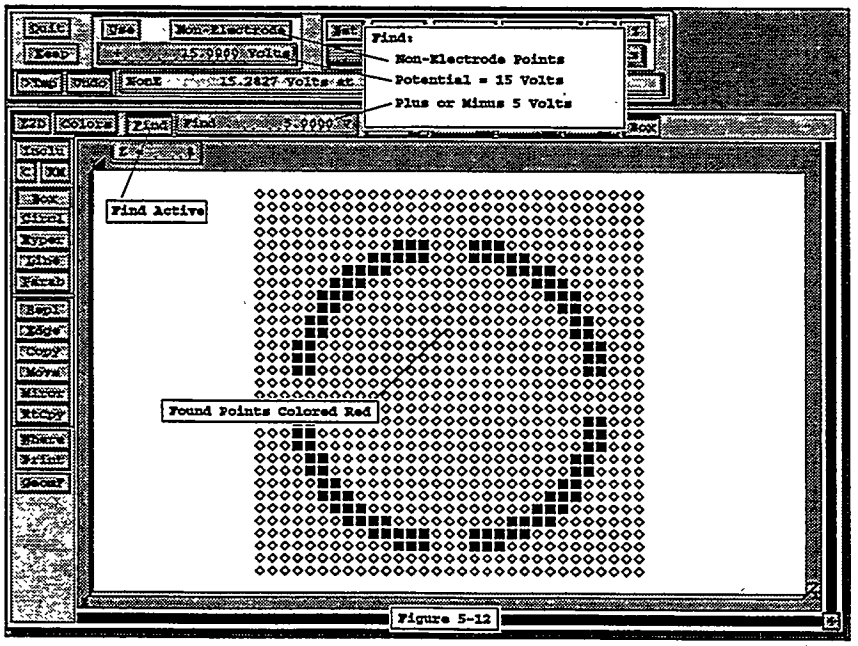
The Function of the Use Button

The **Use** button is provided to allow you to quickly transfer the type ***and*** potential of any point in the view to the currently defined point. ***To transfer the values from an array point in the current view, point to it with the cursor and hit the <U> key.***



Find Function

The **Find** function (*Figure 5-12*) allows finding the currently defined point type and potential (*in the point definition area*) ***plus or minus a delta potential***. The plus or minus delta potential is



new with SIMION 6.0. It allows more flexibility in point selection.

Moreover, when **Find** is active functions like **Replace, Move,** and **Copy** automatically use it to further limit the points selected (*e.g. points moved must be inside include volume, outside exclude volume, and be flagged as found*).

Functions Using Marked Volumes

Modify has a collection of functions that operate on marked volumes (*e.g.* **Replace, Edge, Copy, Move, Mirror,** *and* **Rotate Copy**). Each of these functions is accessed via a button on the left edge of the Modify Screen (*Figure 5-1*). ***Note: A volume (area for 2D arrays) must be marked before accessing any of these functions.*** A point is selected for processing via these functions if it is within the include volume and notin the exclude volume.

Using Find to Augment Selection

SIMION also allows **Find** (*if active - buttion depressed*) to further augment point selection with all of these functions. When **Find** is active a point is selected for processing via the function if it is within the include volume, notin the exclude volume, and is a found point based on the current find criteria (*point type and within potential range*).

The Replace Function

The **Replace** function replaces the selected points with points of the specified type and potential. There are three basic types of point replacement:

Simple Single Point Replacement

This is the simplest form of **Replace.** ***A volume must NOT be marked, and Find status is ignored.*** Set the point type and potential to the desired ***new*** value. Place the cursor at the point you want to replace, hold the **<Ctrl>** key depressed, and click the ***right*** Mouse button. This ***only*** changes the point under the cursor in the current layer (*not all layers*). It is very useful for quick point editing on a layer by layer basis. ***Find, if active, is ignored.***

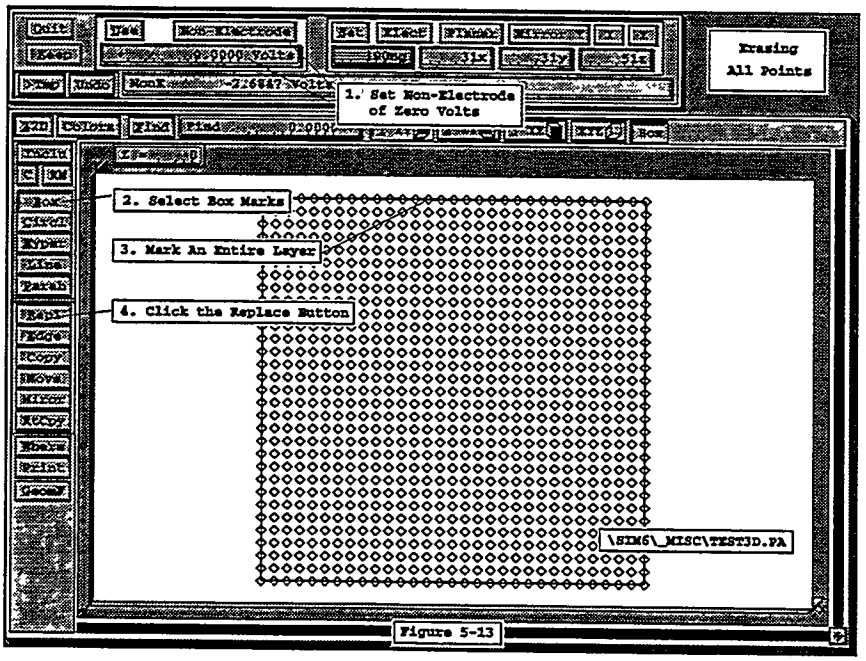
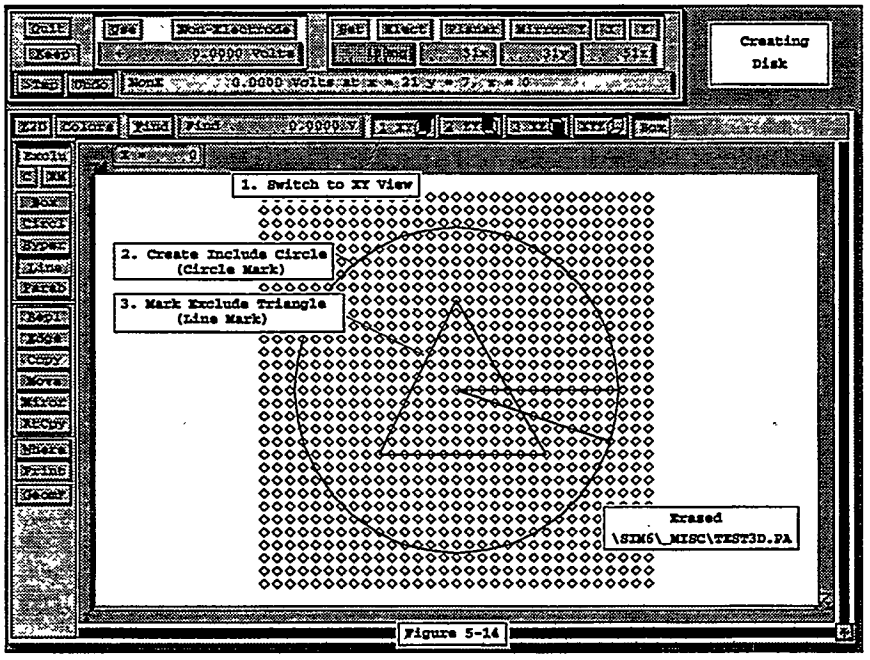


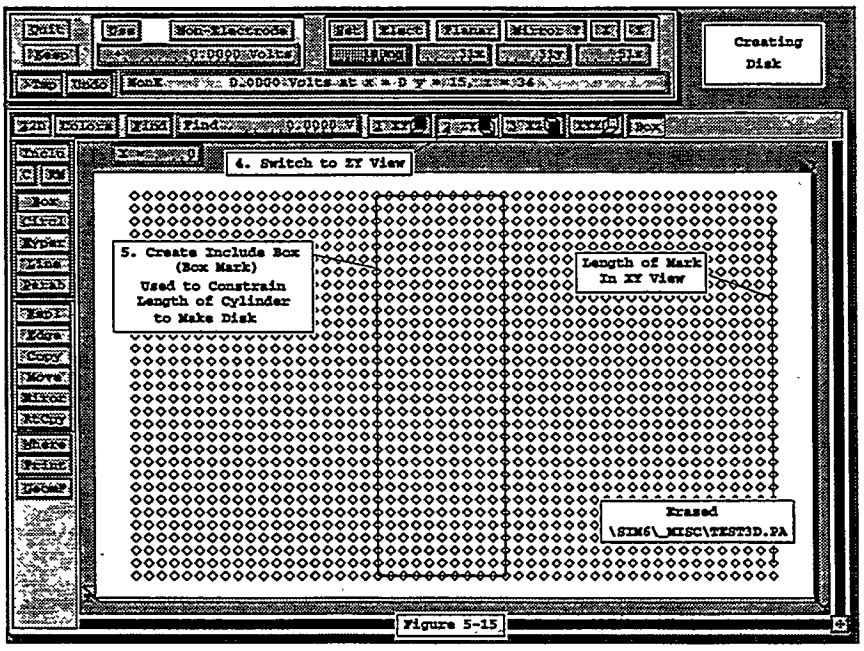
Figure 5-13

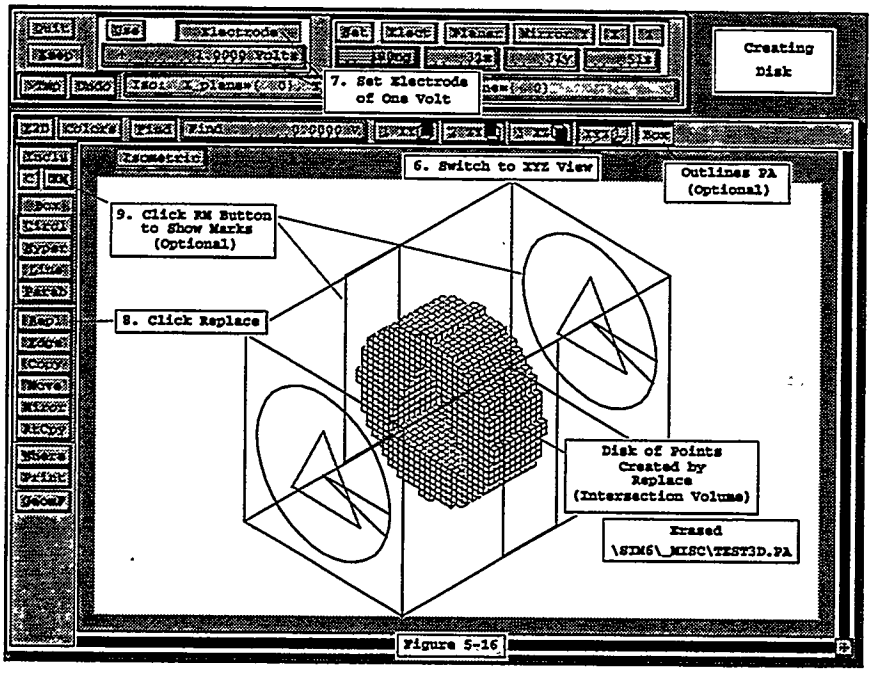
Point Replacement (*Find NOT Active*)

This is probably the most frequently used method of point replacement. The steps are: Mark a volume; set the point type and potential to the desired new value; and click the **Replace** button, hit the **<R>** key, or **<Ctrl>** ***right*** mouse button. SIMION will ask you if you're sure before replacing any points.

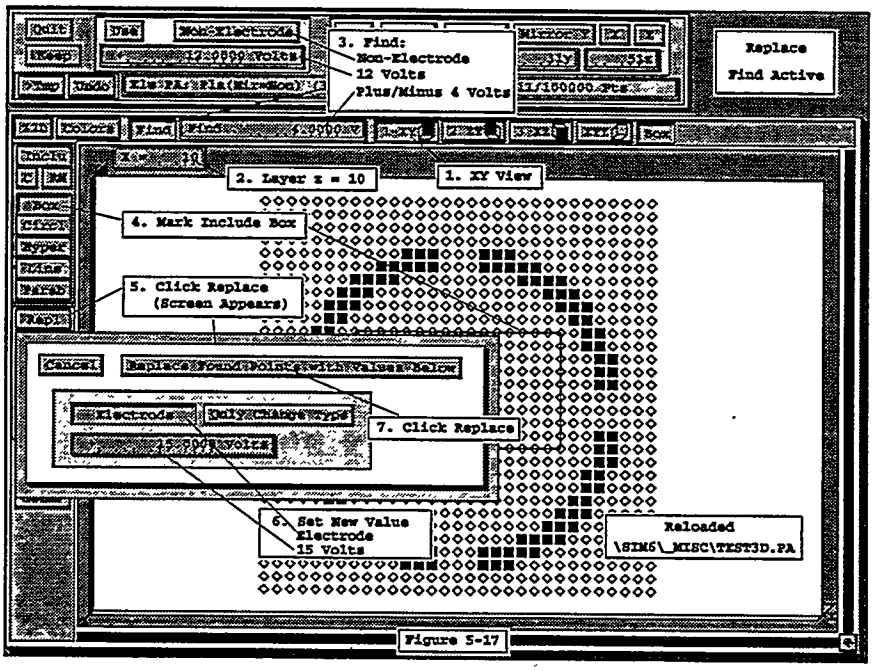
The Replace function can be used to erase all points in the potential array. The steps are: Mark an entire 2D layer with a box mark; set the point type to non-electrode/non-pole and potential to zero; and click the **Replace** button, hit the **<R>** key, or **<Ctrl>** **right** mouse button. Figure 5-13 (*above*) demonstrates this process on a memory copy of **\SIM6_MISC\TEST3D.PA** (*you might want to try this yourself*).







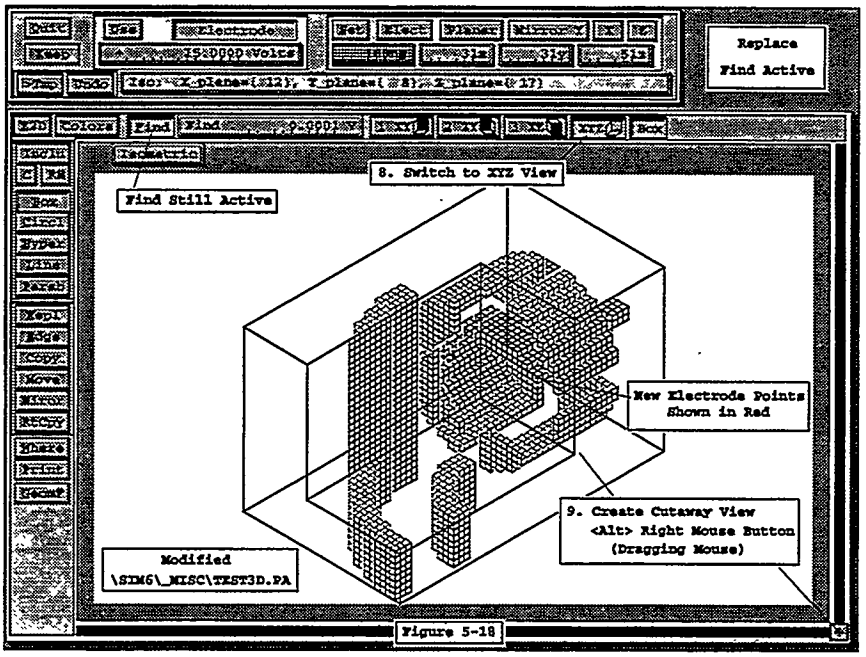
Another example of the Replace function involves creating a cylindrical disk with a triangular hole in it. Figures 5-14 through 5-16 demonstrate how this is done. This example starts with the erased PA above (*try to duplicate this yourself*).



Point Replacement (*Find is Active*)

The steps are: Find desired points (*Set the point type, potential, and range. Click* **Find** *on*); mark a volume; and click the **Replace** button, hit the **<R>** key, or **<Ctrl> right** mouse button. SIMION will display a new point definition screen. Select what you want the selected points to be and click the **Replace Found Points** button on this screen.

Note: You have the option of changing the selected points' type and potential or just changing the selected points' type. The latter is useful for creating fixed boundary conditions after refining (*by converting boundary points to electrode/pole*). Figures 5-17 to 5-18 show the creation of an electrode surface from found non-electrode points using a reloaded copy of **\SIM6_MISC\TEST3D.PA** (*try it yourself - good practice*).



The Edge Function

This function works just like the **Replace** function except that ***only*** the ***boundary points*** are ***replaced***. Thus while circular mark would create a solid cylinder, with replace the edge function creates a one point thick cylindrical shell. This function is useful for creating shaped grids.

Note: The **Edge** *function has no equivalent to* **Replace's** *simple single point replace.*

The Copy Function

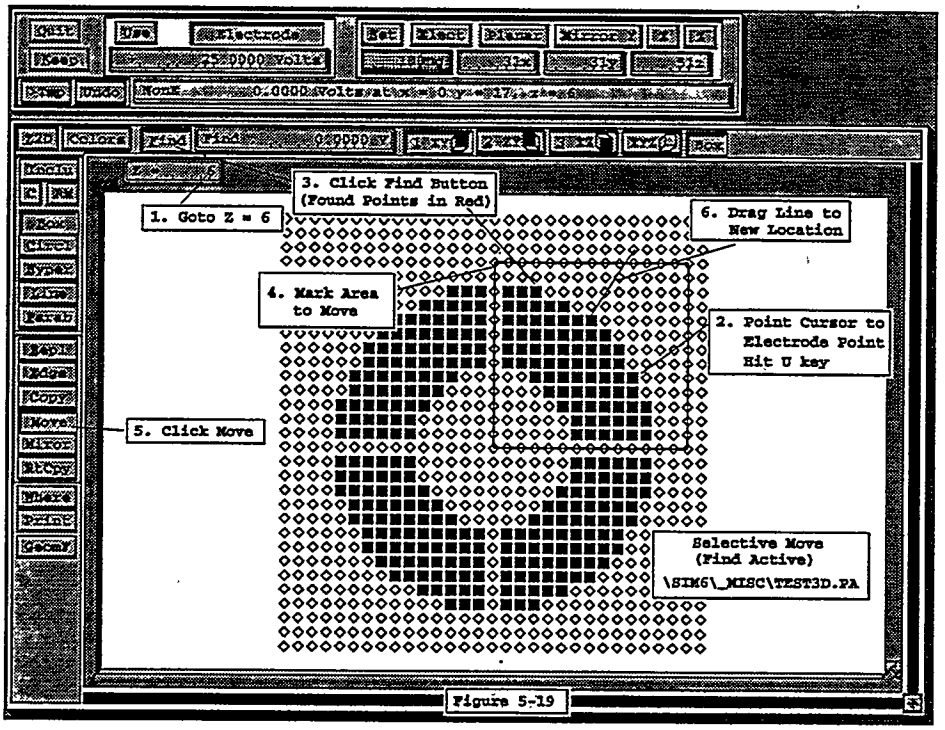
This function allows you to copy selected points. You are allowed full include and exclude volume capabilities as well as having **Find** active to further limit point selection. To copy a group of points make the appropriate volume marks, activate **Find** if desired, and click the **Copy** button. Now draw (*drag with the left mouse button depressed*) a line from the source to the destination to indicate relative location of the copy.

You can click the* RM *button to remark the initial volume if you need to make more than one copy.

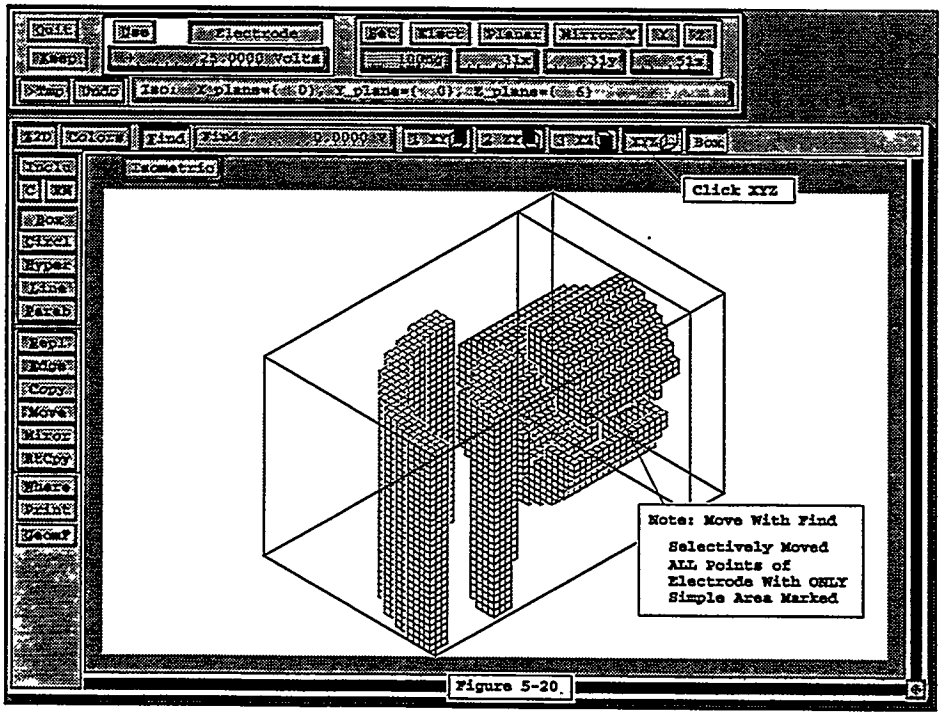
The Move Function

This function allows you to move a collection of points (*as with* **Copy** *above*). You are allowed full include and exclude volume capabilities as well as having **Find** active to further limit point selection. To move a group of points make the appropriate volume marks, activate **Find** if desired, and click the **Move** button. Now draw (*drag with the left mouse button depressed*) a line from the source to the destination to indicate relative location of the move.

Figures 5-19 and 5-20 below, show an example of using **Find** *to selectively move a complex electrode.*



You can click the RM button to remark the volume (Note: The marks have moved with the points) if you need to make more than one move (e.g. 3D move using another 2D view).



The Mirror Function

This function allows you to mirror a collection of points across a mirroring line (*plane in 3D*). You are allowed full include and exclude volume capabilities as well as having **Find** active to further limit point selection. To mirror a group of points make the appropriate volume marks, activate **Find** if desired, and click the **Mirror** button. Now draw (*drag with the left mouse button depressed*) a ***horizontal or vertical mirroring line (defines mirroring plane in 3D).***

*Note: **The mirroring line must be outside the marked volume.***

The Rotate Copy Function

This function allows you to create a volume of revolution via a rotating plane copy process. You are allowed full include and exclude volume capabilities as well as having **Find** active to further limit point selection.

To rotate copy a group of points, mark an include volume that is a single layer (*one point thick*) ***and extends to the pivot point*** (*of rotation*), activate **Find** if desired, switch to the edge 2D view of the marked volume (*marked region appears as a line*), and click the **RtCpy** button. Now start at the pivot point of revolution and draw (*drag with the left mouse button depressed*) the desired arc of revolution (*angle will be displayed on status line*).

The geometry files capability introduced below provides much more powerful rotate copy capabilities.

Geometry Files

SIMION also allows the creation of geometry definitions via an advanced feature called geometry files. The **GeomF** button on the left edge of the Modify Screen is used to access the geometry file development system.

This development system can be used to write geometry instructions in a geometry file (*an ASCII file with a* **.GEM** *extension*). This geometry file can then be compiled with the SIMION's geometry compiler (*accessible from within the geometry development system*) and the resulting geometry definitions inserted in the current potential array.

The process can be highly interactive. The user can erase the potential array from within the development system. Edit the geometry file. Then re-insert the newly re-defined geometry into the array, and immediately examine the results with **Modify**.

Geometry files are most suitable for complex geometry definitions (*2D or 3D*). They are also useful for defining geometry in arrays that may be doubled or halved. The geometry can be re-inserted in a doubled array by doubling the geometry insertion scale factor. This automatically gives the optimum geometry definition for the array's grid density.

See Appendix J for detailed information on the geometry language, defining geometry files, and using the geometry development system.

Refining and Fast Adjusting Arrays - Solving For Fields

Introduction

This chapter covers refining of potential arrays as well as how to make use of SIMION's **Fast Adjust** features. It is assumed that you have read the discussions of potential arrays in Chapter 2, 4, and 5. If not, read the material before proceeding further.

Refining Potential Arrays

Refining, in SIMION, means using the potentials of electrode/pole array points to *estimate* the potentials of non-electrode/non-pole array points. The refining process makes use of finite difference techniques to solve the Laplace equation numerically. ***Appendix E** (computational methods) provides specific details of the refining algorithms used in SIMION.*

A Glimpse of How Finite Difference Works

The finite difference techniques used in SIMION use the average potential of the nearest four (*2D arrays*) or six (*3D arrays*) neighbor points as the basis for estimating the potential of each non-electrode/non-pole point in the array. ***Note: Cylindrical symmetry 2D arrays have a radius correction applied to this averaging scheme.*** The entire array is scanned sequentially (*point by point*) and the estimated potential of each non-electrode/non-pole point is updated using the average of its nearest four or six neighbor points.

Each scan through the array is called an iteration. Each successive iteration (*scan through the array*) propagates the potentials of the electrode/pole points further throughout the potential array. This successive iteration process (*refining*) continues until the array comes into an equilibrium where no point in the array changes by more than a certain potential (*0.005 volts/Mags*) in a single iteration (*the convergence objective*).

At this point we proclaim that the convergence objective has been reached and the potentials of all non-electrode/non-pole points have been determined (*estimated*). ***Even if we iterate to a convergence objective approaching zero, there is no reason to believe that the potentials of all non-electrode/non-pole points have been determined precisely.*** The only exception is electrode/pole geometry that results in linear field gradients. All other fields are approximate. The quality of the estimate relates to how much the field curvature warps the grid cells (*distorts them away from being small flat plates*). ***The less warped the grid cells are*** (*warping is reduced by higher grid densities*), ***the better the potential estimates are for a given convergence limit.***

Using Various Tricks to Speed up the Process

Refining (*by successive iterations*) can involve many iterations. Naturally, the objective is to reach the convergence objective in the minimum number of iterations (*shortest time*). Unfortunately, while the approach described above is direct, it is generally pretty slow. SIMION makes use of various tricks (*techniques*) to speed up convergence.

Over-Relaxation

The first trick SIMION uses is over-relaxation. The over-relaxation technique determines how much a point's potential is to be changed in an iteration and multiplies this potential change by a percentage of over-relaxation:

Used_Delta_Potential = Normal_Delta_Potential (1.0 + Over_Relaxation_Factor)

The value of the over-relaxation factor varies from 0.0 (*none*) to 1.0 (*unstable*). The default value of 0.9 used by SIMION appears to work best for most potential arrays.

Over-relaxation works much like an elevator. ***Things work best if there are no sudden starts and stops*** (*everything is smooth and steady*). Thus SIMION starts refining and ends refining by automatically reducing the over-relaxation factor to help reduce solution kinks. The error term is also used to automatically adjust the over-relaxation factor. To avoid sudden changes in over-relaxation factor, SIMION averages the error term using a exponential moving average technique (*smoothes out changes in over-relaxation factor*). An historical memory factor is used to define the amount of exponential averaging (*default value is 0.7*).

Skipped Point Refining

The use of over-relaxation can reduce the number of required iterations by factors of 10 or more. ***However, even with over-relaxation, the number of iterations required to refine an array is generally proportional to n^2 where n is the number of points in the potential array.*** This can make refining large arrays of a million or more points take hours, days, or more on a 486 class PC.

The skipped point refining technique, developed by the author, yields refine times roughly proportional to n (*an enormous improvement for large arrays*). The technique involves dividing the array dimensions by powers of two into an initially small array (*containing perhaps one in every 16 points in each dimension*). This array is refined. The array is expanded by two. The new array points are estimated using points from the last refine. The array is refined again. This process continues until the array is finally refined without skipping any array points.

The problem with this approach occurs when electrode/pole points are skipped (*unseen*), their effects will have to be propagated when they at last become visible. ***This delay in potential propagation can slow refining later, destroying all the time savings.*** SIMION solves this problem by looking around for skipped electrode/pole points at every level of skipped point refining. If a skipped electrode/pole point is found, a flag is set and SIMION automatically takes the effects of the electrode/pole point into account even though it is not yet visible. The bookkeeping is complex, but the refining is very fast.

How SIMION Handles Array Boundaries

The refining process uses the average of the nearest four (*2D*) or six (*3D*) neighbor points to estimate the new potential for each non-electrode/non-pole point in the array. All this works just fine for points within the interior of the array. ***The problem is how to handle points on the array boundary.***

Points on Non-Mirrored Boundaries

The points on non-mirrored boundaries are handled according to their location:

Corner Points

Corner points use the average of the nearest ***two*** (*2D*) or ***three*** (*3D*) neighbor points to estimate their potentials.

Edge Line Points

Points on an edge line use the average of the nearest ***three*** (*2D*) or ***four*** (*3D*) neighbor points to estimate their potentials.

Edge Plane Points (3D only)

Points on an edge plane use the average of the nearest ***five*** neighbor points to estimate their potentials.

Points on Mirrored Boundaries

The points on mirrored boundaries have the appropriate mirrored points also added into the averaging process. For example, in the case of points along the interior x-axis in a 2D planar array with y mirroring, the average would involve ***four*** points: ***The three neighbor array points plus the point above in y***. Thus the ***plus one*** y point is assumed to also exist as a ***negative one*** y point (*y mirrored*).

This thought process can be extended to logically determine how mirrored averaging works for any point along a mirrored boundary.

What Does All This Mean?

The result of this approach is that SIMION extrapolates at the array boundary. ***If you do not bound the array boundary with electrode/pole points SIMION will struggle on.*** Whether the result is what you intended or not depends on how you defined the points near and on the array boundary.

When Electrode/Pole Shapes Touch the Boundary

When electrode/pole shapes actually touch the array's boundary, SIMION assumes (*using the above averaging scheme*) ***that the electrode/pole extends to infinity***. If this is not a valid approximating assumption, there is a good chance that the refined fields may not be particularly valid either.

When Electrode/Pole Shapes Do Not Touch the Boundary

When electrode/pole shapes do not actually touch the array's boundary, SIMION assumes (*using the above averaging scheme*) that the electrode/pole shapes are bounded. Be sure to leave enough space (*non-electrode/non-pole*) to the boundary so that field propagation around the boundary can be approximated.

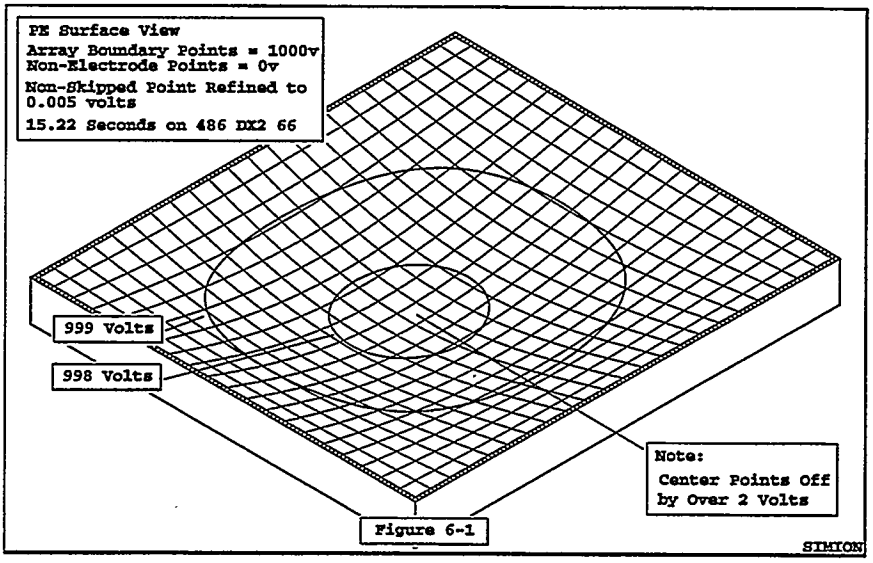
Remembering the Ground Can

Most optics devices live in grounded vacuum chambers. ***You ignore their existence at your peril.*** I have witnessed many examples of *"it doesn't model my problem properly"* that were the result of not including the ground can in the potential array. ***The better your problem is bounded the better your results are likely to be!***

Comparing Skipped Point Verses Non-Skipped Point Refining

It is useful to compare the results of refining with or without skipped point refining active. Figures 6-1 and 6-2 are potential energy surface views of a simple square 2D planar array (*101x by 101y*) refined by each technique. The boundary electrode points were set to 1,000 volts and the interior non-electrode points were set to zero (*0*) volts.

In theory, after refining is completed, all points in the array should have potentials of precisely 1,000 volts. This is definitely ***not*** the case with ***non-skipped point refining*** (*Figure 6-1*). As the Figure 6-1 shows, the center points of the array are only around 997.8 volts (*using default refining parameters - skipped refining turned off*). This is slightly more than 2 volts of systematic error when we have refined to an objective of 0.005 volts. ***How can this be? Remember, the convergence objective is based on change and not accuracy.***



The refining process terminated because no point ***changed*** more than 0.005 volts in the last refine iteration. Because it takes the most iterations (*time*) to communicate potentials to the points furthest away from the boundary points, the center points have not settled out. However, since this becomes almost an exponential approach process for distant points the convergence really slows down as you get close. Thus the voltage changes per iteration become very small and the refine terminates. This process required about 15 seconds on a 486 - 66 machine. ***If you set the convergence objective to 0.5 microvolts (the minimum), the time required to refine increases to about 30 seconds and the center points are now within 0.2 millivolts of 1,000 volts.***

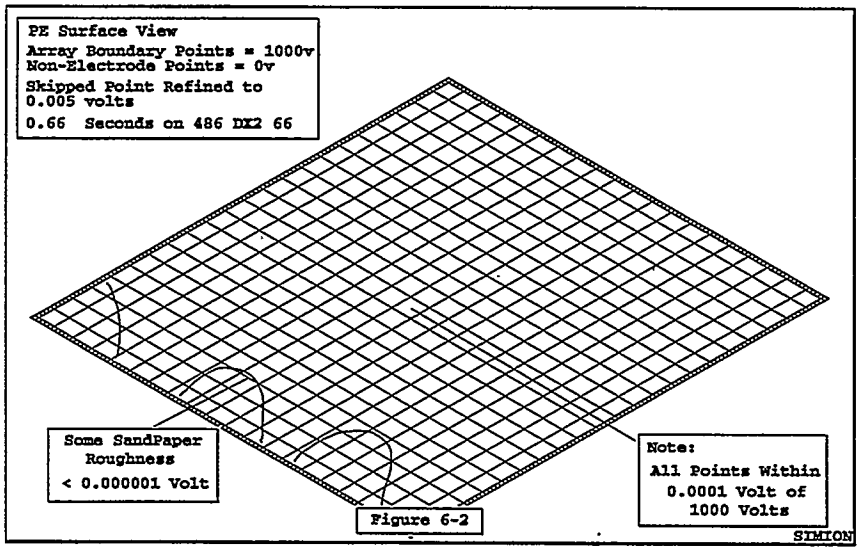


Figure 6-2 shows the results with the default refining parameters (*skipped point refining active*). ***All points are closer than 0.1 millivolts of 1,000 volts. The refine time was only 0.66 sec.*** Thus, in this example, skipped point refining gave a better estimate than non-skipped point refining and was ***about 50 times faster.***

One of the strengths of skipped point refining is that it quickly refines linear gradient field areas in potential arrays. Since Laplace's natural propensity is toward the linear gradient, this is a very

valuable trait. ***Skipped point refining has eliminated the need for presetting linear gradients*** (*a manual trick users employed in earlier SIMION versions*).

The only thing you must keep in mind is the sandpaper effect. Skipped point refining creates local disturbances by estimating the values of new points each time the skip factor is reduced. ***These disturbances need to be refined out too!*** While skipped point refining converges more rapidly to the final solution, it also introduces a certain amount of local surface roughness (*the sandpaper effect*). ***Refining to a lower limit reduces this local surface roughness but will never eliminate it completely.***

Running Refine

Refine is accessed via the Main Menu Screen. To refine a PA (*potential array*), make sure it is the currently selected PA (*its button is depressed*), and then click the **Refine** button or hit the **<R>** key. Note: SIMION only allows two types of potential arrays to be refined: **.PA** (*normal*) and **.PA#** (*Fast Adjust Definition arrays*). ***SIMION refines an array according to its file name extension.*** Thus you must *always* save a Fast Adjust Definition array using a **.PA#** extension or SIMION will consider it to be just a normal **.PA** array.

Refining Normal Potential Arrays (.PA Extension)

The Refine Screen for normal potential arrays (*e.g.* **.PA**) is shown in Figure 6-3. The example below uses the **\SIM6_MISC\TEST3D.PA** file. ***You might want to load it and try refining it yourself for practice.***

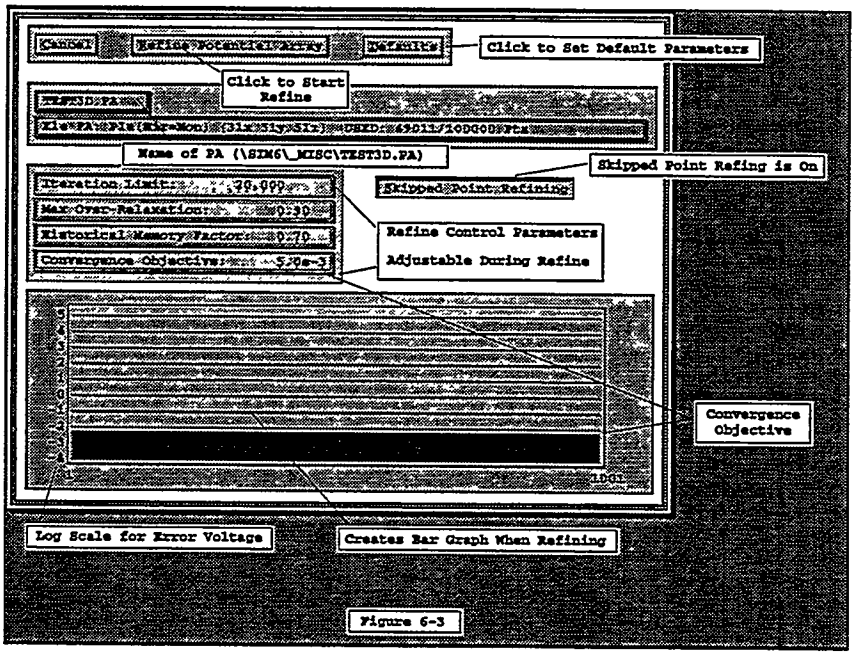


Figure 6-3

Refine Control Parameters

There several refine control parameters you can set before (*and during*) the refining process:

Iteration Limit

The **Iteration Limit** panel controls the maximum number of iterations SIMION will allow before calling a halt to the refining process. This limit exists to prevent SIMION from going on forever trying to refine an array that is approaching its convergence limit very slowly. The default value of 20,000 is more than enough for almost any refine. ***The Iteration Limit panel is accessible while refining either .PA or .PA# arrays.***

Max Over-Relaxation

The **Max Over-Relaxation** panel sets the maximum upper limit to the automatic over-relaxation that SIMION is allowed to apply. This value is normally set to 0.90 (*a good value for most arrays*). ***The Max Over-Relaxation panel is accessible while refining .PA arrays.***

Historical Memory Factor

The **Historical Memory Factor** panel (*default = 0.7*) sets the time constant for an exponential moving averaging algorithm to calculate an average error term (*iteration to iteration*) that is used to control automatic over-relaxation. The higher the factor the smoother the average error term becomes and the slower the changes in over-relaxation factor. ***The Historical Memory Factor panel is accessible while refining .PA arrays.***

The Convergence Objective

The **Convergence Objective** panel (*default = 0.005 volts/Mags*) sets the convergence criteria for terminating the iterative refining process. As discussed above, this parameter does not guarantee any particular level of accuracy. ***The Convergence Objective panel is accessible while refining .PA arrays.***

Skipped Point Refining

The **Skipped Point Refining** button is used to select between skipped point and non-skipped point refining. Skipped point refining is selected when the button ***is depressed*** (*default is skipped point refining*). ***The Skipped Point Refining button is NOT accessible while refining any arrays.***

Selecting Defaults

The **Default** button is used to select the default values for all of the above parameters.

Viewing the Refining Process

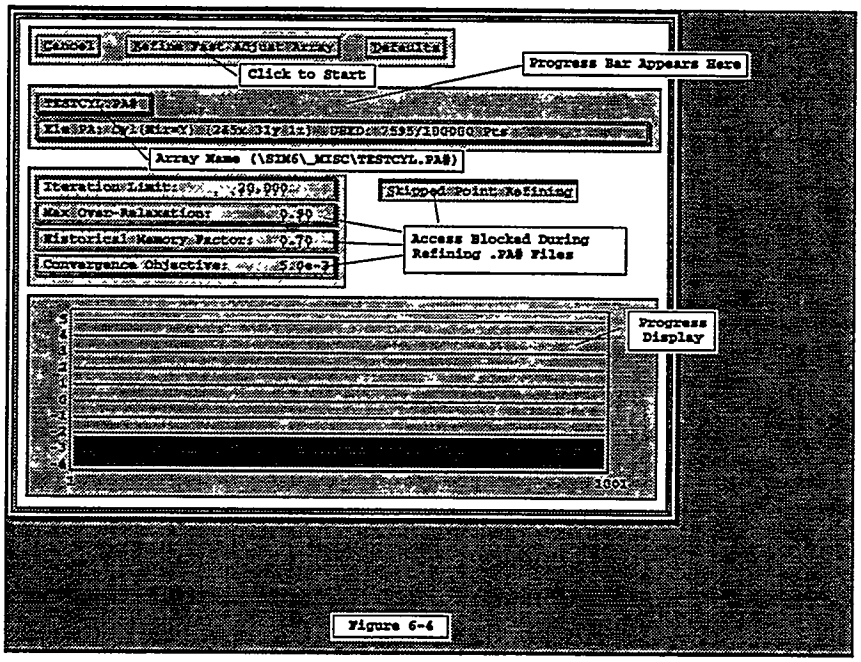
When you click the **Refine Potential Array** button, SIMION first performs ***one*** refine iteration with point skipping turned off to see if the array is already refined to the convergence objective (*and stops if it has been*).

The progress of the refining process is displayed in the semi-log display at the bottom of the Refine Screen. Every ten iterations SIMION draws a bar showing the error value at the end of the most current iteration. As the refining process continues the error bars will indicate progress toward the convergence objective (*green region at the bottom*).

If skipped point refining is active a refine will appear to be a series of sawtooths representing successive refines at lower skip factors. When the skip number displayed is zero, you are on your final refine (*no points are being skipped*). ***Note: The first two skips are refined to values much lower than the convergence objective.*** This helps resolve areas with linear gradients better, so that the refining process will be faster and more accurate.

What is Actually Refined

SIMION only refines the *in-memory* copy of the potential array when it refines a .PA array. This means that you must *save* the refined array if you want this solution to retained between SIMION sessions.



Refining Fast Adjust Definition Potential Arrays (.PA# Extension)

The Refine Screen for fast adjust definition potential arrays (*e.g.* .PA#) is shown in Figure 6-4. The example above uses the \SIM6_MISC\TESTCYL.PA# file. ***You might want to load it and try refining it yourself for practice.***

Refine Control Parameters

There are several refine control parameters that you can set before the refining process (*as described above in* .PA *refining*). ***The only parameter that you can adjust during the refining of*** .PA# ***files is the*** **Iteration Limit** ***panel.***

The Process Involved in Refining a .PA# File

The first and most important thing to understand is that the .PA# ***file is not changed in any way by the refining process.*** It merely acts as the **template** for creating and refining all the required electrode/pole solution arrays required to support the fast adjust process. The following is a step-by-step description of what all the activity on your screen is about:

Save a Copy of the .PA# File

You should make sure the a current copy of the .PA# file has been saved in the current directory *before* starting to refine it. ***The first thing SIMION will try to do is load the disk copy and check it.***

Loading and Checking the .PA# File

SIMION will now load the **.PA#** file from disk and check for adjustable electrode/pole definitions. Remember: Electrode number one must be ***exactly*** one volt (*electrode two = 2 volts and so on*). If you don't define any adjustable electrodes/poles or have skipped some (*e.g. 1,2,3,6,7,8*) SIMION will inform you of your error and abort the refine. At this point SIMION starts its array creation and refining process.

Creating, Refining, and Saving the .PA0 file

The **.PA0** file is the Fast Adjust file. It contains the solution for all non-adjustable electrodes and poles. SIMION loads a copy of the **.PA#** file and coverts it into the **.PA0** file by setting all adjustable electrode/pole points to zero. All non-electrode/non-pole points are also set to zero to insure consistent solution patterns.

The **.PA0** array is then refined using the parameters set before refining began. After refining is completed the file is saved as a **.PA0** in the current project directory. SIMION appends a trailer to this file that contains all the information that **Fast Adjust** will need when fast adjusting the potential array later.

Creating, Refining, and Saving the .PA1-Z files

The **.PA1** file is the specific solution file for adjustable electrode/pole number one. Files **.PA2-Z** contain the specific solution for each adjustable electrode/pole.

SIMION loads a copy of the **.PA#** file and coverts it into the **.PA1** file by setting all electrode/pole number one points to 10,000 volts/Mags and setting ***all*** other electrode/pole points to zero. All non-electrode/non-pole points are also set to zero to insure consistent solution patterns. The **.PA1** array is then refined using the parameters set before refining began. After refining is completed the file is saved as a **.PA1** in the current project directory.

The process above is repeated for each adjustable electrode/pole defined. The displayed file name is changed and a progress bar is displayed to update you on SIMION's progress.

Reloading the .PA# File

When all this activity is completed SIMION will reload the original **.PA#** file and exit to the Main Menu Screen.

Saving the Results

There is no need to save the results you refine a **.PA#** file because SIMION does the saving *automatically* for you.

Fast Adjusting .PA and .PA0 Files

SIMION has specific methods for fast adjusting the solutions of **.PA** and **.PA0** (*or* **.PB0** *and etc.*) arrays. Each of the two types of arrays are fast adjusted differently. ***A .PA file must have been refined for Fast Adjust to serve any purpose.***

The fast adjust process only changes the in-memory copy of the potential array (*e.g.* **.PA** *or* **.PA0**). Be sure to *save* the changed in-memory arrays to disk if you want these changes to persist between SIMION sessions. Note: **.IOB** files save and can restore these potentials too (*see Chapter 7*).

How to Access Fast Adjust

the **Fast Adjust** function can be accessed either from the Main Menu Screen or from within the View Function.

Access From Main Menu Screen

To fast adjust a file from the Main Menu Screen, be sure that the file is selected (*its button is depressed*). Now click the **Fast Adjust** button or hit the **F** key to enter the **Fast Adjust** function. ***If you have selected a .PA# array SIMION will automatically load the refined .PA0 array in its place and fast adjust it (if it exists).***

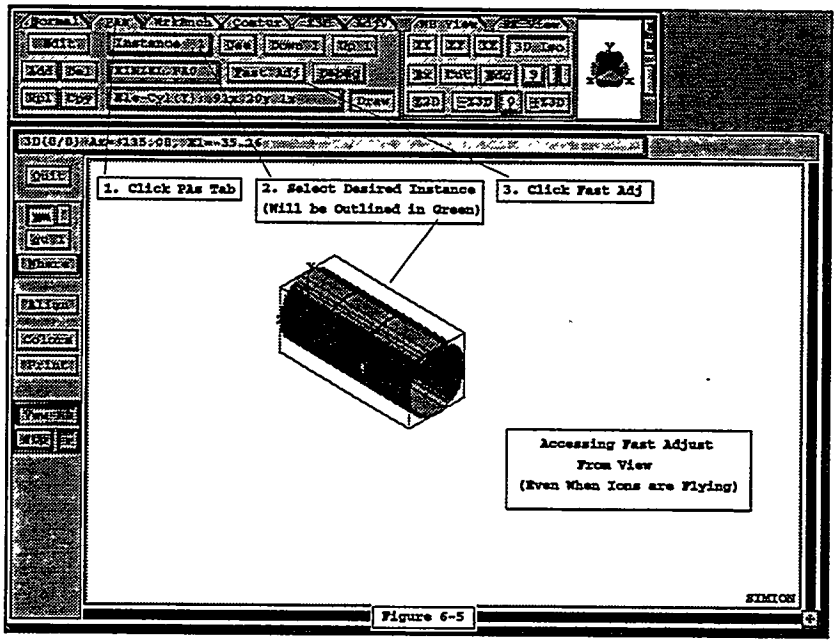


Figure 6-5

Access From the View Function

To fast adjust within **View** (*Figure 6-5 above*) ***even with ions flying***: Click the **PAs** tab, select the desired array with the **Instance** panel (*outlined in green*), and click the **Fast Adj** button. ***Note: Any changes also impact all other instances tied to same potential array.***

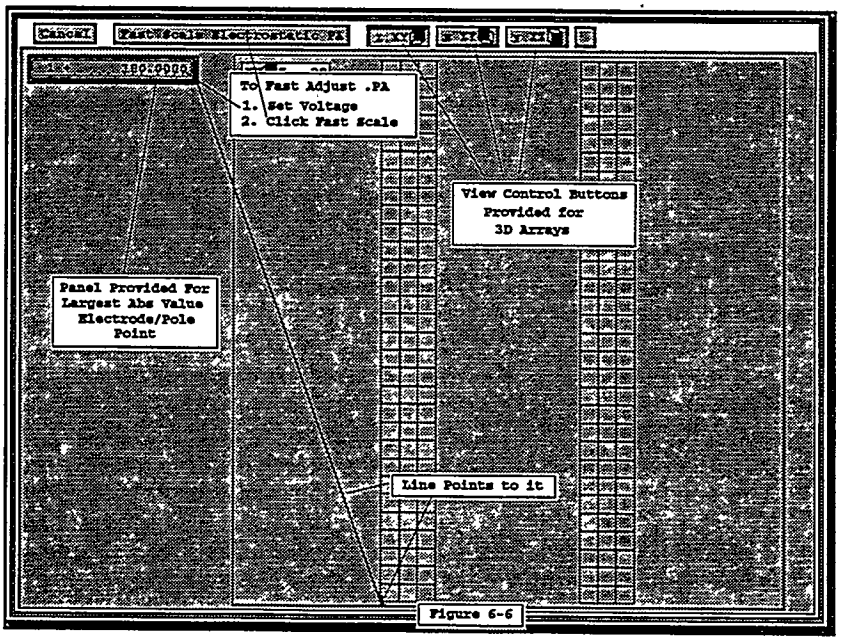
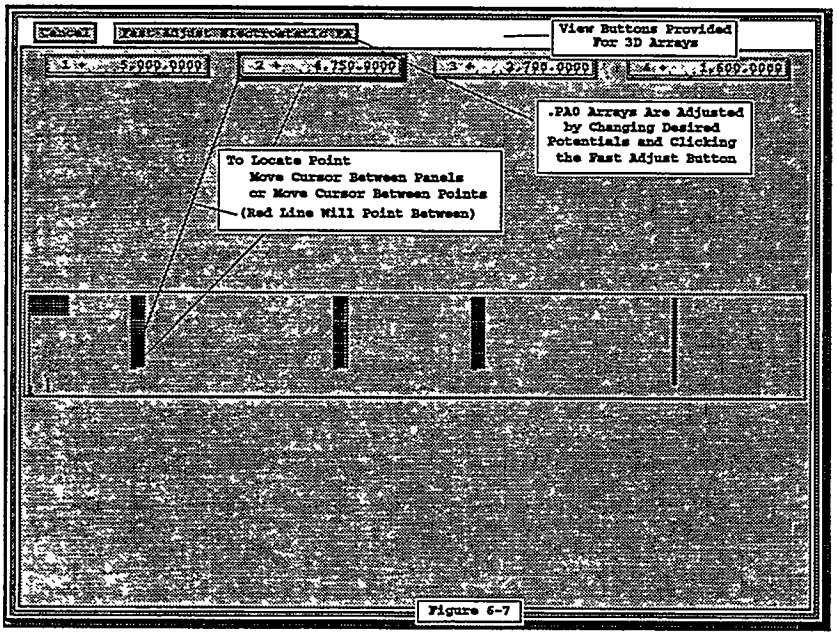


Figure 6-6

How .PA Arrays are Fast Adjusted

SIMION fast adjusts **.PA** arrays by proportional scaling. Each point in the array is multiplied by a scaling factor. When you enter **Fast Adjust** with a **.PA** array SIMION scans the array to find the electrode/pole point with the largest absolute potential.

Fast Adjust then shows a picture of the array and creates an adjustable panel with the point's current potential (*Figure 6-6*). A red line connects the panel and the point on the view so you can see which point serves as the reference. Three view control buttons are provided for 3D arrays (*to assist in seeing point's location*). ***To fast adjust a* .PA, *change the panel to the desired new value (zero is illegal), and click the* Fast Scale *button.***



How .PA0 Arrays are Fast Adjusted

SIMION fast adjusts **.PA0** (*base solution arrays*) by using data from the specific adjustable solution arrays to provide scaling for potential changes of specific adjustable electrodes in the **.PA0** (*or* **.PB0** *and etc. arrays*). ***See Appendix E for specific details.***

What you must do is change the desired adjustable electrode/pole potentials (*via their panel objects*) and click the **Fast Adjust PA** button (*Figure 6-7*). When you point the cursor to a panel it will draw a red line to one of its points. When you point the cursor to point marked with its electrode number, a red line will point to its panel. Three view buttons are provide for any 3D array to aid in electrode viewing.

Remember, you can share the specific solutions arrays among more than one fast adjust base array (*saves disk space and refine time*). This allows you to have multiple instances of the same fast adjust array with independently adjustable potentials via additional **.P?0** base arrays (*where* **?** *is* **B-Z**). To create an additional fast adjust base array (*many can be created*), load the array's **.PA0** file and save it as a **.PB0** file. The **.PB0** file automatically uses the same fast adjust files as the **.PA0** file. In this case, one instance would be linked to the **.PA0** array and another would be linked to the **.PB0** of the same name (*e.g.* **TEST.PA0** *and* **TEST.PB0**). ***The quick way to switch array connections to an instance in* View *is to use the* Rpl *Button in the* PAs Control Screen *to replace the currently selected instance's array with another one.***

Positioning and Viewing Arrays in the Workbench

Introduction

This chapter covers how to position and view potential arrays in SIMION's **View** function's ion optics workbench. It is assumed that you have read the discussions of potential arrays in Chapter 2, 4, 5, and 6. If not, read the material before proceeding further.

Note: Chapter 8 covers how to fly ions within SIMION's ion optics workbench.

> The first seven pages of this chapter present important conceptual information. Although the information density is quite high, you must eventually understand this material to use SIMION at its full power. ***Hang in there! You skip this information at your peril!***

The Ion Optics Workbench

SIMION 6.0 incorporates the concept of an ***ion optics workbench***. The workbench is an imaginary volume in space that you can size and project the ***images*** of potential arrays into. The workbench concept provides the framework to simulate a wide range of problems (*e.g. simple lens systems up to entire instruments*).

The Ion Optics Workbench .IOB Files

Workbench definitions are saved as **Ion Optics Bench - .IOB** files (*e.g.* **EINZEL.IOB** *from the first demo flight - Appendix C*). An **.IOB** file contains definitions of the workbench volume, what arrays are required, their potentials, and where they are to be placed within the workbench volume. When you load an **.IOB** file (*from within* **View**) SIMION automatically loads any potential arrays that are required to support it. ***Saving the .IOB file is important because it conserves workbench definitions and/or changes between SIMION sessions.***

The Workbench Volume

The workbench volume is an imaginary volume of up to plus or minus one kilometer (*8 cubic km maximum*) that you can size to fit the problem at hand. Millimeters (*mm*) are units of length within the workbench.

The Default Workbench

If you enter **View** with a non-empty potential array selected, SIMION will first check to see if this array is referenced in its current workbench definition (**.IOB** *image*). ***If not, it will automatically create the following default workbench for the potential array:***

- A single ***3D image*** (*instance*) of the potential array will be created that projects into the workbench volume.
- The array's physical origin will be positioned at the workbench origin.
- The scale factor will be ***1.0 mm/gu*** (*1.0 millimeter / grid unit*).

- The array's image will be ***aligned*** with the workbench axis (**Az** = **El** = **Rt** = *0.0*).
- The workbench volume will be sized to just contain the array ***image*** (*instance*).

The default workbench simulates what users of previous versions of SIMION have seen in the **View** function. However, it is important to realize that SIMION ***never*** enters **View** without having an active workbench definition (*either existing or by default assumption*).

Projecting Images of Potential Arrays Into the Workbench

SIMION makes use of the concept of ***instances*** to project potential arrays into the workbench volume. An ***instance*** is really a reference that says: ***Project a 3D image of a particular potential array at a designated location in the workbench using a specified scaling and orientation.***

Potential Arrays are Projected as 3D Objects

The projected potential array images are always 3D. They represent the ***volume image*** of the potential array taking into account its ***symmetry*** (*planar or cylindrical*) and ***mirroring.***

SIMION visualizes an array by drawing x, y, and z lines (*in array coordinates*) that connect all electrode/pole points of the ***same*** potential. ***Projected array images appear to be constructed of small wire-frame cubes.*** This particular rendering method makes arrays easy to visualize and allows SIMION to take advantage of powerful tricks to draw hidden line 3D images quickly.

How 2D Cylindrical Arrays Project

A 2D cylindrical array will appear as a ***cylindrical 3D volume*** when projected into the workbench via an instance. If x mirroring is active, the 3D image will appear as two mirror-image cylinders joined at the x = 0 plane.

Note: While cylindrical arrays are surfaces of revolution, ***SIMION always visualizes them as if they were cylinders formed inside 3D planar arrays*** (*with wire-frame cubes*). This is ***only*** for display convenience. ***Although cylindrical apertures may not visualize as smooth circles, SIMION always treats them as smooth circles*** (*for fields and splats*) ***when flying ions through them.***

How 2D Planar Arrays Project

A 2D planar array will appear as a ***rectangular 3D solid.*** Both x and/or y mirroring are allowed. Thus the 3D image may be mirrored across the x = 0 and/or y = 0 planes depending on the mirroring set for the potential array.

The 2D planar assumption implies an infinite extent in z. However, it is impractical as well as inappropriate to project images of 2D planar arrays in this manner. ***For visualization purposes, SIMION assumes z mirroring is active and that the array has a minimum z dimension of the y dimension of the array.*** You have the option of ***increasing*** this z dimension (*by editing the instance definition*). This is useful to increase the volume extent (*z depth*) of a 2D planar array (*e.g. for use in simulating the interior rod portions of a quadruple - see* **_QUAD** *demo*).

How 3D Planar Arrays Project

A 3D planar array will appear as a ***rectangular 3D solid.*** Since x, y, and/or z mirroring are allowed, the 3D image may be mirrored across the x = 0, y = 0, and/or z = 0 planes depending on the mirroring set for the potential array. ***Thus an actual 3D planar potential array might define an eighth of a spherical shell*** (*centered at its origin*)***; but if it were mirrored in x, y, and z; the visualized image would be a complete spherical shell.***

The Concept of Instances

An ***instance***, in the SIMION context, acts to project a ***3D image*** of a potential array into a ***particular location, scale, and orientation*** within the workbench volume. Each instance projects ***one and only one*** potential array into ***one and only one*** location within the workbench.

The Workbench can have Multiple Instances

SIMION allows up to ***200*** instances to project images of potential arrays into a workbench volume. ***Thus up to 200 different potential arrays can be projected into a single workbench simulation.***

Two or More Instances Can Reference the Same Potential Array

Two or more instances are allowed to project images of the same potential array. This allows a single potential array to be used in several locations within the workbench (*e.g. one 90 degree spherical ESA instanced three times to simulate three identical spherical ESAs within the workbench volume*). ***This can save a lot of RAM because only one potential array in RAM is required to simulate several projected images.***

However, there are a few things you must realize:

First, the *potentials* of ***all*** instances that project the ***same*** potential array will be *identical*.

> **Note:** If you want to use a fast adjust potential array (**.PA0**) in two locations (*instances*), but want the adjustable electrodes/poles to have potentials that *differ* between the instances, you must make use of SIMION's multiple fast adjust base array capability (e.g. **.P?0** where **? = B-Z**).
>
> In this case you would return to the Main Menu Screen, save the current **.PA0** array as **.PB0**, reload the **.PA0**, return to **View**, Click the **PAs** Tab, select the instance for the **.PB0** array, click the **Rpl** button and select the **.PB0** file. Now the two instances are tied to *different* fast adjust arrays (*their potentials can now be different*). ***Be sure to save the workbench's .IOB file to remember the changes between sessions.***

Second, instance ***scale factors impact*** electrostatic and magnetic instances ***differently***.

- In the case of ***electrostatic arrays***, the electrostatic field gradients (*and thus ion accelerations*) are controlled by the instance scale factor. The ***smaller*** the projected image of the potential array, the ***stronger*** its field gradients (*assuming the same electrode potentials*). ***This is exactly what you would expect in the real world.***
- ***Instances of magnetic potential arrays are not impacted by scale factor.*** This is because the **Mag** (*the unit of magnetic potential used by SIMION*) is in units of **Gauss * Grid Units**. Thus its gradient (*Gauss*) is independent of instance scaling (*handy*). ***This was done because we normally think in terms of magnetic fields and not magnetic potentials.*** Thus SIMION's **Mag** conserves magnetic field strength (*makes magnetic fields independent of scaling factor*).

How An Instance Projects a Potential Array

Each active instance in the workbench is tied to a ***single*** potential array (*its RAM image*). The ***instance*** projects a 3D image of this potential array into the workbench volume by: ***Positioning, scaling, and orienting it.***

How Instance Images are Positioned

Instances are positioned based on ***three*** points of reference: The ***physical origin***, ***working origin***, and the *currently aligned* ***workbench origin.***

- The ***physical origin*** is the array's ***actual*** origin. For ***all*** arrays this is the array's 0,0,0 point in array grid units (*as you would see in the* **Modify** *function*).
- The ***working origin*** is the position within or near the 3D image of the potential array that you want to consider the origin for ***functional*** purposes (*e.g. for flying ions, array positioning/orientation, and etc.*). The ***working origin*** is ***always*** defined in terms of an ***offset*** in array grid units (*gu*) from the array's ***physical origin***.

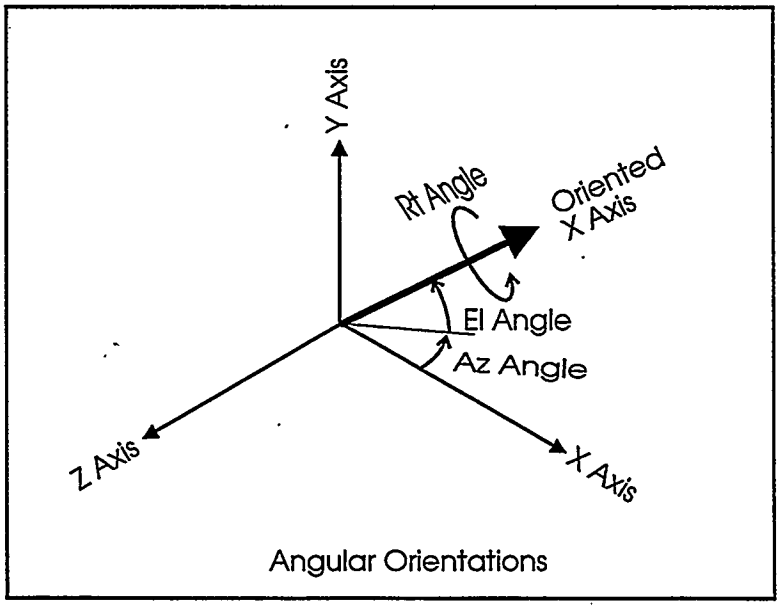
  The default value for this offset is 0,0,0 (*or no offset*). ***This locates the working origin of the array at the physical origin of the array***. There are times when shifting the array's ***working origin*** away from its ***physical origin*** is helpful: To help position a 3D planar array that has a beam line that doesn't go through its physical origin (*common occurrence*), or to move the ***working origin*** to the ion emission point within the potential array (*makes locating ion starting points easy*).
- The ***currently aligned workbench origin*** is the **0,0,0** point in workbench space (*mm*). ***Currently aligned*** means that the position of the ***workbench origin*** is either ***unaligned*** (*normal*) or ***aligned*** (*to the working origin of a selected instance*). The selective alignment of ***workbench coordinates*** is discussed later.

When an array is projected into the workbench, the instance combines the effects of ***two*** user defined ***offsets*** to position it:

1. The first, defines the ***offset*** of the array's ***working origin*** from ***currently aligned workbench origin*** in ***mm***. This offset is the principal way instances are positioned in the workbench volume.
2. The second, defines the ***offset*** of the ***working origin*** of the potential array from its ***physical origin*** in ***grid units***. This offset is defaulted to **0,0,0**: The ***working origin*** and the ***physical origin*** are the same.

How Instance Images are Scaled

Instance images of the potential array are scaled into workbench units (*mm*) via a user defined scale factor that has units of ***millimeters/grid unit*** (*mm/gu*). The ***default*** value for the scale factor is ***1.0 mm/gu*** (*one grid unit equals one mm of workbench distance*).



How Instance Images are Oriented

Instances are oriented about their ***working origins*** by using three orientation angles: ***Az***, ***El***, and ***Rt*** (*in degrees*). These angles assume that the instance's x, y, and z axis are

initially parallel to the *currently aligned workbench coordinate x, y, and z axis* respectively (*e.g.* **Az = El = Rt** = *0.0*). The angular orientations are performed in the following order:

1. The ***Rt*** angle defines the *ccw* rotation of the ***y-axis*** in the ***zy plane*** looking down the positive ***x-axis*** toward the array's ***working origin***.
2. The ***El*** angle defines the *ccw* rotation of the ***x-axis*** in the ***xy plane*** looking down the positive ***z-axis*** toward the array's ***working origin***.
3. The ***Az*** angle defines the *ccw* rotation of the ***x-axis*** in the ***xz plane*** looking down the positive ***y-axis*** toward the array's ***working origin***.

Integrally and Non-Integrally Aligned Instances

An instance can be either integrally or non-integrally aligned with the workbench. An integrally aligned instance has ***each*** axis (*x, y, and z*) ***parallel*** to a workbench axis (*e.g. 0,0,0 or 90,0,-90 for* **Az, El,** *and* **Rt**).

Integrally aligned instances draw faster (*and better*), because SIMION's rendering scheme is highly optimized for them. Features like potential energy surfaces require integral alignment. ***Thus you should strive to have as many of your instances integrally aligned as possible.***

SIMION provides an alternate method to define integrally aligned orientations based on which workbench axis the x and y are parallel to. If **Ax, El,** and **Rt = 0,0,0** the orientations would be **x = +x** and **y = +y** (*x points in +x axis direction and y points in + y axis direction*). A second example, **90,0,-90** would give **x = -z** and **y = -x** (*x points in -z axis direction and y points in -x axis direction*). Selector objects are provided to allow you to select integral alignments in lieu of non-integral alignments.

Note: ***Workbench coordinates*** can be ***temporarily*** aligned with any selected instance. This can be used to integrally align a non-integrally aligned instance for easier viewing or looking at its PE surface (*PE surfaces of instances require integral alignment*). Aligning ***workbench coordinates*** with any instance is discussed below.

Instances Can Overlap

Since we are dealing in virtual reality, PA instances can overlap (*share common volumes*). This is allowed to enable both electrostatic and magnetic effects to be applied to the same volume. ***Please note: You must use common sense here. It is possible to carelessly model the physically impossible.*** If you have a combined electrostatic and magnetic problem, be sure to include the magnetic poles as electrostatic elements (*most likely at ground potential*) in your electrostatic array. ***Just because you overlap electrostatic and magnetic instances don't assume SIMION will automatically take into account the electrostatic effects implied by magnetic pole pieces*** (*it won't*).

Another use of overlapped instances is the positioning of higher grid density (*more accurate*) instances within lower grid density instances to improve accuracy in specific regions. This unfortunately introduces the problem of which instance to use in volumes where two or more of the same type (*e.g. electrostatic*) overlap. ***The instance selection problem is solved by maintaining instances in a list*** (*below*).

Instances Are Maintained in a List

SIMION maintains a numbered list of all currently defined instances. From the list's prospective you can think of each instance as a ***layer***. ***The instance with the lowest number*** (*1*) ***is always on the bottom of the pile.*** The instance with the higher numbers are stacked upon the pile in numerical order. ***The instance with the highest number is always on the top of the pile.***

The Significance of List Order

Instance list order determines the electrostatic and magnetic fields that an ion sees and the electrodes it splats into when the ion is flying in a workbench region where instance volumes overlap. ***Instances with a higher instance number always have a higher use priority.***

Whenever SIMION is flying ions, it ***always*** starts at the ***top*** of the instance list (*highest instance number*) to search for instance(*s*) the ion is flying within. This search ***down*** the instance list (*top to bottom*) ***terminates*** after the ***first*** electrostatic ***and first*** magnetic instance containing the ion have been found (*first one of each type*). **Note:** To define a volume that has ***both*** electrostatic and magnetic fields, an electrostatic ***and*** a magnetic instance must both overlap in the volume's region.

When ions behave strangely and don't seem to be flying through the proper instance. ***Check the instance order.*** There is a good chance that the ions ***may not*** be flying in the instance you think they are.

Changing Instance Order

The PAs Screen (*accessed via the* **PAs** *tab within* **View**) can be used to change an instance's location in the list. Use the ***Instance Select*** panel to select the instance (*its image will be outlined in green*). Now click either the **Up 1** or the **Down 1** button to raise (**Up 1**) or lower (**Down 1**) the instance's priority in the list.

How Multiple Instances Interact

In two words - ***They Don't.*** Each instance is an island. ***It has no knowledge of the existence of any other instance.*** Thus an instance's fields are ***never changed*** by the proximity of another instance.

Transitions Between Instances

It is your responsibility to assure (*through appropriate boundary assumptions*) ***that there is appropriate field continuity in transitions between instances*** (*especially any that overlap*).

Interpolating Electrostatic Fields Between Instances

When an ion is flying between ***electrostatic*** instances (***outside of all electrostatic instances***), SIMION checks forward and back along the ion's ***ballistic trajectory*** for intersections with an ***electrostatic*** source and destination instance. If intersections with ***electrostatic*** source ***and*** destination instances are found (*top to bottom instance list search*), the electrostatic potentials of the source and destination intersection points are determined and the potential gradient is calculated. ***This gradient can only accelerate*** (*or decelerate*) ***the ion along its ballistic trajectory*** (*no refraction - path bending*). ***Thus the ion will never see the effects of an electrostatic instance it is not on a collision path with.***

Copying Electrodes Into Base Instances

Even though instances don't interact, you can copy electrode points between them so that the refining process estimates the fields more appropriately in 3D base instances. The **Cpy** button on the PAs Control Screen can be used to accomplish this (*discussed later in this chapter*).

Aligning the Workbench Coordinates

SIMION treats the ***workbench coordinate*** system as the frame of reference in the workbench. However, there are times when it would be very convenient to be able to ***align*** the workbench coordinates, ***at least temporarily***, with the ***working origin*** (*and angular alignment*) of a particular instance. ***This can make viewing, measuring, instance adjustment/insertion much more direct.***

How to Align Workbench Coordinates with an Instance

This feature is supported via the **Align** button on the left edge of **View**. To temporarily *align workbench coordinates* with a particular instance, select the instance (*depress the* **PAs** *tab and use the* **Instance Number** *panel to select the instance number*), and then depress the **Align** Button.

When you *align workbench coordinates* with an instance, the *workbench's origin* will be positioned at the location of the instance's *working origin* and the *workbench's x, y, and z axis* will be *aligned* with the *instance's x, y, and z axis* (**Az** = **El** = **Rt** = *0.0*). *All* other active instances will have their positioning and orientation parameters automatically adjusted to reference the new *aligned workbench coordinates*.

How to Restore Normal Workbench Alignment

To return to normal *workbench coordinate* alignment, click the **Align** button again (*to raise it*). *All* active instances will have their positioning and orientation parameters restored to reference the *unaligned workbench coordinates*.

How to Make a Workbench Coordinate Alignment Change Permanent

To make the current alignment of the *workbench coordinates* permanent, simply save its **.IOB** file (*using the* **Save** *button in the Normal Screen in* **View**). SIMION will ask if you want to make the current workbench alignment permanent.

Workbench Coordinate Alignment Can be Changed at Anytime

You can change *workbench coordinate* alignment at any time (*even when ions are flying*). ***This allows you the maximum freedom to view, measure, and/or obfuscate whenever you please.***

Workbench Coordinate Alignment and SIMION Features

All features within SIMION *automatically* adjust for and use the *currently aligned workbench coordinates* as their frame of reference. These features are said to work with *currently aligned workbench coordinates*.

The *only* feature that *always* uses *unaligned workbench coordinates* for its definition is the *starting locations and directions of ions*. Whenever the ions' starting parameters are defined in *workbench coordinates* (*they are usually defined in a selected instance's coordinates*), these parameters are assumed to be in *unaligned workbench coordinates*. This prevents you from having to change the starting locations and directions of your ions just because the **Align** button is depressed. SIMION will *automatically* translate the ions' starting locations and directions into the *currently aligned workbench coordinates*.

Accessing SIMION's View Function

Now that you have waded through *all* this preliminary material, you are prepared to get into the *View* function and flail about. Use one of the following recommended approaches for entering *View* from the Main Menu Screen:

Entering View With a Potential Array

Select the desired potential array (*make sure its button in depressed*) and then click the **View** button or hit the **V** key. If the selected potential array is instanced in the current workbench, SIMION will re-enter that workbench. Otherwise, SIMION will create a default workbench definition for the file (*as discussed above*) and enter **View**.

Entering View With an .IOB File

It is suggested that you click the **Remove All PAs From RAM** button before loading a new **.IOB** file (*to remove RAM clutter*). Enter **View** without selecting a potential array (**Empty PA** *button is depressed*). SIMION's GUI File Manager will assist you in selecting an **.IOB** definition file (*e.g.* **EINZEL.IOB**). You should switch to the desired project directory (*click on it - if not currently selected*) and click ***both*** mouse buttons on the selected **.IOB** file name. SIMION will load the **.IOB** file and its referenced potential arrays. It will also ask you if you want ***all*** array potentials restored to what they were when the **.IOB** file was saved (*works for* **.PA** *and* **.PA0** *arrays*).

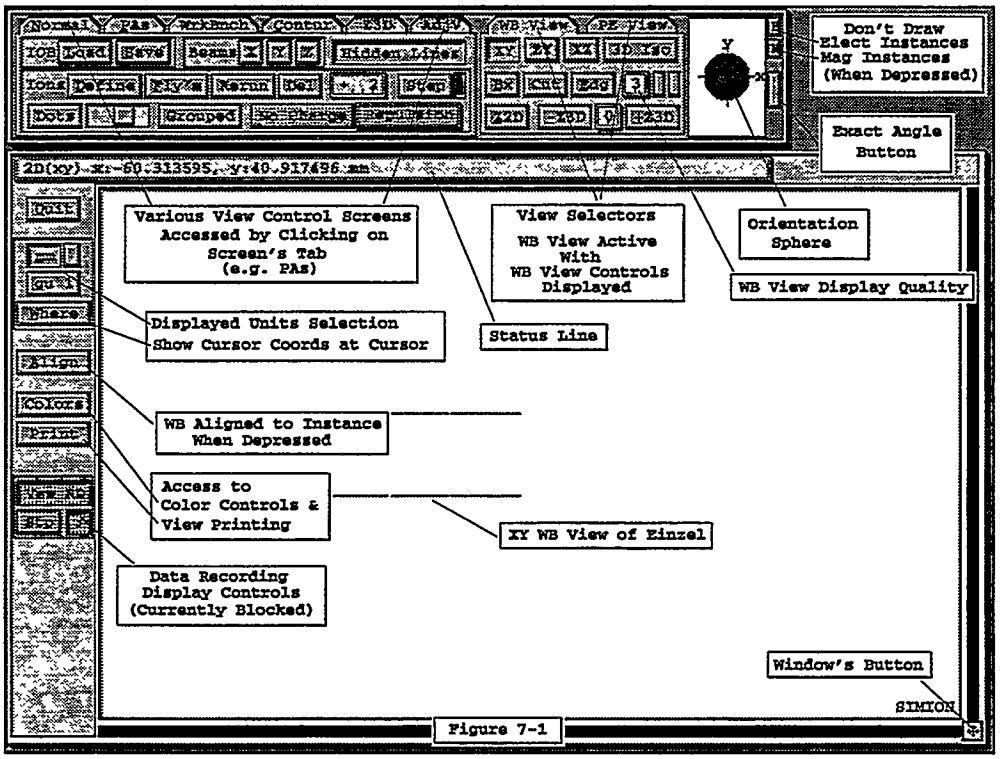


Figure 7-1

Normally, you should click *Yes* (*the default*).

A Tour Around View 's Screen

Figure 7-1 above shows a sample image of **View**'s screen. Controls are located on the left edge and at the top. The currently selected view is displayed in the window. ***Note: SIMION automatically blocks access to screens as well as individual controls whenever their use would be inappropriate.*** In general, most features are accessible even when ions are flying. The objective is to make the **View** function as interactive as possible.

The Controls at the Top of the Screen

There are three groups of control screens at the top of the View Screen: A group of six control screens, a group of two view control screens, and finally an orientation sphere with three view control buttons.

The Six Control Screens Group

The group of six control screens are located in the upper left corner of the display. They are organized as six tabbed folders (*the Normal Screen is currently on top*). The six control screens are:

Normal	Controls for **.IOB** file saving and loading, axis beams, hidden line control, and ion flying controls.
PAs	Controls to create, select, modify, and edit instances.
WrkBnch	Controls for displaying/changing the workbench size.
Contur	Controls for drawing potential and/or gradient contours.
Z3D	Controls for displaying/changing size of current 3D zoom volume. ***Access blocked if not currently 3D zoomed.***
AdjV	Controls for displaying/adjusting user adjustable variables. These are defined in user programs (*Appendix I*). ***Access blocked if not currently flying ions with user programs active that have adjustable variables defined.***

The Two View Control Screens Group

Two view control screens are provided as tabbed folders as above (*the WB View Screen is currently on top*). ***A view type is selected by clicking its tab.***

WB View	Controls for displaying workbench views including: Standard view buttons, box outlining, cutaway clipping, edge only viewing, display quality options, and 2D/3D zooming.
PE View	Controls for displaying potential energy surface views including: Surface polarity, box outlining, grid spacing, display quality options, 2D zooming, and surface relief.

The Orientation Sphere Group

The top rightmost group is composed of an orientation sphere and three display control buttons.

Orientation	The orientation sphere displays/controls the orientation of the displayed view. Normally the sphere automatically swings to the nearest standard view. However, if ***exact angles*** are selected the orientation sphere can adjust the view orientation to the nearest degree.
Exact Angles	The long thin button to the right of the orientation sphere selects exact angles when it is depressed.
E Button	The **E** button turns off the displaying of all electrostatic instances when depressed.
M Button	The **M** button turns of the displaying of all magnetic instances when depressed.

Controls on the Left Screen Edge

A collection of buttons are provided on the left edge of the screen for various functions. These include quitting the **View** function, setting the displayed units, aligning the workbench coordinates, changing colors, printing the current view, and displaying the Data Monitoring Screen (*if active*).

Controlling the View

The **View** function supports both ***workbench*** (**WB**) and ***potential energy*** (**PE**) views. These views are accessed by clicking on their tab (*e.g.* **WB View** *for workbench view*). You can switch back and forth between **WB** and **PE** views even while ions are flying. The following material describes how to control what is shown in each of the two types of view.

Requesting and Halting View Re-drawing

The current view can be redrawn or the current re-draw can be halted at any time. Normally SIMION automatically re-draws the view when appropriate. However, you have the option of forcing the issue.

Requesting a Re-draw of the Current View

To re-draw the current view, move the cursor into the view area or to the orientation sphere and hit the **Enter** key.

To Halt a Re-draw of the Current View

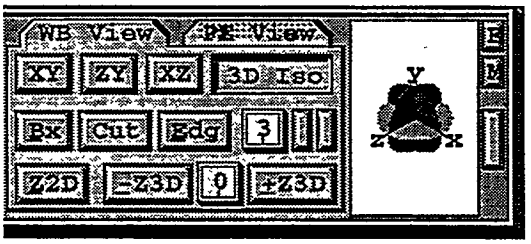
You can also abort a re-draw of the view at anytime by hitting the **Esc** key. Sometimes re-drawing a view can take quite a while (*e.g. display quality set to 9 in an exact angle WB view*). If the drawing is taking too long, hit the **Esc** key and then change the drawing parameters (*e.g. reduce the drawing quality or turn off exact angle drawing*).

Accessing and Controlling Workbench Views

Workbench (**WB**) views are accessed by clicking on the **WB View** Tab to access the WB View Control Screen (*displayed in Figure 7-1 above*).

What is Displayed in the WB View

In **WB View**, you are viewing a 3D volume within the workbench volume. Normally this is the ***entire*** workbench volume. ***However, you have the option of using a 10 level 3D zoom to limit your viewing to smaller volumes within the workbench volume.***



Controlling the Quality of the WB View

The WB View Control Screen has a panel object and two small buttons that can be used to control display quality.

The Display Quality Panel

The WB View Control Screen (*shown above*) has a small panel object (*currently showing the number 3*). This **Display Quality** panel is used to control the quality of the displayed image (*and thus its drawing speed*). The number is adjustable from 0 (*lowest display quality*) to 9 (*highest display quality*). SIMION uses this ***number*** along with the ***projected size*** of each instance to determine how many vectors to draw to represent the instance. Thus when you zoom into an instance, SIMION will automatically draw it in greater detail.

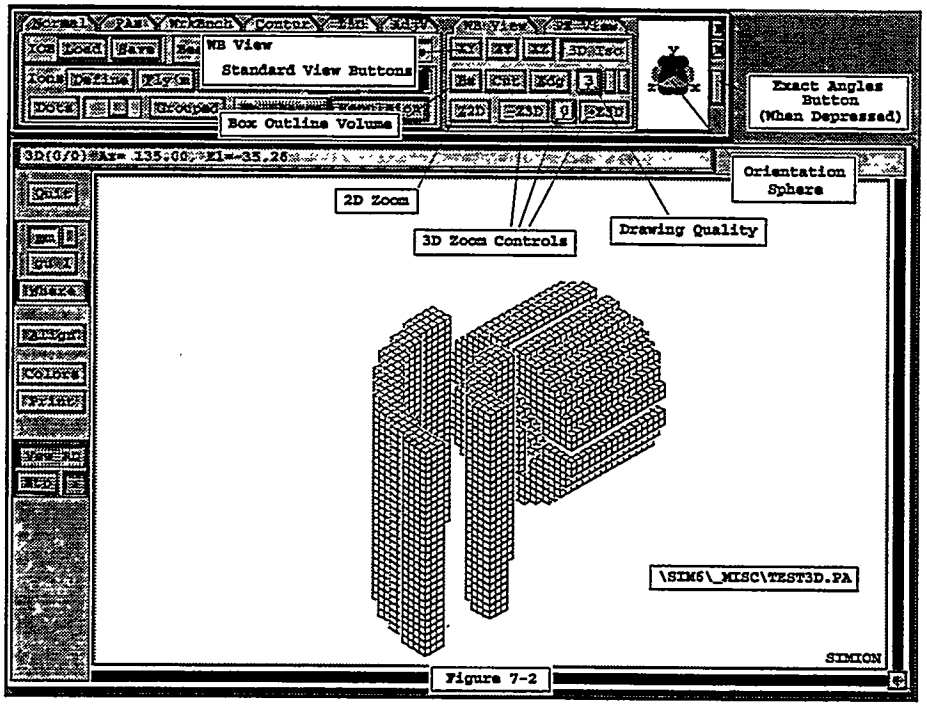
The highest display quality setting (*9*) forces SIMION to draw ***all*** the instances' vectors no matter how big or small the instances' images appear on the screen. ***Warning: A display quality of 9 may result in very slow drawing in exact angle views*** (*discussed below*).

The Two Drawing Control Buttons

Just to the right of the **Display Quality** panel are two small buttons. SIMION visualizes instances via wire grid cubes. Sometimes a 3D isometric image can be made more definitive (*clearer*) when vectors in one direction are not drawn (*cubes require vectors drawn in three directions*). Each of these two buttons blocks drawing of vectors in a particular direction when depressed. ***You may find these buttons useful for seeing certain views more clearly.***

Controlling the Orientation of the WB View

SIMION provides three ways to control **WB View** orientation: Standard view buttons, an orientation sphere, and exact angle viewing. When the view orientation is changed any 2D zooming will be reset to full view. ***Note: All orientations are relative to the currently aligned workbench coordinates.***



Standard View Buttons

The WB View Control Screen has four standard view buttons: **XY, ZY, XZ,** and **3D Iso.** Clicking one of these buttons will access one of three standard 2D and one standard 3D isometric view. The orientation sphere will automatically track these orientations.

Using the Orientation Sphere - Standard Views

The orientation sphere (*to the right of the WB View Control Screen*) can also be used to set the orientation of the view. This is done by moving the cursor to the orientation sphere and ***dragging*** the sphere to the desired orientation with **either** mouse button depressed. The orientation sphere automatically swings to the ***nearest standard view*** when the mouse button is released.

The orientation sphere can access 12 standard 2D views and 8 standard 3D isometric views. ***It is recommended that you try to make used of standard views whenever possible because SIMION's instance rendering system is highly optimized for them.***

Note: Even though you are using a standard view, SIMION will ***only use fast rendering*** for those instances that are ***integrally aligned*** with the ***currently aligned workbench***

coordinates (*discussed above*). Any instance that is ***not integrally aligned*** will be rendered with the ***slower sampling based methods*** used with exact angle views. You can use the **Align** button to ***temporarily align*** the workbench coordinates with any selected non-integrally aligned instance (*speeding its display speed and improving its display quality*).

Exact Angle Views - Using the Orientation Sphere

If you depress the **long** button just to the right of the orientation sphere, SIMION will switch to exact angle views. The orientation object can then be rotated to the nearest one degree of azimuth and elevation angle (*Az and El angles will be displayed within the orientation sphere object*).

*Note: **The drawing speed and quality of the view will erode when exact angles are active** (**in WB Views only - not** **PE Views**).* This is because SIMION must use a ***slower sampling method*** for exact angle instance image rendering. To maintain speed, it automatically reduces the image quality. You can increase the value of the **Display Quality** number to restore quality. However, drawing can become quite slow for the higher numbers (*particularly at 9*). ***Remember, you can <u>always</u> hit the* Esc *key to halt drawing if things have become too slow.***

One trick is to use a display quality of 3 to quickly find the desired orientation, and then increase the drawing quality incrementally (*e.g. 5,6,7 and etc.*) until the rendering is the quality desired. This approach is generally the quickest way to find and display that special exact angle view.

Warning: Exact angle WB Views generate a lot of short vectors. Printing an exact angle **WB View** can swamp your printer's memory. Solution: Get more printer memory or don't print that exact angle **WB View**.

2D Zooming and Scrolling

SIMION provides several ways to 2D zoom and scroll the view. A 2D zoom merely magnifies the current view to help you see details. Scrolling moves the zoomed view about within the full view area.

2D Zooming Using Marked Areas

To 2D zoom (*magnify*), mark the area you want to zoom to (*drag a rectangle with the* **left** *mouse button depressed*) and then click the **right** mouse button (***or click Z2D button or hit the Z key***). To zoom back a level just click the **right** mouse button or **Z2D** button (***without marking***). To zoom back into a previous magnification hold down a **shift** key and click the **right** mouse button (*or click* **Z2D** *button*) ***Up to 10 levels of 2D zoom are remembered.***

To ***delete*** a 2D mark, move the cursor into the view area and hit the **Del** key.

Page View 2D Zooming and Scrolling

To page view 2D zoom, move the cursor to the window's button (*lower right corner of view*). Hold down the **right** mouse button. The screen will be redrawn full view with a cursor that surrounds the current zoomed area. Now hold down a **Ctrl** key and move the mouse (**right** *mouse button still depressed*). Notice that the size of the cursor marked zoom area changes.

To scroll, release the **Ctrl** key and move the mouse (**right** *mouse button still depressed*). Notice that the zoomed area now moves about within the view.

The **Ctrl** key can be depressed or released to instantly switch between zooming and scrolling as desired. Release the **right** mouse button when the desired area is marked and the view will zoom to it.

Current View 2D Zooming and Scrolling

To 2D zoom the current view, move the cursor to the window's button (*lower right corner of view*). Hold down the **left** mouse button. Now hold down a **Ctrl** key and move the mouse (**left** *mouse button still depressed*). Notice that the view can be zoomed in or out.

To scroll, release the **Ctrl** key and move the mouse (**left** *mouse button still depressed*). Notice that the view scrolls.

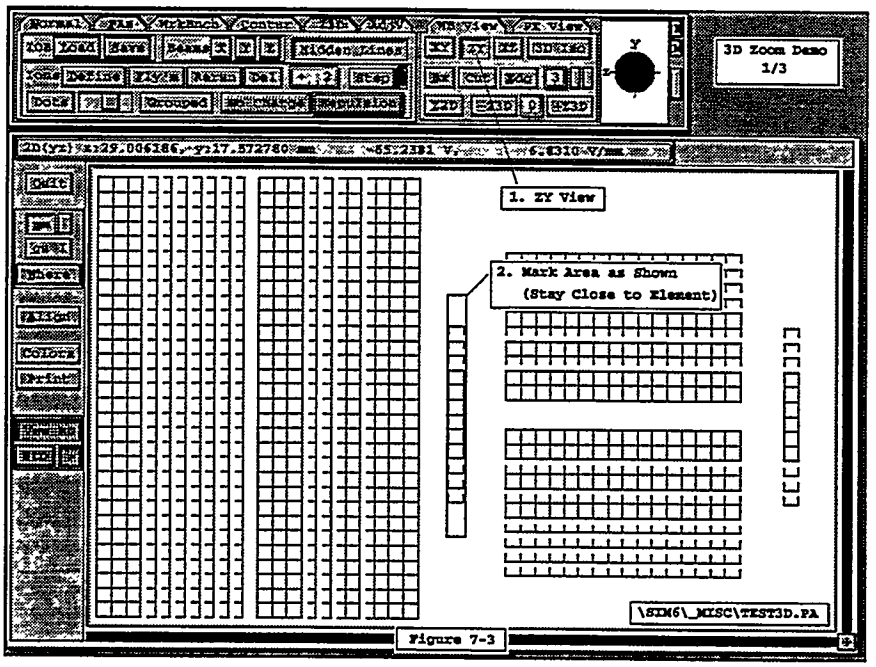
The **Ctrl** key can be depressed or released to instantly switch between zooming and scrolling as desired. Release the **left** mouse button when the desired area is displayed.

Note: You can speed up this process by ***also*** holding down the **right** mouse button. When the **right** mouse button is ***also*** depressed the view is quickly redrawn using box outlines for the instances. To glimpse the actual view, release the **right** mouse button ***momentarily*** and SIMION will re-draw the view.

Zooming the Volume - 3D Zooms

SIMION allows you to define volumes within volumes via a 3D zooming capability (*shown in Figures 7-3 to 7-5 below*). This feature is useful for limiting the size of the viewed volume. You can define a small volume (*e.g. exit aperture*), 3D zoom to it, and watch the ions fly through the volume.

Up to ten levels of ***bi-directional*** 3D zoom are supported. Each successive 3D zoom volume is contained ***within*** the preceding volume. The outermost level (*0*) is always the workbench volume itself. The term ***bi-directional*** means that the zoom volumes are remembered, and you can zoom in and out as appropriate.



The 3D Zoom Controls

The WB View Control Screen contains three objects that control 3D zooming: The **-Z3D** button, **Z3D Level** panel, and the **+Z3D** button.

Z3D Level This panel object displays/selects the current 3D zoom level (*currently displayed volume*). Level zero (*0*) displays the workbench volume.

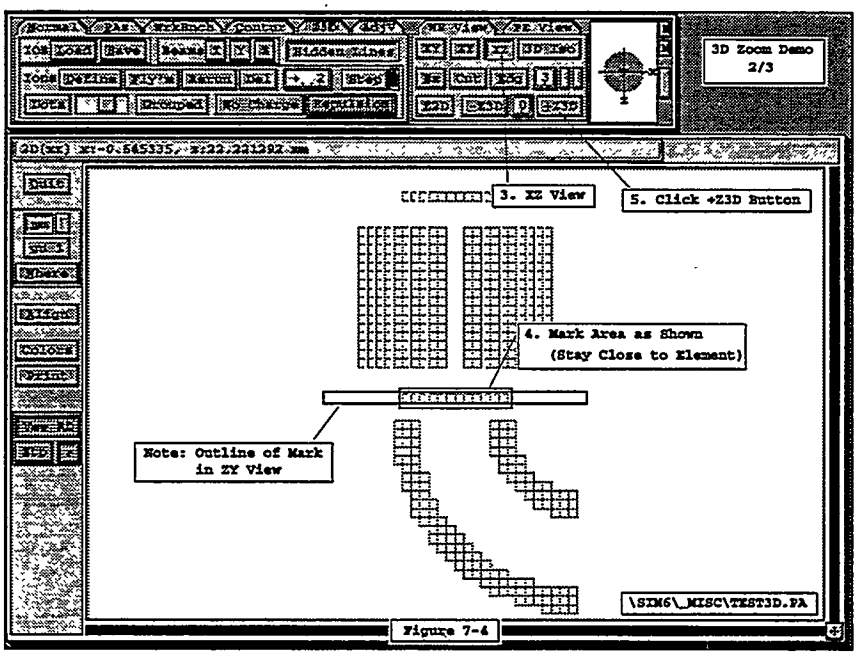
The panel can be used to change between 3D Zoom volumes (*if they are defined*).

-Z3D This button zooms back to the next outer zoom volume (*makes it the currently displayed volume*). Level zero (*0*) displays the outermost volume (*the workbench volume*).

+Z3D If a 3D volume is ***not*** currently marked, clicking this button zooms to the next inner 3D zoom volume (*if defined - else it beeps*).

If a 3D volume is currently marked (*see below*), clicking this button erases all 3D zoom volume definitions below the current level, and then uses the marked 3D volume to create a new 3D zoom volume, adds it to the zoom list, and makes it the current volume.

If the 3D zoom level is already at 9 (*maximum zoom*), the new 3D zoom volume replaces the one that was level 9.



Marking a 3D Volume

View allows you to mark volumes in much the same manner as the **Modify** function (*Figures 7-3 to 7-5*). To mark a volume you must create your marks in ***standard angle*** (***not exact angle***) ***2D views*** in the same manner you would mark for a 2D zoom. When you mark an area in a standard 2D view SIMION assumes the mark extends ***throughout the depth*** of the ***current volume*** (*it is a volume mark*).

Further, if you also mark in one or more of the ***two*** remaining 2D plane directions (*e.g. XY first, then ZY and/or XZ*), SIMION will assume each of these marks is a volume mark too. The actual resulting marked volume is the ***volume of intersection*** common to all of the volumes marked. ***If multiple 2D volume marks have no common intersection volume, SIMION will complain when you try to 3D Zoom.***

SIMION displays outlines of the volume represented by each mark. These marked volumes can be viewed using any standard 3D isometric view.

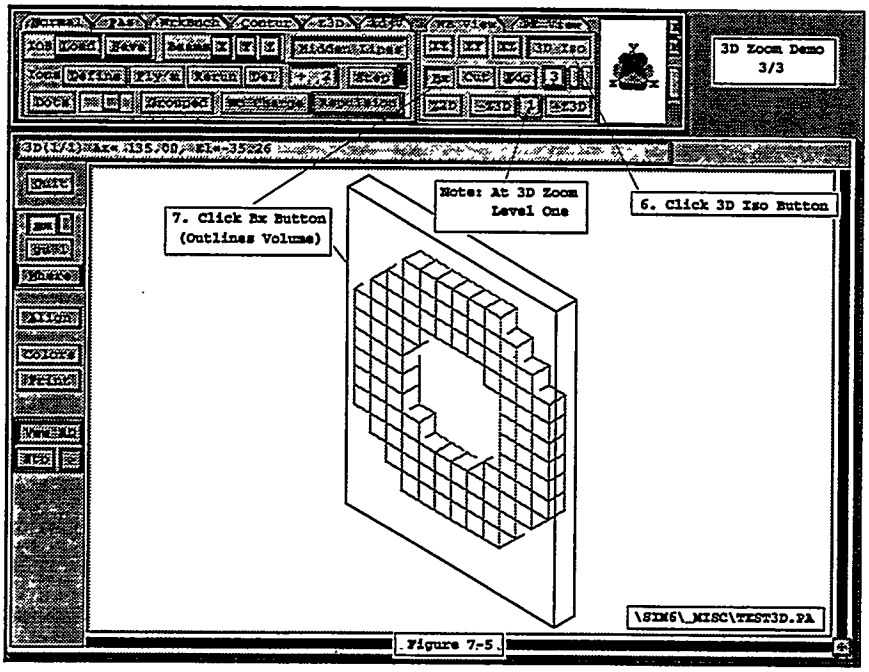


Figure 7-5

Deleting 3D Volume Mark(s)

Volume marks can be ***deleted*** by moving the cursor into the view area and hitting the **Del** key.

Box Outlining

In figure 7-6 below, the **Bx** button was depressed to show the outline of the new 3D volume. Whenever the **Bx** button is depressed SIMION will automatically draw an outline of the current volume (*and the next inner volume if cutaway clipping is active - see below*).

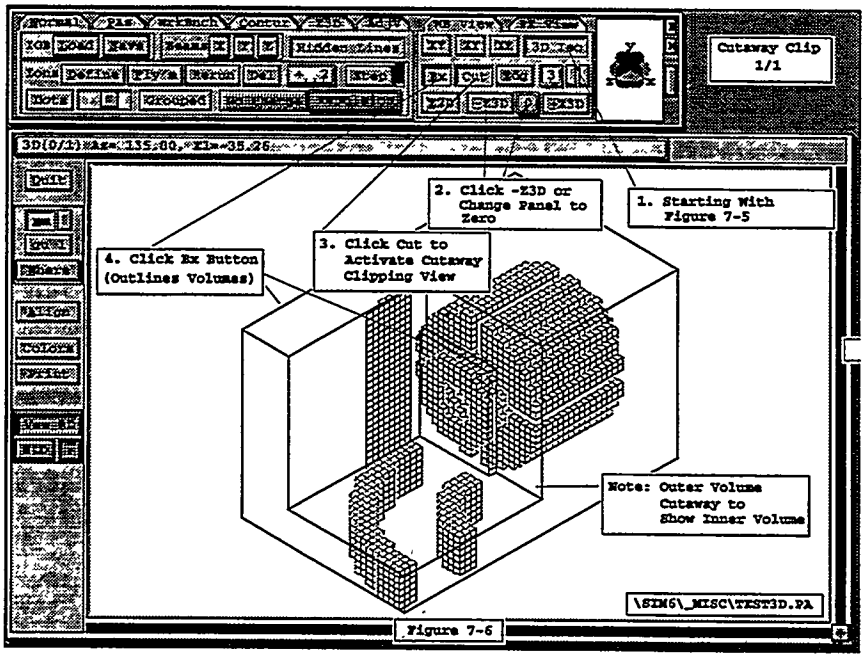


Figure 7-6

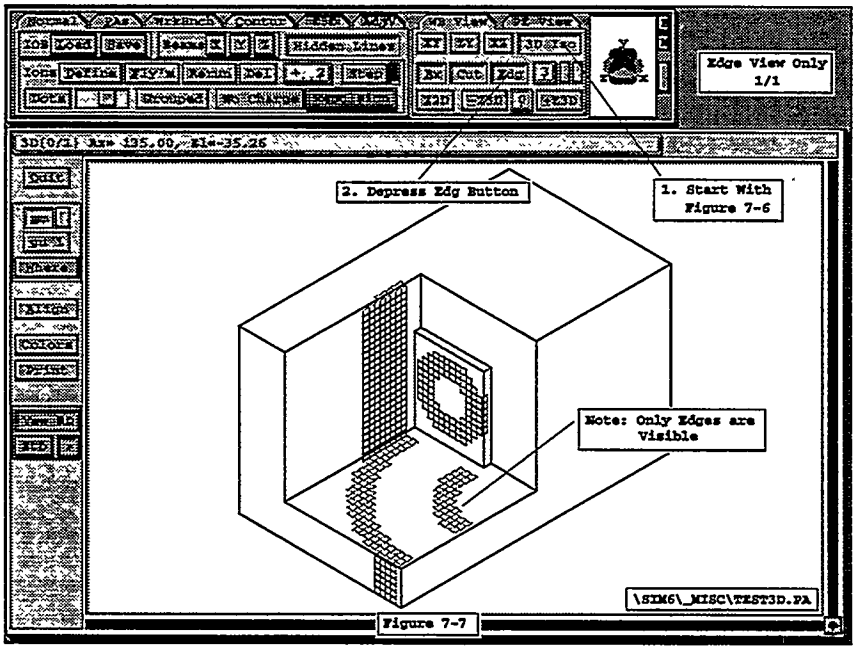
Cutaway Clipping

It is sometimes very helpful to see an inner 3D volume displayed within its parent 3D volume in isometric 3D views (*e.g. see 3D zoom level 4 from within 3D zoom level 3*). The problem is that the parts of the outer volume can, and generally do, block the view of the inner volume. SIMION provides an option called cutaway clipping that cuts away part of the outer volume so that the inner volume is fully visible. SIMION changes what is cut away as you swing between isometric views.

To activate cutaway clipping click the **Cut** button (*so that it is depressed*). If an inner 3D volume is defined, SIMION will automatically cut away portions of it to make the inner volume visible (*Figure 7-6*). If the **Bx** button is depressed, the volumes and cuts will be outlined. If no inner volume is defined, the status of the **Cut** button will be ignored.

Edge Only Views

Normally SIMION shows the interior of volumes (*normal and cutaway*). However, sometimes all you really want to see is the edge of the volume and what intersects it. When the **Edg** button is depressed SIMION only shows the edge vectors that intercept the edges (*Figure 7-7 below*).



The thickness of the edge boundary region can be adjusted by changing the **Drawing Quality** panel. The lower the quality (*e.g. 0*), the thicker the edge boundary region.

Note: Ion trajectories will appear as short line segments. If ion trajectories suddenly appear as dashed lines or are very short, you may have accidentally depressed the **Edg** button.

The **Edg** button is also used (*when depressed*) to signal to the PE View that PE surfaces of 2D cylindrical arrays should use the array's volume boundary layer and not the array's center x-axis layer (*as is the default*).

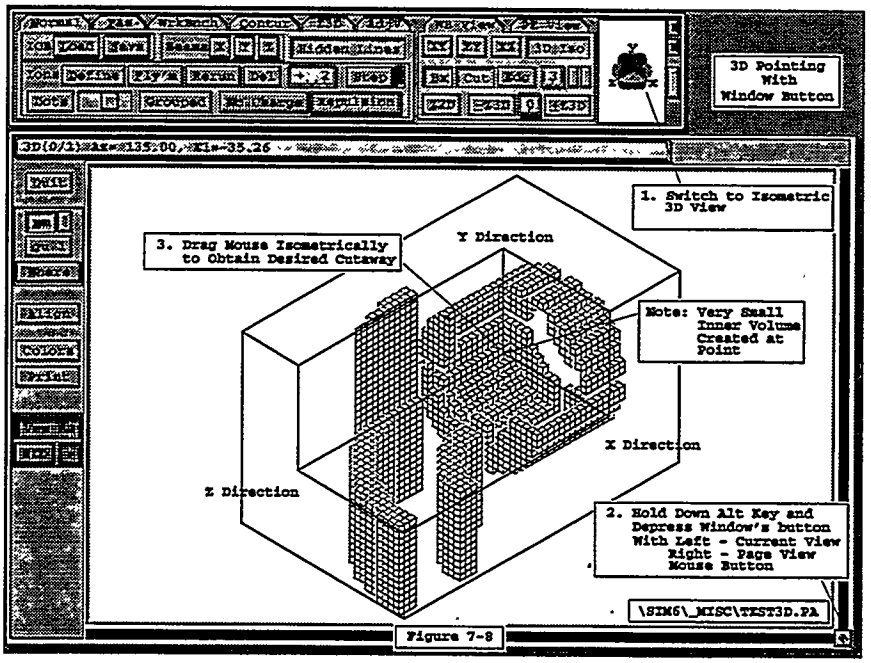
Using 3D Pointing to Create Inner Volumes

You have the option of using 3D pointing to create cutaways and inner volumes.

Creating a Cutaway by 3D Pointing

Figure 7-8 below, shows how to make use of 3D pointing to make a cutaway within the current volume (*current 3D zoom level*).

1. Swing to any ***standard angle*** isometric view.
2. Hold down the **Alt** key (*and keep it depressed*).
3. Move the cursor to the window's button (*lower right corner of view*) and depress the **right** mouse button for page view or the **left** mouse button for current view 3D pointing.
4. Drag the mouse in isometric directions (*e.g. vertical for y motions*) to move the cutaway point to the desired location.
5. You have the option of faster drawing if you depress the **other** mouse button too.
6. Release the mouse button(*s*) when you think you have cutaway what you want, and the view will be redrawn.
7. Repeat from step 2 until you cutaway what you want.

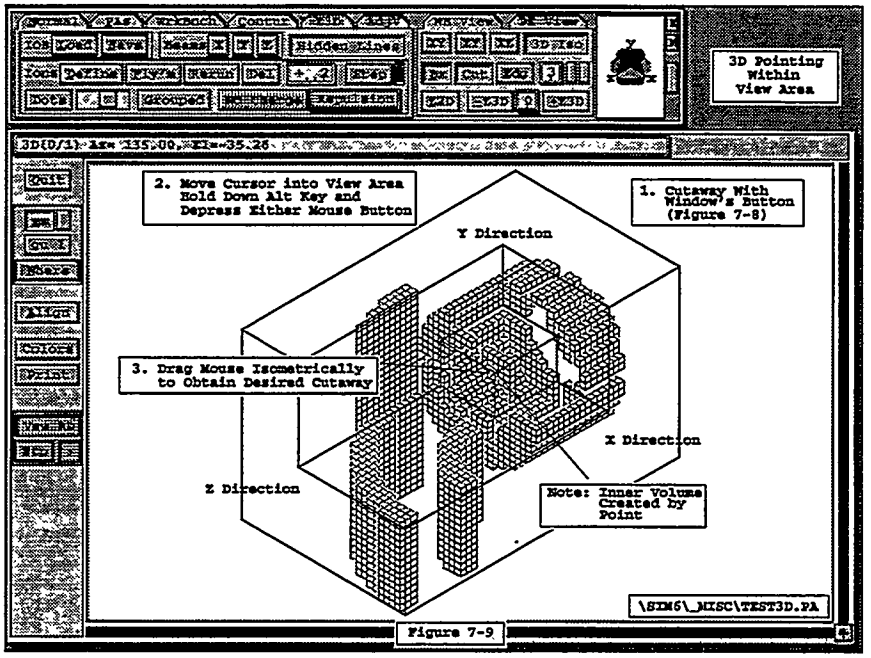


Creating an Inner Volume by 3D Pointing

Figure 7-9 below, shows how to create/size an inner volume via 3D pointing. You may start by either using 3D pointing to create a cutaway as in Figure 7-8 above, or with a existing inner volume displayed (*created by any means - e.g. marking and 3D zoom*) with the **Cut** button depressed.

1. Swing to any ***standard angle*** isometric view.
2. Hold down the **Alt** key (*and keep it depressed*).
3. Move the cursor into the view's area and depress **either** mouse button.
4. Drag the mouse in isometric directions (*e.g. vertical for y motions*) to move the inner volume corner point to the desired location.
5. You have the option of faster drawing if you depress the **other** mouse button too.

6. Release the mouse button(*s*) when you think you have the inner volume you want, and the view will be redrawn.
7. Repeat from step 2 until you have the desired inner volume. ***Note: You have the option of switching to any of the eight standard isometric views.*** Thus you can move any of the inner volume's eight corner points (*just use the proper isometric view*).



Another Way to Adjust the Size of a 3D Zoom Volume

Once you have defined a 3D zoom volume you can adjust its size using the 3D pointing method discussed above or you can make use of the **Z3D** tab to access the 3D Volume Control Screen (*discussed later in this chapter*).

Using the E and M Drawing Control Buttons

Just to the right of the orientation sphere are two small buttons marked **E** and **M**. They can be used to suppress drawing of ***all*** electrostatic (**E**) and/or magnetic (**M**) instances when they are depressed. There are certain cases when this option is useful.

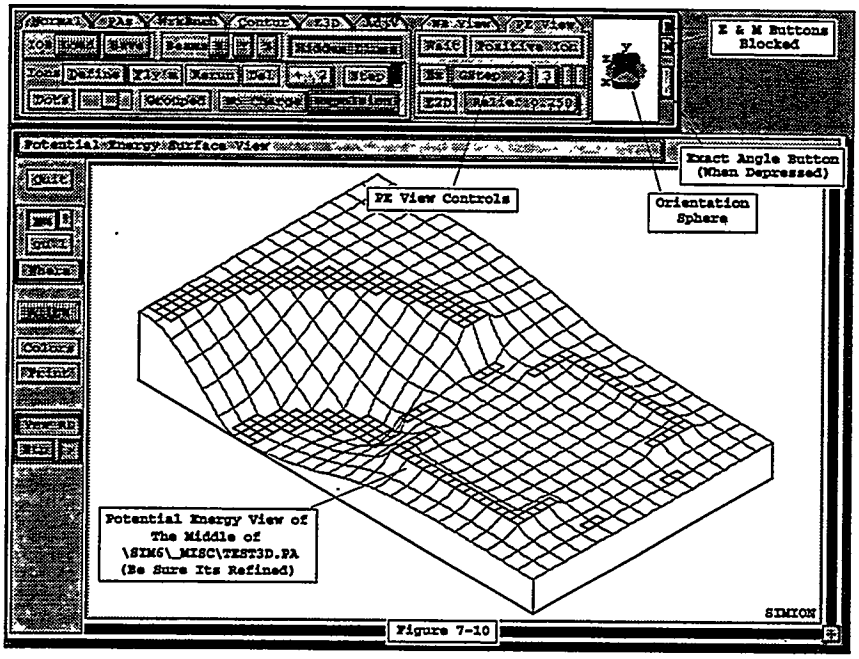
- You want to view electrostatic instance potential energy (*PE*) surfaces and not have magnetic instance PE surfaces confusing the issue (*depress the* **M** *button* ***before*** *entering PE View*).
- You want no instances drawn so you can see the ion trajectories more clearly (*depress both* **E** *and* **M**).
- You have a 3D surface contour and you want to see it without having the image of the instance blocking its view.

You also have the option of turning off the display of any instance individually via the PAs Control Screen (*discussed later in this chapter*).

Note: Although an instance may not be displayed, it doesn't mean that its effect is removed too! Ions fly through and splat into invisible (*not drawn*) instances just the same as when the instances are visible.

Accessing and Controlling Potential Energy Views

Potential Energy (**PE**) views are accessed by clicking on the **PE View** Tab to access the PE View Control Screen (*displayed in Figure 7-10 below*).



What is a PE View

In PE View, you are viewing a potential energy surface of ***one plane*** of the currently displayed 3D volume in the WB View. ***The potential energy surface has real physical significance for electrostatic instances***. A PE View of an electrostatic instance is to an ion what a miniature golf course is to a golf ball. Thus the PE View of an electrostatic instance is very useful for gaining insight concerning why the ions have the trajectories they do. Moreover, it provides an important level of understanding that can lead to design improvements.

Unfortunately potential energy views do not have much significance for magnetic instances. If your workbench contains both electrostatic and magnetic instances you may want to turn off the display of all magnetic instances (*by depressing the* **M** *button*) ***before*** clicking the **PE View** tab.

Special Requirements For Accessing PE Views

There are specific requirements that must be met before SIMION will allow access to PE Views and the display of specific instance PE surfaces.

Required WB Views for Accessing PE Views

SIMION will only allow access to potential energy views when the WB View is displaying a ***standard 2D view*** (*using either the three 2D view buttons or any 2D view of the orientation sphere*), and exact angle viewing is ***off*** (**Exact Angle** *button is not depressed*). SIMION will ***automatically*** block access to the **PE View** tab whenever, the current WB View is ***illegal*** for accessing a PE View.

Required Instance Alignment

To be displayed in a PE View, an instance must be ***integrally aligned*** with the ***currently aligned workbench coordinates***. An instance is integrally aligned if its x, y, and z axis are ***all parallel*** to ***any*** axis of the ***currently aligned workbench coordinates***. When an instance is integrally aligned a 2D view of the current workbench volume is a 2D view of the instance too. Thus SIMION can create a potential energy surface for it.

Note: the **Align** button can be used to ***integrally align*** the workbench coordinates with any instance (*thus allowing its PE surface to be displayed*). To temporarily align the workbench coordinates with a specific instance:

1. Click the **PAs** tab to access the PAs Control Screen.
2. Use the **Instance Selector** panel to select the desire instance. The currently selected instance's view image will be outlined in green.
3. *Depress* the **Align** button and the workbench coordinates will be ***temporarily*** integrally aligned with the selected instance's.
4. Now you can click the **PE View** tab and the instance's PE surface will be displayed.
5. Be sure to ***turn off*** this temporary alignment when no longer needed (*click the* **Align** *button to raise it*).

Instances Must be in the Current Volume to be Visible

To be visible in a PE View, an instance ***must be visible*** within the currently displayed WB View volume (*at the current 3D zoom level*). Instances that fall outside the current WB display volume ***will not be displayed*** in a PE View.

Only the portion of the instance that actually falls within the current WB display volume will appear in a PE View. ***This means that 3D volume zooms can be used to control the region of the instance that is displayed as a PE surface.***

What PE Surface Does SIMION Show?

When SIMION displays a PE surface, it is only displaying the PE surface of ***one layer*** of each integrally oriented instance that is visible within the current WB View volume. ***The question is: Which layer?*** The following explains the layer selections rules:

Rules for 2D and 3D Planar Instances

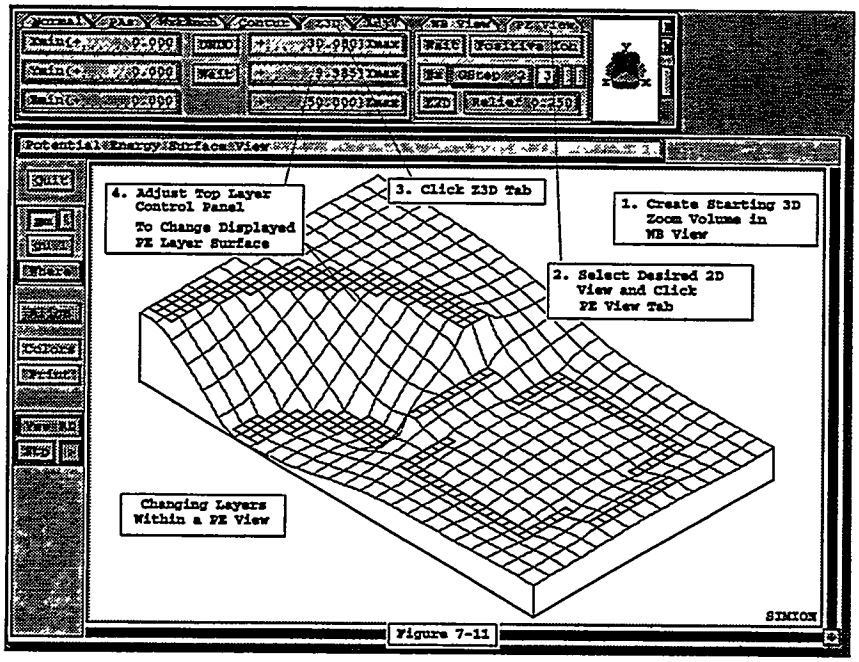
SIMION uses a simple layer selection rule for 2D and 3D Planar instances. Start inward from the front surface of the current WB View volume and display the first instance layer encountered. An instance layer is just like a layer in the **Modify** function (*a plane of array points*). ***SIMION always uses the plane array points nearest to the display screen for its selected PE surface layer.***

This means you can see the PE surfaces of layers inside an instance by cutting inside it with a 3D Zoom (*as in Figure 7-10 above*).

Rules for 2D Cylindrical Instances

2D cylindrical instances present a special problem for PE Views. What you normally want to see is the ***on x-axis layer*** (*central plane PE view*) assuming you're ***not*** looking at the instance from an end-on view. ***When the view of a 2D cylindrical instance is not an end-on view, SIMION shows the central plane (x-axis layer) PE view.***

If you want to view other layers of the volume in a ***side view*** of a 2D cylindrical array, click the **Edg** button ***before*** entering the PE View. ***SIMION now treats the 2D cylindrical array as if it were actually its pseudo equivalent 3D Planar dual for layer selection and PE surface generation.***



Using 3D Zoom Tricks to Select PE Display Layers

As stated above, the 3D zoom feature of WB Views can be used to select which instance layer is viewed as a PE surface. However this approach is unsatisfactory if you want to change the displayed PE surface layer from within a PE View.

The trick is to 3D zoom into a starting volume and then enter the PE View. Now from within PE View click the **Z3D** tab to access the 3D Zoom Volume Control Screen. Change the position of the front of the 3D volume's surface (*using the appropriate panel control*) and SIMION will automatically re-draw new PE surface layers as required. Figure 7-11 shows an example of this.

The Various PE View Controls

The PE view controls consist of the orientation sphere, exact angle button, and other controls on the PE View Control Screen (*Figure 7-10 above*). When you switch between WB and PE views SIMION ***remembers the settings*** of these controls (*including orientation sphere and exact angles button*) and ***automatically restores them*** when you re-enter the view. The following material discusses the various PE View controls:

Orientation Sphere and Exact Angle Button

The orientation sphere works much the same way it did in WB Views. In the case of PE views there are only four standard views. These are the ***four isometric views*** with ***positive*** elevation angles.

If you want some other view, depress the exact angle button. You will then be allowed any view with a ***positive*** elevation angle. ***Note: The use of exact angles doesn't slow or complicate displays as it does in WB Views.***

Wait Button

The **Wait** button is provided to allow you to change several things without having SIMION automatically re-draw the PE view after each change. To stop auto-display updating *depress* the **Wait** button. ***Now make your changes.*** Then click the **Wait** button *again* and the display will be updated.

Positive or Negative Ion Surfaces

SIMION provides a **Positive Ion/Negative Ion** button to display the potential energy surface that an ion of the selected charge polarity would see. ***This is very helpful to understand the differences in trajectories for ions of opposite polarities.*** To select the desired surface polarity, click this button until it shows the desired ion polarity.

The Bx Button

The **Bx** button is used to display the outline of the current PE surface volume when depressed. This volume has the size of the 2D image of the current volume for its horizontal dimensions and the maximum potential for its vertical dimensions.

The GStep Panel

The **GStep** panel sets the grid spacing in terms of displayed electrode point spacing. The default value of 2, means draw one potential grid line (*green*) for every two drawn electrode lines (*black*). Higher values (*10 is the maximum*) result in a courser potential grid surface.

The Drawing Quality Panel

The next object to the right is the **Drawing Quality** panel. It has the same general function as the equivalent object in the WB View Control Screen. The default value is three. A display quality value of 9 forces all designated vectors to be drawn no matter how small the image of the instance on the screen. ***Note: Unlike the WB View the use of a drawing quality of 9 with exact angles <u>will not</u> adversely affect either drawing or printing speeds.***

The Two Small Buttons

Two small buttons are provided just to the right of the **Drawing Quality** panel. These buttons suppress the drawing of grid lines (*green*) in a particular direction (*when depressed*). SIMION will allow you to suppress drawing one, ***but not both***, directions of grid lines. ***Suppressing the drawing of grid lines in one direction sometimes improves the surface view.***

The Z2D Button

The **Z2D** button supports a ***10 level bi-directional 2D zoom*** in the same manner as in WB View. To zoom in, mark a 2D zoom area with the mouse (**left** *button depressed*); and zoom by clicking the **right** mouse button, clicking the **Z2D** button, or by hitting the **Z** key. Zoom outs are made by clicking the **right** mouse button (*or as above*) without marking an area first. To zoom back in, hold down the **shift** key and click the **right** mouse button (*or as above*) without marking an area first.

You also have the option to 2D zoom and scroll using the window's button in the same manner described for WB Views.

The Relief Panel

The **Relief** panel controls the vertical scaling factor or relief of the potential energy surface. Increasing the value for relief increases the y-height of the higher potential areas. The default value is 0.250. This is the control you make use of along with the orientation controls to obtain just the right vantage point.

Controls on the Left Edge of the View Screen

The View Screen has a collection of controls on the left edge of the screen (*Figure 7-12 below*). Their functions will be described below:

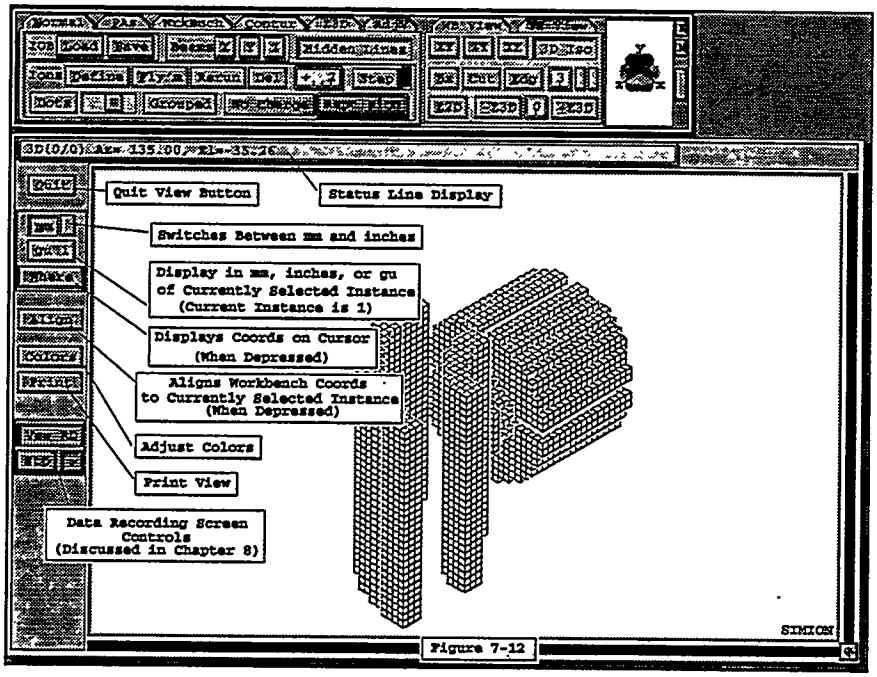


Figure 7-12

The Quit Button

Clicking the **Quit** button or hitting the **Esc** key (*when ions are not currently flying and a screen re-draw is not currently in progress*) will signal to SIMION that you want to exit the **View** function. You will be asked if you're sure (*to prevent* **Esc** *happy exits*). Click **Yes** or hit the **Y** key and the current **View** session is history.

The Status Line Display

The status line display shows various messages. It is most frequently used for displaying cursor information within WB views (*doesn't display cursor information in PE Views*). What is displayed depends on the view:

Any 3D Isometric and All Exact Angle WB Views

The status line displays the azimuth and elevation orientation angles of the current view. ***If the cursor is over an electrode/pole, the potential of the electrode/pole will be displayed.***

Any Standard 2D WB View

The status line displays the 2D coordinates of the cursor in the current display units (*discussed below*). Moreover, if the cursor is over an instance the potential and gradient of the point under the cursor will be displayed. The feature displays the potential and gradient of the point on the first layer of the instance it encounters while looking inward from the screen. ***This works just like PE View layer selection*** (*discussed above*).

The Display Units Controls

Just below the **Quit** button is a group of buttons that control the units displayed on the status line and on the cursor (*when activated by the* **Where** *button*). Their functions will be discussed button by button:

The mm/in Button

The ***mm/in*** button when depressed forces the displayed units to be in **mm** or **inches** from the ***currently aligned workbench origin.***

The Blank Button

Just to the right of the **mm/in** button is a blank button. When this button is depressed, it switches the **mm/in** button (*if depressed*) from **mm** to **inches.**

Trick! When the **Where** button is depressed (*coordinates shown on cursor*), depress the **mm/in** button and point the cursor to a point on a 2D standard WB View. Let's say that the current display is in mm. Now hit the **Spacebar** without moving the mouse. The display is now in inches. ***Hitting the* Spacebar *toggles between inches and mm.***

The gu ? Button

The ***gu ?*** Button (***?** is* **1** *in Figure 7-12*) designates that the displayed units be in grid units from the working origin of the currently selected instance. In Figure 7-12 the currently selected instance is 1.

To change the currently selected instance, click the **PAs** tab to access the PAs Control Screen. Now use the **Instance Selector** panel to designate the current instance number (*instance image in view will be outlined in green*). Finally, click the **Normal** tab to exit PAs Control Screen.

The Where Button

When the **Where** button is depressed the coordinates of the cursor will be displayed on the cursor whenever the cursor is within a 2D standard angle WB View. The units displayed are selected via the unit controls described above.

The Align Button

The **Align** button is used to ***temporarily*** align the workbench coordinates with the working origin of the currently selected instance when depressed. ***This can be done at any time - even when ions are flying.***

How to Align the Workbench With an Instance

To align the workbench with an instance you must first select the desired instance (*the currently selected instance*) and ***then*** depress the **Align** button.

To select an instance, click the **PAs** tab to access the PAs Control Screen. Now use the **Instance Selector** panel to designate the current instance number (*instance image in view will be outlined in green*).

What is Changed?

When the workbench coordinates are aligned with an instance, the following changes are made:

- The workbench origin is located at the instance's working origin.
- The workbench x, y, and z axis are aligned with the instances x, y, and z axis respectively.
- The positions, scaling, and orientations of ***all*** other instances are changed to reflect the new workbench orientation.

- The workbench dimensions are expanded, ***if needed,*** to contain the non-aligned workbench volume within the aligned workbench.

How to Return Back to Normal Alignment

The transformation can be reversed at any time by clicking the **Align** button (*raising it*). ***Note: All editing of instances done while the workbench is aligned with an instance: Adding, deleting, moving, sizing, and/or orienting will be conserved in the reverse transformation*** (*when the* **Align** *button is clicked again - to raise it*).

How to Make the Alignment Permanent

You also can keep the current alignment permanent by saving the current workbench definition to its current (*or create a new*) **.IOB** file. Click the **Normal** tab to access the Normal Controls Screen and click the **Save** Button. SIMION will ask if you want the **.IOB** saved with the current alignment. Click the **Yes** button and save the **.IOB** file. ***Both the saved .IOB file and current workbench image retain the alignment as the normal unaligned orientation.***

Uses for the Align Button

The following examples should serve to get you thinking of the many important uses of the **Align** button:

- To force an instance into integral alignment so that it can be viewed by WB View's fast rendering.
- To force an instance into integral alignment so that its PE surface can be seen in a PE View.
- To temporarily align on one instance so that positioning or orientation of another instance (*along perhaps an off-axis beam line*) can be more easily accomplished.

The Colors Button

The **Colors** button is provided to allow you quick access to SIMION's color palette controls. Changing colors can sometimes help improve visualization.

Note: There is also a **Colors** button on the Print Options Screen. It is provided to facilitate changing colors while printing.

The Print Button

The **Print** button accesses the print and annotate function to vector print the current view. ***See Appendix G for information on your printing options.***

Be sure to click the **Options** button to make sure the printer port (*e.g.* **PRN**) and printer language (*e.g. PostScript*) are appropriate for your printer and its connections. Remember! If you want color output, depress the ***B/W*** button so it will switch to the word **Color.**

Trick! If you want to pixel print the entire screen (*view and all*), hit the **<Ctrl F1>** key to access the GUI sketch/print utility, hit the **P** key for print, click the **NO** button to select full screen printing. Have Fun!

The Data Recording Display Screen Controls

The lowest area on the left of the view screen contains a collection of buttons used to control the Data Monitoring Screen (*when active*). Figure 7-13 shows access to these controls as blocked. ***This* View *feature will be discussed in Chapter 8.***

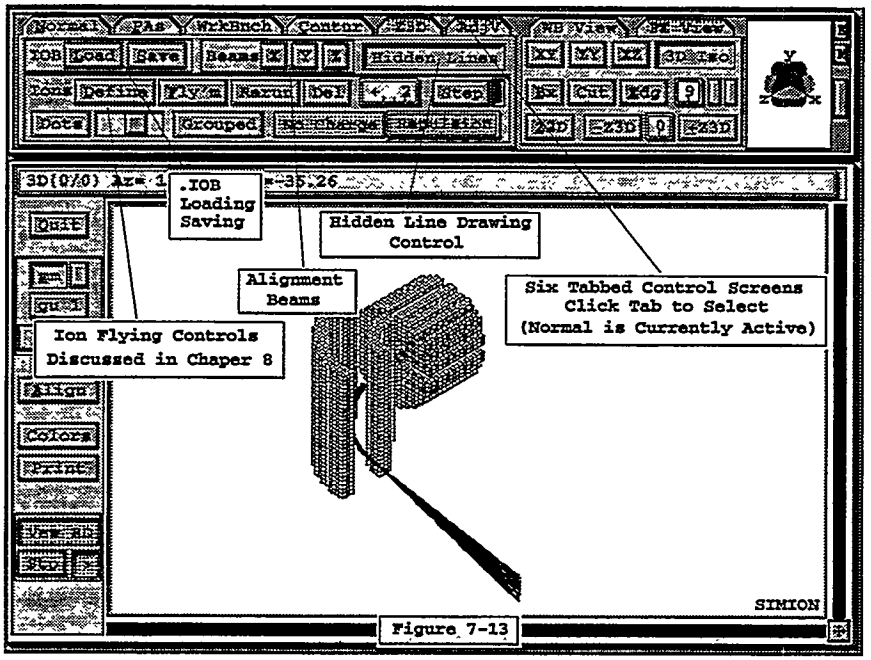


Figure 7-13

The Six View Control Screens

The final topics for discussion with the **View** function involve the six tabbed control screens: **Normal, PAs, Wrkbnch, Contur, Z3D,** and **AdjV** (*Figure 7-13 above*). Each of these control screens are accessed by clicking on their tab or hitting an access key (*e.g.* **N** *for* **Normal**) if allowed. Access to these control screens is normally allowed at any time, even when ions are flying. ***However, whenever access would be inappropriate, the access tab will automatically be blocked*** (*e.g.* **Z3D** *and* **AdjV** *tabs in Figure 7-13 above*).

The Normal Control Screen

The Normal Control Screen is accessed by either clicking the **Normal** tab or hitting the **N** key. Access to this control screen is allowed at any time. The Normal Control Screen provides access to **.IOB** file saving and loading, alignment beams, hidden line drawing control, and the various ion flying controls (*discussed in Chapter 8*).

.IOB File Saving and Loading

The Normal Control Screen has **Save** and **Load** buttons for saving and loading **.IOB** files. An **.IOB** file contains a complete workbench definition: Workbench size, definition of active instances, the names of the potential arrays each instance references, and their potentials (**.PA** *and* **.P?0** *files only* - **? = A - Z**).

The Save Button

The **Save** button is used to save the current workbench definition as an **.IOB** file. Saving the workbench definitions to an **.IOB** file is important if you want these definitions to persist between SIMION sessions.

If any array has been fast adjusted recently, you will be asked if you want to save it too. The only reason for saving these changes is if you plan to load it outside of an **.IOB** file (*e.g. with* **Old** *or* **Load**) and want the potentials conserved. ***Remember the .IOB contains the active potentials at the time of .IOB file saving, and SIMION can restore these potentials whenever the .IOB is loaded.***

Also, if the **Align** button is currently depressed, you will be asked if you want the workbench saved with the current alignment active.

The Load Button

The **Load** button is used to load an **.IOB** file and its workbench definitions and make them the current workbench definitions for the **View** function. Any required potential arrays that are *not* already in memory will automatically be loaded. SIMION will ask if you want the array potentials *restored* to the values active when the **.IOB** was last saved/replaced. If you select **Yes** (*the default*) SIMION will automatically fast scale **.PA** files and fast adjust **.P?0** files to restore their potentials.

It is *not* recommended that you use the **Load** button to routinely load workbench definitions. The suggested method is to exit **View**, remove all PAs from RAM, and then re-enter **View** - forcing an automatic **.IOB** load request. ***This approach minimizes the clutter of PAs in RAM.***

Alignment Beams

The **X**, **Y**, and **Z** buttons on the Normal Controls Screen activate x, y, and z axis alignment beams. The alignment beams pass through the working origin of each instance and are aligned with their PA's x, y, or z axis respectively. Thus if two instances exist and the **X** alignment beam button is depressed *two* x-axis alignment beams will be drawn - ***one for each instance.***

Alignment beams are often useful for checking instance and ion beam positioning.

Hidden Line Drawing Control

The **Hidden Lines** button is used to control hidden line removal for ***alignment beams***, ***contours***, and ***ion trajectories*** (*only*). When this button is depressed (*its default*) SIMION uses hidden line drawing methods to remove portions of alignment beams, contours, and ion trajectories that are visually blocked by instance images.

The Ion Flying Controls

The Normal Controls Screen also has a collection of ion flying controls. ***These controls are discussed in Chapter 8.***

The PAs Control Screen

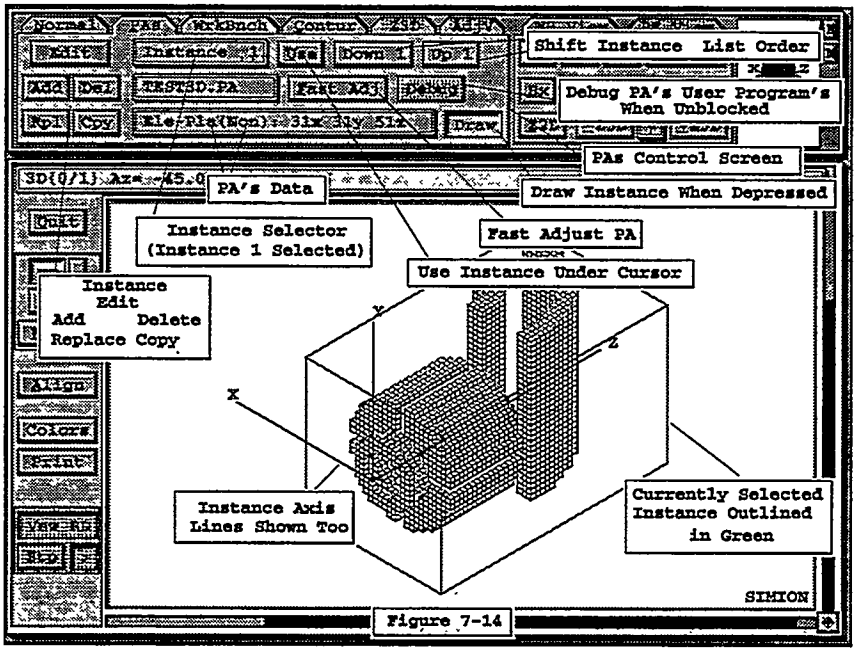
The PAs Control Screen is used to add, delete, and edit instances of potential arrays (*Figure 7-14 below*). The control screen is accessed by clicking on the **PAs** tab or hitting the **P** button.

Instance Selection, Information, and Access Controls

The right two thirds of the PAs Control Screen contains controls to select an instance, display its PA's description, and access/control various items.

The Instance Selection Panel

The **Instance Selection** panel is used to display/select the currently selected instance. An instance must be selected before it can be acted on by operations like **edit** and **delete**. The currently selected instance is outlined by a green box and its x, y, and z axis are drawn to aid in visualizing its orientation.



The Use Button

The **Use** button is useful for quickly selecting an instance with the cursor in *any* WB View. Just point the cursor at the desire instance and hit the **U** key. SIMION will select the nearest instance to the cursor location as the currently selected instance and outline it in green.

The Down 1 Button

Moves the currently selected instance down one in the instance list towards *lower* priority (*e.g. was 2, now 1*). ***Instances with lower priority are not as likely to be visible to ions.***

The UP 1 Button

Moves the currently selected instance up one in the instance list towards *higher* priority (*e.g. was 1, now 2*). ***Instances with higher priority are more likely to be visible to ions.***

The PA Data Displays

Two display objects are provided to describe the potential array that is referenced by the currently selected instance. In Figure 7-14, the array's name is **TEST3D.PA** and it is an electrostatic planar non-mirrored 3D array with dimensions of : 31x, 31y, and 51z.

The Fast Adj Button

The **Fast Adj** button is used to access the fast adjust function for the array referenced by the currently selected instance. If the referenced PA is not fast adjustable or fast scaleable (*e.g.* **.PA#**) access to the button will be blocked. Note: The actual PA (*memory copy*) is fast adjusted (*or fast scaled*). ***If more than one instance references the same PA, any fast adjust changes in one of these instances will affect the others too!***

The Debug Button

The **Debug** button is provided to allow quick access to any user programs associated with the instance's potential array. This button is a direct link to the user program development system. Access to the button will be blocked if the potential array doesn't have any associated user programs (*e.g. a* **TEST.PRG** *file for a* **TEST.PA** *array file*) *or* ions are currently flying. ***See Appendix I for information on user programs.***

The Draw Button

The **Draw** button controls the drawing of the instance. If the button is depressed (*the default*) the instance will be drawn if and only if its associated E or M button ***is not*** depressed. If the **Draw** button ***is not*** depressed the instance ***will not*** be drawn.

Just because an instance is invisible to you, doesn't mean SIMION won't see it. ***Ions fly through and collide*** (*splat*) ***with invisible instances just as if they were visible.***

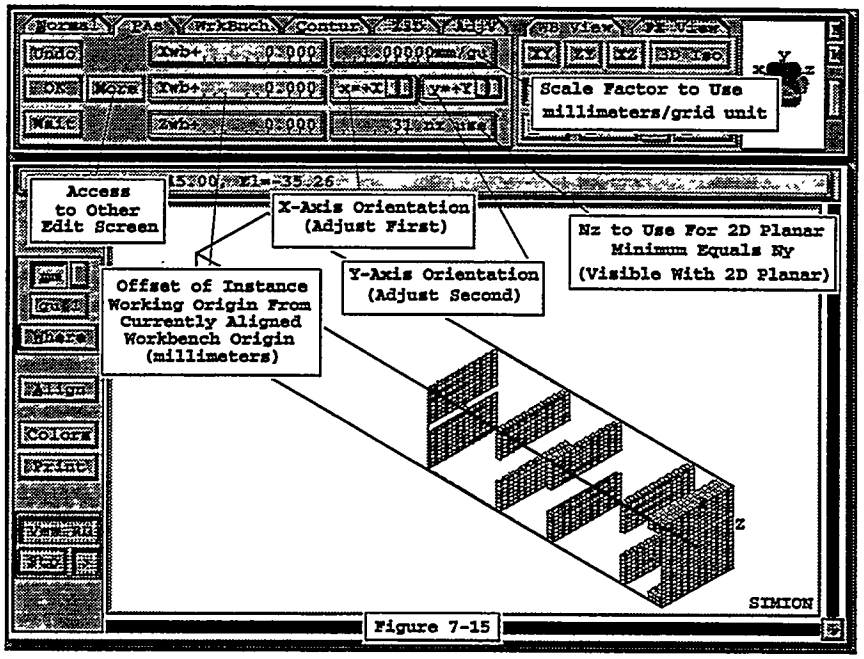


Figure 7-15

The Edit Button

The **Edit** button is used to edit the positioning, scaling, and orientation of the currently selected instance (**Instance Selection** *panel*). To edit an instance, select it with the **Instance Selection** panel and then click the **Edit** button. SIMION automatically switches to the first of two instance parameter screens (*Figure 7-15 above*).

The Undo Button

The **Undo** button is visible on ***both*** edit screens. It is used to restore the positioning, scaling, and orientation to what they were when the **Edit** button was ***last*** clicked.

The OK Button

The **OK** button is visible on ***both*** edit screens. Clicking the **OK** button accepts the changes and returns you to the PAs Control Screen. Unless the **Wait** button is depressed, switching to another control screen (*e.g. the Normal Controls Screen*) is the same as clicking the **OK** button (*any instance editing changes are kept*).

The Wait Button

The **Wait** button is visible on ***both*** edit screens. When it is depressed any editing changes are delayed until the **Wait** button is clicked again or the **OK** button is clicked.

Normally SIMION changes the instance interactively as you edit its parameters. This allows you to see the results of your editing immediately. The **Wait** button is provided to allow you to make several changes without creating a flurry of re-draws.

The Xwb, Ywb, and Zwb Panels

These three panel objects display/control the offset position of the instance's ***working origin*** from that of the ***currently aligned workbench origin***. The units of offset are millimeters.

The Scaling Panel

The **Scaling** panel displays/controls the scaling of the instance units (*gu*) into workbench units (*mm*) via the ***mm/gu*** scale factor. The default value is 1.0 millimeter/grid unit.

The Two Orientation Selectors

There are two **Orientation** selectors (*Figure 7-15 above*). These are used to display/control the instance's angular orientation. ***All*** angular rotations are made about the instance's ***working origin***. These selectors are coupled with the **Az, El**, and **Rt** panels on the second edit screen (*accessed via the* **More** *button*).

Use the **Orientation** selectors to insure that the instance will be integrally aligned with the ***currently aligned workbench coordinates***. If the instance is not integrally aligned the **Orientation** Selectors will display ***x= ?*** and ***y= ?***.

When adjusting the **Orientation** selectors, adjust the **x=** selector ***first*** and the **y=** selector ***second***. The **x=** selector allows six orientations (*+x, -x, +y, -y, +z, and -z*). The **y=** selector has four orientations that depend on the orientation of the **x=** selector.

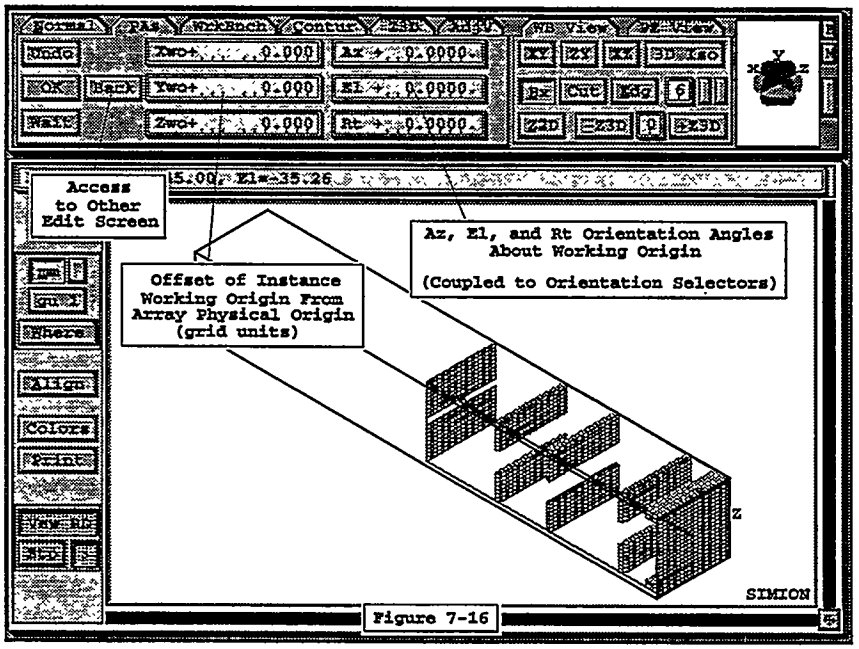


Figure 7-16

The More Button

The **More** Button is used to access the second instance edit screen (*Figure 7-16 above*). The second edit screen has controls to display/adjust the offset of the working origin

from the array's physical origin and the azimuth, elevation, and rotation angles for orienting the instance in the ***currently aligned*** ***workbench coordinates***.

The Back Button

The **Back** button is used to return to the first edit screen (*Figure 7-15 above*).

The Xwo, Ywo, and Zwo Panels

These three panels are used to display/control the offset of the ***working origin*** from the ***physical origin*** of the potential array. The units of offset are potential array grid units. This offset is normally 0,0,0 - the ***working origin*** is the array's ***physical origin***.

The Az, El, and Rt Panels

These three panels are used to display/control the orientation angles (*in degrees*) of the instance relative to the ***currently aligned*** ***workbench coordinates*** (*Figure 7-16 above*). A diagram of how these angles work is at the beginning of this chapter. ***These controls are normally used to define a non-integrally aligned orientation.***

The **Az, El,** and **Rt** controls are interconnected with the **Orientation** selectors on the first edit screen (*Figure 7-15 above*). In order for an instance to be integrally aligned with the workbench coordinates each of these angles must be integrally divisible by 90.

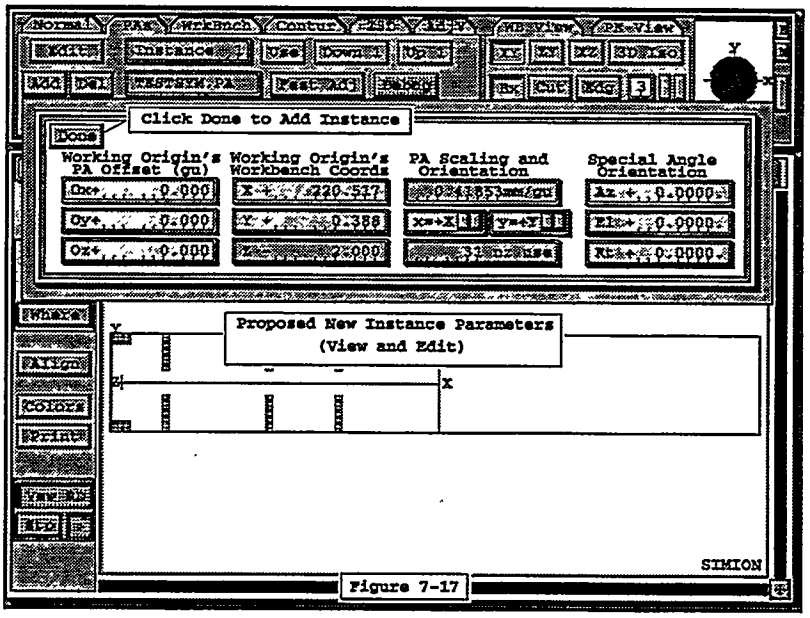


Figure 7-17

The Add Button

The **Add** button on the PAs Control Screen is used to add an instance to the workbench. The following procedure is used to add an instance:

1. Mark the volume you want the instance to be in. If you don't define a volume, SIMION will assume that you want the instance to fill the current workbench volume. ***You may want to increase the workbench size first.***
2. Click the **Add** button. SIMION will use the GUI File Manager to assist you in selecting the potential array to couple to the instance. Point to the desired potential array's button and click ***both*** mouse buttons (*only **refined** **.PA** and **.P?0** array files are useful for ion flying*). SIMION will automatically load the potential array if it is not already in RAM.

3. You will now see an instance edit screen that contains the parameters that SIMION plans to use for the instance (*Figure 7-17 above*). Edit the values of these parameters as desired. Now click the **Done** button and the instance will be added to the end of the list (*highest numbered instance = highest priority*).

You can then click the **Edit** button to edit the instance further as described above, use the **Up 1/Down 1** buttons to change its priority order in the instance list, or use the **Del** button to delete it (*assuming you really blew it*).

The Del Button

The **Del** button is used to delete the currently selected instance. To delete an instance, use the **Instance Selection** panel to select the desired instance. Click the **Del** button. SIMION will ask if you're sure. ***Click* Yes *and it's all over but the shouting.***

The Rpl Button

The **Rpl** button is used to change the potential array that is referenced by an instance ***without*** changing any of the instance's positioning, scaling, and orientation parameters. Thus **Rpl** allows you to substitute one potential array for another. One could use **Rpl** to connect a **.PB0** array into an instance that formerly was connected to a **.PA0** array. This allows different instances to of the same array to have different potentials (*discussed above*).

To replace an array, use the **Instance Selections** panel to select the desired instance. Click the **Rpl** button. SIMION will use the GUI File Manager to assist you in selecting the replacement array. Point to the array's button and click ***both*** mouse buttons.

The Cpy Button

The **Cpy** button provides a method to copy overlapping electrode (*or pole*) points from instances (*of the same type - e.g. electrostatic to electrostatic*) to the ***equivalent*** locations on the ***currently selected 3D (non-mirrored) potential array instance. This is an advanced SIMION feature used to help model interactions between instances.***

For example, let's say we have a collection of instances inside a larger 3D potential array instance (*the can*). Let's also assume that the instances interact electrostatically (*e.g. it is not field-free between them*). What we need is a way to model the effects of the interior instances on the fields of the containing 3D potential array instance. ***We could do this if we could project the appropriate electrode points from the interior instances into their equivalent 3D array instance points.*** We would then refine the 3D array to estimate the fields outside the interior instances.

For this problem, let's also assume that the 3D array is a **.PA0** fast adjust and that we want to be able to vary the effective voltages of the interior instance electrodes too. The first step would be to create the externally equivalent potential arrays for the internal instances. In most cases (*good design*) cylindrical potential arrays are enclosed by a tube. This means that the array looks like a solid tube to a outside observer. Thus one would use **Modify** to create a solid tube (*bar*) starting with the original array. If the tube is to be fast adjustable it should be given a potential appropriate to mesh properly with the 3D array's **.PA#** file and the file saved with a different name. Now load the desired **.IOB** file and use **Rpl** (*above*) to load the 3D **.PA#** file into ***its instance*** as well as the solid cylinder array into its appropriate instance. Now switch to the 3D instance, use the **Cpy** Button to copy the points from the solid cylinder. Now exit **View** and **Refine** the **.PA#** file. You can now get back into **View** and use **Rpl** to restore the original potential arrays to their proper instances (*be sure to save the .IOB file afterwards*).

Note: Because the destination 3D array is ***generally*** at much lower resolution than the source array you may have problems with ions splating on the crude array below as they exit the higher resolution array above (*in instance list order*). This problem can be avoided by slightly expanding the size of the interior potential arrays with an expanded boundary area (*volume*) of non-electrode points. This helps ions transition beyond any copied points (*and thus no splat*).

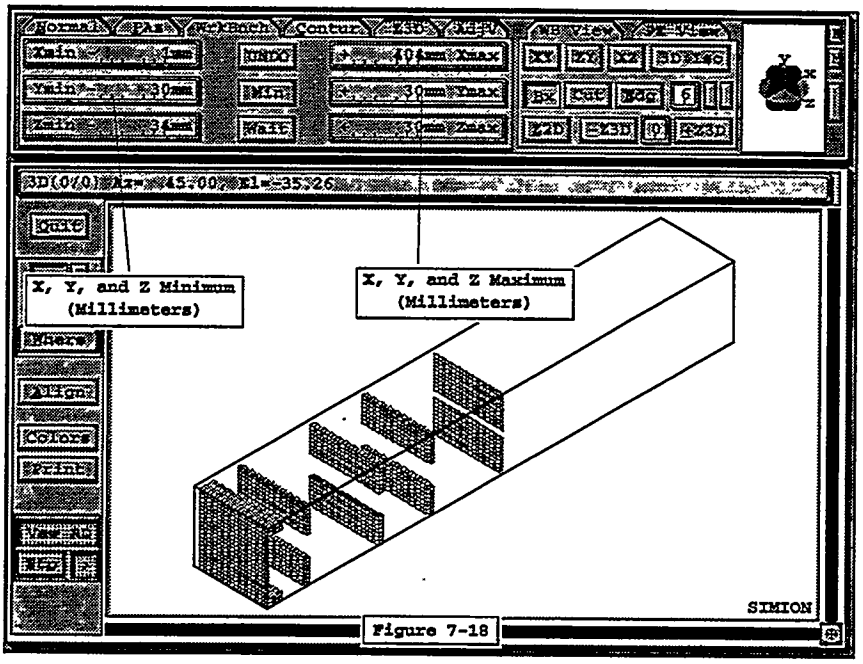


Figure 7-18

The Workbench Control Screen

The Workbench Control Screen is accessed by clicking the **WrkBnch** tab. This screen has objects that display/control the size of the current workbench (*Figure 7-18 above*).

The Undo Button

Clicking the **Undo** button restores the workbench size that was active when the Workbench Control Screen was entered (*when the* **WrkBnch** *tab was last clicked*).

The Min Button

Clicking the **Min** button reduces the size of the workbench to the ***minimum volume*** that the currently defined instances will just fit within.

The Wait Button

Normally SIMION changes the workbench volume interactively as you change the workbench volume parameters. However, if you depress the **Wait** button, SIMION will allow you to adjust the workbench dimension panels without immediately applying the changes to the workbench. ***Remember!*** You must click the **Wait** button (*to raise it*) ***before*** your modifications will be applied to the workbench.

The Xmin, Ymin, Zmin, Xmax, Ymax, and Zmax Panels

These six panel objects display/control the dimensions of the workbench in ***currently aligned workbench coordinates***. The units are in millimeters. You ***will not be allowed*** to adjust the workbench to a volume that would be ***smaller*** than the volume required to contain the currently defined instances.

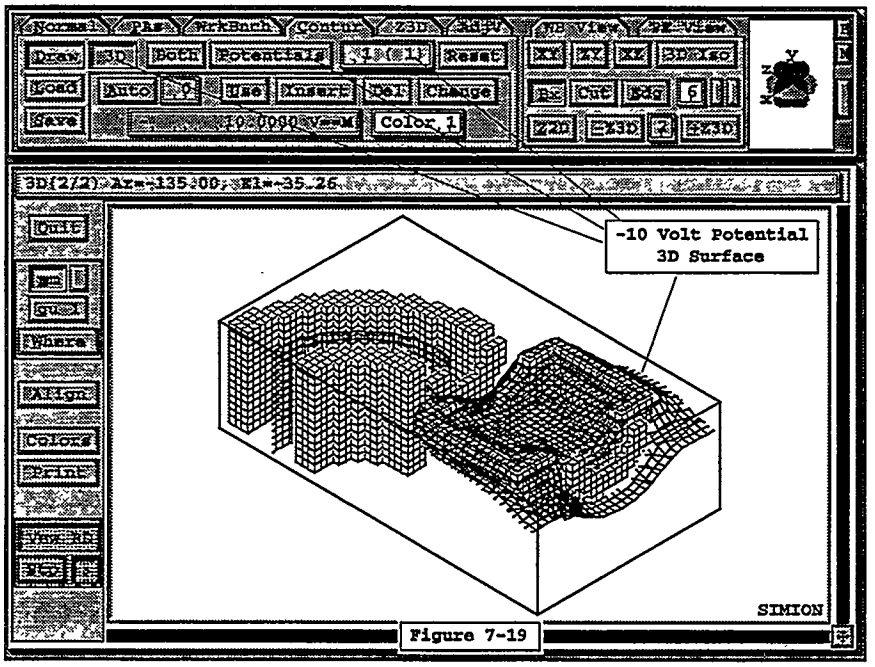


Figure 7-19

The Contour Control Screen

Click the **Contur** tab or hit the **C** key to access the Contour Control Screen (*Figure 7-19 above*). The Contour Control Screen is used to control the drawing of both potential and gradient contours and 3D contour surfaces.

Instances Must be Integrally Aligned

Contours will ***only*** be displayed for instances that are ***integrally aligned*** with the ***currently aligned workbench coordinates***. You can integrally align any instance (*that is not currently integrally aligned*) by making it the currently selected instance via the **Instance Selection** panel on the PAs Control Screen (**PAs** *tab*), and then depressing the **Align** button.

Types of Contours

SIMION will draw potential (*volts or Mags*) ***and/or*** gradient (*volts/mm or Gauss*) contours of electrostatic and magnetic instances. Up to ***99*** contours of each type (*potential and gradient*) can be currently defined. Each contour is defined by its potential or gradient value and its color (*index*). Note: Contour colors are separately controllable.

Contours can be drawn either as lines or 3D surfaces. Your contouring options depends on the view selected (*e.g. WB or PE*).

Contour Lines

Contour lines can be drawn in any WB or PE View.

Contour Lines in PE views

PE Views support the drawing of contour lines (*only*). These contour lines are the same as shown in the equivalent ***standard angle 2D*** WB View.

Contour Lines in 2D WB Views

Contour lines are drawn ***standard angle 2D*** WB Views (*the* **Exact Angle** *button must be off*). The status of the **3D** button is ***ignored.*** The instance must be drawn (*e.g. its* **Draw** *button depressed and its type button raised -* **E** *or* **M**) for ***standard angle 2D*** WB View contours to be drawn.

When SIMION displays contour lines in ***standard angle 2D*** WB Views, it is ***only*** displaying them for ***one layer*** of each integrally aligned instance that is visible within the current WB View volume. ***The question is, which layer?*** The following explains the layer selections rules (*same rules as for PE View surfaces*):

Rules for 2D and 3D Planar Instances

SIMION uses a simple layer selection rule for 2D and 3D Planar instances. Start inward from the front surface of the current WB View volume and display the first instance layer encountered. An instance layer is just like a layer in the **Modify** function (*a plane of array points*). ***SIMION always uses the nearest plane of array points for its selected contour surface layer.***

This means you can see the contour lines of layers inside an instance by cutting inside it with a 3D Zoom.

Rules for 2D Cylindrical Instances

2D cylindrical instances present a special problem for contouring. What you normally want to contour is the ***on x-axis layer*** (*central plane view*) assuming you're ***not*** looking at the instance from an end-on view. ***When the view of a 2D cylindrical instance is not an end-on view, SIMION contours the central plane (x-axis layer) view.***

If you want to view other layers of the volume in a side view of a 2D cylindrical array, click the **Edg** button. SIMION now treats the 2D cylindrical array as if it were actually its pseudo-equivalent 3D Planar dual for layer selection and contour generation.

Contour Lines in 3D WB Views

Contour lines are drawn in ***standard or exact angle 3D isometric*** WB Views when the **3D** button (*on Contour Control Screen*) is ***not*** depressed or the **Edg** button ***is*** depressed. ***Contours of 3D WB views will be drawn whether the instance itself is drawn or not.***

The three visible boundary planes of the ***current volume*** (*workbench or 3D zoom*) are used for determining the layer of the instance used for contouring. The instance layers that are used for contouring are those layers that ***actually intersect*** one of the three visible boundary planes. ***If no instance layer intersects a visible boundary plane no contour lines from that instance will be drawn in that boundary plane.***

3D Contour Surfaces

3D Contour Surfaces are only drawn in ***standard or exact angle 3D isometric*** WB Views when the **3D** button (*on Contour Control Screen*) ***is*** depressed and the **Edg** button ***is not*** depressed. ***Contour surfaces of 3D WB views will be drawn whether the instance itself is drawn or not.*** Thus you can make an instance invisible and SIMION will ***still*** draw its 3D contour surface(s).

Unlike contour lines, 3D contour surfaces will be drawn for all instances that appear within the ***current volume*** (*workbench or 3D zoom*). Use the **Hidden Lines** button on the Normal Controls Screen to determine whether hidden line drawing is active for 3D contour surfaces.

Contouring Controls

The Contouring Control Screen contains a collection of control objects. Note: SIMION uses the current **Drawing Quality** panel's value (*0 lowest - 9 highest*) of the current view type (*WB or PE*) to determine the quality level to draw any contours. The following is a tour of these objects to help you understand how to use them.

The Draw Button

The **Draw** button forces the drawing of all contours (*if defined*) when *depressed*. Once this button is depressed, contours will be drawn whenever the view is redrawn. To stop contour drawing click the **Draw** button to raise it.

Remember, if contouring is taking too long, you can always hit the **Esc** key to halt contour drawing on the current view (*seldom necessary*).

The Load and Save Buttons

SIMION allows you to save contour definitions to **.CON** files. The **Load** and **Save** buttons on the left edge of the Contour Control Screen are used to load and save **.CON** files (*via the GUI File Manager*). A contour file (*e.g.* **TEST.CON**) contains the definitions for all the potential ***and*** gradient contours defined when the file was last saved.

The Both Button

When the **Both** button *is* depressed, both the potential and gradient contours will be displayed. If the button to the right displays potentials, then the gradient contours will be drawn first (*if defined*) followed by the potential contours (*if defined*). Otherwise, the drawing order will be reversed.

When the **Both** button ***is not*** depressed only the contours of the currently selected type (*e.g. Potentials in Figure 7-19*) will be drawn.

The 3D Button

The **3D** button (*when depressed*) tells SIMION to draw 3D surface contours in ***standard and exact angle 3D*** WB views. If the **Edg** button is depressed the **3D** button status will be ignored. ***It is recommended that you not draw more than one 3D surface contour*** (*things can get pretty confusing*).

The **Hidden Lines** button (*on the Normal Controls Screen*) can be used to control the hidden line removal drawing of the 3D surface contours. You may also want to suppress drawing of the instance itself to aid in viewing the 3D surface contours (*via the instance's* **Draw** *button or the* **E** *or* **M** *button*).

The Potentials / Gradients Button

To the right of the **Both** button is the **Potentials / Gradients** button. It is used to select the type of contours to be drawn, defined, and/or edited. Click this button to switch between contour types. Up to ***99*** contours of each type can be defined at the same time.

The Contour Selection Panel

Immediately to the right is the **Contour Selection** panel. It displays two numbers. The ***leftmost*** number is the currently selected contour number, and the ***rightmost*** number is the total number of currently defined contours of the selected contouring type (*potentials or gradients*).

SIMION automatically keeps all contours of the same type in a sorted list from 1 (*smallest value*) to n (*highest value*). When you use the panel to select a contour by its number, SIMION will automatically display its value and color in panels at the bottom of the control screen.

The Reset Button

The **Reset** button is used to reset (*erase*) all contour definitions of the currently selected contouring type (*potentials or gradients*). SIMION will ask if you are sure before erasing these contour definitions.

The Auto Button

The **Auto** button is used to automatically generate ***potential*** contours. Contour definitions will be generated using the currently selected instance (*PAs Control Screen via* **Instance Selection** *panel*). There are two types of automatic contours depending on the value of the **Automatic Contour Number** panel immediately to the right.

The normal procedure for automatically creating contours is to select the color desired for the automatic contours (*via the* **Color** *panel*), select the number and type of automatic contours (*via the* **Automatic Contour Number** *panel*), and then click the **Auto** button. SIMION will scan the currently selected instance, generate the list of potential contours (*replacing any that are currently defined*), and then depress the **Draw** button (*if required*) to force the new contours to be drawn.

The Automatic Contour Number Panel

This panel controls the number and type of automatic potential contours generated depending on its value.

A Value of Zero (0)

If this panel has the value of zero, SIMION will scan the currently selected instance for electrode/pole points and remember their potentials. It will then create contours at each of these potentials ***and*** at potentials 10% above and below these potentials to the next higher and lower potential. If the number of contours thus generated would exceed 99, SIMION will reduce the number of contours defined.

Values Greater Than Zero

If this panel has a value greater than zero (*e.g. 9*), SIMION will scan the currently selected instance for the range of electrode/pole potentials. It will then create the requested number potential contours at equally spaced intervals between the maximum and minimum potentials found.

For example, if the maximum potential is 100 volts and the minimum is 0 volts, a panel value of 9 would result in contours being created at 10, 20, 30, 40, 50, 60, 70, 80, and 90 volts.

The Use Button

The **Use** button provides a second way to define contours. Switch to a ***standard angle 2D*** WB View, select the desired contouring type (*potentials or gradients*), set the desired color (**Color** *panel*), move the cursor to the point the contour will cross, and hit the **U** key. SIMION will determine, insert, and draw (*if the* **Draw** *button is depressed*) the new contour. Its value and color will be displayed on the **Contour Value** panel and **Color** panel objects.

There is a second way to make use of the **Use** button. In this case you mark an area, and then click the **Use** button or hit the **U** key. SIMION will draw the contour through the ***center*** of the marked area.

The Insert Button

The **Insert** button provides a third way to define contours. Simply select the contour type (*potential or gradient*), enter its value in the **Contour Value** panel, enter its color in the **Color** panel, and then click the **Insert** button. If the **Draw** button is currently depressed the new contour will be drawn.

Note: SIMION will refuse to insert a contour that duplicates the ***value*** of any currently defined contour.

The Del Button

The **Del** button is used to delete contour definitions. There are two ways to make use of this feature. The first approach is to select the contour type (*potential or gradient*), select the desired contour using the **Contour Selection** panel, then click the **Del** button to delete it. If the **Draw** button is currently depressed the contour will be erased. ***This works in any view*** (*WB or PE*)

In the second approach use the mouse to mark the contour (***move the cursor to a location on or near the contour line then click the left button***). Now click the **Del** button to delete the selected contour. ***This only works in standard 2D WB Views.***

The Change Button

A contour's value and/or color can be changed via the **Change** button. Select the contour type (*potential or gradient*), select the desired contour using the **Contour Selection** panel (***the value and color will be displayed on the*** **Contour Value** *panel and* **Color** *panel respectively*). Edit the contour's value and color as desired. Now click the **Change** button and the selected contour is changed. If the **Draw** button is currently depressed the contour will be redrawn as changed.

Note: SIMION will refuse to change a contour to a ***value*** that duplicates any other currently defined contour.

The Contour Value Panel

The **Contour Value** Panel displays the value (*potential or gradient*) of the contour currently designated by the **Contour Selection** panel. ***This is merely a copy of the contour's value.*** If you change it's value, nothing will happen until you click either the **Insert** or **Change** buttons.

The Color Panel

The **Color** panel displays the color of the contour currently designated by the **Contour Selection** panel. ***This is merely a copy of the contour's color.*** If you change it's color, nothing will happen until you click either the **Insert** or **Change** buttons.

The 3D Zoom Volume Control Screen

The 3D Zoom Volume Control Screen is used to control the size of a 3D zoom volume. To access this screen you must currently be 3D zoomed (*level 1 or higher*). Select the 3D zoom level to access and click the **Z3D** tab. SIMION will automatically ***delete*** any inner 3D zooms and display the 3D Zoom Volume Control Screen (*Figure 7-20 below*).

The 3D Zoom Volume Control Screen allows you to set the exact size of the 3D zoom volume. This can be done while ions are flying. The feature is very useful for seeing how moving a plane of the current volume changes the contours or potential energy surfaces. You can, in effect, sweep through the 3D zoom volume.

The UNDO Button

The **UNDO** button is used to restore the size of the selected 3D zoom volume to what it was before the **Z3D** tab was clicked to enter the 3D Zoom Volume Control Screen. ***Note:*** The **UNDO** button ***will not restore*** any lower level 3D zooms deleted when the **Z3D** tab was clicked.

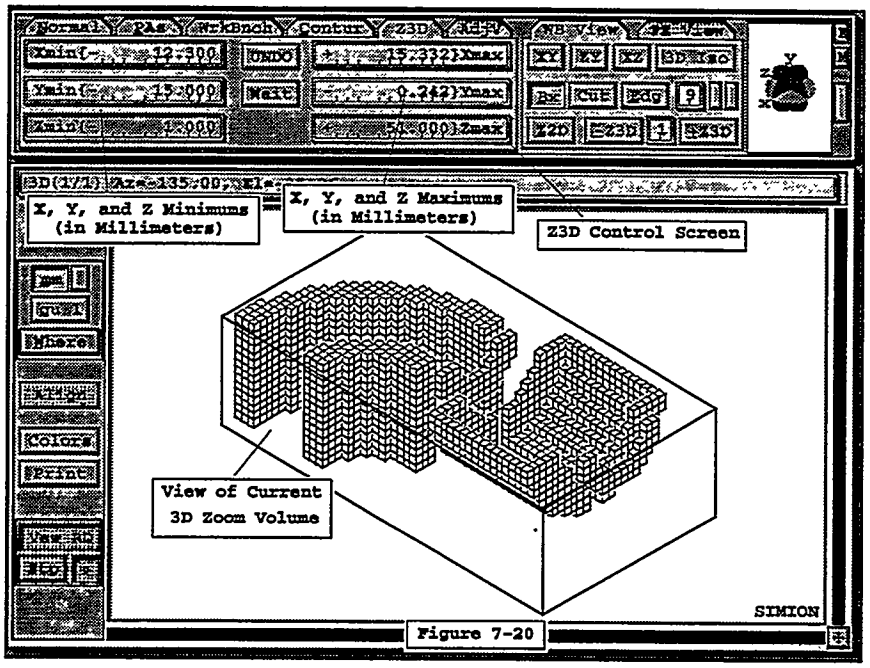


Figure 7-20

The Wait Button

SIMION normally adjusts the 3D zoom volume interactively as you change any of its dimensions. However, there may be times when you want SIMION to wait. Depress the **Wait** button and SIMION will not make any changes in the 3D zoom volume until you click the **Wait** button again to raise it.

The Xmin, Ymin, Zmin, Xmax, Ymax, and Zmax Panels

These panel objects are used to display/change the current 3D zoom volume dimensions (*in millimeters*). The panel objects will not let you do anything illegal (*like negative size or a larger size than the next outer volume - workbench or 3D volume*).

To change a dimension, select the appropriate panel and change its value. SIMION will re-size and re-draw the view as you change the panel's value. If you want SIMION to wait, depress the **Wait** button, make your changes, and then click the **Wait** button again (*to raise it*).

The Adjustable Variables Control Screen

The **AdjV** tab is used to access the Adjustable Variables Control Screen (*Figure 7-21 below*). This tab is normally blocked ***unless*** ions are flying with user programs active that have adjustable variables defined.

When you have user programs active that have adjustable variables defined, you can use this screen to change the values of these adjustable variables while the ions are flying. This is a useful feature if you want to adjust frequency, damping, or a wide range of other useful parameters.

Each adjustable variable has its own control panel with its name and value. A slider is provided to allow access to many adjustable variables - three at a time.

This control screen will be discussed further in Chapter 8 and Appendix I.

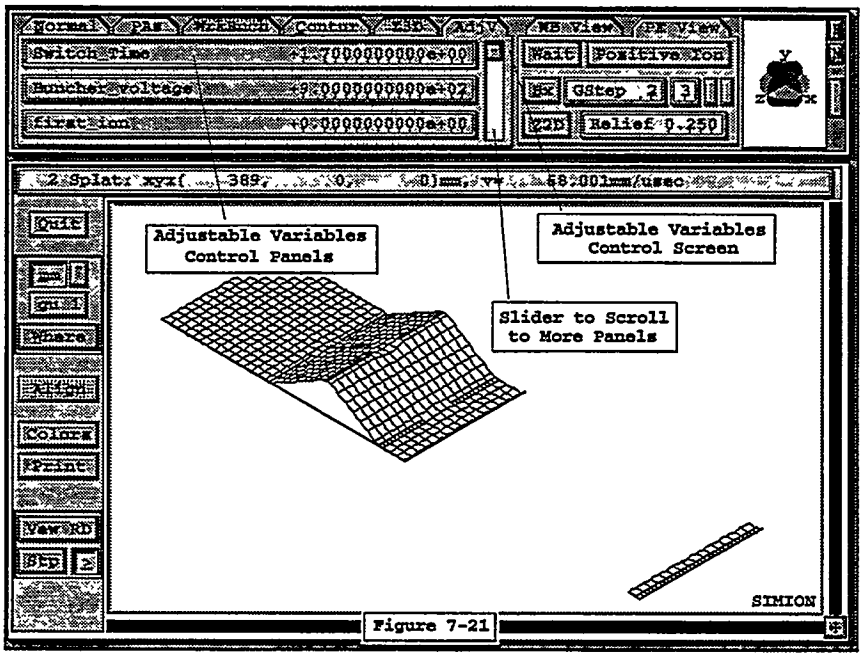


Figure 7-21

Flying Ions

Introduction

This chapter covers how to fly ions in SIMION's ion optics workbench. All Ions are flown within the View function. The material below assumes that you know how to create potential arrays, refine them, and project instances of them into a workbench volume. It is assumed that you have read the discussions of potential arrays in Chapter 2, 4, 5, 6, and 7. If not, read the material before proceeding further.

Chapter Organization

This chapter covers ion flying in the following order:

- Defining the ions to fly.
- Flying the defined ions.
- Selectively recording ion flight data.
- An introduction to user programs.

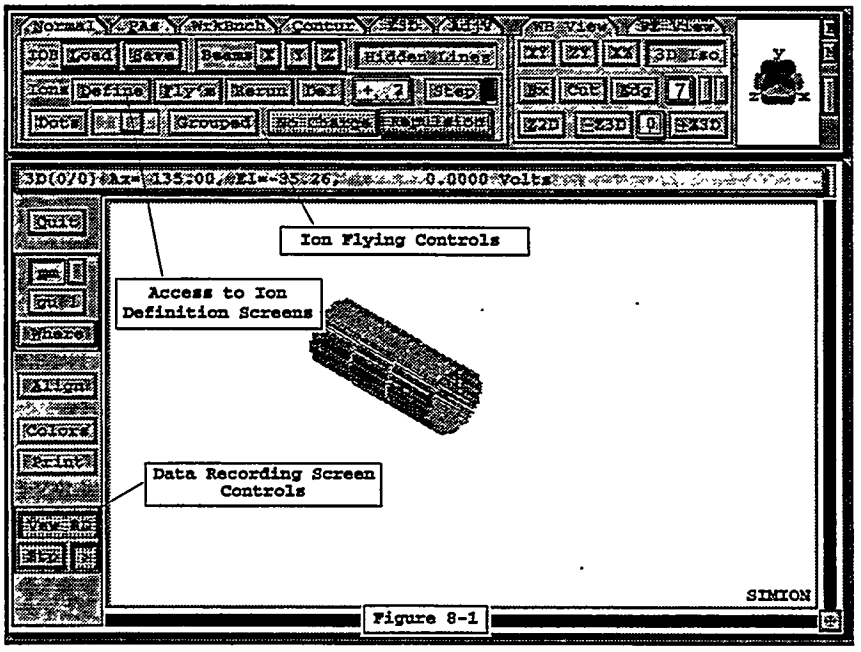


Figure 8-1

SIMION's Unit System

SIMION uses the following ***internal*** unit system for all its ion flying:

Mass	*Unified* atomic mass units (*amu*).
Charge	Elementary charge units (*e.g. electron has a charge of -1.0*).
Energy	Electron Volts (*eV*).
Time	Microseconds from the start of the **Fly'm** clock.
Distance	Millimeters in the ***currently aligned*** ***workbench coordinates***.
Velocity	Millimeters/microsecond in the ***currently aligned*** ***workbench coordinates***. This is the same as ***meters/millisecond*** or ***km/second***.

Defining the Ions to Fly

The Normal Controls Screen, in **View**, contains a collection of ion flying controls (*Figure 8-1 above*). The **Define** button is used to access the Ion Definition Screen.

Ion Definition Parameters

SIMION uses a collection of parameters to define each ion. The following material discusses each parameter and its use:

Ion's Mass (amu)

The mass of the ion is defined in ***unified*** atomic mass units (*amu*). This is the ***rest*** mass of the ion.

Ion's Charge (elementary charge units)

The ion's charge is specified in elementary charge units. For example, the charge on an electron is -1.0 elementary charge units.

Ion's Starting Kinetic Energy (eV)

The ion's starting kinetic energy is specified in electron volts (*eV*). SIMION translates this starting kinetic energy into initial ion velocity (*mm/microsecond*) using the ion's mass and applying relativistic corrections as required. This velocity will initially be in the ion's starting direction (*defined below*).

Ion's Starting Location

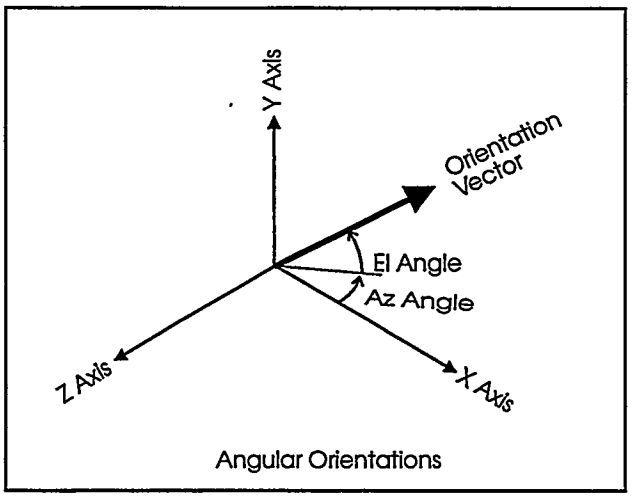
This is the x, y, and z coordinates of the location where the ion first appears. SIMION always uses the ***currently aligned*** ***workbench coordinates*** for ion locations. However, you must define the ion's starting location in one of the following coordinate systems (***the same coordinate system for all ions defined***):

1. ***Unaligned*** ***workbench coordinates*** (*mm*). These are independent of the current status of the **Align** button (*handy*).
2. In grid units (*gu*) from the ***working origin*** of a user specified instance.

SIMION will ***automatically*** convert ion starting locations defined in any of the above coordinate systems into the ***currently aligned*** ***workbench coordinates*** for its internal use.

Ion's Starting Direction

The ion's starting direction is defined through the use of azimuth (*Az*) and elevation (*El*) angles (*in degrees*). These angles are ***relative*** to the coordinate axis system used to define the ion's starting location (*diagram on right*). For example, if the ion's starting location was specified in instance number one's coordinates, the orientation angles would be relative to instance number one's x-axis, y-axis, and z-axis.



Ion's Time of Birth (TOB)

SIMION starts the elapsed time clock when you click the **Fly'm** button to start flying the ions (*units of time are in microseconds*). Normally all ions are created at time equals zero. However, you have the option of a delayed time of birth for each ion. The time of birth parameter specifies this delay in microseconds.

Ion's Color

SIMION allows individual ions to have different colors. Any of SIMION's 16 colors (*color index 0-15*) can be selected.

*Note: **Selecting a color of 15 will make the ion invisible.*** This is useful if you are using charge repulsion on a group of ions and only want to see the trajectories of one or more selected ions to keep down the clutter.

Ion's Charge Weighting Factor (CWF)

SIMION 6.0 supports certain types of charge repulsion. To speed calculations, each ion normally represents a cloud of ions. However, the exact relative number of ions that each ion represents may be different. For example, if each ion represents the ions in rings of cylindrical slices, and the ***actual*** ions are uniformly distributed, the ions in the outer rings probably represent more physical ions than the ions in the inner rings.

The charge weighting factor (*CWF*) is provided to allow you to tell SIMION that this ion represents 50% more actual ions (*CWF = 1.5*) than the normal ion (*CWF = 1.0 - default value*).

The Top Row of Controls on the Ion Definition Screen

The top row of controls perform the following functions (*Figure 8-2 below*):

The Cancel Button

The **Cancel** button discards any ion definition change and restores the ion definition active when the **Define** button (*on the Normal Controls Screen*) was clicked.

The OK button

The **OK** button exits the Ion Definition Screen and keeps any ion definition changes as the current ion definitions.

The Coordinate System Selector

The **Coordinate System** selector is used to select the coordinates to use for locating the ions. The options are: ***Unaligned*** workbench coordinates or instance coordinates of ***one*** of the currently defined instances.

In Figure 8-2 (*below*) the coordinates chosen are instance number one's coordinates. The ***second*** number one (*1- to the right*) indicates that there is ***only one*** instance defined in the current workbench. Thus the ion starting locations are in grid units (*gu*) from instance number one's ***working origin***. Also, the orientation angles are relative to the instance number one's x, y, and z axis.

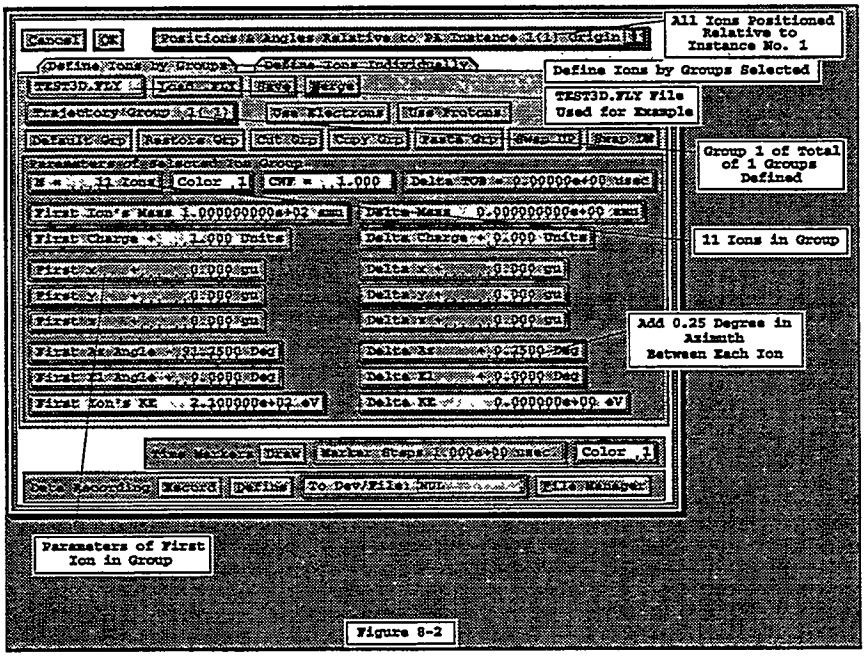


Figure 8-2

How Ions Are Defined

Ions are defined either in ***groups*** of similar ions or ***individually***. Each ion definition method has advantages for certain types of simulations. SIMION provides access to either definition method via tabs on the Ion Definition Screen (*Figure 8-2*).

Defining Ions by Groups

Clicking **Define Ions by Groups** tab on the Ion Definition Screen accesses the Group Definition Screen (*Figure 8-2 above*). This definition screen allows you to define ions in groups of ***similar*** ions. Up to **25** of these groups can be defined at one time.

A group of ions is defined by a starting ion definition (*ion number one in the group*) and ***parameter increments*** for use in defining the additional ions in the group. For example, in Figure 8-2, there is one group of ions defined. This group has 11 ions. The only difference between the ions is that the azimuth angle increases 0.25 degree between successive ions.

Saving, Loading, and Merging Ion Group Definitions

The top four objects on the Group Definition Screen are used for saving, loading, and merging ion group definition **.FLY** files. *Note:* In addition to the definitions of up to 25 ion groups, the **.FLY** files *also* retain the selected coordinate system to use (*from the* **Coordinate System** *selector*).

The Current .FLY File Name

The first object contains the name of the **Current .FLY** file (*if any*). In Figure 8-2, the name is **TEST3D.FLY**.

The Load .FLY Button

The **Load .FLY** button accesses the GUI File Manager to load an ion group definition file. The name of the file loaded will be displayed as the **Current .FLY** file.

The Save Button

The **Save** button accesses the GUI File Manager to save the current collection of ion group definitions as a **.FLY** file. The name of the file saved will be displayed as the **Current .FLY** file.

The Merge Button

The **Merge** button accesses the GUI File Manager to select a **.FLY** file to merge with the current ion group definitions. *Note:* The groups defined in the selected merge file will be ***added to the end*** of any groups currently defined. ***Only*** as many groups will be added as will fit into the ***maximum 25 group limit*** of a **.FLY** file.

Trajectory Group Selection and Group Editing Buttons

A panel object and nine buttons are provided for editing functions involving an ***entire*** ion group.

The Trajectory Group Panel

The **Trajectory Group** panel displays/selects the number of the currently displayed ion group. There are *two* numbers on this panel object. The ***leftmost*** is the number of the ***currently selected ion group***. This number can be changed to select the desired ion group to display. The ***rightmost*** number is the total number of ion groups currently defined (*from 1 to 25*).

The Use Electrons Button

The **Use Electrons** button makes all ions in the ***currently selected ion group*** have the rest mass and charge of an ***electron***.

The Use Protons Button

The **Use Protons** button makes all ions in the ***currently selected ion group*** have the rest mass and charge of a ***proton***.

The Default Grp Button

The **Default Grp** button changes the group definition of the ***currently selected ion group*** to a SIMION default group definition.

The Restore Grp Button

The **Restore Grp** button restores the group definition of the ***currently selected ion group*** to what it was when the group was most recently selected via the **Trajectory Group** panel.

The Cut Grp Button

The **Cut Grp** button removes the ***currently selected ion group*** from the ion group definition list and places a copy of it in the ion group paste buffer. ***Note: SIMION will not allow you to cut the last group.***

The Copy Grp Button

The **Copy Grp** button puts a copy of the ***currently selected ion group*** in the ion group paste buffer.

The Paste Button

The **Paste Grp** button inserts a copy of the ion group paste buffer (*one group definition*) immediately after the ***currently selected ion group. SIMION will refuse to paste a group when 25 groups are already in the group definition list.***

The Swap Up and Swap Dn Button

The **Swap Up** and **Swap Dn** buttons are used to move the ***currently selected ion group*** up (*towards ion group one*) or down (*towards the last ion group*) within the ion group list. ***Ions are flown individually in group order.***

Parameters of Selected Ion Group

Below the ion group editing buttons are the parameters of the ***currently selected ion group***. These parameters can be viewed and edited via their panel objects. The following is a discussion of each parameter.

The Number of Ions Panel

The **Number of Ions** panel is used to specify the number of ions in the ***currently selected ion group***. The number must be one or greater. ***Warning: Ions take memory***. If you request a ***huge*** number of ions SIMION may be forced to virtual (*or refuse to fly the ions*).

The Color Panel

The **Color** panel is used to specify the color for ***all ions*** in the ***currently selected ion group***. Any of SIMION's 16 colors (*color index 0-15*) can be selected.

Note: Selecting a color of 15 will make the ion invisible. This is useful if you are using charge repulsion on a group of ions and only want to see the trajectories of one or more selected ions to keep down the clutter.

If you want ions to have different colors you must define them in different groups.

The CWF Panel

The **CWF** panel is used to specify the ***charge weighting factor*** to use for ***all ions*** in the ***currently selected ion group***. As discussed above, this factor is only used to help in charge repulsion situations. The default value is 1.0.

The Delta TOB Panel

The **Delta TOB** panel is used to set the ***change*** of time of birth between successive ions in the ***currently selected ion group***. The first ion of each group is ***always*** born at time zero. If the ***Delta TOB*** panel value is greater than zero, then each successive ion's birth is delayed by that amount in microseconds.

The First Ion's Mass and Delta Mass Panels

These two panels display and set the ***rest*** masses for ***all ions*** in the ***currently selected ion group*** (***in unified atomic mass units***). The first ion in the group has the mass of

the **First Ion's Mass** panel. The mass of each successive ion has a **Delta Mass** panel increment applied.

The First Charge and Delta Charge Panels

These two panels display and set the electrostatic charge for ***all ions*** in the ***currently selected ion group*** (*in elementary charges*). The first ion in the group has the charge of the **First Charge** panel. The charge of each successive ion has a **Delta Charge** panel increment applied.

The First X, First Y, First Z, Delta X, Delta Y, and Delta Z Panels

These six panels display and set the starting positions of ***all ions*** in the ***currently selected ion group***. The units and locations depend on the current setting of the ***Coordinate System*** selector (*as described above*). The first ion in the group has the position of **First X, First Y,** and **First Z** panels. The position of each successive ion has the **Delta X, Delta Y,** and **Delta Z** panel increments applied.

The First Az Angle, First El Angle, Delta Az, and Delta El

These four panels display and set the starting flight direction of ***all ions*** in the ***currently selected ion group***. The units of these angles are in degrees relative to the current setting of the **Coordinate System** selector (*as described above*). The first ion in the group has the orientation of **First Az Angle** and **First El Angle** panels. The orientation of each successive ion has the **Delta Az** and **Delta El** panel increments applied.

The First Ion's KE and Delta KE Panels

These two panels display and set the starting kinetic energy of ***all ions*** in the ***currently selected ion group***. The units of kinetic energy are electron volts (*eV*). The first ion in the group has the kinetic energy of the **First Ion's KE** panel. The kinetic energy of each successive ion has a **Delta KE** panel increment applied.

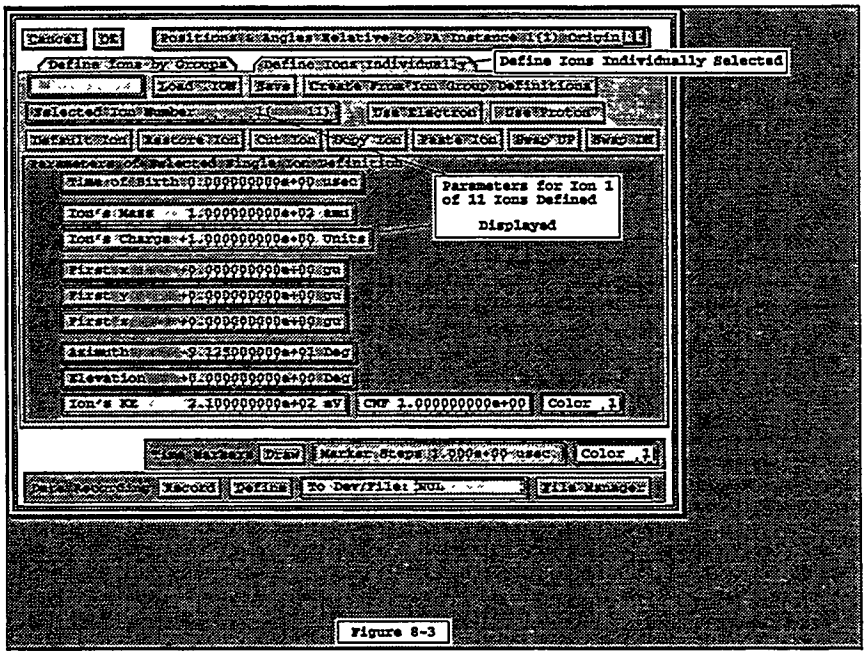


Figure 8-3

Defining Ions Individually

Clicking **Define Ions Individually** tab on the Ion Definition Screen accesses the Individual Ion Definition Screen (*Figure 8-3 above*). This definition screen allows you to define ions individually. ***This means that each ion's parameters can be totally independent of any other ion's parameters*** (*e.g. color, mass, and etc.*).

The Number of Individual Ion Definitions Allowed

By default, SIMION allows up to ***500*** individually defined ions at one time. If you want to define more than 500 ions you must return to the Main Menu Screen, remove all PAs from RAM, and then adjust the **Allocate Memory for a Maximum of 500 Ions** panel to the desired number of ions. ***Remember not to be too greedy - ions take RAM.***

Saving and Loading Individual Ion Definitions

The three objects at the top left of the Individual Ion Definition Screen are used for saving, loading ion definition **.ION** files. These **.ION** files have an ASCII format to allow you to ***also*** create them outside of SIMION (*see Appendix D*). Moreover, SIMION's data recording feature (*discussed later in this chapter*) can create **.ION** format files too.

Note: Unlike .FLY files, .ION files just contain ion definitions. Thus, you will probably need to set the **Coordinate System** panel to the desired coordinate system *before* flying ions loaded from an **.ION** file.

The Current .ION File Name

The first object contains the name of the **Current .ION** file (*if any*).

The Load .ION Button

The **Load .ION** button accesses the GUI File Manager to load an individual ion definition file. The name of the file loaded will be displayed as the **Current .ION** file.

The Save Button

The **Save** button accesses the GUI File Manager to save the current individual ion definitions as a **.ION** file. The name of the file saved will be displayed as the **Current .ION** file.

Converting Ion Group Definitions to Individual Ion Definitions

The **Create From Ion Group Definitions** button is provided to allow you to use the current ion group definitions (***all currently defined groups***) to create the a list of individual ion definitions. The ions thus created replace any that were in the individual ion definition list. If there are more ions than can be currently defined individually (*500 is the default*), SIMION will warn you and only transfer the number of ions that will fit.

Ion Selection Panel and Editing Buttons

A panel object and nine buttons are provided for editing functions involving an entire ion definition.

The Selected Ion Number Panel

The **Selected Ion Number** panel displays/selects the number of the currently displayed ion. There are ***two*** numbers on this panel object. The ***leftmost*** is the number of the ***currently selected ion***. This number can be changed to select the desired ion to display. The ***rightmost*** number is the total number of individual ions currently defined (*from 1 to 500 or more*).

The Use Electron Button

The **Use Electron** button makes the ***currently selected ion*** have the rest mass and charge of an ***electron.***

The Use Proton Button

The **Use Proton** button makes the ***currently selected ion*** have the rest mass and charge of a ***proton.***

The Default Ion Button

The **Default Ion** button changes the definition of the ***currently selected ion*** to a SIMION default ion definition.

The Restore Ion Button

The **Restore Ion** button restore the definition of the ***currently selected ion*** to what it was when the ion was most recently selected via the **Selected Ion Number** panel.

The Cut Ion Button

The **Cut Ion** button removes the ***currently selected ion*** from the individual ion definition list and places a copy of it in the ion paste buffer. ***Note: SIMION will not allow you to cut the last ion.***

The Copy Ion Button

The **Copy Ion** button puts a copy of the ***currently selected ion*** in the ion paste buffer.

The Paste Button

The **Paste Ion** button inserts a copy of the ion paste buffer (*one ion*) immediately after the ***currently selected ion. SIMION will refuse to paste an ion when the maximum number of ions*** (*typically 500*) ***are already in the individual ion definition list.***

The Swap Up and Swap Dn Button

The **Swap Up** and **Swap Dn** buttons are used to move the ***currently selected ion*** up (*towards ion one*) or down (*towards the last ion*) within the individual ion list. ***Ions are flown individually in ion definition order.***

Parameters of Selected Single Ion Definition

Eleven panel objects are provided to display and edit the definition of the currently selected ion (*via the* **Selected Ion Number** *panel*). The following discusses the use of each of these panels.

The Time of Birth Panel

The **Time of Birth** panel displays the time of birth for the ***currently selected ion*** in microseconds from the start of the **Fly'm** clock.

The Ion's Mass Panel

The **Ion's Mass** panel displays the ***rest*** mass of the ***currently selected ion*** in ***unified*** atomic mass units (*amu*).

The Ion's Charge Panel

The **Ion's Charge** panel displays the electrostatic charge of the **currently selected ion** in elementary charge units.

The First X, Y, and Z Panels

The three First X, Y, and Z panels display and set the starting position of the ***currently selected ion***. The units and locations depend on the current setting of the *Coordinate System* selector (*as described above*).

The Azimuth and Elevation Panels

These two panels display and set the starting flight direction of the ***currently selected ion***. The units of these angles are in degrees relative to the current setting of the **Coordinate System** selector (*as described above*).

The Ion's KE Panel

The **Ion's KE** panel displays and sets the starting kinetic energy of the ***currently selected ion***. The units of kinetic energy are electron volts (*eV*).

The CWF Panel

The **CWF** panel is used to specify the ***charge weighting factor*** to use for the ***currently selected ion***. As discussed above, this factor is only used to help in charge repulsion situations. The default value is 1.0.

The Color Panel

The **Color** panel is used to specify the color for the ***currently selected ion***. Any of SIMION's 16 colors (*color index 0-15*) can be selected.

Note: Selecting a color of 15 will make the ion invisible. This is useful if you are using charge repulsion on a group of ions and only want to see the trajectories of one or more selected ions to keep down the clutter.

Defining Ions Outside SIMION

You have the option of creating your own **.ION** files outside SIMION. These files are ASCII files (*see Appendix D for their format requirements*). The **.ION** files can be generated either manually (*by you with an editor*) or automatically (*by a program you created*).

Defining Ions Via User Programs

SIMION allows you to redefine ions within the **Initialize** and **Other_Actions** program segments of user programs. ***See Appendix I for more information.***

This **Initialize** program segment allows you to program SIMION to ***redefine*** the ions for you. The **_RANDOM** demo serves as an example of having user programs randomize your ions for you. The **GROUP.PRG** file in the **_TRAP** demo also provides another example of random ion generation. The best way to make use of this feature is to study how they work and either copy these program segments into your own user program or create one yourself that does just what you want.

The **Other_Actions** program segment allows you to change the mass and/or charge of the ion while it is flying. This is useful if you want to simulate neutralization, fragmentation, or other interesting effects.

Flying Ions

Now that we have defined the ions we want to fly we need to learn how to fly them. SIMION has a collection of ion flying controls on the Normal Controls Screen (*Figure 8-4 below*).

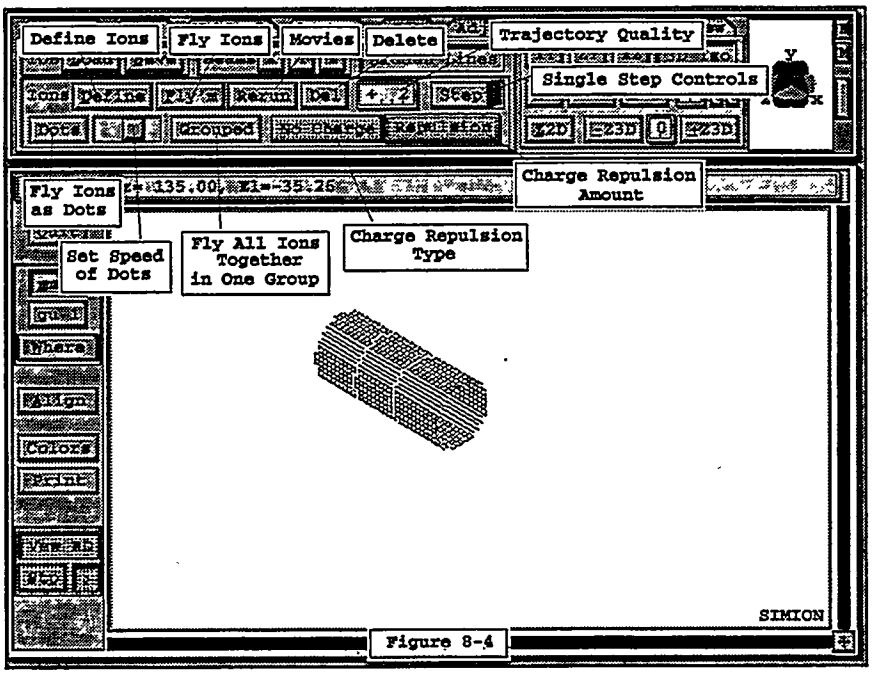


Figure 8-4

Background Information

The following is provided to help you better understand the ion flying controls and how to effectively use them.

What Can be Done While Ions are Flying

The user interface has been designed to be as interactive as possible. This means that most anything (*e.g. changing views, moving instances, aligning on an instance, and etc.*) can be done ***even*** while ions are flying. This allows you the maximum freedom to change things in a natural way just as if SIMION was a real optics bench that you could fiddle with at any time.

Ions are Flown Individually or in Groups

SIMION allows you to fly each ion defined individually (*one at a time*) or as a group (*all at once*). Generally, flying ions individually is faster. However, flying ions together as a group offers additional options.

SIMION's Clock

SIMION's clock records the ion's time of flight in microseconds. This clock is started (*reset to zero*) each time an ion is flown ***individually*** or when the flying of an entire group starts.

If an ion has a delayed time-of-birth ***and*** is flying individually, SIMION will automatically advance the clock to the ion's time-of-birth and start the ion flying.

How Ion Trajectories are Computed

Basically, ion trajectory calculations involve an integration process. SIMION makes use of a highly modified 4th order Runge-Kutta technique (*for details see Appendix E*). In general, SIMION varies the time step used so that an ion moves a fixed amount of distance in a single time step. This distance is normally one grid unit within the current instance.

However, there are times when crossing a boundary, high field curvature, or velocity reversals require smaller time steps to conserve energy properly. SIMION normally checks for velocity reversals and high field curvature (*e.g. grid transitions*) and automatically reduces its time steps via a binary boundary approach method. This strategy provides a robust method for automatically putting the time steps where needed without wasting time steps in simple regions (*e.g. linear fields*).

You have the option of controlling the integration methods used. This allows you to speed calculations in problems that don't require as much protection. Remember, it is ***your*** responsibility to verify that such compromises are justified (*e.g. the trajectories are not significantly different*).

The Recording of Trajectories

SIMION normally saves the trajectory position data of the ion trajectories in a single ***Fly'm*** to a temporary trajectory plot file. This allows you to view these trajectories ***after*** the end of the flight in different views (*e.g. WB or PE*) and in different ways (*e.g. lines or dots*). ***However, trajectory data can take an enormous amount of disk space***. This is particularly true if your ions are wandering about in an ion trap.

You have the option of turning off (*and erasing the existing trajectory plot file*) the recording of trajectory image data via the **Rerun** button (*see below*).

Viewing Ions as Lines or Dots

Ions are normally displayed as lines. However, it can sometimes be very useful to view them as moving dots (*especially if they are flying in groups*). SIMION allows you to view ions either way via the **Dots** button (*see below*).

Discussion of Individual Ion Flying Controls

The following material gives detailed information about each ion flying control object on the Normal Controls Screen (*Figure 8-4 above*).

The Define Button

The **Define** button allows access to the Ion Definition Screens (*discussed above*). It also allows access to time marker controls as well as SIMION's data recording controls.

The Fly'm Button

The **Fly'm** button starts the currently defined ions flying when ***depressed.*** Clicking the **Fly'm** button again or hitting the **Esc** key ***stops*** the current ion flying process.

Be sure to select the desired flying options (*e.g. Grouped and charge repulsion*) ***before*** starting to fly ions. Selecting this type of option while ions are flying will automatically terminate the current flight.

If any of the instances have user programs active SIMION will automatically compile them when you depress the **Fly'm** button. If there are any adjustable variables defined in these user programs, SIMION will display a window that allows you to adjust their value(*s*) ***before*** the ions start flying.

The Rerun Button

The **Rerun** button can be clicked during a **Fly'm** and ***will not*** cause an automatic flight termination. ***Note: The* Rerun *button can also be controlled by user programs*** (*handy*). This button has three important functions:

1. **First,** it tells SIMION that you want to keep rerunning the trajectories when the **Rerun** button is ***depressed*** at the ***end*** of a **Fly'm.** This feature is handy for movie style animation (*SIMION just keeps flying ions*) that allows you to change voltages or adjust instance positions without having to periodically start the ions flying again.
2. **Second,** when the **Rerun** button is depressed it automatically ***erases*** the current trajectory image file (*if any*) and ***stops*** the recording of trajectory flight image data. This is a handy way to avoid swamping your hard drive with ion image data (*during long ion flights: e.g. in traps*). If you click the **Rerun** button (*raising it*), SIMION will automatically resume saving trajectory image data.
3. **Third,** if data recording to a file is active, SIMION will take note of the status of the **Rerun** button ***at the instant*** when the **Fly'm** button is clicked to ***start*** flying ions. If the **Rerun** button is depressed (*at that instant*) then no recorded data will be sent to the user designated data record file during the ***entire*** **Fly'm.** However, recorded data will always appear on the Data Monitoring Screen (*if and when displayed*).

The Del Button

The **Del** button ***terminates*** any active ion flying and ***erases*** the current trajectory image file (*if any*). This is a good way to remove pesky trajectory images from your field of view.

The Trajectory Quality Control Panel

The **Trajectory Quality Control** panel (*showing the value of 2 in Figure 8-1*) is used to control the methods SIMION uses to compute ion trajectories (*see Appendix E*). The default value of 2 turns on ***all*** the computation quality features and provides good trajectory estimates (*at possibly the cost of taking longer*). The value selected for this control panel sets the computational quality in the following manner:

Quality = 0 When the **Trajectory Quality Control** panel is set to zero, SIMION computes ion trajectories at minimum quality (*maximum speed*). Time steps are ***fixed*** for integration steps of one grid unit (*in current instance*) per time step. ***Velocity reversal detection, edge detection, and binary boundary approach are all turned off.*** This level of quality will give totally satisfactory trajectories for many applications.

Quality < 0 When the **Trajectory Quality Control** panel is set to a value less than zero (*e.g. -2*), time steps are ***fixed*** for integration steps of ***1/(1- Quality)*** grid units per time step. Thus a value of -2 would result in time steps of 1/3 grid unit per time step. ***Velocity reversal detection, edge detection, and binary boundary approach are all turned off.***

Quality > 0 When the **Trajectory Quality Control** panel is set to a value greater than zero (*e.g. 2 - the default*), ***Velocity reversal detection, edge detection, and binary boundary approach are all turned on.*** As the quality value increases the edge detection criteria (*CV*) becomes more sensitive. For qualities from 1 to 100 the ***nominal*** time step is set for one grid unit per time step. For qualities above 100 the ***nominal*** time step is set for ***1/(Quality -100)*** grid units per time step.

The Step and Single Step Buttons

Just to the right of the **Trajectory Quality Control** panel are the **Step** and **Single Step** buttons. Their function is to halt and single step the trajectory calculations.

The Step Button

The **Step** button, when depressed, halts trajectory calculations. When the **Step** button is raised trajectory calculations resume. One use for this button would be to halt a calculation, depress then raise the **Rerun** button to erase any trajectory images, raise the **Step** button for a moment to capture a trajectory section, and then depress it again to halt execution. The trajectory image section then could then be viewed and printed (*if desired*).

The Single Step Button

When the **Step** button is depressed, the button to its immediate right will automatically be raised. This is called the **Single Step** button. Each time you click the **Single Step** button SIMION will execute a single time step. This is useful for close examination of ion movement from time step to time step.

Another use would be if you are flying ions as dots in a trap and you want to print a picture of the ions at a point in time. Click the **Step** button to halt, depress then raise the **Rerun** button to erase any trajectory images (*or just raise it if it was already depressed*), adjust **Trajectory Quality Control** to -500 (*for the smallest time step*), click the **Single Step** button *once* to record a single time step, and then print the desired view.

The Dots Button and the Dots Speed Slider

At the beginning of the next line is a **Dots** button and a **Dots Speed** slider.

The Dots Button

When the **Dots** button is depressed, ions will appear as moving dots during trajectory calculations. This is particularly useful when flying ions in ***groups*** to see how the groups of ions behave (*e.g. in time-of-flight instruments or traps*).

If the **Rerun** button is ***not*** depressed, ions will also appear as dots during display redrawing (*because their images have been recorded*). Depress **Dots** and ions will be redrawn as dots. Click **Dots** again and ions will be redrawn as lines.

If you print trajectories with **Dots** active an ion dot will normally be printed at each time step (*this could be a lot of dots*).

One way to avoid all this clutter is to make use of a time markers trick. Use the **Define** button to access the Ion Definition Screen. Now depress the time marker **Draw** button, set the desired time step (*in microseconds*), and set the marker ***color*** to ***15*** (*the trick*). SIMION will now use each ion's own color for its time markers on any future **Fly'm**s. Further, if **Dots** are active (*button depressed*), ***only*** the time markers (*and **not** the normal dots*) will be sent to the printer (*useful*).

The Dots Speed Slider

The **Dots Speed** slider is used to insert a bit of time delay between time steps when ions are flown as dots. The ***leftmost*** setting inserts a lot of delay (*slowest dots*) while the ***rightmost*** setting inserts no delay (*fastest dots*).

This slider ***is*** accessible during normal trajectory calculations, but ***is not*** accessible when the screen is being redrawn (*e.g. new view or after Fly'm ends*). If you are re-drawing and the dot speed is too slow, hit the **Esc** key to halt drawing, adjust the slider for a faster speed (*move toward right*), move the cursor into the view area, and hit the **Enter** key to force a another view re-draw.

The Grouped Button

The **Grouped** button flies all the ions together in a single group when depressed. This is somewhat slower than flying ions individually, because the slowest ion (*the one with the shortest time step*) is leader of the pack (*or anchor if you prefer*).

This can really slow things down for positive values of **Trajectory Quality**. It is recommended that you always try to use zero or negative values (*fixed steps*) of **Trajectory Quality** to minimize this slow down.

Flying ions in groups is useful to observe their relative movement along the flight line. ***Ions must be flown in groups if charge repulsion is to be used.***

The Charge Repulsion Controls

Access to the charge repulsion controls (*to the right of the* **Grouped** *button*) is blocked unless the **Grouped** button is depressed. These controls are a button that selects the type of charge repulsion used and a panel object that is used to set the amount (*even while ions are flying*).

Grouped ions are normally flown ***without*** charge repulsion (*e.g.* **No Charge** *button displayed*). However, SIMION also supports the ***estimation*** of charge repulsion effects. Note: These computations are useful for determining if you may have charge repulsion problems and where they are occurring. ***It is not intended to model heavily space charged environments accurately***. See Appendix E for more details.

Types of Charge Repulsion Supported

Two basic types of charge repulsion can be modeled (*selected by clicking the* **Repulsion Type** *button and setting the* **Repulsion Amount** *panel*): Ion Cloud and Ion Beam.

Ion Cloud Repulsion

Ion cloud assumes that the defined ions are representative of an actual cloud of ions. Each ion is considered to represent a small cloud of ions (*thus ions are modeled as clouds and not point charges*). The mutual attractions or repulsion between ions (*each mimicking a small ion cloud*) are included in trajectory calculations when ion cloud repulsion is active. The objective is to use a smaller number of grouped ions to simulate a larger group of ions. Two approaches are supported: Coulombic and factor repulsion.

Coulombic repulsion is activated when **Coul Repl** is displayed on the **Repulsion Type** button. SIMION performs coulombic repulsion by dividing a total charge amount in coulombs (*set by user on* **Repulsion Amount** *panel)* among the ions defined (*based on their charge and Charge Weighting Factor*).

Factor repulsion is activated when **Fact Repl** is displayed on the **Repulsion Type** button. SIMION performs factor repulsion by applying a charge multiplication factor (*set by user on* **Repulsion Amount** *panel*) to each ion defined (*Charge Weighting Factor included too*). In this manner one ion can be made to represent 1,000 or more ions in terms of the repulsion effects it has on other ions.

Both coulombic and factor repulsion use ***time coherent integration***. This means all the grouped ions are ***kept together in time***.

Ion Beam Repulsion

The second type of repulsion is beam repulsion. It is selected when **Beam Repl** is displayed on the **Repulsion Type** button. Beam repulsion is modeled by considering ions as lines (*actually - small cylinders*) of charge by using their apportioned current and velocity to compute a charge per unit length. Thus each ion repels or attracts other ions as if it represented a small cloud of line charge (*radius of line is finite*). In order to accomplish this, SIMION apportions the ***ion beam current*** among the group of ions according to their charge and Charge Weighting Factors as above.

Beam repulsion requires a ***special*** type of group flying called ***space coherent integration***. This is due to the fact that ions will not fly at the same speed, but we really need all ions to remain in a plane normal to the axis ion's trajectory so that line repulsion retains some meaning. SIMION accomplishes this by assuming that the

first ion of the *first* group (*e.g. first ion defined*) is the axis ion (*at the center of the pack*). ***It is your responsibility to insure this.*** It then integrates the trajectory of this ion step by step. At each time step it calculates the normal plane to this ion. All other ions are given the appropriate time steps to keep them roughly on the surface of the plane. ***This means that each ion is really represented at a different time.*** However, since this is a beam of ions, one could argue that what we are seeing is the particular ion in the line of ions that currently satisfies the plane intersection requirement.

How to Use Charge Weighting Factors

When a family of ions are initialized you have the option of defining a geometrical weighting factor for each ion group (*or each ion in individual ion definitions*). This is used for apportioning charge around differently when some ions may represent larger initial areas or volumes than others. The formula below shows how CWF effects weighting.

Effective Weight = abs(ion's_charge) * CWF / sum_of_all_ions(abs(ion_charge) * CWF)

How Good is SIMION's Charge Repulsion?

I'm not confident that these calculations are particularly accurate. However, charge repulsion onset levels appear realistic. Moreover, increasing the number of ions used to model the effects doesn't seem to have a dramatic impact on the apparent results (*a small group of ions seems to work pretty well*). ***Beware, this approach doesn't model the space charge effects in ion source regions.***

Flying with Charge Repulsion

The best way to fly with charge repulsion is to use **Rerun** mode. This allows you to keep changing the charge, factor, or current while the ions are flying and have them automatically re-fly with the new settings.

Before clicking **Fly'm**, select **Grouped** and click the **Repulsion Type** button (*to the right*) to select the desired form of charge repulsion: **No Charge** (*repulsion off*), **Beam Repl**, **Coul Repl**, or **Fact Repl**. Now set the current, charge, or factor depending on the charge repulsion mode chosen by using the **Repulsion Amount** panel. ***Click the*** **Fly'm** ***button and they're off.*** Note: you may now adjust the amount of repulsion (*current, charge, or factor*) as the ions fly and if **Rerun** is active see the full effects on the next fly cycle. ***Have fun, but don't be too quick to believe what you see.***

Drawing Time Markers

Near the bottom of the Ion Definition Screen (*Figure 8-5 below*) there is a group of controls for generating time markers:

The Draw Button

When the **Draw** button is depressed time markers will be drawn on ion trajectories when ions are flown.

The Marker Steps Panel

The **Marker Steps** panel is used to set the time steps for markers in microseconds.

The Marker Color Panel

The **Marker Color** panel sets the color of all markers (*including time markers*). Its function is equivalent the that used to set ion colors.

Trick: If you use the color 15 (*White*) for markers, SIMION will actually use the color of the ion for the marker color. Moreover, if you ***print*** trajectories (*having markers drawn with color 15*) with the **Dots** button depressed, SIMION will ***only*** print the ***actual*** markers and not the flying dots themselves (*useful*).

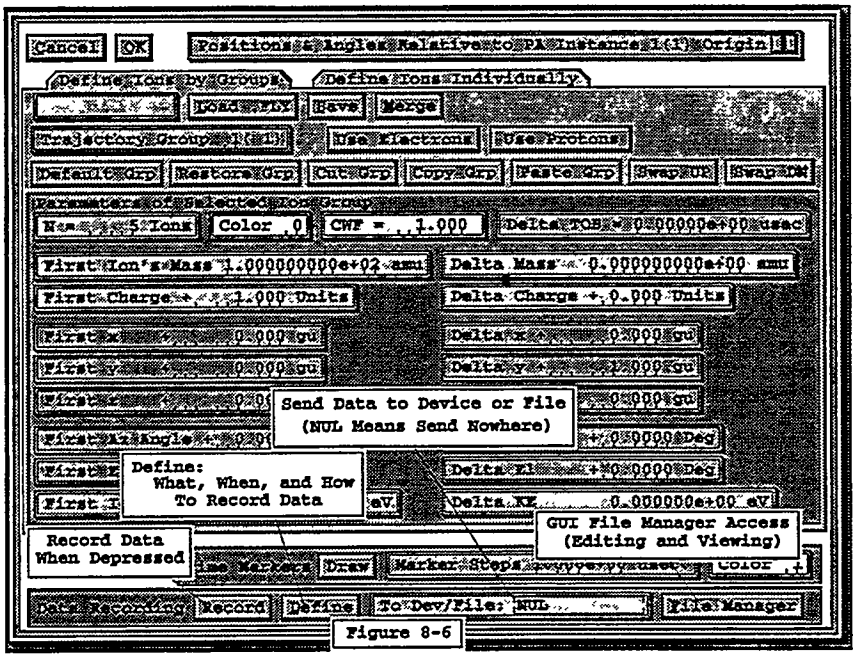


Figure 8-6

Time Markers are Considered Events

Each time marker triggers a data recording marker event. Thus time markers can be used as triggering events for data recording (*discussed below*).

Data Recording

SIMION 6.0 has a powerful data recording option that allows you to record/view specific data, occurring at specified events, in a designated format. Access to these options is via the Ion Definition Screen (*Figure 8-6 above*). At the bottom of this screen are controls for data recording:

The Record Button

If you depress the **Record** button data will be recorded (***and possibly sent to a device or file***) whenever ions are flown. Note: Data is ***never*** sent to a device or file (***but it can be viewed***) when the **Rerun** button is ***depressed prior*** to clicking the **Fly'm** button.

The To Dev/File Ioline

The **Dev/File** ioline object allows you to specify the destination device (*e.g. PRN*) ***or*** file for recorded data. Data is ***always appended*** to the target device/file. ***This means that the recorded data is added to instead of replacing any current data in the target device/file.***

If you output to a device (*e.g.* **PRN, LPT1, COM1**, *and etc.*) ***do not*** put a colon (**:**) after the device name (*e.g.* **PRN:**).

Note: The default device is **NUL**. This means data is not actually sent to a device/file but still can be viewed on the Data Monitoring Screen.

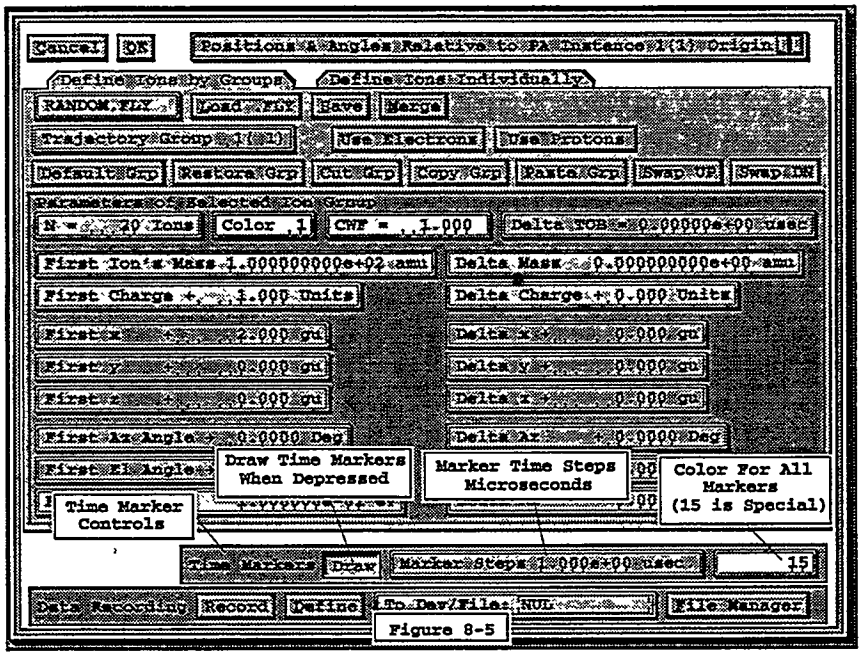


Figure 8-5

The File Manager Button

This button accesses the GUI File Manager. This is useful if you want to inspect the contents of a data recording file with an editor like **EDY** (*supplied with SIMION*). To edit a data recording file click the **File Manager** button, depress the button with the name of the file you want to edit (*with a single mouse button*), and click the **Edit** button on the file manager.

The Define Button

If you click the **Define** button a Data Recording Definition Screen will appear (*Figure 8-7 above*). On it you can define what to record, when to record, and how to record.

The Cancel and OK Buttons

The **Cancel** button exits the Data Recording Definition Screen and ***restores*** the data recording definitions that were active at the time the **Define** button was clicked.

The **OK** button exits the Data Recording Definition Screen and ***keeps*** the current data recording definitions for use in any future **Fly'm**.

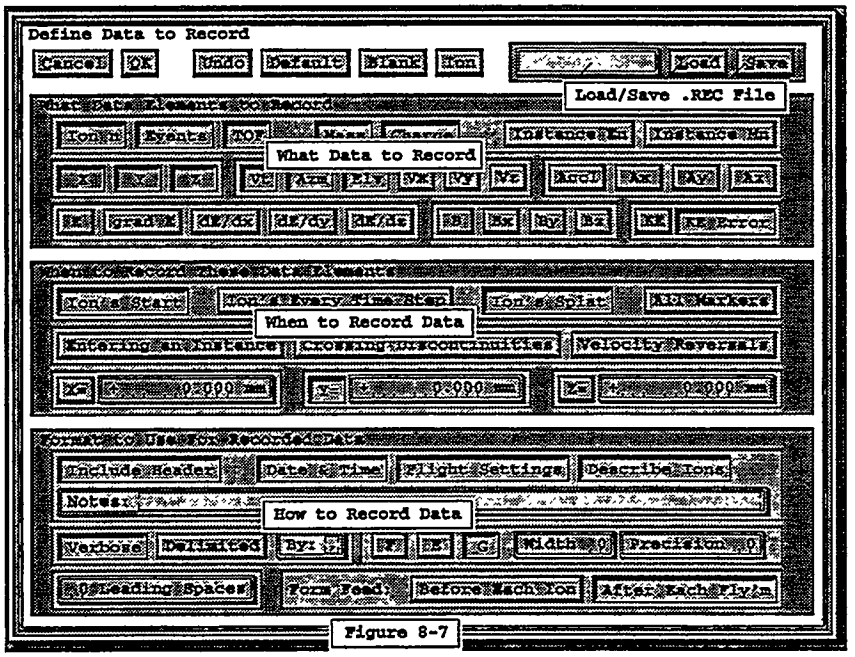


Figure 8-7

Special Editing Buttons

There is a row of special editing buttons at the top center of the Data Recording Definition Screen.

The Undo Button

The **Undo** button restores the data definitions to what they were when the **Define** button was last clicked.

The Defaults Button

The **Defaults** button sets the SIMION default data recording definitions.

The Blank Button

The **Blank** button erases all data recording definitions. This is useful if you want to insure that all options are off ***before*** creating a new set of recording definitions.

The Ion Button

The **Ion** button defines a data recording format that can be saved as an **.ION** file (*it is your responsibility to name it properly - e.g. with an* **.ION** *file extentsion*). This is useful if you want to save the termination conditions of your ions for perhaps flying in another workbench (*via the use of hugging and chalking methods*).

Saving and Loading .REC Files

Three objects are provided to allow you to save and load data recording definitions in **.REC** files.

The .REC Filename Ioline

The **.REC Filename** ioline is provided to display the name of the last **.REC** file saved or loaded (*if any*).

The Load Button

The **Load** button is used to load data recording definitions saved in a **.REC** file. The GUI File Manager will automatically assist you in file selection.

The Save Button

The **Save** button is used to save the current data recording definitions in a **.REC** file. The GUI File Manager will automatically assist you.

Selecting What to Record

A collection of buttons are provided to let you select the data elements to record. A data element is selected by ***depressing*** its button. ***The specific help for each button explains the data element and gives its units of measure.***

The Ion n Button

The **Ion n** button requests the recording of the ion's number.

The Events Button

The **Events** button requests that the event (*or events*) that caused the data to be recorded be recorded too. In **Verbose** format a description will be provided. In **Delimited** format a number will be provided that is the sum of the following ***causing*** event(s) related bit flags:

1	Ion Created
2	Next Time Step
4	Hit Electrode
8	Dead in Water
16	Outside Workbench
32	Ion Killed
64	Marker
128	Entering an Instance
256	Field Discontinuity (*CV Limit*)
512	Velocity Reversal
1024	Crossed X = Value Plane
2048	Crossed Y = Value Plane
4096	Crossed Z = Value Plane

The TOF Button

The **TOF** button requests that the time-of-flight (*time since* **Fly'm** *clock started*) in microseconds be recorded.

The Mass Button

The **Mass** button requests that the current ***rest*** mass of the ion in ***unified*** atomic mass units be recorded.

The Charge Button

The **Charge** button requests that the current charge of the ion in elementary charge units be recorded.

The Instance En Button

The **Instance En** button requests that the number of the electrostatic instance (*if any*) that the ion is ***currently*** within be recorded. ***If the ion is not currently in an electrostatic instance a value of zero will be recorded.***

The Instance Mn Button

The **Instance Mn** button requests that the number of the magnetic instance (*if any*) that the ion is ***currently*** within be recorded. ***If the ion is not currently in a magnetic instance a value of zero will be recorded.***

The X, Y, and Z Buttons

The **X**, **Y**, and **Z** buttons request that the current x, y, and z positions in millimeters relative to the ***currently aligned workbench coordinates*** be recorded.

The Vt Button

The **Vt** button requests that the ion's current speed in millimeters per microsecond be recorded.

The Azm and Elv Buttons

The **Azm** and **Elv** buttons request that the ion's current velocity azimuth and/or elevation orientation angles (*in degrees*) relative to the ***currently aligned workbench coordinates*** be recorded.

The Vx, Vy, and Vz Buttons

The **Vx**, **Vy**, and **Vz** buttons request that the ion's velocity components (*in millimeters per microsecond*) relative to the ***currently aligned workbench coordinates*** be recorded.

The Accl Button

The **Accl** button requests that the ion's current total acceleration in millimeters per microsecond squared be recorded.

The Ax, Ay, and Az Buttons

The **Ax**, **Ay**, and **Az** buttons request that the ion's acceleration components (*in millimeters per microsecond squared*) relative to the ***currently aligned workbench coordinates*** be recorded.

The E Button

The **E** button requests that the electrostatic potential (*in volts*) at the ion's current location be recorded.

The grad E Button

The **grad E** button request that the total electrostatic gradient (*in volts per millimeter*) at the ion's current location be recorded.

The dE/dx, dE/dy, and dE/dz Buttons

The **dE/dx**, **dE/dy**, and **dE/dz** buttons request that the components of electrostatic gradient (*in volts per millimeter*) relative to the ***currently aligned workbench coordinates*** at the ion's current location be recorded.

The B button

The **B** button requests the total magnetic field (*in gauss*) at the ion's current location be recorded.

The Bx, By, and Bz Button

The **Bx**, **By**, and **Bz** buttons request the magnetic field components (*in gauss*) relative to the ***currently aligned workbench coordinates*** at the ion's current location be recorded.

The KE and KE Error Buttons

The **KE** button request that the ion's current kinetic energy (*in electron volts*) be recorded.

The **KE Error** button requests that ion's current kinetic energy error (*in electron volts*) be recorded. This error is based on conservation of energy assumptions using the starting and current potential and kinetic energies of the ion. ***Note: This parameter has no meaning if time-changing fields are being applied via user programs.***

Selecting When to Record

The middle collection of buttons designate events that ***will*** trigger a data record when depressed. SIMION will place a dot at the point in the trajectory where the data recording event occurred. The color of the dots will be the color selected for time markers. ***Note: Not all events will be flagged at any computational quality.***

The Ion's Start Button

The **Ion's Start** button designates the start of an ion's flight (*time-of-birth*) as a data recording event when depressed.

The Ion's Every Time Step Button

The **Ion's Every Time Step** button designates ***every*** time step as a data recording event when depressed. This is useful for detailed tracking and troubleshooting. ***Warning: This can generate an enormous number of records.***

The Ion's Splat Button

The **Ion's Splat** button designates the ion's death (*by whatever means*) as a data recording event when depressed. ***Note: Trajectory Qualities greater than zero*** (*e.g. the default of* 2) ***are required for good event accuracy.***

The All Markers Button

The **All Markers** button designates all ***time*** markers and ***user program generated*** markers as a data recording events when depressed.

The Entering an Instance Button

The **Entering an Instance** button designates a data recording event when an ion ***just*** enters ***any*** instance (*when depressed*). ***Note: Trajectory Qualities greater than zero*** (*e.g. the default of* 2) ***are required for good event accuracy.***

The Crossing Discontinuities Button

The **Crossing Discontinuities** button designates a data recording event when an ion crosses through a binary boundary (*CV detection based*). ***Note: Trajectory Qualities greater than zero*** (*e.g. the default of* 2) ***are required for this event.***

The Velocity Reversals Button

The **Velocity Reversals** button designates a data recording event when an ion reverses a Vx, Vy, or Vz velocity component in ***currently aligned workbench coordinates. Note: Trajectory Qualities greater than zero*** (*e.g. the default of* 2) ***are required for this event.***

The X, Y, and Z = Plane Crossings

You can designate an event when an ion crosses a specified x, y, or z = value plane. These planes are relative to the ***currently aligned workbench coordinates.*** To designate an event plane, click the desired plane's button (**X**, **Y**, *or* **Z**) ***and*** set the desired value (*position*) for the plane in ***currently aligned*** *workbench coordinates.* ***Note: Trajectory Qualities greater than zero*** (*e.g. the default of* 2) ***are required for good event accuracy.***

Selecting the Recording Format

The bottom collection of buttons designate the data recording format. There are two groups of buttons designating: Data Recording Header and Record Format

Data Recording Header Controls

SIMION can optionally supply a Data Recording Header at the beginning of recorded data for each **Fly'm**. The following objects are used to control its output:

The Include Header Button

The ***Include Header*** button Includes a Data Recording Header at the beginning of data recorded for a single **Fly'm** when ***depressed***. The header will include the information requested by the other header control objects.

The Date & Time Button

The date and time of the start of the **Fly'm** will be recorded when this button is depressed.

The Flight Settings Button

The header will include information concerning grouped flying and repulsion settings when this button is depressed.

The Describe Ions Button

The header will include a description of the ions flown when this button is depressed.

The Notes Ioline

The **Notes** ioline object is included to allow you to include one line of your own personal notes in the header. If the line is blank, no notes will be included in the header.

Data Record Format Controls

The next group of control objects designate the record format used for each data record:

The Verbose and Delimited Buttons

The **Verbose** and **Delimited** buttons are used to select ***either*** verbose ***or*** delimited format data records. The verbose format outputs each data item's name and its value (*designed to be easy for you to read*):

mass(100.0) charge(-1)

The delimited format outputs each data item as a number with a designated delimiter used (*e.g.* ,) to separate each data item. ***A delimited data record is always a single line***. Delimited records are useful for importing data into other programs (*e.g. spreadsheets*).

500,22,15.6

The By: Ioline

The **By:** ioline is used to specify the delimiter to be used to separate data items in delimited format output. The *comma* (,) is the default delimiter.

The F, E, and G Buttons

The **F**, **E**, and **G** buttons are used to designate the format to be used for numerical data. ***Note: The user program*** Message ***cmd makes use of these number format settings too***.

The **F** button creates standard C language f format numerical output using designations for width and precision (*-ddddd.ddddd floating point notation*).

The **E** button creates standard C language e format output using designations for width and precision (*-dddd.ddde-ddd scientific notation*).

The **G** button creates either f or e format output depending on which one is more compact.

The Width and Precision Panels

The **Width** and **Precision** panels are used to set width and precision values that follow C language conventions (*e.g. %width.precision f*). Setting both width and precision to zero gives variable widths under program control (*default*).

The Leading Spaces Panel

You also have the option of inserting a fixed number of leading spaces in front of each line output via the **Leading Spaces** panel.

Form Feed Controls

Buttons are also provided to request form feeds ***before*** each ion flies and ***after*** each **Fly'm**.

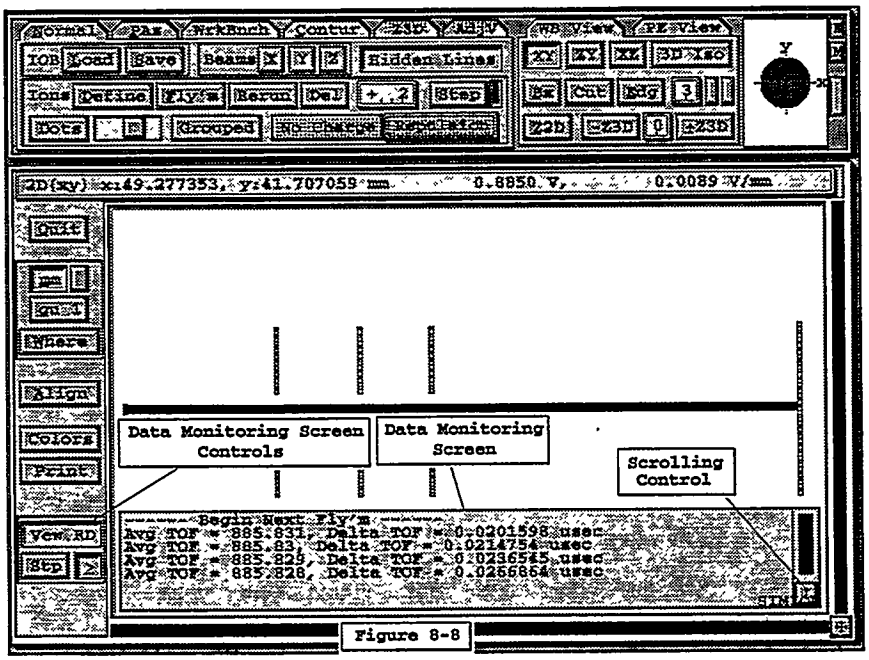


Figure 8-8

Using the Data Monitoring Screen

When data recording is active (*the* **Record** *button is depressed on the Ion Definition Screen*) ***or*** when any ***active*** user programs have **Message** or **R/S** commands you will be allowed access to the Data Monitoring Screen controls on the View Screen (*Figure 8-8 above*).

The View Recorded Data Button

The **View RD** button displays the Data Monitoring Screen in the lower left corner of the view window when **depressed.** ***The Data Monitoring Screen is on by default.*** The **View RD** button can be used to turn the screen off and on.

Pausing at Each Recording Event

In order to facilitate data monitoring, a **STP** button has been provided. When this button is *depressed* the Ion(*s*) will fly to the next recording event and *stop* with the event's data displayed on the Data Monitoring Screen. You can then examine the data at your leisure. To step to the next recording event click the raised '>' button (*to the right of the* **STP** *button*). You may turn **STP** on and off during a flight. ***Note: you may adjust your view, print, or whatever when you are paused.***

Scrolling the Data Monitoring Screen

The Data Monitoring Screen (*visible within the view window when data is being recorded*) has a scroll button (*Figure 8-8*). This button allows you to access the most recent **Fly'm's** last **500** lines of recorded output. ***Access is allowed even while the ions are flying.***

If you scroll with the ***left*** mouse button depressed the data screen will be updated as you scroll. If you scroll with the ***right*** mouse button depressed the data screen will be updated when you release the mouse button.

The view region bar above the scrolling button is normally ***blue*** indicating that ***all*** lines since the start of the last **Fly'm** can be viewed. The view region bar changes to ***red*** when more than **500** lines have been output to the monitor to warn you that only the ***last*** **500** lines can be viewed (*e.g. records have scrolled off the top of the Data Monitoring Screen*).

Flying Ions with User Programs Active

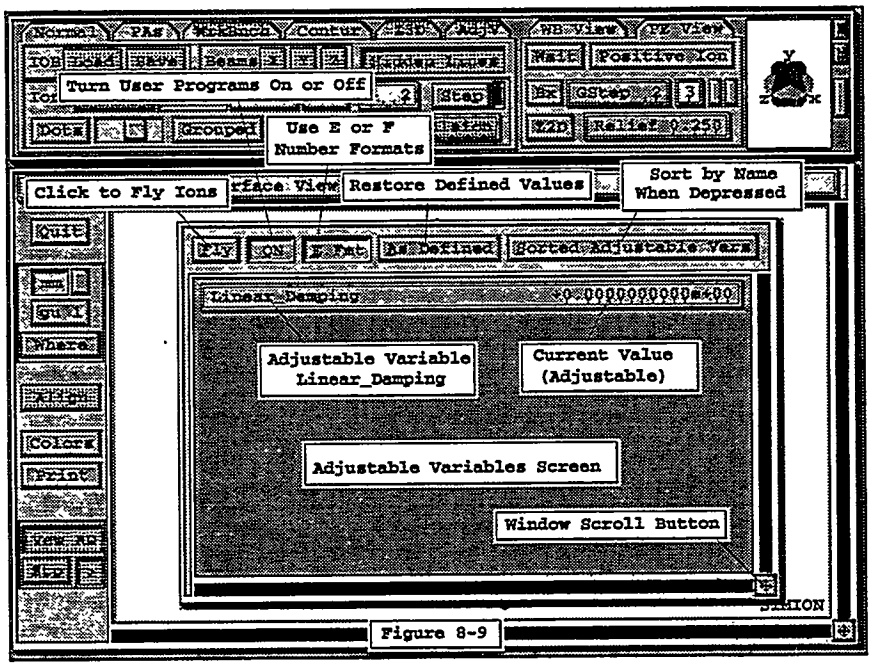
Whenever you click the **Fly'm** button to start ions flying, SIMION quickly checks each instance to see if its referenced potential array has a user program file (*e.g. for* **TEST.PA** *there is a* **TEST.PRG** *file in project directory*). If any potential array has a user program file (**.PRG**), SIMION will ***automatically*** compile its user program segments and use them while flying ions. ***See Appendix I for information on user programs.***

The material presented in this chapter is limited to discussing the special screens you will encounter when flying ions with user programs active. The emphasis will be on how to ***use*** these screens effectively, as ***opposed*** to how to ***write*** user programs.

The Adjustable Variables Screen

If SIMION compiles user programs and finds Adjustable Variables defined therein, it will display an Adjustable Variables Screen just prior to actually flying ions (*Figure 8-9 below*). You can use this screen to change the value of any Adjustable Variable before the ions begin to fly.

Note: Any changes made in Adjustable Variable values will be retained as the proposed value for the next **Fly'm.**



The List of Adjustable Variables

Adjustable Variables are displayed as a list of panel objects in a window. The name of the Adjustable variable is on the left side of its panel object, and the value of the Adjustable Variable is on the right side of its panel object.

You can scroll this window vertically via normal GUI methods if more Adjustable Variables are defined than will fit in the window's area.

The Fly Button

The **Fly** button is clicked after you have adjusted the values of the appropriate Adjustable Variables. If you want to abort the **Fly'm**, just hit the **Esc** key.

The ON/OFF Button

The **ON/OFF** button is used to turn *all* user programs on or off. The default value is **ON**. This button allows you to turn user programs off for the current **Fly'm**. ***It is useful for testing and debugging purposes.***

Note: You must have Adjustable Variables defined to access this screen and the* ON/OFF *button. Hint: Create a dummy Adjustable Variable to force SIMION to display this screen.

The E Fmt Button

The **E Fmt** button displays Adjustable Variable values as *e* format numbers ***when depressed*** and *f* format numbers ***when <u>not</u> depressed.*** Your preference will be remembered throughout the current SIMION session.

The As Defined Button

Clicking the **As Defined** button restores the values of ***all*** Adjustable variables to the initial values defined for them in the user programs.

The Sorted Adjustable Variables Button

When this button is *depressed* the list of Adjustable Variables will be sorted alphabetically. Otherwise, the list of Adjustable Variables will be in the order they were encountered during user program compilation.

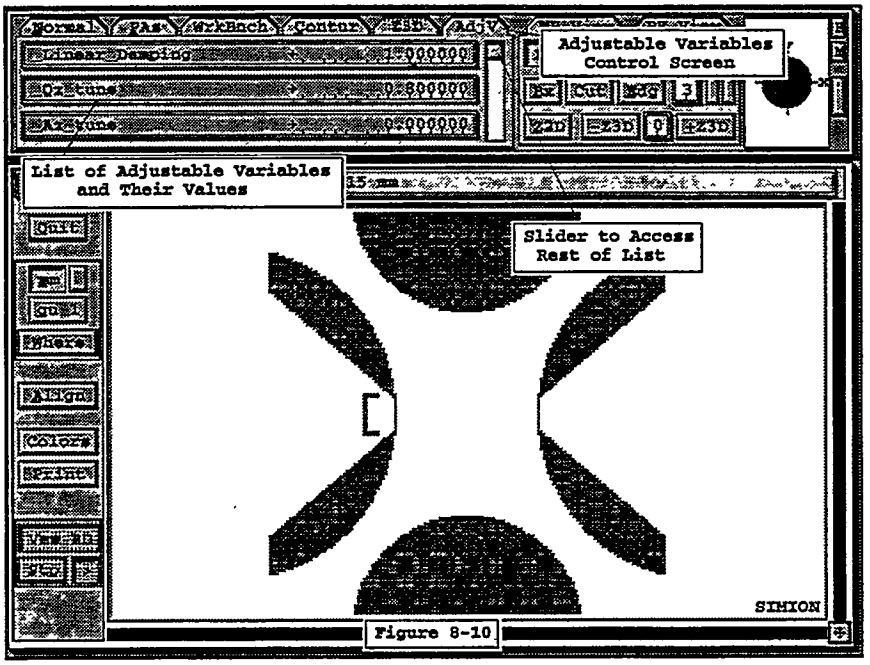


Figure 8-10

Accessing Adjustable Variables While Ions Are Flying

There are occasions when it would be very helpful to be able to access Adjustable Variables while ions are flying (*e.g. to change a tune point or damping factor*). SIMION supports this ability via the Adjustable Variables Control Screen accessed via the **AdjV** tab on the View Screen (*Figure 8-10 above*).

This tab is normally blocked. However, whenever ions are flying with user programs that have Adjustable Variables, the tab is ***unblocked.***

The List of Adjustable Variables

The Adjustable Variables Control Screen displays a list of the Adjustable Variables defined. The number display format is either e or f depending on what is currently optioned (*via* **E**. **Fmt** *button in the Adjustable Variables Screen above*).

The List Access Slider

If there are more than three Adjustable Variables to view, SIMION will automatically provide a ***List Access*** slider on the right edge of the screen.

How the Adjustment Works

SIMION allows you to adjust these variables while the ions are flying. However, it ***does not retain*** any changes you make on this screen. These changes are assumed temporary to the current **Fly'm**.

Moreover, if user programs happen to change the value of any of these Adjustable Variables the displayed value ***will not*** reflect this change.

Thus you should avoid displaying Adjustable Variables that are changed in value by user programs as ions fly. *This can be accomplished by using the trick described below.*

Selecting the Adjustable Variables to Display

If you create names for Adjustable Variables that begin with a leading underscore (*e.g.* **_Linear_Damping**), SIMION will detect this and ***only*** display Adjustable Variables with leading underscores in the Adjustable Variable Control Screen. Otherwise it will display a list of ***all*** Adjustable Variable defined. Thus if you want to limit the list, ***use leading underscores*** for any Adjustable Variables you want displayed.

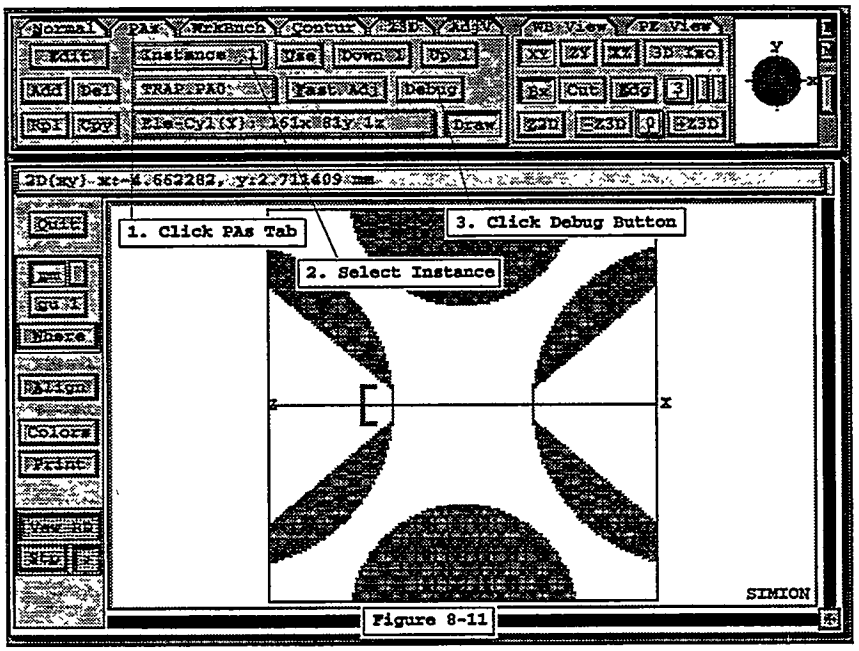


Figure 8-11

Accessing the User Program Debugger From Within View

If you develop user programs you will have bugs and new features you will want to fix or add as quickly as possible. SIMION provides access to the user program development system from within the View Screen (Figure 8-11).

To access the user program development system from within **View**, click the **PAs** tab, use the **Instance Selector** panel to select the instance with the user programs you want to debug, and click the **Debug** button.

Advanced Strategies and Tactics

Introduction

This chapter provides a few tricks and advanced concepts for getting the most out of SIMION 6.0. SIMION has enough power and flexibility that no one person will probably ever discover all the tricks or use all of its power. The objective of this chapter is to get you well on the road to discovering your own SIMION tricks. SIMION is the tool, *you* are the creator. ***You are mostly limited by your imagination.***

Doing More With Instances

The following suggestions provide interesting ways to make use of instances:

Instance Scaling Can Cause Instance Skipping

SIMION normally attempts to move an ion one grid unit per time step (*unless the absolute value of the Trajectory Computational Quality is high*). This is normally not a problem. However, if you have a small instance with an image size that is smaller than one grid unit of the instance it is within, SIMION may accidentally skip over it (*nearsighted as it is*).

You can verify that an instance is being skipped by using data recording. Create a data record with the ion number (*and location if desired*). Select **Entering an Instance** as the recording event and send verbose output to the Data Monitoring Screen. If the instance's entering record is missing then the instance was accidentally skipped during ion flying.

The easiest way to avoid instance skipping is to have the size of the smallest instance a least as big as two or more grid units of the largest instance. If this is not possible, you have the option of turning up the computational quality or making use of user programming tricks (*discussed below*).

Speeding Up Ion Flying Between Instances

When SIMION flies ions outside instances it uses time steps based on the largest scaling factor used for ***any*** instance. Thus if the largest instance scale factor defined is 100 mm per grid unit, SIMION will normally assume 100 mm grid steps when flying outside of instances.

The problem is: If all your instances have very fine grid spacing, ions will fly slower outside instances than if one of your instances had a large scaling factor. The trick is to create a dummy array (*very small*) and insert it with a large scaling factor within the workbench. Make sure the instance is out-of-the-way (*ions will not fly through it*). ***Note: Don't make the scale factor too large or SIMION might accidentally skip a small instance*** (*as discussed above*).

Using Instances as Portable Beam Stops

It is sometimes useful to be able to insert a beam stop at any location along a beam line. Beam stops help you examine beam shapes, times of arrival, and lots of other things.

The easy way to do this is to create a potential array that is a solid electrode (*no non-electrode points*). A good starting point is a 2D cylindrical array that is 3 grid units thick in x and perhaps

50 units thick in y. Fill the entire array with zero volt electrode points. Save the array with a name like **BEAMSTOP.PA.** You don't need to refine the array because there are no non-electrode points.

Now use the **Add** function within **View** to insert this array into the current workbench. Make sure that the instance has the highest priority (*e.g. highest number in instance list*). Now move, size, and orient the beam stop instance to the desired location within the workbench. ***That's all there is to it.***

Since the presence of one instance doesn't change the fields in another instances, the ions will splat into the beam stop without having their trajectories changed in any way by its presence. You should use a positive value (*e.g. 2*) for trajectory computational quality so that binary boundary approach will be active and the impact points of the ions will be calculated accurately.

This trick works best when you project the beam stop ***within*** other instances. If the beam stop is ***outside*** all other instances you must be sure its potential is the ***same*** as that of the last instance the ions flew through. Otherwise the ions will accelerate or decelerate into the beam stop. ***The way to avoid this problem is to make the beam stop array a magnetic array of zero Mag pole points.***

Modeling Field Interactions Between Instances

SIMION does not compute field interaction effects between instances. Each instance is a island unto itself. It is your responsibility to model these interactions by setting the appropriate array boundary conditions. The following is a collection of methods and tricks you can use. These ***will not*** solve ***all*** your field interaction problems but they should be helpful.

Fields Between Abutted Instances

There are times when the inner portion of a lens component has a simple 2D symmetry, while its ends require 3D asymmetrical modeling. One could use a large 3D array for the entire component. However, the 3D array may turn out to be impossibly large or not model certain aspects of the problem particularly well. In these cases you should consider abutting instances together to form the component. An example of this is the quadrupole demo in the **_QUAD** directory.

The problem is how to match the fields properly in regions where instances abut. The most obvious solution would be to break the component at grid boundaries (*have instances abut at grid boundaries*). This assumes the component has plane shaped grids that allow this trick. If it does not have grids, you might well ask yourself why not? Even if there can be no grids, there may be portions with good linear gradients. In this case you could model the problem with grids (*placed well into the linear gradient*) to isolate instances without incurring significant errors when simulating a real-world grid-less system.

Another approach is to recognize the depth of your asymmetries. In a quadrupole, the end regions of the rods are quite asymmetrical. However the central portion of the rods do not see the end regions at all and thus can be modeled by a 2D array. The problem is where to transition between instances. The question is how far do the external fields penetrate. ***You can use SIMION to determine this for your problem.*** However, as a rule of thumb, it is normally safe to assume that external fields do not penetrate a gap to a depth of more than three to five times the gap's width. The quadrupole demo abuts the instances in this region.

Transferring Fields from Outer to Inner Instances

The first question you should ask yourself is why? It is normally good design practice to isolate components as much as possible from each other. This usually involves placing a can

around the inner components. When you do this, the problem normally reduces to that of the abutting problem discussed above.

However, there are cases where isolation will not work. SIMION has only limited tools to support these cases. Whether you can make use of them depends on the nature and symmetry of your problem.

If you want to model an emission point at higher resolution within a larger volume, you could create a high resolution array for the point and superimpose it's instance upon the larger volume's instance. The problem is how to set the boundary conditions properly. The suggested approach would be to create the outer volume's array first. Model the emission point within it as best you can. **Refine** the outer volume's array and save it. Now use the **Modify** function to change the points along what will be the interface boundary for the inner instance into electrode points without changing their potentials. This is done by turning **Find** on, setting a large delta find potential, marking an actual boundary line, clicking the **Repl** button, and only changing point type to electrode. Now reduce the size of the array to this bounded portion (*by moving and sizing as required*). **Keep** the changes. Use the **Double** function (*with electrode interpolation active*) to increase the array density one or more times, **Refine** the array, and **Save** it with a new name. This method is discussed in Chapter 5 (**Double** *function*).

The approach will work only in certain cases. One: Both arrays are 2D cylindrical and the point is centered on the x-axis. Two: Both arrays are 3D asymmetrical (*less constraints - more RAM*).

Transferring Fields from Inner to Outer Instances

For example, let's say we have a collection of instances inside a larger 3D potential array instance (*the can*). Let's also assume that the instances interact electrostatically (*e.g. it is not field-free between them*). What we need is a way to model the effects of the interior instances on the fields of the containing 3D potential array instance. We could do this if we could project the appropriate electrode points from the interior instances into their equivalent 3D array instance points. We would then refine the 3D array to estimate the fields outside the interior instances.

The **Cpy** button (*PAs Control Screen in* **View**) provides a method to copy overlapping electrode (*or pole*) points from instances (*of the same type - electrostatic or magnetic*) to the equivalent locations on the ***currently selected 3D (non-mirrored) potential array instance.***

For this problem let's also assume that the 3D array is a **.PA0** fast adjust and that we want to be able to vary the effective voltages of the interior instance electrodes too. The first step would be to create the externally equivalent potential arrays for the internal instances. In most cases (*good design*) cylindrical potential arrays are enclosed by a tube. This means that the array looks like a solid tube to a outside observer. Thus one would use **Modify** to create a solid tube (*bar*) starting with the original array. If the tube is to be fast adjustable it should be given a potential appropriate to mesh properly with the 3D array's **.PA#** file and the file saved with a different name. Now load the desired **.IOB** file and use the **Rpl** button (*PAs Control Screen*) to load the 3D **.PA#** file in its instance as well as the solid cylinder in its instance. Now select the 3D instance, use the **Cpy** button to copy the points from the solid cylinder. Now exit **View** and **Refine** the **.PA#** file. You can now get back into **View** and use **Rpl** to restore the original potential arrays to their proper instances (*be sure to save the* **.IOB** *file afterwards*).

Note: Because the destination 3D array is generally at much lower resolution than the source array you may have problems with ions splating on the crude array (*instance below*) as they exit the higher resolution array above. This problem can be avoided by slightly expanding

the size of the interior potential arrays with a boundary area (*volume*) of non-electrode points. This helps ions transition beyond any copied points (*and thus no splat*).

User Programs

If you have played with the demos you have probably seen user programs in action (*e.g. the trap demo*). It is recommended that you read (*or at least scan*) Appendix I before continuing with this section.

User Programs and Potential Arrays

User programs are associated with potential arrays. For example the array **TEST.PA**'s user program array would be named **TEST.PRG**. Unless an ion is ***actually within*** the volume of an instance (*using instance selection rules*) that has an array ***with*** user programs, the ion will ***not*** be impacted by user programs.

This means that when an ion is flying outside of all instances, ***no*** user programming control can be exerted on it. This is very important to understand. ***Most of the it-doesn't-seem-to-work-at-all problems relate to ions being out of the span of control of the defined user programs.***

Extending the Span of User Program Control

One method to insure that ions can be controlled anywhere within the workbench volume is to create an instance that fills the current workbench volume. This instance ***must be*** instance number ***one*** and should use a low density 3D array (*conserves RAM*). If this array has user programs associated with it, then ions flying outside of higher priority instances will automatically be within its span of control.

Easy Voltage Control With User Programs

When flying ions, it would be nice to be able to fast adjust voltages without having to use the **Fast Adjust** function (*accessible from within* **View**). The ideal would be a voltage on a panel object that could be adjusted directly while ions are flying. User programs will allow us to do just that.

The following user program can be used with **TEST.PA0** file in the **MYSTUFF** directory (*example in Chapter 2*). Create the user program file with an editor and save it in the **MYSTUFF** directory as **TEST.PRG**:

```
DEFA    Lens_Voltage          1000    ;Adjustable variable and value
DEFS    Prior_Voltage         1000    ;Static variable for change detection
SEG     Fast_Adjust                   ;voltage adjustment segment
        RCL Lens_Voltage              ;get current voltage requested
        STO Adj_Elect02               ;set electrode two to desired voltage
        Exit                          ;exit program segment
SEG     Other_Actions                 ;used to update PE surface
        RCL Prior_Voltage             ;get last applied voltage
        RCL Lens_Voltage              ;get current requested voltage
        x=y Exit                      ;exit segment if the same
        STO Pior_Voltage              ;update prior to current
        1 STO Update_PE_Surface       ;request PE surface update
        Exit                          ;exit program segment
```

Use **View** to load the **TEST.IOB**. Fly the Ions **Grouped** in **Rerun** mode. Click the **AdjV** tab, change the voltage on the **Lens_Voltage** variable, and watch the ions fly. Switch to a potential

energy view. Notice that the potential energy surface changes each time you change the **Lens_Voltage**. ***This might be a handy user program to modify and use with your simulations.***

Randomizing Ions Via User Programs

It is often desirable to use randomized ions. The **Initialize** program segment can be used to randomize ions. The **_RANDOM** and **_TRAP** demo user programs contain various random ion generators. The basic approach is to randomize the desired parameters of currently defined ions. Thus the randomizing routines will randomize whatever ions you define. Take the time to study each demo randomizer program segment.

The randomizer in the **_RANDOM** demo varies an ion's energy as a random factor of its starting energy and orientation within a random cone angle about its starting angle. The **GROUP.PRG** file in the **_TRAP** demo also randomizes starting location and time-of-birth.

These demos should serve to get you started. Remember, you can apply any probability distribution (*as defined by you*) to the randomization processes.

Data Recording and User Programs

User programs can be used to record data. There are basically two approaches you can use: The **Mark** and the **Message** commands:

The Mark Command

The **Mark** command (*only legal in* **Other_Actions** *program segments*) is considered a legal data recording event. Thus you can define the parameters you want to record and the format to use. Depress the **All Markers** event button. Now whenever an **Other_Actions** segment executes a **Mark** command, a data record will be created (*assuming that the* **Record** *button is depressed*).

The Message Command

The **Message** command (*legal in* **Initialize, Other_Actions,** *and* **Terminate** *segments*) can be used to output data records directly. You control the record format by what you include in the message. All numbers will be output using the currently defined data recording number format (*e.g.* **e, f,** *or* **g** *plus width and precision*). When **Message** commands are defined in user programs SIMION will ***automatically*** enable the Data Monitoring Screen.

The demos have some good examples of using the **Message** command. The **_BUNCHER** demo computes and displays ion hit time spreads (*in* **TARGET.PRG**). The **_TUNE** demo computes and displays focus quality and informs the user of the tuning process status.

Controlling Time Steps With User Programs

User programs can control time steps. The following examples demonstrate a couple of important uses:

To Approach a Time Boundary

The **_BUNCHER** demo makes use of a time boundary. Before the time boundary, the deceleration voltage is on. After the time boundary, the deceleration voltage is off. The trick is to switch the deceleration voltage at exactly the right time ***and*** have the ion time steps synchronized exactly with this time switch. The **BUNCHER.PRG** has code that accomplishes this. Take the time to see how this works. It may be useful for you too!

Limiting the Maximum Time Step

There are times when automatic time steps can be a problem. In the **_TRAP** demo, when ions are being flown in groups to form crystals, ions slow down and the time step dilates. This can create a stability problem if the time step starts jumping RF cycles. Thus the **TRAP.PRG** has a **TSTEP_ADJUST** segment that limits the maximum time step to 0.1 RF cycle. The protection is important for zero computational quality level trajectories.

Modifying Ion Motions and Accelerations

User programs allow you complete control of ion motions and accelerations. You can even use SIMION to model any arbitrary acceleration problem (*not even ion related*). This capability is quite useful for modeling viscous or collisional cooling.

Viscous Cooling

Appendix I shows listings of how to model viscous cooling. The **_TRAP (GROUP.PRG)** and **_DRAG** demos make use of viscous cooling. Study the user programs and try it for yourself.

Collisional Cooling

Two **_TRAP** demos (**INJECT.PRG** and **TICKLE.PRG**) use a simple collisional cooling model. It uses mean-free-path and the mass of the cooling gas as cooling control parameters. All collisions are assumed to be elastic. Further the model assumes cooling collisions don't change direction of ion trajectories (*a somewhat dubious assumption*). The model gives reasonable results and could serve as starting point for your efforts in this area.

Changing the Ions' Colors

The **_TRAP (TICKLE.PRG)** and **_BUNCHER (BUNCHER.PRG)** demo change the ions' colors as they fly. This is useful if you want to display a change. In the case of the trap demo the color tracks the polarity of the end-cap tickle voltages (*color indicates direction of tickle force - useful*). ***Changing ion colors can be very useful.***

Controlling the Rerun Button With User Programs

The **_TUNE** demo controls the **Rerun** button via the **Rerun_Flym** reserved variable. This allows the user program to keep re-flying the ions until the desired tuning objective has been obtained. It then turns off the **Rerun** button and makes one last flight at the tuned voltage so that the ions trajectories will be recorded. This is an interesting example of user programs used as controllers.

Geometry Files

Appendix J gives an ***intense*** discussion of geometry files. These are quite useful if you have complex geometry definitions. Geometry files have ***three*** very real benefits:

1. Array geometry of arbitrary complexity can be defined with geometry files. Geometry files are much more powerful than normal **Modify** methods.
2. You fix an error by editing a file. With normal **Modify** methods you ***might*** have to re-enter all the geometry if you make a really bad error.
3. Once defined, a geometry file can be easily scaled to fit different array sizes. Thus an array can be doubled without introducing the jags.

Hardware and Software Requirements

Hardware Requirements

This program requires:

CPU	Minimum:	386 or above with numerical coprocessor
	Recommended:	486, Pentium, or P6 (*the faster the better*)
RAM	Minimum:	8 MB Intel Extender Version
		16 MB Rational Extender Version
	Recommended:	32 MB or more
DISK	Minimum:	100 MB - 50 Megs free
	Recommended:	1,000 MB or more - most free
Video	Minimum:	VGA - 14 inch color monitor
	Recommended:	VGA (*w VESA BIOS SVGA*)
		20 inch color monitor (*1280 x 1024*)
Printer	Minimum:	Any printer that supports:
		PostScript, HPGL, HPGL2, or PCL5
	Recommended:	A newer printer with RISC speed
		(*Color would be very nice too*)

Software Requirements

Operating systems: Runs as a 32 bit extended DOS program in DOS (*5.0 or above*), Windows (*3.xx, NT - Intel, or 95*), or OS/2 (*2.x or Warp*). **Mouse drivers: Requires that a DOS mouse driver be active (*e.g.* MOUSE.COM).**

Note: SIMION 6.0 is totally self-contained. This means that its 32 bit DOS extender, device drivers, and GUI are all bound into its **.EXE** file.

The Two Versions of SIMION

Two different DOS extender versions of SIMION are included in this distribution package (*Intel and Rational*). Each has its strengths:

Extender Version	*When to Use*
Intel Version	Minimum RAM (*8 megabytes*) DOS 386 or 486 Automatic virtual memory support
Rational Version	Lots of RAM (>= *16 megabytes*) Windows (*any*), OS/2 (*any*), or DOS 486, Pentium, or above (*fastest SIMION version*) Or if having problems with Intel version

THIS PAGE INTENTIONALLY LEFT BLANK

Installing/Troubleshooting SIMION 6.0

Introduction

SIMION 6.0 is distributed on two diskettes. These diskettes contain *two* different extender versions of SIMION and the EDY editor (*Intel and Rational*) along with a collection of examples to help you get started with SIMION.

The Two Different DOS Extender Versions of SIMION

Intel and Rational DOS extender versions of SIMION and EDY (*an editor*) are provided. Each has its strengths. The reason for this is to help insure a bit more stability, versatility, and utility. Note: Each version is complete. Each has its extender, device drivers, and GUI bound into a single **.EXE** file. The only way to tell these programs apart is their speed and installation quirks.

Extender Version	*When to Use*
Intel Version	Minimum RAM (*8 megabytes*) DOS 386 or 486 Automatic virtual memory support
Rational Version	Lots of RAM (>= *16 megabytes*) Windows (*any*), OS/2 (*any*), or DOS 486, Pentium, or above (*fastest SIMION version*) Or if having problems with Intel version

It is very important to recognize that programs based on different DOS extenders often do not like working together. This is true with the two SIMION and EDY versions shipped in this package. ***You will have problems if you try to run different extender versions of SIMION and EDY together.***

The Intel DOS Extender Versions

The installation program installs the Intel DOS Extender versions of SIMION and EDY as the default option. These were chosen for the default installation because the Intel extender is automatically virtual, very easy to install, requires less memory, ***but*** is slower.

> *The Intel DOS extender supplied with SIMION requires the following when HIMEM.SYS is loaded.*
>
> *DOS is loaded LOW or*
>
> *EMM386.EXE is active*
>
> ***If this is not the case, your computer will crash when you click the* Edit *button.***

The Rational DOS Extender Versions

The installation program can also install the Rational DOS extender versions of the programs. These versions are created by a Watcom C compiler with Pentium optimizations. They are quite fast, more tolerant of your DOS environment, require more RAM, and do not virtual (*in DOS*) without explicitly using SET Commands. ***However, they do virtual automatically in Windows and OS/2.***

Which One Should You Use

Start learning SIMION with the Intel extender version. If you have compatibility problems, have a Pentium (*or a fast 486*), or want to access more than 64 Megs of physical RAM then switch to the Rational Extender version. ***The installation program can be used to easily switch between extender versions.***

Installation of SIMION 6.0

Preliminaries

Make Sure the VESA BIOS is Active

Note: If you want to take advantage of screen resolutions above VGA be sure that your video card's VESA BIOS is loaded and active before proceeding (*e.g. run* **VVESA.COM** *for many ATI cards*). See your video card's documentation for more information.

Make Sure a DOS Mouse Driver is Active

Make sure a DOS mouse driver is loaded (*e.g.* **MOUSE.COM** *in your AUTOEXEC.BAT file*) ***before starting your installation.*** Just because the mouse works in Windows it doesn't mean there is a mouse driver active for DOS.

If a DOS mouse driver is not active, SIMION will refuse to run and tell you that a DOS mouse driver must first be loaded.

Installation Steps

To install SIMION, put **Diska** in your diskette drive, switch to that drive (*e.g.* **A:**), and type **INSTALL** and press **<Enter>**. The installation program will allow you to select either the Intel or Rational version and whether you want the demos loaded (*highly recommended*). You will be prompted to load **Diskb** as appropriate.

The programs will load into the **C:\SIM6** program directory (*you can select another drive and/or directory name*). Any demonstration files will be loaded into subdirectories created below the program directory (*e.g.* **C:\SIM6**) by the installation program. ***Note: The installation program will also install SIMION, Walk-About, and EDY in your* C:\DOS *subdirectory so you can access SIMION from anywhere on your disk if you prefer.***

When the installation is complete hit **<Enter>** and the above program files will be copied to **C:\DOS** and SIMION will auto-execute as you exit the installation program. ***If you hit* <Esc>** ***you can exit without copying the files to* C:\DOS *and auto-executing SIMION.***

> *Note: The files* **EDY.EXE** *and* **EDYSET.EDY** *must be somewhere in the current search path (e.g.* **C:\DOS**) *if you expect to be able to call the editor from within SIMION (required for user program and geometry file creation).*

Running SIMION for the First Time

If you are starting SIMION manually, type **SIMION** and hit **<Enter>**.

The Program Banner should appear. Be sure to read the GUI introduction banner and take the time to learn about help (**F1** *key*) and object help. SIMION will now scan your VESA BIOS (*if active*) to determine what screen resolutions it can support beyond VGA.

You will be given a screen with resolution selection buttons (*or a VGA available only screen*). Pick the desired resolution and you're on your way. As a general rule 16 color drivers run faster than 256 color drivers. If your computer has a local bus video card (*e.g. PCI*) 256 colors may run just as fast as 16 colors. ***However, SIMION only makes use of 16 colors*** (*using a 256 driver gains you nothing but perhaps access to some higher screen resolution - 1280x1024*).

Note: You can change video resolution from within SIMION ***at any time*** by clicking **Adjust** button (*Main Menu Screen*) and then clicking the **Video Adjust** button (*Adjust Menu Screen*).

> *The first time SIMION executes it automatically creates the GUI support directory* **C:\FILES.GUI**. *This directory holds files that contain directory trees, error message files and user preference files* (e.g. video and printer). ***If you somehow get all messed up, erase all the files in this directory and restart SIMION to start with a clean slate.***

If you have launching problems check the troubleshooting section at the end of this appendix.

The Various Demo Directories

If you elected to install the SIMION demos, the installation program created a collection of demo subdirectories below the program installation directory (*e.g.* **C:\SIM6**). Each of these directories contains one or possibly more SIMION demos or projects. These should serve to get you started and help demonstrate some of SIMION's power.

Appendix C contains a few step-by-step examples to start learning SIMION by using some of these demos. This is a good way to get a little cockpit experience before wading through the rest of the manual.

> *Note: Many of the demo directory names begin with a leading underscore* (e.g. _DRAG). *In order to maximize the compression of the demo files* (to facilitate distribution) *the potential arrays in these directories were not refined* (all non-electrode points were set to zero). **This means that you must perform certain actions on the files in these directories before their demos can be run successfully.** *Each directory contains a* **README.DOC** *file that explains what actions you must take to prepare their demos.* **Be sure to read this file and perform the required actions before trying to run any demos in a leading underscore directory.**

What is Walk-About?

Walk-About is a separate functioning version of the GUI File Manager used in SIMION. It is provided because users may like the GUI File Manager and want to be able to make use of it outside of SIMION. To execute Walk-About simply enter **WA** and **<Enter>** at the DOS prompt (*this assumes you let the installation program install Walk-About in your* ***C:\DOS*** *directory*).

Running in Windows (3.xx, NT, 95) or OS/2 (2.x or Warp)

SIMION can be run in Windows or OS/2. The Rational extender version seems to work best in these environments. You can run SIMION from the DOS prompt, make an Icon for it, or use a **.PIF** file (*Windows 3.xx - example in* **C:\SIM6 - SIMION.PIF**).

The main problem you will have relates to flaky Windows/OS/2 video drivers. If you context switch between programs (*e.g.* **<Alt Tab>** *in Windows*), your screen may look weird when you transfer back into SIMION. These video problems usually manifest themselves as trash on the bottom of the screen (*video RAM not restored properly*), weird *colors (color palette and/or video registers not restored properly*), or a completely trashed *screen (wrong video mode restored*). More video problems seem to

occur at higher SIMION screen resolutions (*problems vary by: Video card, computer, and OS - even 95 does it too*).

If and when this happens to you, enter an <Alt V> or <Ctrl V> key (for restore Video). SIMION will blank the screen, set the correct video mode, restore the color palette, and re-draw your screen properly. ***Thank Microsoft and IBM for their robust system software!***

Virtual Memory

Virtual memory is no substitute for RAM. ***It only allows what flat wouldn't fit before, to now proceed at a infuriatingly slow pace.*** The following material provides information on using virtual memory with the two supplied DOS extenders.

Virtual Memory and the Intel Extender

The Intel version of SIMION is initially set to support a 20 Meg region (*RAM and/or Disk*). If you don't have enough RAM, SIMION will use disk for RAM (*much slower*). If you see your disk light flashing when SIMION is redrawing a view (*and things are really slow*) welcome to virtual land. ***This is normally a clue that you need to buy more RAM.***

The program installation directory (*e.g.* **C:\SIM6**) has a program called **REGION.EXE**. You can use it to change the size of the virtual region (*memory limit*) allocated to SIMION:

REGION SIMION.EXE (*example of command line to use*)

Note: With the INTEL extender, only the first 64 Megs of physical RAM will be used no matter how large the region.

Virtual memory makes use of a swap file. If you are operating outside Windows the swap file will normally be **C:\XMSWAP.TMP**. You can specify another file with the DOS SET command:

SET SWAP=D:\SWAPFILE (*declares d:\swapfile as the swap file to use*)

Virtual Memory and the Rational Extender

The Rational extender versions are hard to virtual in DOS, but easy to virtual in Windows, and OS/2.

Virtual Memory in DOS

The Rational extender version of SIMION can virtual in DOS, but you must tell it to do so via a DOS SET command. ***If the proper SET commands are not provided, the Rational extender version will be real memory only in DOS.*** It however can access all your available physical RAM (*unlike Intel*).

The use of virtual memory is far less direct.

SET DOS4GVM = [option[#value]] [option[#value]]

Options:	(*sizes are in kbytes*)
MINMEM	The minimum amount of RAM managed by VMM. The default is 512 KB.

MAXMEM	The maximum amount of RAM managed by VMM. The Default is 4 MB.
SWAPMIN	The minimum or initial size of the swap file. If this option is not used the size of the swap file is based on VIRTUALSIZE (*see below*).
SWAPINC	The size by which the swap file grows.
SWAPNAME	The swap file name. The default name is "DOS4GVM.SWP". By default the file is in the root directory of the current drive. Specify the complete path name if you want to keep the swap file somewhere else.
DELETESWAP	Whether the swap file is deleted when your program exits. By default the file is not deleted. Program startup is quicker if the file is not deleted.
VIRTUALSIZE	The size of the virtual memory space (*RAM and DISK*). The default is 16 MB.

SET DOS4GVM=MAXMEM#128000 VIRTUALSIZE#256000 DELETESWAP

(Use 128 MB of physical RAM, virtual to 256 MB, delete swap when program exits)

Obviously you're in for an adventure here! More RAM looks pretty cheap.

Virtual Memory In Windows and OS/2

Since SIMION requests memory increases as heap allocations both Windows (*all versions*) and OS/2 (*all versions*) automatically provide virtual **DPMI** memory support. ***The limit is generally related to the Windows or OS/2 swap file size.***

When you are using a large amount of virtual memory relative to RAM size you may have to do the following to convince Windows 3.xx to cooperate. Type the following parameter in the **[386enh]** section of the **SYSTEM.INI** file in the Windows directory:

- **PageOverCommit=4**

This will allow Windows 3.xx to allocate a virtual memory limit for SIMION that is up to 4 times the RAM available (*e.g. 16 MB RAM will allow up to 64 MB of virtual memory at the above setting of 4*).

Windows 95 generally works OK if memory allocation methods for DOS are all set to Auto. For small RAM machines or large RAM needs, you may need to specify the **DPMI** memory explicitly (*e.g. 48000 for 48 Megs of DPMI memory*). Note: You will also have to increase the Intel version's region size to match (*default region size is 20 Megs*).

Virtual memory, SIMION, and OS/2 appear to work for the sites that use OS/2 (*provided that you set DPMI memory allocation and other parameters properly*). OS/2 Warp is reported to be faster and more compatible than earlier versions.

General Troubleshooting

The following is provided to assist you in troubleshooting:

Some General Questions

What is the quickest way to see what something does or how to use it?

> *Point to the object or area and hit the* **<F1>** *key.*
>
> *If the help screen has an* **Object Help** *button click it if you want to learn how to use that class of objects properly.*
>
> *If all else fails, read the manual!*

The number panel values change too quickly for my taste when I try to adjust them using the mouse buttons. Can I adjust the mouse button repeat rates?

> *Yes! Access the Delays Adjust Screen: Adjust Delays (from Main Menu Screen). Click the* **Repeat Delays** *button and move slider more to the* **right.** *Check by hitting the letter* **t.** *Click* **OK** *to return.*

The GUI File Manager seems to always scan each drive I select at least one time in a SIMION session. This takes time (*I have a very large disk*). Can I turn this off?

> *Yes! Access the Other Options Adjust Screen: Adjust Other (from Main Menu Screen). Click the* **Scan ON** *button so it reads* **Scan OFF.** *Click* **OK** *to return.*

Users of Prior SIMION Versions Problems

I marked an area in Modify, clicked the <u>right</u> mouse button and the screen zoomed. I was expecting the area to be filled.

> *SIMION 6.0 uses the* **right** *mouse button for 2D view zooming. The equivalent function in* **Modify** *is now the* **<Ctrl>** **<u>right</u>** mouse button, *clicking the* **Replace** button, *or just hitting the* **<R>** *key.*

What happened to the RPA program? How do I create my fast adjust files?

> *If you* **Refine** *a* **.PA#** *file, SIMION will automatically invoke the equivalent of RPA in* **Refine.**

Startup Problems

If you had trouble getting off the ground the following may be of help:

I get a message about no mouse.

> *SIMION requires that a DOS mouse driver be active (e.g.* **MOUSE.COM**). *Just because Windows works OK doesn't necessarily mean that a DOS mouse driver is active.*
>
> *Find your DOS mouse driver and load it before trying to run SIMION. You should probably load the driver in the* **AUTOEXEC.BAT** *file.*

I get a message about not enough RAM. I have 8 Megs what is wrong?

> *The Intel version of SIMION is set by default for a 20 Meg region. This means that when loading it will automatically try to create a scratch virtual file of 12 Megs or so. If your disk is too full, it cannot succeed and will give up.*
>
> *You have two choices: Clean up your disk to free enough space or run the REGION program on SIMION to reduce its region size (see virtual information above).*

For whatever reason I just cannot seem to get SIMION to start running properly.

The best suggestion is probably back to the basics. Save copies of your CONFIG.SYS and AUTOEXEC.BAT files and then gut them back to basic DOS. Yes, remove all your favorite bells and whistles. If SIMION will then run, try adding things back until it breaks.

Another option would be to try the Rational extender version of SIMION. It appears to be much better behaved in strange environments.

Kick-out or Lockup While Running Problems

I started playing around with the GUI File Manager, clicked the Edit button, and my machine locked up.

The Intel version of SIMION has probably declared war on **HIMEM.SYS**. *Your DOS must be set low or* **EMM386.EXE** *should be active. Then things should work.*

I was in the GUI File Manager, clicked the Edit button, the screen flashed, but no editor appeared.

SIMION's editor **EDY.EXE** *and its personality file* **EDYSET.EDY** *must be on the current search path (e.g.* **C:\DOS***) to be found and used by SIMION. Adjust your search path or copy these files into a directory that is on the current search path (e.g.* **C:\DOS***).*

I did something in SIMION and suddenly I am out at the DOS prompt. What happened?

If you are using the Intel version, it may mean that you are very close to the region limit (virtual memory limit). Run REGION.EXE on SIMION.EXE and increase your region size. If that fails, try using the Rational version of the program.

If you are using the Rational version it probably relates to being on the ragged edge of your virtual memory limit. Increase the virtual size in DOS or swap file in windows (see material above on how to convince windows to allocate you more memory).

If you are in Windows or OS/2 it may be a result of a faulty Windows/OS/2 video driver. Make sure yours is the most recent attempt by the board maker.

Strange or Erratic Behaviors

SIMION seemed to work OK then I started having trouble saving files or scanning for directories or things just seem broken.

You may be out of disk space. ***Check this first!*** *You should have at least 20 megabytes free at all times.*

SIMION stores temporary files in the current drive (for ion trajectories and etc.). ***It is possible that you have filled up your disk with ion trajectories.*** *This can be avoided by depressing the* **Rerun** *button when flying ions (ion trajectory images are then not saved in a temporary file).* ***This trick is very useful for watching ions stranded in an ion trap for extended periods of time.***

If you are on a network there is a chance that for some reason (unrelated to SIMION) your network has starting saving files as read-only (YES this has happened). Use SIMION's file manager (or Walk-About) to determine and reset any read-only file attributes. Be sure to check the GUI's private directory too, **C:\FILES.GUI**. ***If you find read-only files, make sure network support finds and fixes your problem.***

If all else fails, erase <u>all</u> the files in the **C:\FILES.GUI** *directory and/or re-install SIMION from the distribution diskettes.*

Video Display Problems

I <u>only</u> seem to be able to get VGA resolution. What is Wrong.

There's a good chance that the VESA BIOS is not loaded (if your video card has one - most new ones do). Check your video card's documentation or give their support line a call. ***If your computer's video card doesn't support VESA BIOS extensions then you are stuck with VGA.*** *The only remaining option for you would be to get a new video card that supports VESA BIOS extensions.*

I installed SIMION on a portable computer (*or perhaps some other computer*). I checked the Adjust Video and clicked the 1024 / 768 resolution and the screen went blank (*or displayed trash*). I can't seem to get SIMION to come up anymore. The screen just sort-of brightens (*or all I see is trash*).

Just because your portable (or regular computer) has a VESA BIOS it doesn't mean your little screen (or full size monitor) can support higher resolutions. With portables the higher resolutions are normally for external monitors only. Your problem is that SIMION has taken you at your word and tries to launch with an unusable screen resolution.

Not to worry! Reboot your machine. When you start SIMION hold either **Shift** *key depressed (**and keep it depressed**) to force the GUI to come up in VGA mode with the screen resolution adjustment screen appearing first.*

Now you can experiment to see just what enhanced VESA resolutions will actually work (if any).

I am having problems running SIMION within a window in Windows or OS/2.

Although it may be possible to run SIMION from within a window (with VGA resolution) it is <u>not</u> recommended. ***You should flag SIMION or all DOS programs to be displayed full screen.***

I am using SIMION from within Windows or OS/2 and see screen trash when I toggle back into SIMION.

*When you toggle between SIMION and Windows (or OS/2) these operating systems try to save a screen snapshot of SIMION's screen image to allow you to toggle back quickly. Unfortunately, things do not always work as planned (**this is a chronic feature of Windows and OS/2 - all versions**).*

Hit either the **<Alt V>** *or* **<Ctrl V>** *to restore* **Video.** *SIMION will blank the screen set the proper video mode, color palette, and re-draw the screen.*

If that fails (e.g. machine lockup or blown out to the DOS prompt), you have a totally defective Windows or OS/2 video device driver (not that uncommon). Contact the manufacturer of your video card for the latest attempt at a video driver for your operating system. ***Use VGA or just don't context switch.***

Potential Array Problems

SIMION refuses to allow me to adjust the array dimensions to create a large 100 x 100 x 100 potential array in the New function.

> *You have to increase the size in the* **Max PA** *panel (on* **NEW** *screen) to at least 1,000,000 points. Then the desired size can be set. SIMION may still balk because it is out of RAM or virtual space (see discussion of virtual memory above).*

SIMION refuses to load a potential array - gives out of memory error.

> *If you are loading into an existing PA (not into an empty PA) the memory required for the PA file to be loaded exceeds the memory allocated to the existing PA.* ***You will either have to load the PA into the empty PA region or remove all PA's from RAM and then load the PA into the empty PA region.***
>
> *If you are loading into an empty PA region, you have reached a RAM or virtual limit. If you are using the Intel version you can increase the region size. If it is the Rational version the solutions are: More RAM, activate virtual in DOS, run SIMION from Windows/OS/2, or increase the DPMI memory allocation. See discussion on virtual memory above for more information.*

SIMION refuses to allow the Double function to resize an array or the nx, ny, or nz adjust panels within the Modify function don't allow me to increase PA dimension to those I desire.

> *When a potential array is created or loaded into an empty PA, SIMION allocates a fixed amount of memory (thus max size) for the potential array. Your resizing efforts were blocked because they would have exceeded the memory allocated for the PA's region.*
>
> *The best way to deal with this is to save the current potential array (if it has not already been saved). Now click the* **Remove All PAs From RAM** *button (on the Main Menu Screen), Compute desired size for the to be expanded array (maximum number of points). Enter this size in the* **Max PA** *panel object (Main Menu Screen). Now reload the desired PA into the empty PA region. SIMION will allocate the amount of expansion memory you desire and you can proceed with expanding the size of the potential array.*

Printing Problems (Read Appendix G)

I tried to print something and all I got was pages of strange looking text.

> *SIMION assumes that your printer is* ***PostScript*** *by default. You are probably seeing PostScript output on your non-PostScript Printer.*
>
> *Use the* **Options** *button on the Print Control Screen to access the options and use the selector to option your printer's language.*

I can't seem to get SIMION to output to my el-cheapo matrix printer.

> *SIMION requires that your printer functions with one of the following printer languages:* ***PostScript*** *(color or B/W),* ***PCL5*** *(color or B/W),* ***HPGL2****, or* ***HPGL****.*
>
> *If this is not the case, you have two choices: One get a new printer (recommended), or obtain a translation program from somewhere that converts one of the printer languages above to something your current printer understands.*

I can't get SIMION to output in color to my PostScript, PCL5, or HPGL2 printer or plotter.

> *The Print Control Screen has a* **B/W** *button. Click this button to toggle to* **Color**. *Now your output should be in color.*

Can I change the width of lines output in B/W?

Yes! SIMION translates each color into a line width. Use the **Options** *button on the Print Control Screen and adjust the desired colors' width control panels (at the bottom of the screen). Note: Line widths are adjustable for all printer languages except HPGL.*

Can I change the width of lines output in color?

Yes! SIMION uses the width of the color white (color 15 - lower right corner) for all colors in colored vector printing. Adjusting this width can have positive or negative impacts on what you get.

How do I send printer output to a file?

Use the **Options** *button on the Print Control Screen and change the device/file name to the file name you want (e.g.* **PRN** *to* **TEST.EPS**). *Note: SIMION appends (adds) all printer output to the designated device or file.* ***Normally you will only want to output a single frame to a print file.*** *Thus, you should change file names before each print to file.*

I am having trouble importing SIMION PostScript files into my word processor or graphics program.

First, be sure that you have selected ***encapsulated*** *PostScript for your printer language in SIMION. Next, save only one image per file - margins set to zero.*

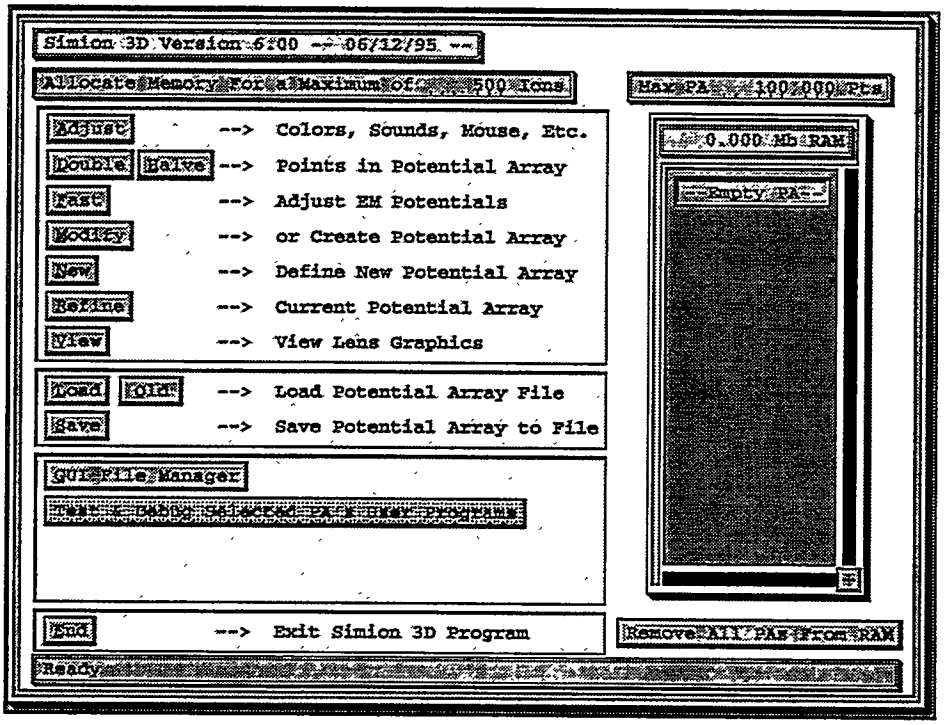
Note: ***Many graphics programs pretend to be able to read encapsulated PostScript.*** *What they really read is their own particular dialect. We can't help with this problem. As a fallback position, try using HPGL when outputting to a file. Most programs support HPGL format files (although you may suffer some image quality loss).*

Sample SIMION 6.0 Runs

Launching SIMION for the First Time

Please read (*or at least scan*) Appendix F ***now***. ***It is important that you have at least some understanding of the GUI before you proceed.***

It is assumed that you have installed SIMION as per the instructions in Appendix B. Make sure you are in SIMION's program directory (*e.g.* **C:\SIM6**). Type **SIMION** and press the **<Enter>** key.



Getting the Lay of the Land

If all has gone well you should be looking at the Main Menu Screen (*shown above*). Note that it is full of buttons. To select a menu option, click its button or if the first letter of the button's label has a red underline, you can access it directly from the keyboard by entering the label's first letter (*e.g. <A> for* A*djust*).

Playing with Adjust Preferences

Use the **Adjust** button to adjust preferences. The Adjust Options Screen will allow you to adjust color, sounds, delays, other options (*e.g. mouse*), and video resolutions. Take the time to try each. Use the help screens (*including object help*) to get familiar with the GUI and how to use SIMION.

GUI File Manager

The GUI has a very powerful and easy to use file manager that is used to load and save your SIMION files (*Appendix F discusses the GUI and its file manager*). The Main Menu Screen's **GUI File Manager** button gives quick access to the file manager itself. It is suggested that you take the time to get acquainted ***NOW***.

Learn about the controls, create a subdirectory, copy a few files, and you're ready to start using SIMION.

> *The program Walk-About* (**WA.EXE**) *is a stand-alone version of the GUI File Manager. Walk-About can be used outside of SIMION. It is provided for those who decide to adopt the GUI File Manager for more general use.*

Scan the Manual and Its Illustrations

It probably would be a good idea to quickly scan the main manual ***NOW!*** There are annotated figures throughout the main manual. These figures serve to show you where things are and how to perform tasks with SIMION. ***Remember, if all else fails, read the manual.***

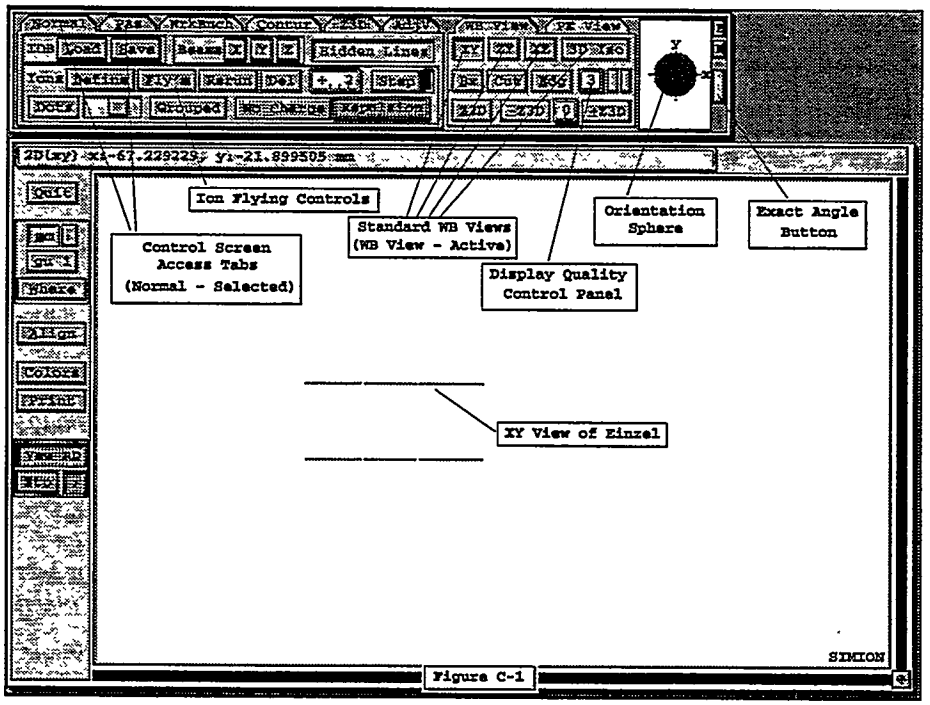


Figure C-1

Your Maiden Voyage With SIMION

1. Start SIMION from its program directory (*if you haven't already done so*).

2. Press the **View** Button (*or hit the* **<V>** *key*) to access the **View** function (*no PAs loaded*).

3. The GUI File Manager will be invoked to select an **.IOB** file (**I***on* **O***ptics* **B***ench*). Click on the **EINZEL** sub directory (*just below the* **C:\SIM6** *sub directory*). Now place your cursor over the **EINZEL.IOB** button and click ***both*** mouse buttons together, and the **.IOB** will be loaded along with its PA files. Click *Yes* to restore PA potentials. ***You should now be looking at an XY 2D view of an einzel lens (Figure C-1 above).***

4. Now click the **3D ISO** button to get a quick 3D isometric view of the einzel lens. ***The drawing may look a bit crude.*** Put your cursor on the display quality panel (*currently showing the number* **3**) and hit the **9** key. ***Now the drawing quality looks pretty good.***

   Click the **XY**, **ZY**, and **XZ** buttons too. These give you quick access to the standard 2D views.

5. Use the orientation sphere to change the view's orientation (*point to the sphere and drag it about with either mouse button depressed*). Try all the indexed 2D (*12 views*) and 3D (*8 views*) isometric views. Now set the drawing quality to 6 and click the exact angle button. Try various exact angles using the orientation sphere. Vary the drawing quality to see the effect on quality and drawing time (*9 can take a very long time*). ***Remember you can hit the <Esc> key to stop a re-draw if it is taking too long!***

6. Try 2D zooming. Mark an area (*drag with the* **left** *mouse button depressed*) and click the **right** mouse button (*to 2D zoom*). Click the **right** mouse button again to return to the previous view. Now hold either **Shift** key depressed and click the **right** mouse button to zoom back in. ***Up to 10 levels of 2D zoom are remembered.***

7. Try 3D zooming (*volume zooming*). Switch to the 2D end view of the einzel (*looking down the cylinders - hint -* **ZY** *button*), mark a region that ***encloses slightly more*** than the right half of the cylinder (*we don't want to cut off any future ion trajectories in the center*), and click the **+Z3D** button. Now rotate to a 3D view orientation (*hit:* **3D Iso** *button or use orientation sphere*). ***You are now looking inside the einzel cut in half along its length.***

8. Now let's fly some ions! Click the **Define** button (*for define ions screen*), then the **Load** button, click *both* mouse buttons on the **EINZEL.FLY** button, and click the **OK** button. Now click the **Fly'm** button (*to start the ions flying*). The ions should fly one at a time. ***You can change views, zoom or whatever as the ions fly.***

9. To fly the same ions as a group click the **Grouped** button and then the **Fly'm** button.

10. To continue to re-fly the same ions as a group, click the **Rerun** button too. Now click the **Fly'm** button. If you like, now click the **Dots** button to see the ions fly as dots. Adjust their speed with the slider control (*just to the right of the* **Dots** *button*). Adjust the trajectory quality panel to zero (*was initially 2*). Notice that the trajectories are faster. Try other trajectory quality levels to see their effect. Hit **<Esc>** or click the **Fly'm** button to stop the ions flying.

11. You can adjust voltages while the ions are flying. Get the ions flying as rays or dots **Grouped** in a **Rerun** mode. Set trajectory quality to zero for fast ion flying. Now click the **PAs** tab and Click the **Fast Adj** button. Set the voltages you want, Click the **Fast Adjust** button, the voltages are adjusted, and **View** returns with the ions still flying (*powerful*). Click the **Normal** tab to return to the Normal Controls Screen.

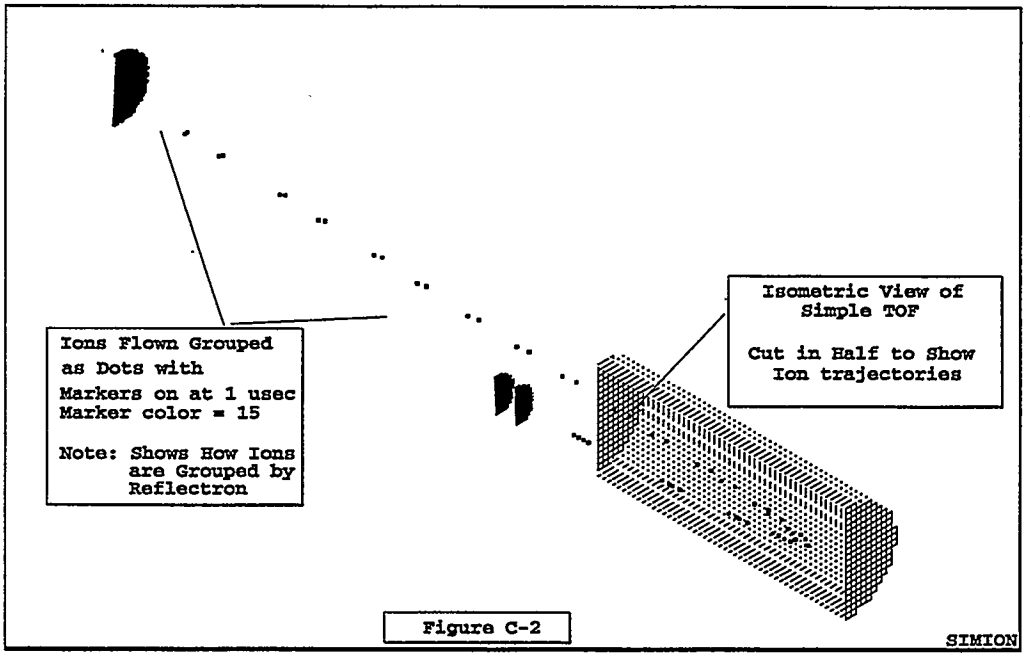
12. For beam repulsion, make sure that **Grouped** is depressed, select **Beam Repl** (*click button to right of* **Grouped** *three times*), **Rerun**, and turn off **Dots** (*optional*). Now **Fly'm**. As the ions fly, adjust beam current to 10^{-6} amps (*panel to right of* **Beam Repl**) and note the beam divergence. Try to find the threshold of the effect and note where the beam deviates the most.

Your Second Voyage With SIMION

1. Start SIMION from the program directory (*e.g.* **C:\SIM6**) or click the **Remove All PAs from RAM** button (*Main Menu Screen*) to remove your previous adventure.

2. Press the **View** Button (*or hit the* **<V>** *key*) to access **View** (*no PAs loaded yet!*).

3. The GUI File Manager will be invoked to select an **.IOB** (*Ion Optics Bench*) file. Click on the **TOF** sub directory (*below the program directory*). Now place your cursor over the **TOF.IOB** button and click ***both*** mouse buttons together, and the **.IOB** file will be loaded along with its PA files. Click *Yes* to restore PA potentials. You should now be looking at an XY 2D view of a three element Time-of-Flight (*TOF*) system. Turn



display quality to 6.

4. Swing to an end view and cut the view in half with a 3D zoom. Note: We want to see inside the reflectron. Be sure that ***more than half is visible*** so you haven't cut away the ion trajectories themselves (*as in Figure C-2*).

5. Click the **Define** button and load the **TOF.FLY** file. Set **Grouped, Dots,** and **Rerun.** Now **Fly'm.** Four ions will be flying. Ions of the same color have the same mass (*but different energies*). Note how the ions reflect in the reflectron and time converge at the detector.

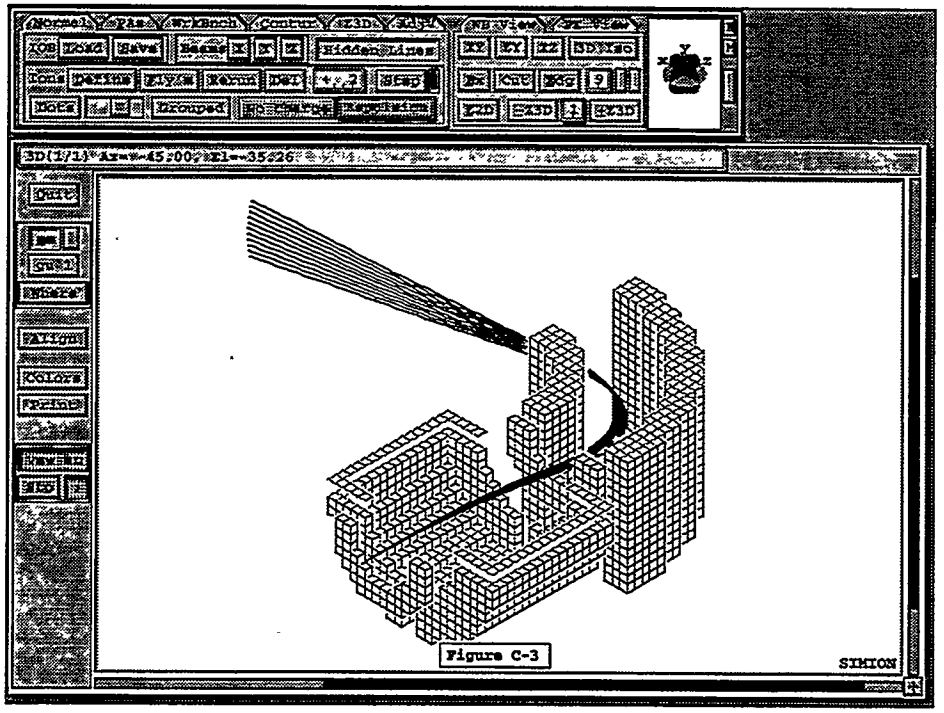
6. Let's look at some numbers. Click the **Define** button to access the data recording controls. Click the **Define** button (*in the Record Data Area*) to access the Data Recording Definition Screen. Use the **Load** button to load the **TOF.REC** file. Click the **OK** Button, then the **Record** button, and then the **OK** button. Now **Fly'm.** Notice that a data screen appears and the ion impact times are displayed. Note the control area on the lower left edge of your screen is now unblocked. You can turn the data display screen on and off as well as step from event to event. Use the **F1** help key to learn how.

7. Change the computational quality from 2 to 0. This turns off the boundary checking features. Note that time convergence erodes when quality is set to zero.

8. Try other views (*2D and 3D*) including exact angles. Try zooming in to see details (*2D and 3D zooms*).

9. From the XY 2D view (**XY** *button depressed*) click the **PE View** tab and watch the ions reflect in the potential energy field of the reflectron. Note that only the PE surface of the reflectron is visible. This is because the other two instances are off about 3 degrees (*non-integrally aligned*).

10. To view the PE surface of the source element, exit PE view (*click* **WB View** *tab*), click the **PAs** tab and use the instance selector to select the source element PA (*green box will surround it*). Now click the **Align** button and *then* the **PE View** tab. Note: Ions could have been flying during this process. If you have turned them off, click the **Normal** tab and click **Fly'm** button to resume ion flying.



Your Third Voyage With SIMION

Most of SIMION's demos are found in directories with leading underscores (*e.g.* **_MISC**). This means that you are required to do some preparation before these demos can be used (*normally just refining arrays*). These directories all have **README.DOC** files to tell you what to do. ***Always read the README.DOC file and perform the preparations before trying to run the demos.***

The third voyage (*Figure C-3 above*) involves the leading underscore directory **_MISC**. ***The instructions below will tell you how to prepare the selected demo.***

1. Start SIMION from the program directory (*e.g.* **C:\SIM6**) or click the **Remove All PAs from RAM** button to remove your previous adventure.

2. These are the instructions for preparing this demo (*first time only*). ***Skip this step on any subsequent running of this demo.***

   Click the **Load** button (*or hit the* **<L>** *key*) to load a potential array directly. The GUI File Manager will be invoked to select a PA (*potential array*) file. Click on the **_MISC** sub directory (*below the program directory*). Now place your cursor over the **TEST3D.PA** button and click ***both*** mouse buttons together, and the **.PA** file will be

loaded. Click the **Refine** button and then press the **<Enter>** key to refine the potential array. After the refining is completed the main menu will return. Click the **Save** button and press the **<Enter>** key. You will be asked if you want the file replaced - click the **Yes** button (*or hit the* **<Y>** *key*). A file memo will appear for you to edit. Skip this by clicking the **right** mouse button. Now click the **Remove All PAs from RAM** button to remove the potential array from RAM for a clean start.

3. Press the **View** Button (*or hit the* **<V>** *key*) to access **View** (*no PAs loaded yet!*).

4. The GUI File Manager will be invoked to select an **.IOB** file. Click on the **_MISC** sub directory (*below the program directory*). Now place your cursor over the **TEST3D.IOB** button and click ***both*** mouse buttons together, and the **.IOB** file will be loaded along with its PA files. Click *Yes* to restore PA potentials. You should now be looking at an XY 2D view of a 3D potential array. Swing to an Isometric view to verify. Turn display quality to 9.

5. Click the **Define** button and load the **TEST3D.FLY** file. Make sure **Grouped** and **Rerun** are OFF. Now **Fly'm**.

6. Now swing to a 2D view and make a 3D cut to show the lower half of the workspace. Look at this with the various 3D views.

7. From an isometric view use page view (*right mouse button depressed on window's button*) and the **<Alt>** key to do some 3D pointing and cutting. Refer to the manual for details.

8. Try to 3D zoom to a small volume inside the lens system. Now use the **Cut** button to get cutaway views of the process.

9. Explore the window's and view's help screens (*they are different*). See any new tricks? Try them.

10. Another thing to try is to cut the volume in half with a 3D zoom (*as above*). Look in from above (*2D view*). Switch to a **PE View**, click the **Z3D** tab, and then move the top surface of the 3D cut up and down (*using the sizing panels*). See how the shape of the PE surface changes?

Other Adventures

You will note that there are a number of demo directories below your program directory. Snoop around and try them all. If your PA memory becomes too cluttered Click the **Remove All PAs from RAM** button to clean the mess up.

> *Note: Many of the demo directory names begin with a leading underscore* (e.g. **_DRAG**). *In order to maximize the compression of the demo files* (to facilitate distribution) *the potential arrays in these directories were not refined* (all non-electrode points were set to zero). **This means that you must perform certain actions on the files in these directories before their demos can be run successfully.** *Each directory contains a* **README.DOC** *file that explains what actions you must take to prepare their demos.* **Be sure to read this file and perform the required actions before trying to run any demos in a leading underscore directory.**
>
> ***If you run the icrcell or quad demos you may have to increase the region of the Intel version to around 30 megabytes.***

Note: You can now start creating your own PAs (*2D and 3D*) or load older version SIMION PA files. ***Be sure to create a project directory first and use it for your initial floundering.*** Have fun!

Files Used by SIMION 6.0

Introduction

SIMION creates and uses a collection of files. Each type of file generally has its own unique extension (e.g. .PA) to signify to you and SIMION what it is. Files are used to store electrode geometry, potential arrays, contour intervals, trajectory parameters, and more.

It is important that you observe SIMION's file conventions. The following is a list of the files SIMION uses:

Files Found in Project Directories

The following is a list of files that may appear in a project's directory:

File: *.CON (All files with .CON extensions)

Created by:	**SIMION in the CONTOUR MODE**
Used by:	**SIMION in the CONTOUR MODE**
Function:	**A Collection of Contour Intervals**

SIMION creates ***.CON** files via the **Save** button on the Contour Mode Screen. These files contain a list of potential and gradients contour intervals for later use via the contour file **Load** button.

File: *.FLY (All files with .FLY extensions)

Created by:	**SIMION in the TRAJECTORY DEFINITIONS**
Used by:	**SIMION in the TRAJECTORY DEFINITIONS**
Function:	**Parameters for Grouped Ion Definitions**

When you enter the parameters for a specific collection of grouped ion definitions SIMION allows you to save these parameters as a ***.FLY** file. This allows you to later recall the grouped ion definitions without having to remember and re-enter them.

File: *.GEM (All files with .GEM extensions)

Created by:	**YOU with Editor (e.g. EDY) as ASCII file**
Used by:	**SIMION for Defining Electrode/Pole Geometry**
Function:	**Electrode/Pole Geometry Definition Files**

Geometry files are used as an alternate way to define complex 2D and 3D electrode/pole geometry. This is an advanced SIMION feature. *See Appendix J for more information.*

File: *.IOB (All files with .IOB extensions)

Created by:	**SIMION in the VIEW OPTION**
Used by:	**SIMION in the VIEW OPTION**
Function:	**Ion Optics Bench Definition File**

When you define an ion optics workbench of instances the **Save** button on the Normal Mode Screen (*in View*) allows you to save this definition as an ***.IOB** file. This allows you to later load the ***.IOB** file and have SIMION automatically establish the workbench for you (*including loading potential arrays and restoring potentials*).

File: *.ION (All files with .ION extensions)

Created by:	**SIMION in the TRAJECTORY OPTION**
Used by:	**SIMION in the TRAJECTORY OPTION**
Function:	**Parameters for Individual Ion Definitions**

When you enter the parameters for a specific collection of individual ion definitions SIMION allows you to save these parameters as an ASCII ***.ION** file. This allows you to later recall the individual ion definitions without having to remember and re-enter them. These ASCII files can be created externally by you with an editor or another program. The format used by ***.ION** files is described at the end of this appendix.

File: *.PA? (All files with .PA? extensions)

Created by:	**SIMION**
Used by:	**SIMION**
Function:	**Holds an entire potential array**

A **.PA?** file contains all the data for an entire potential array in machine binary image storage format. These files preserve potential arrays between SIMION sessions. The **?** signifies any legal file naming character (e.g. **.PA, .PA1, .PAA**). The format used by ***.PA** files is described at the end of this appendix to allow reading and creation via your own C programs.

File: *.PRG

Created by:	**YOU with Editor (e.g. EDY) as ASCII file**
Used by:	**SIMION for User Programs**
Function:	**Holds User Programs**

These files contain user programs generated by the user. For example: The file **TEST.PRG** is the user program file for the potential array **TEST.PA** (*or* **TEST.PA0**). User programming is an advanced SIMION feature. ***See Appendix I for more Information.***

File: *.REC

Created by:	**SIMION in DATA RECORDING**
Used by:	**SIMION in DATA RECORDING**
Function:	**Holds Data Recording Definitions**

These files contain data recording definitions. The data recording feature of SIMION allows you to save and load data recording definitions as ***.REC** files.

Important GUI Files in the C:\FILES.GUI Sub Directory

The following GUI support files are found in the **C:\FILES.GUI** sub directory. It is useful to know about them.

File: GUI?????.TMP

Created by:	**GUI**
Used by:	**GUI**
Function:	**GUI's temporary GUI Error File.**

The **GUI?????.TMP** file retains status and error information for a currently running GUI application (*this is done to avoid sharing violations when multiple GUI applications are concurrently running*). Automatically managed by GUI based programs. ***Program lockups and rude behavior on your part can create a collection of .TMP files. Feel free to erase them to clean out the clutter.***

File: ERRORS.GUI

Created by:	**GUI**
Used by:	**GUI**
Function:	**GUI's Error File.**

The **ERRORS.GUI** file retains status and error information for a GUI application. ***If something strange is happening with lots of beeping, quit SIMION and take a look at this file.*** Automatically managed by GUI based programs.

File: INI.GUI

Created by:	**GUI**
Used by:	**GUI**
Function:	**GUI's Personality file.**

The **INI.GUI** file retains the color palette, delays, sounds, mouse speed, and other appearance items. Automatically managed by GUI based programs. ***If you erase this file, the GUI will assume its default personality.***

File: PLOTTERS.GUI and ANNOTATE.GUI

Created by:	**GUI**
Used by:	**GUI**
Function:	**GUI's Print Personality & Annotations files.**

The **PLOTTERS.GUI** file retains the current printing options. Automatically managed by GUI based programs. *If you erase* **PLOTTERS.GUI**, *the GUI will assume its default printing personality*. The **ANNOTATE.GUI** file retains the last annotations defined in a GUI based program session.

File: VESA.GUI

Created by:	**GUI**
Used by:	**GUI**
Function:	**GUI's Video Personality file.**

The **VESA.GUI** file retains the current video options. Automatically managed by GUI based programs. ***If you erase this file, the GUI will assume its default video resolution - VGA.***

File: ?_DIR.GUI

Created by:	**GUI**
Used by:	**GUI**
Function:	**GUI's Directory Scan for a Drive.**

The **?_DIR.GUI** file retains the last directory scan for drive ? (*e.g.* **C_DIR.GUI** *for* **C** *drive*). Automatically managed by GUI File Manager (*recreated with* **Scan** *button*). ***If you erase this file, the GUI File Manager will automatically rescan the drive.***

The Format Used with SIMION 6.0s .PA Files

The following is provided for those who may want to generate or read SIMION 6.0 **.PA** or **.PA#** files. ***The definitions below assume C conventions***:

```
#define  PLANAR          1        /* planar symmetry */
#define  CYLINDRICAL     0        /* cylindrical symmetry */

#define  MIRROR_X        1        /* array mirrored in x */
#define  MIRROR_Y        2        /* array mirrored in y */
#define  MIRROR_Z        4        /* array mirrored in z */
#define  MAGNETIC_PA     8        /* is magnetic potential array z */

int ng = 100;                     /* number of grid points between poles */

struct header_3d                  /* header image for SIMION 6.0 PA files */
        {
        long mode;                /* mode must be -1 */
        long symmetry;            /* PLANAR or CYLINDRICAL */
        double max_voltage;       /* max voltage allowed for pa */
        long nx;                  /* array's x dimension size */
        long ny;                  /* array's y dimension size */
        long nz;                  /* array's z dimension size */
        long mirror;              /* (MIRROR_X | MIRROR_Y |
                                  MIRROR_Z | MAGNETIC_PA) +
                                  (ng << 4) */

        };
```

Note: mirror contains mirroring flags, magnetic PA flag, and shifted copy of ng.

File Format:

```
struct header_3d                  /* byte packed - no skipping */
double points [ nz ] [ ny] [ nx]; /* array in internal binary format */
```

The values of the points are as follows:

Non-electrode points	=	Potential;
Electrode points	=	Potential + 2.0 * max_voltage

ASCII Format Used in Individual Ion Definition Files .ION

The ASCII file format has one (*and only one*) ion definition per line with parameters in the order below separated by commas. Parameters can be skipped (*using commas*) or omitted (*trailing parameters*). ***SIMION will automatically assume default values for any skipped/omitted parameters.***

TOB,	**MASS,**	**CHARGE,**	**X,**	**Y,**	**Z,**	**AZ,**	**EL,**	**KE,**	**CWF,**	**COLOR**	
0,	100,	1,	0,	0,	0,	0,	0,	0.1,	1,	0(*black*)	**Default Values**

Computational Methods

Introduction

This appendix describes the methods used for computations within SIMION 3D Version 6.0. The discussion is somewhat generalized to avoid writing a tome on the subject.

SIMION 6.0 is an electrostatic and magnetic field modeling program. This means it is designed to model the electrostatic and magnetic fields and forces created by a collection of shaped electrodes given certain symmetry assumptions. Electrostatic and magnetic fields can be modeled as boundary value problem solutions of an elliptical partial differential equation called the Laplace equation.

The specific method used within SIMION is a finite difference technique called over-relaxation. This technique is applied to two or three dimensional potential arrays of points representing electrode (*pole*) and non-electrode (*non-pole*) regions. The objective is to obtain a best estimate of the potentials for those points within the array that depict non-electrode (*non-pole*) regions.

Relaxation techniques use iteration, a technique of successive approximation. Relaxation has the advantage that normal numerical computation errors are minimized, solutions are quite stable, and computer memory storage requirements are minimized. However, its disadvantage is that the exact number of iterations required for a given level of refining is quite variable and initially unknown for each specific solution.

In the discussions below references will be made mostly to electrostatic arrays. ***These discussions apply in the same manner to magnetic arrays unless otherwise noted.***

The Structure of the Potential array

In SIMION, the potential array is a one dimensional array of double precision points (*referencing methods make it appear two or three dimensional to the user*). Each array element represents a point within the two (*or three*) dimensional array. Each array element typically stores two pieces of information. The first is the current voltage at the point, and the second is whether or not the point is an electrode. Electrode points are marked by adding two times the maximum allowable array voltage to their actual voltage. ***SIMION has the ability to automatically re-scale a potential array with a larger offset voltage if the maximum allowable array voltage is exceeded.***

The array is considered packed when each point holds both voltage and electrode flag information. Certain functions like refining operate faster if the electrode flag information is held in a separate array. Each PA in RAM has its own separate buffer to support the unpacked potential array when needed.

The Relaxation Method

The relaxation method uses nearest neighbor points to obtain new estimates for each point. SIMION uses the nearest four points for 2D (*nearest six points for 3D*). The example below demonstrates the 2D array (*3D adds a point 6 above in z and a point 5 below in z for a total of six*):

P_4

P_1 P_0 P_2 2D $P_{0new} = (P_1+P_2+P_3+P_4)/4$

3D $P_{0new} = (P_1+P_2+P_3+P_4+P_5+P_6)/6$

P_3

The equations above (*used to approximate Laplace Equation solutions*) estimate the new value of a point from the ***average*** value of its nearest four (*or six - 3D*) neighbors.

If each point within the potential array is estimated with this technique (*except electrode points*), we have performed a single iteration. Each time we scan through the array again (*another iteration*) changes made in previous iterations are propagated further throughout the potential array. ***If we iterate enough times, non-electrode points change less and less between successive iterations.*** At some point, when these changes are small enough, the potential array can be said to be refined close enough for some purpose.

Over-Relaxation

If the equation above is used for refining potential arrays, a large number of iterations will be required. Over-relaxation speeds the refining process by increasing each voltage adjustment by some factor. Let's say that the new value of a point is computed to be 101 volts and its current value is 100 volts. The required adjustment would be +1 volt for simple relaxation. Over-relaxation would change this adjustment by multiplying it times a factor that ranges from 1 (*relaxation*) to 2 (*unstable over-relaxation*). ***SIMION's over-relaxation factor assumes the number one is added to it.*** Thus in SIMION, an over-relaxation factor of 0.9 applied to a +1 volt change results in a correction of:

1.9 volts = 1 * (1.0 + 0.9)

Dynamically Self-Adjusting Over-Relaxation

It turns out that for each different potential array there exists a unique over-relaxation factor that gives the fastest solution convergence. These ideal factors are generally in the range of 0.9 but they are different for each array. SIMION makes use of a dynamically self-adjusting factor to help avoid looking for the ideal over-relaxation factor. A normal SIMION refine uses the following factors:

ITERATION	OVER-RELAXATION FACTOR	
1	0.40	
2-10	0.67	
>10	F	= (F_{old} * Hist) + (1 - Hist) * (F_{max} - DF * Error)
Where:	F_{old}	= factor used in prior iteration
	Hist	= History factor (e.g. 0.70)
	F_{max}	= Max over-relaxation (e.g. 0.90)
	DF	= F_{max} - 0.40
and:	Error	= 1/(1 + (DEL/(2*DELM)))
Where:	DEL	= Max prior abs single point change
	DELM	= Convergence Goal (e.g. 0.005)

The self-adjusting characteristics use the maximum single point absolute voltage change to drive the factor. A maximum over-relaxation factor limits the maximum value, and a history factor limits the dynamic rate of change. ***As the array approaches convergence the over-relaxation factor is reduced to help smooth the final voltage estimates.***

Planar 2D Symmetry Refining Equations

The voltage change estimates for ***planar 2D non-mirroring*** potential arrays are calculated as follows:

Interior points	P_0	=	$(P_1+P_2+P_3+P_4)/4$
All corner points	P_0	=	$(P_2+P_4)/2$ **{lower left corner}**
All edge points	P_0	=	$(P_1+P_2+P_3)/3$ **{top edge}**

The voltage estimate changes for ***x mirroring*** are as follows:

LL corner point	P_0	=	$(P_2+P_2+P_4)/3$
Left edge points	P_0	=	$(P_2+P_2+P_3+P_4)/4$

The voltage estimate changes for ***y mirroring*** are as follows:

LL corner point	P_0	=	$(P_2+P_4+P_4)/3$
Bottom edge points	P_0	=	$(P_1+P_2+P_4+P_4)/4$

The voltage estimate changes for ***x & y mirroring*** are as follows:

LL corner point	P_0	=	$(P_2+P_4)/2$

Planar 3D Symmetry Refining Equations

Interior points	P_0	=	$(P_1+P_2+P_3+P_4+P_5+P_6)/6$
All corner points	P_0	=	$(P_2+P_4+P_6)/3$ **{LLL corner}**
All edge plane points	P_0	=	$(P_1+P_2+P_4+P_5+P_6)/5$ **{Bottom xz)**
All edge line points	P_0	=	$(P_2+P_4+P_5+P_6)/4$ **{LL edge line}**

The voltage estimate changes for the various x, y, and z mirroring follow from the 2D planar examples above.

Cylindrical 2D Symmetry Refining Equations

Cylindrical 2D symmetry equations reflect the fact that each of the four nearest neighbor points **do not** contribute equally to the best estimate of the central point. SIMION uses an effective area weighting function. In this approach, each of the four neighbor points contribute as a function of the area of view (*considering cylindrical symmetry*) it has with the central point.

Interior points	P_0	=	$(P_1+P_2+P_A+P_A)/4$
Lft/Rht edge pts	P_0	=	$(P_2+P_A+P_A)/3$ {Left edge points}
Interior axis pts	P_0	=	$(P_1+P_2)/6 + 2*(P_4)/3$ {y = 0}
Corner axis pts	P_0	=	$(P_2)/5 + 4*(P_4)/5$ {LL Corner}
Upper corner pts	P_0	=	$(2*y*P_2+(2*y-1)*P_3)/(4*y-1)$ {upr lft}
Top edge points	P0	=	(2*y*(P1+P2+P3) - P3)/(6*y-1)
Where:	PA	=	P3 + F*(P4-P3)
	F	=	(2+(1/y))/4
	y	=	**y position of center point in grid units**

The voltage estimate changes for x mirroring follow from the 2D planar examples above.

Skipped Point Refining

SIMION 6.0 employs a skipped point refining technique (*developed by the author*) that greatly speeds up refining. With skipped point refining active, most arrays refine in times proportional to their number of points (*as opposed to n squared for normal finite difference techniques*).

Skipped refining makes use of a rather direct approach that becomes very complex in practice. This approach is to initially refine a smaller array by skipping points, estimate the values of intermediate points (*double the array density*), and refine again. The process continues until no points are being skipped (*the final refine*). ***The best approach is to use powers of two point skipping.***

SIMION begins by looking at the size of the array to be refined. The maximum initial skipping is limited by the smallest array dimension. SIMION wants at least 4 points as a minimum dimension in the starting skip level. This means that square or cubic arrays produce the greatest initial skip factors for their size (*and usually faster refines*).

Another benefit of skipped point refining is that SIMION automatically refines the initial and next skip to orders of magnitude lower error than the user specification. This quickly finds and establishes the linear gradient fields within the array (*improving accuracy and greatly speeding refining*). ***Thus there is no need to preset linear gradients (a previous speed up trick) as in older versions of SIMION.***

Unfortunately, it is not all that simple. The big problem is that of electrode visibility. When you are skipping points you are most likely skipping electrode points here and there. If you blithely ignore this issue, the skipped point refine solutions may be quite wrong. This error will persist until the skip length reduces to the point that the troublesome electrode points are visible. Now the program has to propagate these newly visible effects around the array. ***The invisible electrode point problem typically kills any speed advantage simple minded skipped point refining has.***

SIMION 6.0 attacks this problem by scanning for and flagging any skipped electrodes at the beginning of each level of skipped point refining. Now SIMION knows which points have one or more invisible electrode points lurking near them and where they are. These special points are refined with a star weighting function that converges properly on linear gradients (*nature's preferred gradient*).

This approach combined with special compensations for irregular shifts in array boundaries as skipping changes makes for a very complex refining system. ***However, it also turns out to be dramatically faster for large arrays than the previous SIMION methods.***

Array Doubling

The **DOUBLE OPTION** works in the following way: Dummy points are inserted between each existing point. This means each initial square area of four points becomes a nine point square with the original four points acting as the corner points of the new square.

The problem is what to do with the new dummy points? Are they to be electrodes or non-electrodes? Points are typed in the following manner:

1. Scan alternate lines (*y=0, y=2, and etc.*) from left to right. The dummy point is converted to an electrode ***if and only if*** the point to the left and right are both electrode points ***of the same value***, or else the dummy point is considered to be a non-electrode.

2. Scan each column (*x=0, x=1, and etc.*) from bottom to top looking at the odd line dummy points (*y=1, y=3, and etc.*). The dummy point is converted to an electrode ***if and only if*** the points below and above are both electrode points ***of the same value***, or else the dummy point is considered to be a non-electrode.

Note: If Interpolated Electrodes Mode is active and any new point has neighboring electrode points of different values the new point will be an electrode of the average value of its neighboring electrode points.

SIMION handles 3D arrays in an equivalent manner.

Note: New non-electrode points are always set to zero. The doubled array must be refined after doubling to be usable.

Fast Voltage Adjustment Method

The fast voltage adjustment method used in SIMION makes use of the additive solution property of the Laplace equation. This property allows separate solutions for each boundary (*or collections of boundary elements*) to be combined in a simple scaling/additive process to obtain the potential at each point in the potential array.

For example, if a potential array has five electrodes we could create and refine five potential arrays (*one for each electrode*). Each array would be the specific solution of the Laplace equation for a particular electrode. In potential array number one, all electrode points ***except*** those associated with electrode number one would be set to zero volts. The electrode points representing electrode number one would be set to the desired voltage for electrode number one. When this array was refined it would represent the specific solution for the effects of electrode one with the selected voltage.

The remaining four electrode potential arrays are created in a similar fashion. The composite potential would be computed as follows:

Composite potential:

$$P_n = P_{n1} + P_{n2} + P_{n3} + P_{n4} + P_{n5}$$

Where:

n = **Any common array location**

P_{n1} = **Electrode one's solution array**

SIMION includes an additional potential array, the base array, to support the fast adjustment process. This array initially *only* contains the solutions for any electrode points that do not have their own individual potential arrays. **If all electrode points are associated with a particular individual solution potential array then all the points of the base array are initially at zero volts.**

Fast adjustment works by using each reference electrode potential file and a scaling factor to change the non-electrode point voltages in the base potential array:

Voltage Adjustment:

$$V_{new} = V_{old} + F * V_{ref}$$

Where:

V_{new} = **Adjusted point voltage - base array**
V_{old} = **Current point voltage - base array**
V_{ref} = **Current point voltage - Elect array**
F = $Ve_{new}/Ve_{ref} - Ve_{old}/Ve_{ref}$
Ve_{new} = **New electrode voltage**
Ve_{old} = **Current electrode voltage**
Ve_{ref} = **Reference electrode array voltage**

The voltage adjustment is accomplished by loading portions of a reference electrode array and using the portion's points to adjust the equivalent points in the base potential array. During this process all electrode points of the specific electrode are changed to their new value. This process continues one electrode file at a time until the required adjustments have been made.

Trajectory Algorithms

Ion trajectory calculations are a result of three interdependent computations. First, electrostatic, magnetic, and charge repulsion (*if active*) forces must be calculated based on the current position and velocity of the ion(s). These forces are then used to compute the current ion acceleration and used by numerical integration techniques to predict the position and velocity of the ion at the next time step. Moreover, the time step itself must be continuously adjusted to maximize trajectory accuracy while minimizing the number of integration steps per trajectory.

Instance Selection Rules

Unlike previous versions, SIMION 6.0 can have up to 200 potential array instances in its workspace at one time (*instead of just one*), and it must decide which instance to apply at each trajectory calculation point. This problem is solved by keeping instances in an ordered list. SIMION searches down this list from the last instance on the list (*highest priority*) toward the first (*instance 1 - lowest priority*). SIMION will always use the first electrostatic (*and magnetic if defined*) instance(s) it encounters while search down the instance list that contains ion to compute the forces on the ion.

SIMION's Unit System

SIMION uses the following internal unit system:

Time	=	**microseconds**
Length	=	**mm**
Position	=	**mm**
Velocity	=	**mm/ microsecond**
Acceleration	=	**mm/ microsecond^2**
Elect Gradient	=	**volts/ mm**

Electrostatic Force Computation

Electrostatic forces are initially computed in terms of volts per grid unit. As the ion progresses through the potential array it moves from within one square of grid points into another. SIMION automatically generates a small 16 point array that represents the current four point grid and the 12 grid points around it (*64 points for 3D arrays*) . If the ion happens to be in a boundary grid location SIMION estimates the potential of any grid point that falls outside the current potential array. The values of these grid points are determined by symmetry assumptions and grid point location.

P_4

P_1 P_{ION} P_2 **One <u>half</u> grid unit separations**

P_3

Voltage gradients are calculated by first estimating the potentials at four points (*six points for 3D*) as pictured above. These points are exactly 0.5 grid units from the current ion position.

The potential at each point is normally calculated via linear interpolation using the four grid points bounding the grid square it falls in (*8 point cube for 3D*). However, electrode boundaries present a problem.

If one of the four points (*6 points 3D*) has an electrode boundary between it and the ion position, SIMION adjusts its calculations. In this case the value of the point in question is calculated by assuming a straight line extrapolation using the boundaries of the ion's grid square (*assuming continuation of the current gradients*). ***This correction improves SIMION's ability to model fields near electrodes and ideal grids.***

Electrostatic Forces:

Planar 2D

$F_x = V_1 - V_2$ {sign corrected}
$F_y = V_3 - V_4$ {sign corrected}
$F_z = 0$

Planar 3D

$F_x = V_1 - V_2$ {sign corrected}
$F_y = V_3 - V_4$ {sign corrected}
$F_z = V_5 - V_6$ {sign corrected}

Cylindrical 2D

$F_x = V_1 - V_2$ {sign corrected}
$F_r = V_3 - V_4$ {sign corrected}
$F_y = F_r * y/r$ {sign corrected}
$F_z = F_r * z/r$ {sign corrected}

Where:

V_n = potential (in volts) at P_n
r = sqrt(y*y + z*z)

Electrostatic Acceleration:

Now Transform E Forces into workbench orientation (Forces initially calculated in PA orientation)

$A_x = F_x / (1.03642722 \times 10^{-2} * M * S)$
$A_y = F_y / (1.03642722 \times 10^{-2} * M * S)$
$A_z = F_z / (1.03642722 \times 10^{-2} * M * S)$

Where:

M = Ion's relativistic mass (amu) / unit charge
S = Scale in millimeters/grid unit

Electrostatic Forces Outside Instances

When an ion is outside all electrostatic instances SIMION looks both directions along its current trajectory path for the closest electrostatic instance of intersection in both directions. If the current trajectory intersects electrostatic instances in both directions, SIMION will determine the potentials at the points of intersection and estimate the resulting electrostatic acceleration (*if any*) assuming a linear gradient. ***Note: Charge repulsion calculations are not effected by the ion's instance (or non-instance) status.***

Magnetic Force Computation

Magnetic forces are calculated (*if magnetic instances are defined*) by searching the list of instances to see if the ion is presently within a magnetic instance (*searching last to first in instance list*).

Magnetic Forces:

Planar 2D

$B_x = ng * (P_1 - P_2)$ {sign corrected}
$B_y = ng * (P_3 - P_4)$ {sign corrected}
$B_z = 0$

Planar 3D

$B_x = ng * (P_1 - P_2)$ {sign corrected}
$B_y = ng * (P_3 - P_4)$ {sign corrected}
$B_z = ng * (P_5 - P_6)$ {sign corrected}

Cylindrical 2D

$B_x = ng * (P_1 - P_2)$ {sign corrected}
$B_r = ng * (P_3 - P_4)$ {sign corrected}
$B_y = F_r * y/r$ {sign corrected}
$B_z = F_r * z/r$ {sign corrected}

Where:

ng = user adj magnetic scale factor
P_n = potential (in Mags) at P_n
r = sqrt(y*y + z*z)

Magnetic Acceleration:

Now Transform B field into workbench orientation
(Forces initially calculated in PA orientation)

$A_x = C * (V_yB_z - V_zB_y) / M$
$A_y = C * (V_zB_x - V_xB_z) / M$
$A_x = C * (V_xB_y - V_yB_x) / M$

Where:

C = $9.648453082 \times 10^{-3}$
V_x = Ion's velocity in workbench axis direction
M = Ion's relativistic mass (amu) / unit charge

Charge Repulsion Forces

SIMION supports three estimates of charge repulsion: Beam, coulombic, and factor. ***The basic approach is to let a few ions represent many.*** In general, charge repulsion estimations involve determining the forces between the current ion in question and all other currently flying ions. These forces are then scaled by a charge scaling factor and accelerations are obtained due to each interaction. The sum of the accelerations represent the estimate of charge repulsion effects on the current ion.

Beam Repulsion

Beam repulsion is estimated via line charge assumptions. Each ion is assumed to represent a line charge. The line charge density coulombs/mm is determined by apportioning the beam current between the ions according to their charge adjusted by their charge weighting factor and dividing it by the ion's velocity:

coulombs /mm = (coulombs/usec) / (mm / usec)

Beam repulsion requires that all ions be flown via ***space coherent integration.*** This is accomplished by using ion number one as leader of the pack. At each time step for ion number one a plane containing the ion and normal to its velocity is computed. This plane is then used to control the time steps of all other ions so that they will fall within the plane too. The following calculates the charge repulsion between to representative weighted ions.

Beam Repulsion Acceleration:

re	=	r/r_{avgmin}
rfactor	=	$re^2 / ((0.341995 + re^3) * r_{avgmin})$
scale	=	$-1.73433341x10^{+9}$ * Amps * chg /(m * total_chg)
accel	=	scale * tchg * tcwf * rfactor / tv

Acceleration components computed using unit vector for r between the two ions

Where:		
amps	=	total beam amps
chg	=	charge on current ion (real charge)
m	=	relativistic mass of current ion (amu)
total_charge	=	sum of all ion's cwf * their absolute charge
tchg	=	test ion's charge (real charge)
tcwf	=	test ion's charge weighting factor
r	=	distance between the two ions
r_{avgmin}	=	current average min distance between ions
tv	=	speed of the test ion

Coulombic Repulsion

Coulombic repulsion is estimated via point charge assumptions. Each ion is assumed to represent a point charge. The total coulombs of charge is apportioned between the ions according to their charge adjusted by their charge weighting factor. Coulombic repulsion requires that all ions be flown in ***time coherent integration.***

Coulombic Repulsion Acceleration:

re	=	r/r_{avgmin}
rfactor	=	$re^2 / ((1.0 + re^4) * r_{avgmin})$
scale	=	$-8.67166704x10^{+14}$ * C * chg /(m * total_chg)
accel	=	scale * tchg * tcwf * rfactor

Acceleration components computed using unit vector for r between the two ions

Where:		
C	=	total coulombs of charge
chg	=	charge on current ion (real charge)
m	=	relativistic mass of current ion (amu)
total_chg	=	sum of all ion's cwf * their absolute charge
tchg	=	test ion's charge (real charge)
tcwf	=	test ion's charge weighting factor
r	=	distance between the two ions
r_{avgmin}	=	current average min distance between ions

Factor Repulsion

Factor repulsion is estimated via point charge assumptions. Each ion is assumed to represent a point charge. Each point's effective charge is determined by multiplying its charge by the charge multiplication factor adjusted by its charge weighting factor. Factor repulsion requires that all ions be flown in *time coherent integration*.

Factor Repulsion Acceleration:

re	=	r/r_{avgmin}
rfactor	=	$re^2 / ((1.0 + re^4) * r_{avgmin})$
scale	=	$-1.389354835 \times 10^{-4} * F * chg / (m * avg\_cwf)$
accel	=	scale * tchg * tcwf * rfactor

Acceleration components computed using unit vector for r between the two ions

Where:		
F	=	charge multiplication factor
chg	=	charge on current ion (real charge)
m	=	relativistic mass of current ion (amu)
avg_cwf	=	average all ion's charge weighting factors
tchg	=	test ion's charge (real charge)
tcwf	=	test ion's charge weighting factor
r	=	distance between the two ions
r_{avgmin}	=	current average min distance between ions

Corrections for Ion Clouds

In each of the three repulsion types above each ion is actually representing a cloud of ions. If for some reason an ion found itself in the middle of the cloud of another ion it should have no forces on it. However, if the standard $1/r^2$ (*Factor and Coulombic repulsion*) or $1/r$ (*Beam repulsion*) distances were to apply then forces would be infinite and everything would blow up.

SIMION compensates for this problem by using radius factors (*rfactor above*) that are $1/r^2$ or $1/r$ if r is large and diminish to zero as r approaches zero. The method used closely models the effect of having

the ion near a cloud of ions (*spherical - Coulombic and Factor, or cylindrical - Beam*) with an effective radius of r_{minavg}.

The value of r_{minavg} is set to the average minimum distance between all currently flying ions. SIMION updates this value each time step. Thus the effective radius of the ion clouds change as the ions move about. ***The single exception is when Factor repulsion is set to 1.0.*** Then SIMION ***always*** uses 3.0e-11 mm (*10 times the classical electron radius*) for the effective cloud diameter.

Numerical Integration Method

A standard fourth order Runge-Kutta method is used for numerical integration of the ion's trajectory in three dimensions. This approach has the advantage of good accuracy and the ability to use continuously adjustable time steps. This, to some extent, compensates for the added time required to calculate accelerations at four points for each time step.

Adjustable Time Steps

A good self-adjusting time step method is just as important as the interpolation and integration methods. Each depends on the other if reasonably accurate trajectories are to be calculated in a minimum of computer time.

Large time steps work well in regions of the potential array where voltage gradients are highly linear and have a minor impact the ion's rate of change in kinetic energy. **However in other areas it is very critical to have a small time step to maintain accuracy in high gradient areas. If the time step is significantly too long an ion can overshoot its energy null point (*e.g. reflector fields*) and gain enough false kinetic energy to gradually make the subsequent trajectory meaningless.**

SIMION utilizes a two step process to estimate an appropriate time step for each integration step. Before entering the first step of Runge-Kutta integration, the total velocity and acceleration terms are calculated at the current ion location.

The user has previously entered a value for the largest distance step permitted. **The default (*and maximum*) value for this distance step is one integration step per grid unit of length.**

This distance step is used with the current ion velocity to estimate the time step assuming zero acceleration. Likewise the distance step is used with the current ion acceleration to estimate the time step assuming zero initial velocity. The composite estimate is obtained by using a parallel effect assumption (*to avoid iterative solution techniques*).

Longest time step computation:

$$
\begin{aligned}
t_v &= d / v \\
t_a &= \mathrm{sqrt}(2 * d / a) \\
t_{step} &= t_v * t_a / (t_v + t_a) \quad \{a!=0,\ v!=0\} \\
t_{step} &= t_v \quad \{a = 0\} \\
t_{step} &= t_a \quad \{v = 0\}
\end{aligned}
$$

Where:

d	=	**Maximum distance step**
v	=	**current ion velocity**
a	=	**current ion acceleration**

Notice that t_{step} is calculated three ways depending on the presence of velocity and acceleration.

If both acceleration and velocity are present a second computation is performed to see if the time step should be shortened. A quantity called stop length is computed. Although it seldom indicates the actual ion's stopping distance (*e.g. magnetic accelerations*) it is a very good indicator of the rate of trajectory curvature. The stop length if shorter than 10 grid units is used to compute a reduced time step.

Reduced time step:

$$
\begin{aligned}
S_{dist} &= v * v / (2 * a) \\
\text{If } S_{dist} > 10 \quad & t_{step} = t_{step} \\
\text{If } 1 < S_{dist} < 10 \quad & t_{step} = t_{step} * S_{dist} / 10 \\
\text{If } S_{dist} < 1 \quad & t_{step} = t_{step} / 10
\end{aligned}
$$

Thus the normal time step can be as much as 10 times shorter if the ion is in a region of high trajectory curvature. This method appears to be quite successful in maintaining accuracy while minimizing total integration steps.

Edge Detection and Boundary Approach

Two other problems must also be considered. First, SIMION's trajectory algorithms are quite blind. They can serenely propel an ion right over a sharp gradient edge (*electrostatic or magnetic*) and detect it only after the fact. This can create all sorts of problems with energy conservation and thus compromise the accuracy of ion trajectories. Second, once a boundary (*of whatever type*) has been detected (*by whatever method*) how do you efficiently approach it (*being basically blind*).

Binary Boundary Approaches

SIMION 6.0 makes use of a binary approach method. SIMION looks ahead one time step to see if a boundary has been crossed. If so, the integration time step is cut in half and the integration calculation step is repeated. This process continues until the boundary is no longer crossed. At the next integration calculation SIMION remembers that a boundary exists and starts off with the last successful time step and halves it as necessary to avoid the boundary. Of course this cannot continue indefinitely because the boundary would never be crossed. SIMION limits the smallest time step to 0.0001 of the stop length corrected time step. When this limit is reached the boundary is crossed whatever the eventuality.

SIMION's boundary detection involves the tentative calculation of the next ion position. If any of the four Runge-Kutta terms blow up (*hits an electrode or is outside of grid*) a warning flag is set and if the time step is above the lower limit the time step is halved and the process loops.

Field Curvature Detection

The next boundary test is the gradient of acceleration. The coefficient of variation squared is calculated for the four Runge-Kutta Ax, Ay, and Az acceleration terms. If the C_v for any acceleration term is above 1/(accuracy level) (*typically 1/2 or 0.5*) the binary approach is used until all values are less than the upper limit or the minimum time step size has been reached.

Likewise, when the C_v is less than 1/2 the upper limit SIMION doubles the time step a step at-a-time until the stop length time is reached or a higher Cv forces another time step reduction. This reduces errors on sharp edges (*grids and magnet boundaries*) and in higher field curvature areas.

SIMION has additional looping triggers for electrodes, grid boundaries, velocity reversals, and other events. Together these algorithms provide SIMION with accuracy improvements while minimizing the number of integration steps.

How Trajectory Quality is Controlled

SIMION uses a **Trajectory Computational Quality** panel (*Normal Controls Screen - accessed via the* **Normal** *tab*) to control how trajectories are calculated. The default value for this parameter is 2.

Quality < Zero

Provides control of relative time steps. The step distance is set to 1/(1 + abs(quality)) of the normal stop length time step. CV, boundary, and reversal checking are turned off.

Quality set to Zero

This is provides the fastest calculations at the expense of accuracy. Time steps are adjusted to move the ion the normal stop length time step (*typically one grid unit*). CV, boundary, and reversal checking are turned off.

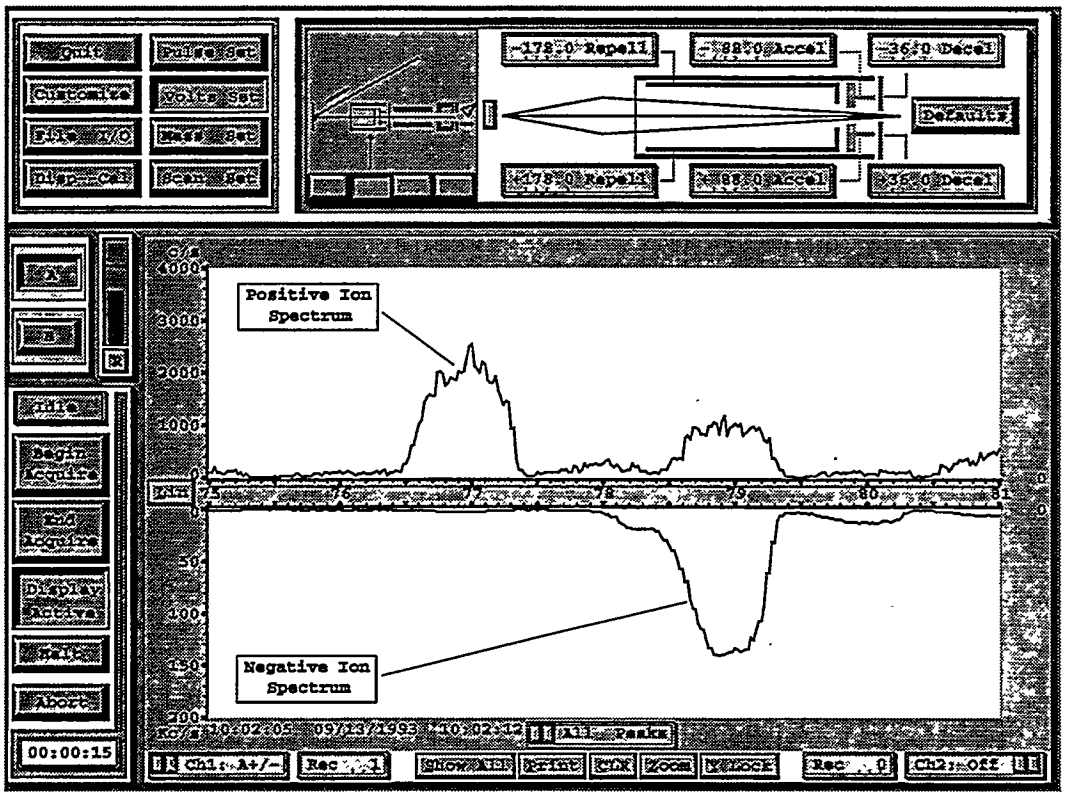
Quality > Zero and < 100

Time steps are adjusted to move the ion the normal stop length time step (*typically one grid unit*). ***CV, boundary, and reversal checking are turned on.*** CV limits are based on 1.0/quality.

Quality > 100

Provides control of relative time steps too. The step distance is set to 1/(1 + abs(quality -100)) of the normal stop length time step. ***CV, boundary, and reversal checking are turned on.*** CV limits are based on 1.0/quality.

SIMION's GUI



Introduction

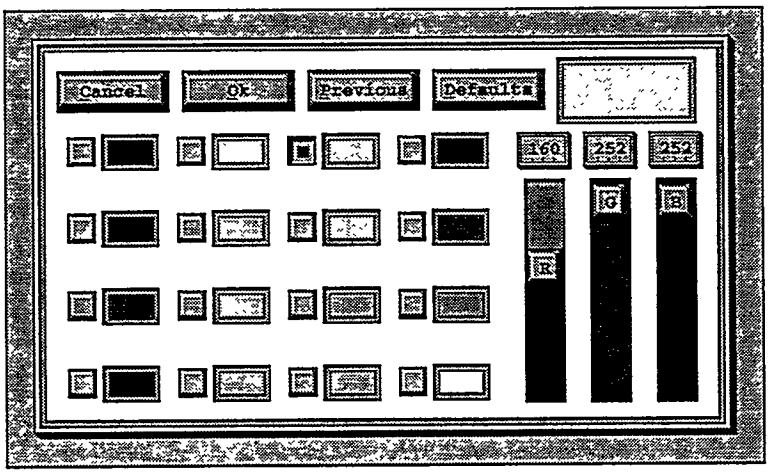
SIMION makes use of a *G*raphics *U*ser *I*nterface - ***GUI*** originally created by the author for developing data systems (*as illustrated above*). The GUI has gradually evolved into a powerful platform for applications like SIMION 6.0.

Nevertheless, the decision to place SIMION 6.0 on the GUI's platform was a difficult one. In the final analysis the author felt that the GUI's performance (*very high speed video*), features (*e.g. full 32 bit virtual memory DOS Protected Mode Interface - DPMI*), and stability (*stand alone DOS and Windows compatibility*) made it a prudent choice.

Whatever! This appendix is devoted to acquainting you with SIMION's GUI and its many impressive features.

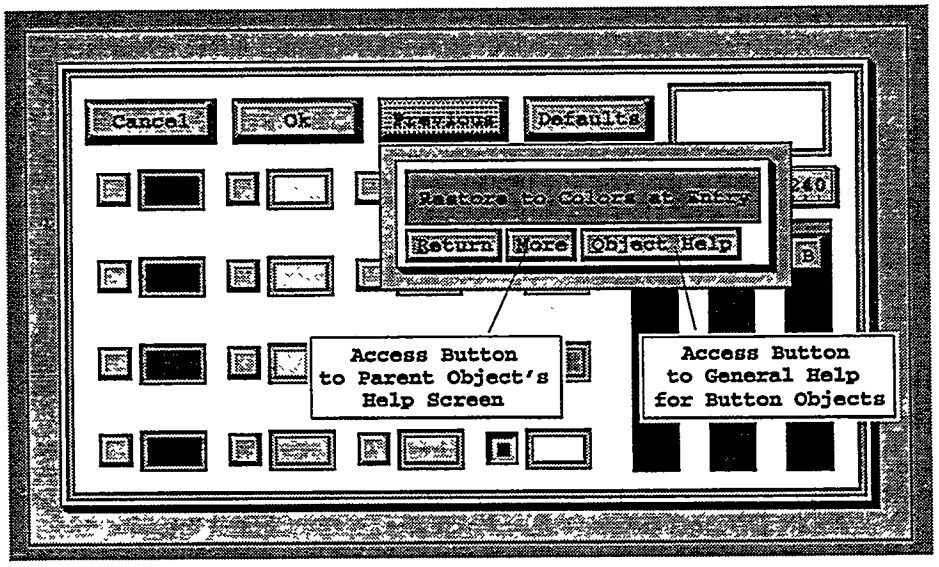
The Philosophy of SIMION's GUI

SIMION's GUI represents a departure from where most current windowing GUI's are today, but it also appears to point in the general direction that GUI's are evolving. The philosophy behind this GUI can be summarized as follows: ***Make it familiar, easy to learn, easy to use, and powerful.*** The illustrations below serve to demonstrate this philosophy:



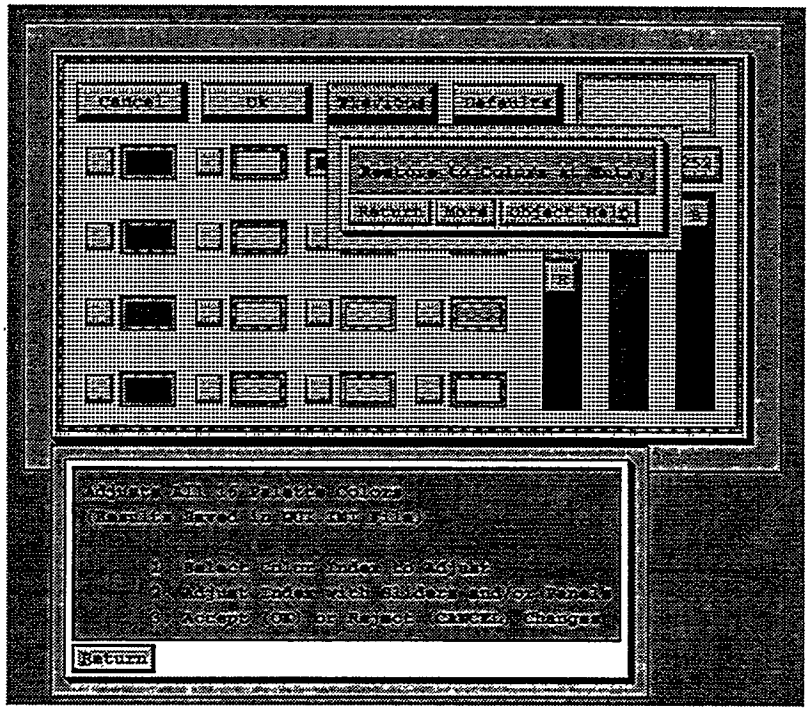
First: Make it Familiar

The GUI makes ***extensive use of familiar objects*** like ***buttons***, ***sliders***, and the like. These objects are combined to ***resemble the familiar interfaces of the electronics devices*** used around us.



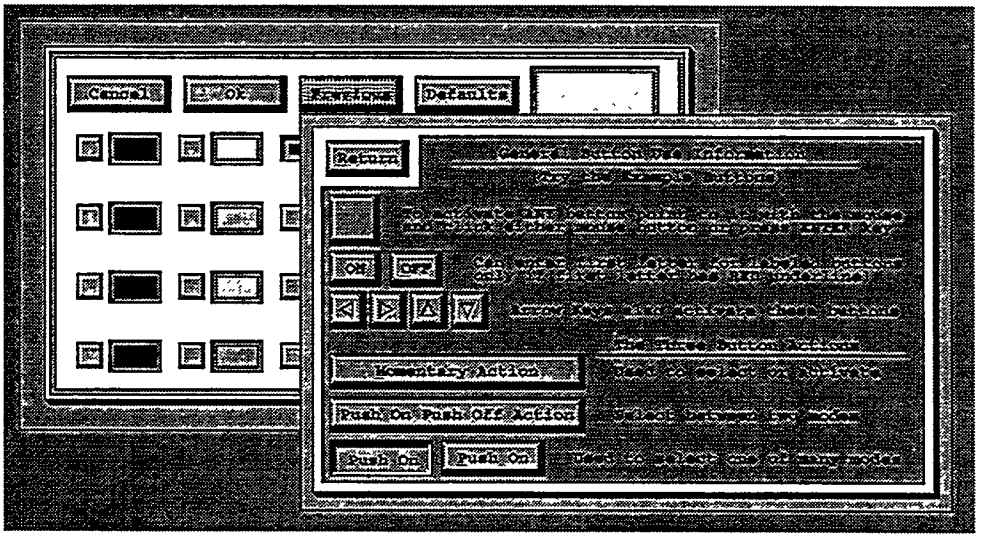
Second: Make it Easy to Learn

Since people don't typically read manuals, <u>each</u> object on the screen has its own help screen to explain its function and use. Just point the cursor at an object and press the **<F1>** help button. Most objects also provide additional levels of help *via the* **More** *and* **Object Help** *buttons on their help screen.* The illustrations below are examples of using these additional buttons to gain access to these additional levels of help:



The More Button

The **More** button (*when provided*) gives direct access to the help screen of the object's parent (*and so on up the parental hierarchy*). SIMION's ***help strategy*** is designed to go from the ***specific to the general*** (*a more natural direction*).



The Object Help Button

The **Object Help** button (*when provided*) accesses a ***help screen for the selected object class*** (*e.g. button class*). The object class help screen explains the features of the class of objects and how to use them. ***Usable example objects of the class are provided on the help screen to serve as learning aids*** (*victims*).

Third: Make it Easy to Use

Many features like: ***Esc key aborting, assisted mouse pointing,*** and ***direct keyboard access*** capabilities assist in the GUI's ease of use:

The Esc Key - Your Yellow Brick Road

If you're lost or just want to stop something in progress (*e.g. a printout*), hit the <Esc> key. ***Each time you press the <Esc> key the GUI will stop what it's currently doing and back out a layer.*** Of course, there is built in protection to prevent you from aborting right out of something important (*e.g. like the program itself*).

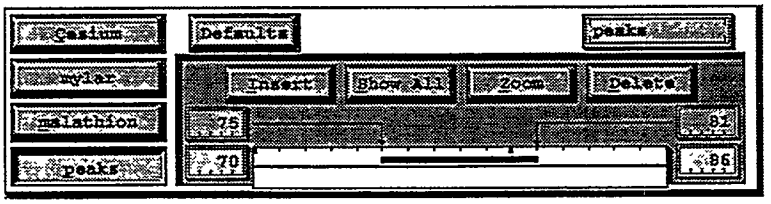
Assisted Mouse Pointing

The GUI knows where the objects are on your screen and assists you in pointing to them. Visual and audio cues (*object blinking, cursor changes, and sounds*) are provided to further reinforce pointing feedback.

In query screens, the ***GUI moves the cursor to the default object*** (*or answer button*), ***and then returns the cursor to its starting point*** when the query screen is removed.

If your inputs are currently limited to some special screen, ***the GUI will automatically limit (lock) your cursor's movements*** to that area. This prevents you from *falsely* assuming that you might have access to objects outside the locked area.

The result is easier and faster program access via the mouse.



Direct Keyboard Access to Buttons

While mice may be *nice, interfaces that also support simple keystroke entry provide the fastest program access for the experienced user.*

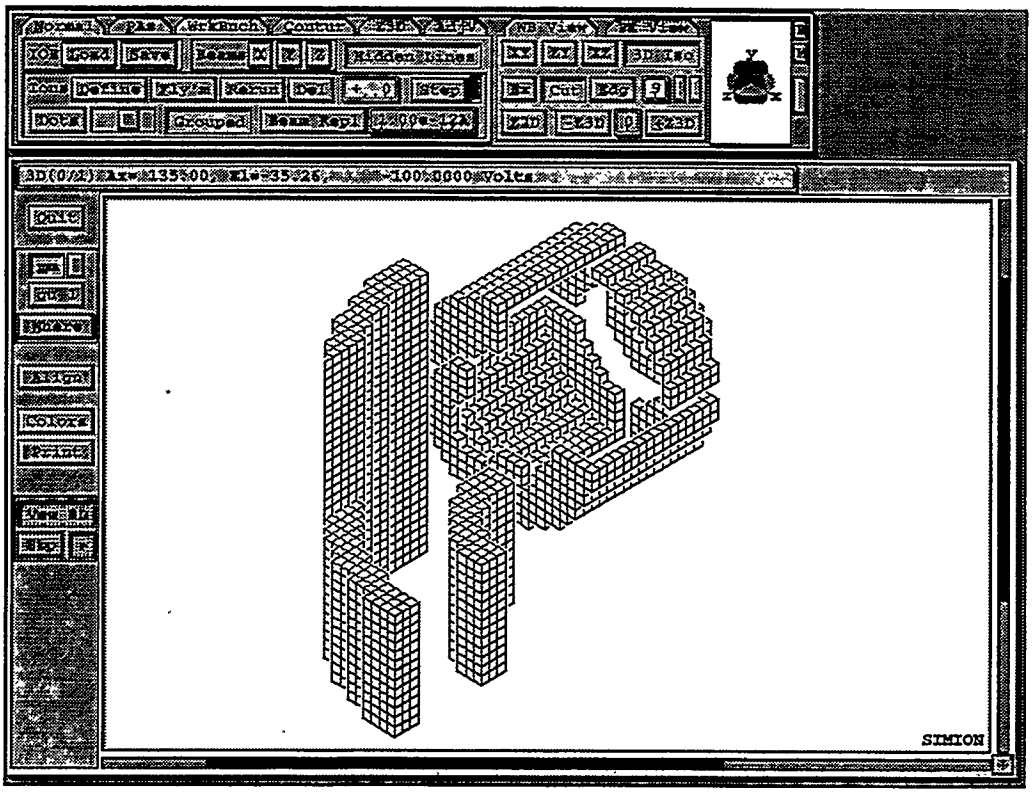
In previous versions of SIMION a potential array was saved by pressing the <S> key (*for Save command*) entering the name of the file and then pressing <Enter> to save it.

The same keystrokes can be used in SIMION 6.0. This is because many (*though not all*) labeled buttons can be directly accessed by entering the first letter of their label on the keyboard (*e.g.* <S> *for* Save).

Whenever a labeled button is currently keyboard accessible the GUI will automatically place a red underline below the first letter of the button's label.

Keyboard accessibility of labeled buttons depends on their order of creation and on the location of the mouse cursor. The most recently created buttons (*on highest level screen*) have the highest priority for direct keyboard access. ***However, if the cursor is in an ioline object (e.g. used for entering a filename), all keyboard entries are processed by the ioline object.***

Whenever a labeled button is not currently keyboard accessible the GUI will automatically remove the red underline from its first letter (e.g. when the cursor enters an ioline object).



Fourth: Make it Powerful

The GUI includes innovative features like ***one button windows, trapped cursor scrolling,*** and ***bi-directional memory zooms*** that really help make SIMION 6.0 easy to use.

One Button Windows

While other GUIs control their windows with lots of little objects *controlling horizontal and vertical motions separately*, this GUI uses just *one* button for ***full bi-directional control.***

Drag this button with the **left** mouse button depressed and you have ***full 2D scrolling capability.*** Hold down the **<Ctrl>** key while you drag the mouse and the ***view dynamically zooms.*** Hold down the **<Alt>** key while you drag the mouse for ***true 3D pointing and volume cutting*** capabilities in SIMION.

Drag this button with the **right** mouse button depressed and you have all the functions above in ***full page*** (*world*) view.

Trapped Cursor Scrolling

If you hold down the **<Ctrl>** key when the cursor is within the view area of a window the window will automatically scroll the view when the cursor hits a view boundary (***2D demand scrolling***).

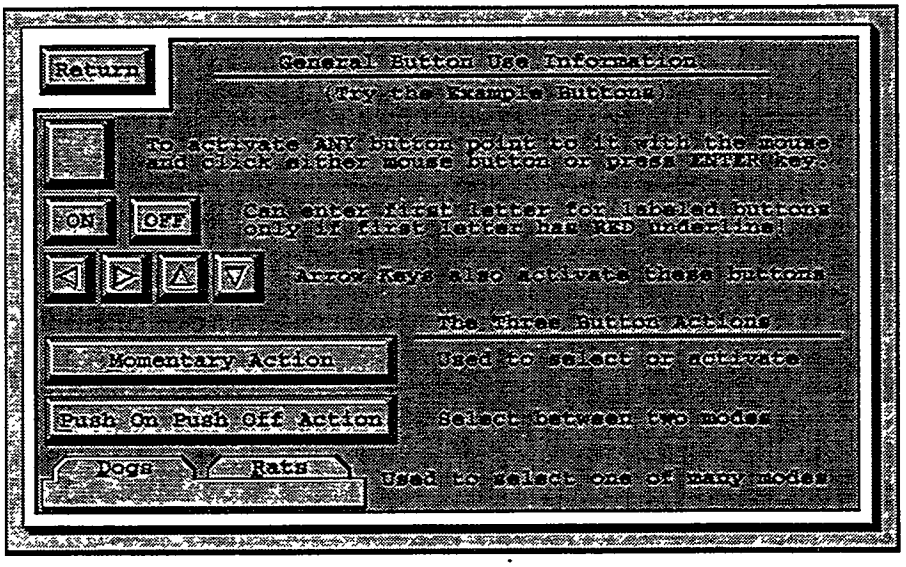
Quick Bi-directional Memory Zooms

To zoom in, just ***mark the desired zoom area*** by dragging in the view area with the **left** mouse button depressed and then click the **right** mouse button to zoom. If an ***area is not marked, clicking the right mouse button*** in the view area will ***zoom out*** to the prior zoom level (*up to 10 levels remembered*). However, if either **<Shift>** key is ***depressed*** clicking the **right** mouse button with no area marked will ***zoom in*** to the next zoom level (*if defined*).

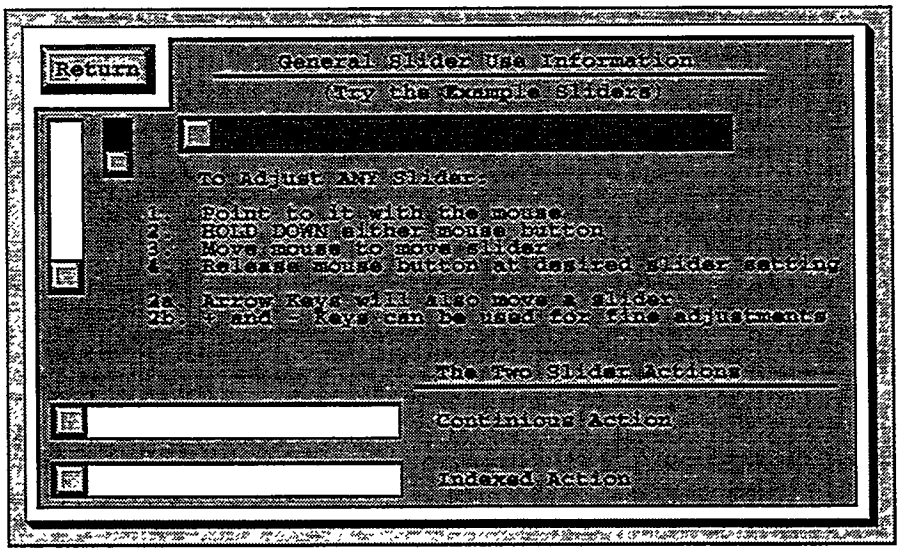
Standard GUI Objects

The GUI screen is composed of layers of objects. Some of these objects are simple surfaces, others are standard functional objects like buttons, while others are custom objects designed for special tasks within an application (*e.g. the view orientation control sphere in SIMION's view function*). The discussion here will be limited to standard functional objects: ***Buttons, Sliders, Iolines, Panels, Selectors, and Windows***. Each object's general help screen will serve as the basis for explaining it:

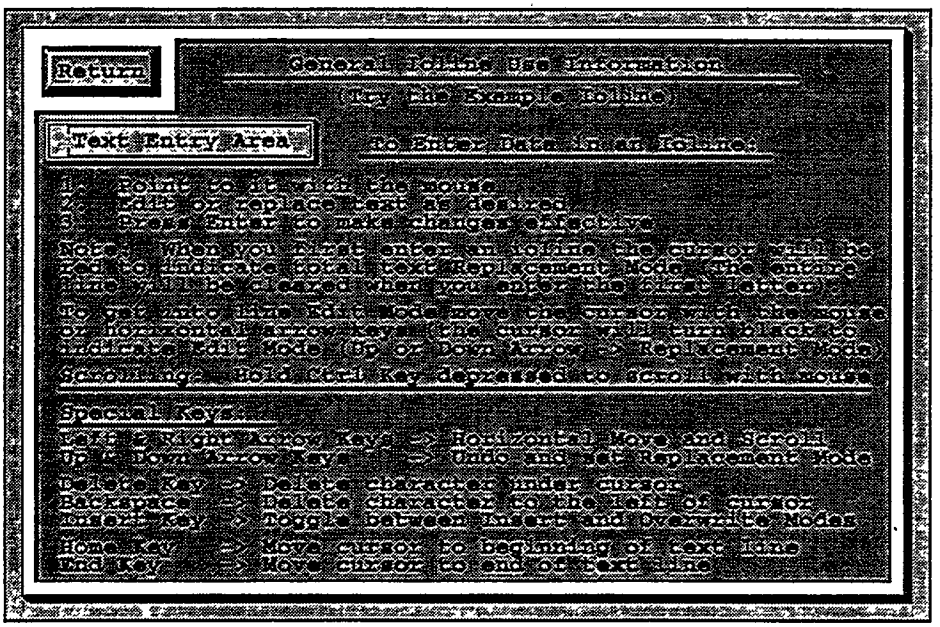
Button Objects



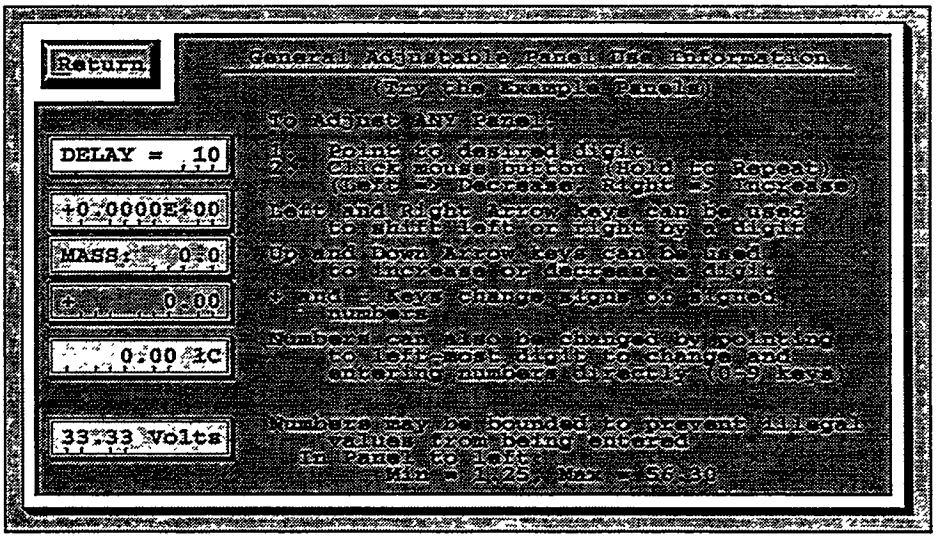
Slider Objects



Ioline Objects



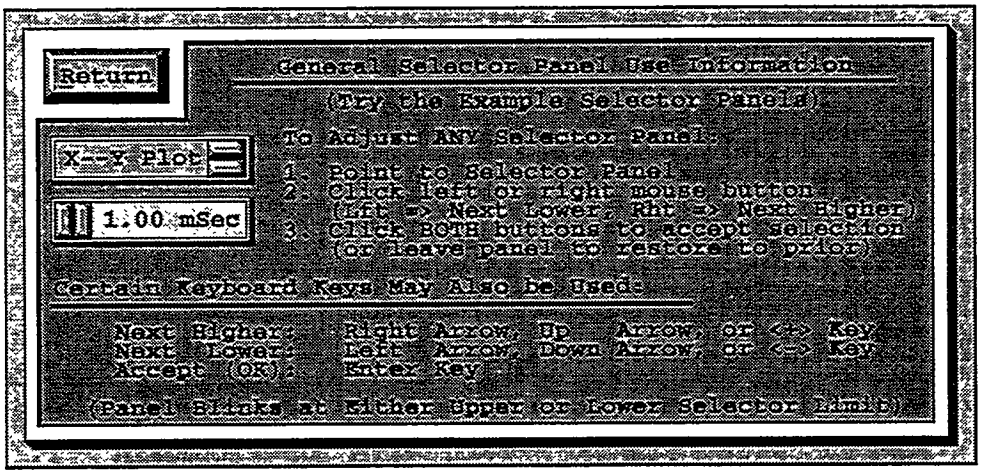
Panel Objects



The Two types of Panel Objects

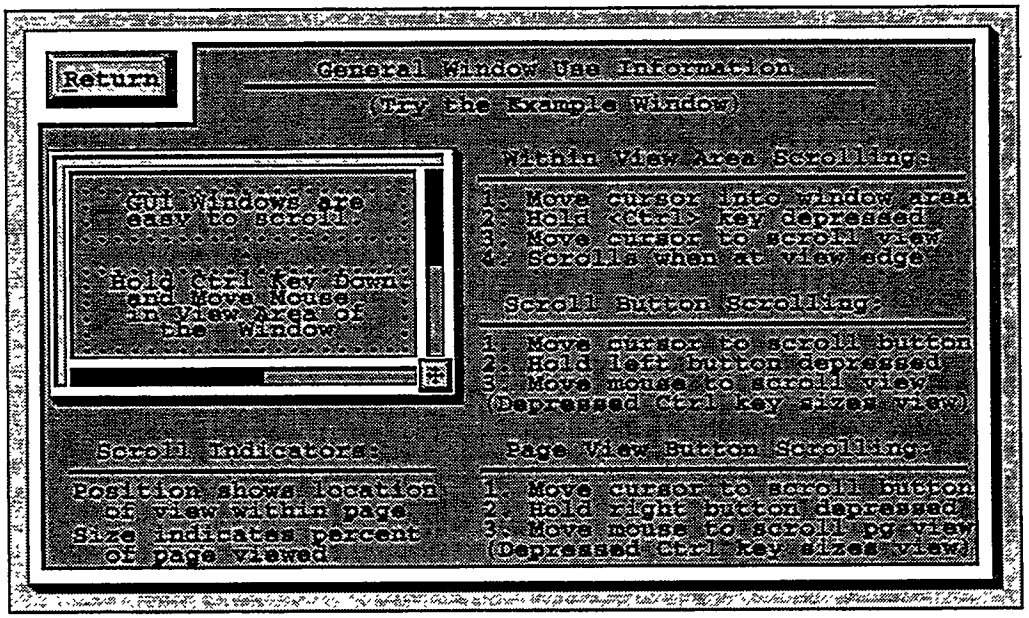
There are really two types of panel objects: ***Adjustable and non-adjustable.*** The ***adjustable*** variety have ***small vertical lines below each digit*** that can be adjusted (***as in the example adjustable panels above***). Non-adjustable panels are used to display strings of text (***e.g. status and warning indicators***).

Selector Objects



Note: Some selector objects automatically accept the selection (*e.g. Print mode selector*). ***Automatic acceptance occurs to speed up the process and when no adverse consequences would occur from user fiddling with the selector.***

Window Objects



Note: The GUI uses line drawing characters in the IBM extended character set to draw text boxes. Unfortunately, PostScript doesn't support these characters in its extended character set. ***Thus you see '.' (periods) for the line drawing characters in the illustration above.***

Normal Verses Page View

The GUI window button in both text and graphics windows provides access to two views: ***Normal view*** and ***page view***. ***Scrolling***, ***Zooming***, and ***3D pointing*** function in either view.

Normal View - (Current Position and Magnification)

Is accessed by activating the window button with the **Left** mouse button.

Page View - (World View)

Is accessed by activating the window button with the **Right** mouse button. The area of the normal view is shaded or outlined. ***Page view is useful for fast scrolls and zooms.***

Text Windows (text only windows)

Scrolling Text Windows

Scrolling is supported if the page area is larger than the visible area.

Window Button Scrolling: Depress (*and hold down*) the window button and move the mouse in the direction you want the view to scroll.

Cursor-In-View Demand Scrolling: Hold **<Ctrl>** key depressed when cursor is in window's view area. View will scroll when cursor hits a view boundary.

Graphics Windows (lines and other graphic elements)

Scrolling Graphics Windows

Scrolling is supported as in text windows (*above*).

Zooming Graphics Windows

Window Button Zooming: Hold **<Ctrl>** key depressed and depress (*and hold down*) the window button. Move the mouse ***away to zoom out, inwards to zoom in.***

Cursor-In-View Zoom in: Mark area to zoom into using **left** mouse button. Click **right** mouse button to zoom in (*ten level memory zooming supported*).

Cursor-In-View Zoom out: Don't mark area. Just click **right** mouse button to zoom out.

Cursor-In-View Zoom back in: Don't mark area. Hold either **<Shift>** key depressed and click **right** mouse button to zoom in (*if defined*).

3D Pointing (Isometric 3D Views only)

3D Isometric Cut Plane Control: Hold the **<Alt>** key depressed and depress (*and hold down*) the window button. Move the mouse in the ***appropriate isometric direction*** to select an isometric cut plane and move it in the desired direction.

3D Cursor-In-View Pointing: Hold the **<Alt>** key depressed with cursor in window's view area. Move the mouse (*with* **left** *button depressed*) in the ***appropriate isometric direction*** to move cursor in 3D and define the inner volume.

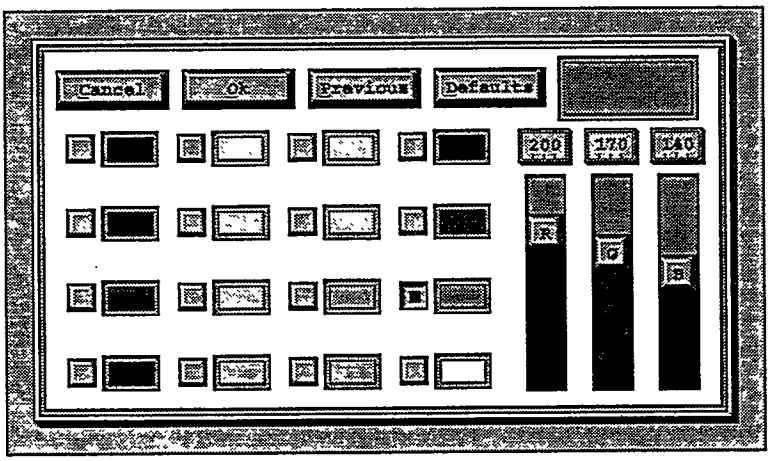
Quick Draw Scrolling, Zooming, and 3D Pointing

You can ***temporarily*** tell the GUI to eliminate (*or at least minimize*) redrawing. To activate ***quick draw*** just depress the ***other*** mouse button during the ***scrolling***, ***zooming*** or ***3D pointing*** process.

GUI Customizing Screens

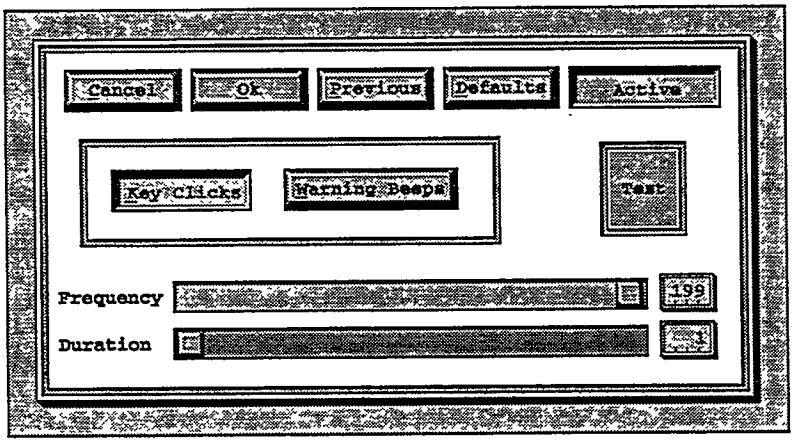
The GUI has five customizing screens that allow you to change ***colors***, ***sounds***, ***delays***, ***other items***, and ***video resolution.*** In SIMION, you gain access to these screens via the **Adjust** button on the main screen. ***Any changes you make are retained (between sessions) in the*** **C:\FILES.GUI\GULINI** ***file.***

Color Customizing Screen



This screen sets the 16 primary colors used by the GUI. The SIMION **Modify** and **View** functions as well as the Print Options Screen provide a ***Color*** button to allow you to change colors for viewing and printing preferences (*e.g. for emphasis*).

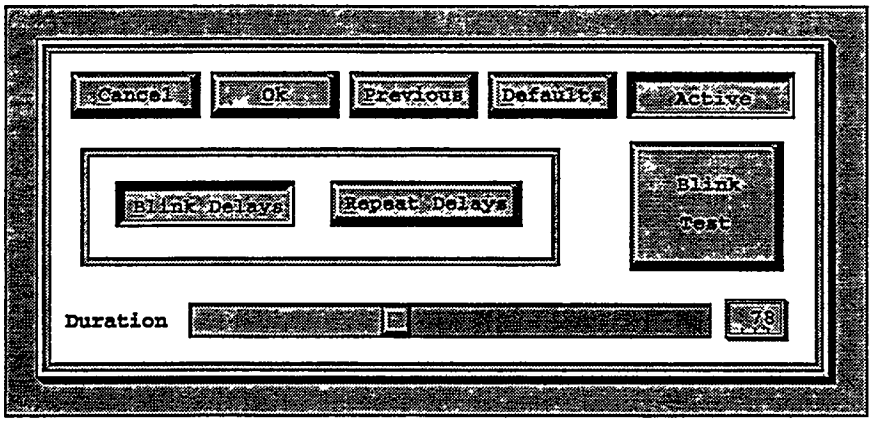
Note: The three shades of brown (*rightmost column*) are used for simulating surface illumination in 3D isometric views.



Sound Customizing Screen

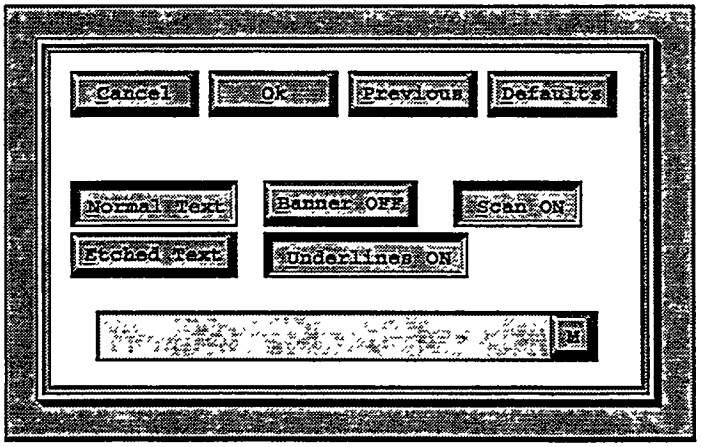
The ***sound customizing*** screen is used to adjust the ***warning beeps*** and ***key clicks***. You can adjust each for ***frequency*** and ***duration***. Moreover, there is an **Active** button (*upper right corner*) that can be used to turn off clicks and/or beeps.

Delay Customizing Screen



The delay screen is used to control the ***blink*** and ***repeat delays***. When the cursor enters an object that is capable of action it normally blinks the object. The repeat delay determines how fast a momentary button object releases when activated by a keyboard entry. ***Repeat delay also controls the mouse button repeat rate when used to adjust panel objects.***

Other Items Customizing Screen



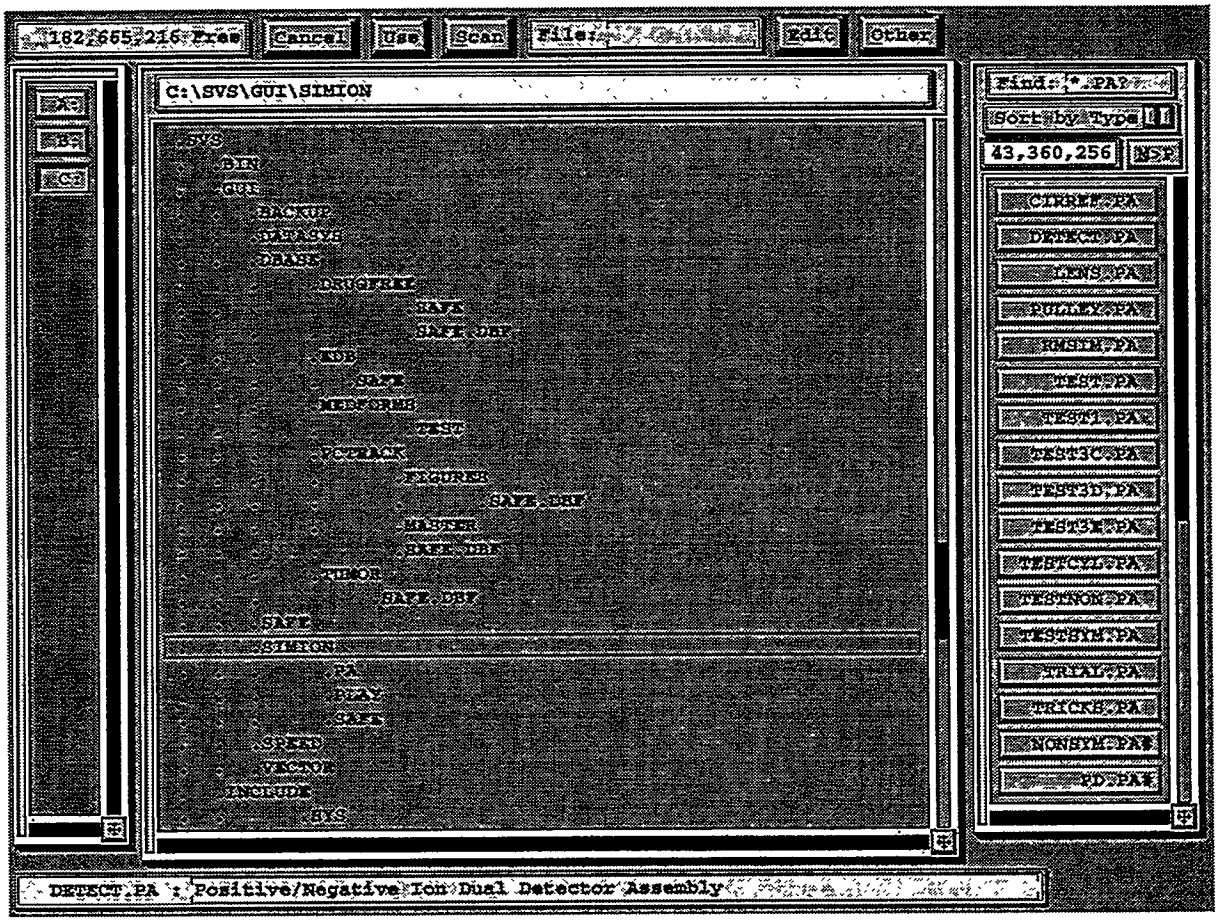
Two buttons (*on the left*) select between **normal** or **etched** text on button objects (*the screen above demonstrates etched text*). The **Underlines** button controls whether the GUI is allowed to tell you (*with red underlines*) if a labeled button is currently keyboard accessible (*buttons still remain keyboard accessible*). The **Scan ON/Scan OFF** button tells the GUI File Manager whether to ***automatically scan*** a drive for directories the ***first time it is selected*** in the current program session. The slider object adjusts mouse sensitivity: left (*low*) to right (*high*).

Video Resolution Screen

The GUI supports higher screen resolutions via ***VESA Video BIOS extensions.*** If your video board supports the VESA standards and its VESA BIOS is active (*loaded*), the GUI will interrogate your video card and determine what resolutions it can support. ***These available resolutions will appear as buttons on your screen.*** To select a resolution click its button. ***Note: If at any time the GUI's Video gets trashed (e.g. by Windows or OS/2) click <Alt V> or <Ctrl V> to restore it.***

The GUI has built in support for VGA (*640 x 480*) and VESA (*800 x 600*, *1024 x 768*, and *1280 x 1024*) in both 16 and 256 colors. ***Note: 256 color resolutions are generally slower than 16 color resolutions*** (*except on local bus video machines - e.g. PCI video cards*).

The GUI File Manager



Note: The GUI uses line drawing characters in the IBM extended character set to display directory trees. Unfortunately, PostScript doesn't support these characters in its extended character set. ***Thus you see '.' (periods) for the line drawing characters in the illustration above.***

Introduction

The GUI's File Manager is illustrated above. This GUI feature is used extensively within SIMION for loading and saving files. However, you will find that it is a ***powerful tool*** for doing all sorts of things. The information below only gets you started. ***Use the <F1> help key on the GUI File Manager's objects to learn the details.***

A Quick Tour of the File Manager Windows

There are three windows in the screen above:

Drive Display Window

The *leftmost window* is used to view and select the *current drive* (*e.g.* **C:**). Note: A drive is normally scanned for directories the *first time* it is *selected* in the *current* program *session*. Use the **Scan ON/Scan OFF** button (*Other Options Customizing Screen*) to suppress auto-scanning.

A drive is selected by clicking its button.

Directory Tree Window

The **middle window** shows the **complete directory tree** for this drive. The **current directory is marked** (*e.g.* **C:\SVS\GUI\SIMION**) and its path displayed above the window.

A directory is selected by clicking on it.

Files Found Window

The *rightmost window* contains a listing of the *files found* in the current directory.

One or more files are selected and de-selected by clicking on the files' buttons.

There are four objects at the top of the files found window. The *topmost* is an *ioline* that contains the *current search criteria* (*e.g.* ***.PA?** - *you can change it*). In this case the file search was limited to files having "PA" as the first two characters in their file extension.

The next object is a *selector object* that determines the current *file sorting order*: By type (*by extension then name*), by name (*by name then extension*), by creation date (*youngest on top*), or by file size (*largest on top*).

Just below is a display *panel* that shows the *disk space used by the found files. Note: This number includes the wasted cluster space.*

The fourth (*and last object*) in the *files found* window is a small **M>P** button. This button is used to *output file memos* for the *currently found files* to a printer/file (*ASCII image only*).

> ***Note: If you move the cursor to a file's button, the current directory's path display will switch to displaying the directory information for that file (e.g. size, date, and attributes).***

File Memos

The GUI File Manager provides a powerful file memo capability. Each file can have a memo (*up to 70 characters long*) associated with it (*memos are saved in the* **MEMOINFO.GUI** *file in the current directory*). ***At the bottom of the file manager screen is a long ioline for creating and viewing file memos.*** In the example above, the file memo for the file **DETECT.PA** is displayed.

Viewing File Memos

To view a file's memo (*if any*) just *point to its file button* in the *found files* window.

Creating File Memos

To create (*or edit*) a memo for a file, ***point the cursor to its button.*** Now *move* the *cursor* to the *left off the file's button.* If you are successful, the *file's name should still be in the memo ioline.* Now use the ioline object to *enter or edit the memo.*

The GUI File Manager and Memos

The GUI File Manager automatically manages your memos for you. If you use the file manager to copy a file with a memo its memo will be copied too. Likewise, deleting a file with the GUI File Manager will delete its memo too. ***Moral: If you use memos use the GUI File Manager.***

Memos for files are saved in a **MEMOINFO.GUI** memo file in the current directory.

The Top Row of Objects

The Disk Free Space Ioline Object

The top left panel object displays the ***free space in bytes*** of the ***currently selected drive***.

The Cancel and Use Buttons

The **Cancel** button is used to abort the file manager. You are ***returned to the drive and directory at the time you entered the file manager***. The **Use** button returns the filename you selected and ***stays in the currently selected drive and directory***.

The Scan Button

The **Scan** button is used to ***rescan the current drive for directories***. If you create or delete directories ***outside*** of the file manager (***e.g. in DOS***) or change disks in a drive, these changes ***may not*** be reflected in the directory tree displayed. **Scan** is ***more*** important when **Scan OFF** is active.

The File Ioline Object

The **File** ioline object (*when provided*) is used for entry of a file name. Note: You can point the cursor to the file's button and click ***a*** mouse button (***click both for select and Use exit***).

The Edit Button

The file manager provides an **Edit** button when the use of a file editor is allowed. To edit a file, select its button (*leaving it depressed*) and then click the ***Edit*** button. ***The GUI uses the EDY editor by default***. If you prefer another editor, use the **DOS SET** command to link SIMION to your editor. The line below makes **EDLIN** the active editor (*or non-editor*):

SET GUI_EDITOR=EDLIN

The Other Button

The **Other** button supports a wide variety of operations from file copying to directory renaming. ***Note: For operations on files, select them (depress their buttons) before clicking Other.***

Some Examples of File Manager Operations

Deleting One or More Files

Select File(*s*), click **Other** button, and click **Delete** button (*you will be asked if you're sure*).

Copying or Moving One or More Files

Select File(*s*), click **Other** button, click **Copy** button, and select copy or move. Now select the destination drive and directory and click the **Proceed** or **Cancel** button. ***Note: The file manager will always ask permission before overwriting any files.***

Inserting a New Directory

Click on the new directory's parent directory. Now click the **Other** button, and enter the new directory's name and hit **<Enter>**.

Deleting an Existing Directory

Click on the Directory, click the **Other** button, and click the **Delete Directory** button. If the directory is not empty you will be warned. ***If you persist (be very careful) the file manager can be convinced to delete the directory, its files, and all of it sub-directories***.

Printing Options

Introduction

SIMION provides powerful printing capabilities. ***These capabilities include: Multiple printer language support, color and B/W, image composition, and image annotation.*** The material below is provided to assist you in using your printer effectively with SIMION.

Remember: Most of this information is available from SIMION's specific and object help screens. Just point at the object in question and press the <F1> key.

Supported Printers

SIMION supports printer languages in lieu of specific printers. This approach provides better driver longevity at the price of not supporting certain unique printer features.

The GUI has internal support for the following printer languages: **PostScript (*B/W and Color*), PCL 5 (*B/W and Color*), HP-GL/2, and HP-GL.** Your printer (*or plotter*) must support at least one of these printer languages to be used directly with SIMION.

For example, a LaserJet II is an early PCL language device (*not supported*). If you want to use a LaserJet II printer it must either be upgraded to PostScript or you must obtain a conversion program (*if available elsewhere*) that can convert a file containing one of the above printer formats into the printer format required.

Your printer's speed and resolution are important! The GUI's printer drivers create the printed image quickly. Your printer is the real speed and quality limitation. The minimum printer resolution should be 300 dpi (*600 dpi or above recommended*). Further, most newer printers feature RISC processors that speed up graphics output from minutes to seconds. First class printers are relatively inexpensive. Your time is valuable. Don't waste it waiting for a slow low quality hard copy.

A Word About DOS Printer Connections

Your printer is probably connected to either a parallel or serial port. DOS monitors serial and parallel port activities to detect device hang-ups. While undoubtedly an admirable goal, it can create irritating ***abort, retry, ignore*** messages on your screen. This can become especially irritating with older PostScript printers (*with slow processors*) that seem to stop and pant for extended periods during output. You can avoid this problem by including special ***MODE*** command modifications in your ***AUTOEXEC.BAT*** file:

MODE LPT1,,P	The **P** tells ***DOS*** to Persist. ***DOS*** will not check for device hang-ups on ***LPT1***.
MODE COM1,96,N,8,1,P	The **P** tells ***DOS*** to Persist. ***DOS*** will not check for device hang-ups on ***COM1***.

Remember: Serial ports must be initialized properly via a ***MODE COM?*** command (*as above*) to communicate properly with any printer connected to a serial port. Make sure your ***AUTOEXEC.BAT*** file contains the proper ***MODE*** command if your printer is connected to a serial port.

Connecting Your Printer to SIMION

The GUI initially assumes that you have a ***PostScript*** printer connected to the ***LPT1*** parallel port (*PRN*). If this is true, no action on your part is required.

An alternate printer language must be specified if your printer is not PostScript compatible. Further, the device port must be changed if your printer is connected to a serial port or you want to direct your printer output to a file. ***Printer language and device port/file specifications are accomplished via the Print Options Screen discussed later in this section.***

The Printer Pipeline

When you print something from SIMION a switch is set within the GUI and the desired object is redrawn. Each drawing command is automatically routed to ***both*** the GUI screen driver ***and*** the currently active printer driver. ***Thus you can see the printer's progress by just looking at your screen!***

If something is wrong or you want to abort the current printout, just hit the <ESC> key.

The GUI actually has two printing pipelines: A vector pipeline and a pixel pipeline. The differences between the two printing strategies are discussed below. However, all printer output whether pixel or vector is eventually converted to a vector format and transmitted to the printer.

This means that both vector and pixel output images can be positioned and sized on the page as desired via the GUI's print option image composer. The actual quality of the printed image will depend on the quality of the printer, image size on the page, and whether the output was originally vector or pixel based.

Pixel Verses Vector Printing

The GUI supports both pixel (*e.g. screen printing*) and vector (*e.g. isometric views*) printing.

Pixel Printing

Pixel printing can be thought of as transferring copies of your screen's pixels (*dots*) to your printer. This means that the printed image shares the resolution and quality of the image you see on your screen. ***All lines are drawn at the equivalent width of one screen pixel.*** Therefore, if SIMION is working in VGA mode (*640 x 480 pixels*) any pixel printing will be of lower detail quality than if SIMION is in a high resolution SVGA mode (*e.g. 1280 x 1024 pixels*).

Note: Because the GUI always converts pixel output into vector format before shipping it to your printer, the jagged diagonal lines visible on your screen will appear as smooth diagonal lines on your printer (*the better the printer, the smoother the lines*).

Vector Printing

Vector printing involves drawing lines with widths defined in printer space (*points or mm*) rather than in screen space (*pixels*). This means that an isometric view that might look absolutely shabby in VGA screen resolution may look quite acceptable if shipped to a good PostScript or PCL 5 printer.

Where Vector and Pixel Printing are Used

SIMION uses pixel printing for all screen printing, and for any 2D view within **Modify**. Isometric views within **Modify** and all printing within **View** use vector printing.

How to Request a Printout

There are two types of program printouts: Window and screen. Each is requested differently.

Requesting a Window Printout

SIMION provides a **Print** button on any screen where window printout is supported (*e.g.* **Modify** *and* **View**). Just click the **Print** button to print the window's current view. ***The Annotate and Print Screen will appear.***

Requesting a Screen Printout

Any screen or selected screen object(s) can be printed via the GUI screen print utility. To access screen printing hold the **<Crtl>** key depressed and press the **<F1>** key. A cross hairs cursor will appear with a panel object in the upper left corner giving its position.

To exit from this mode just hit the <ESC> key.

Printing the Entire Screen

To print the entire screen just hit the **<P>** key. A small screen will ask if you want to print the shaded object or the entire screen. Click the **No** Button to print the entire screen. The Annotate and Print Screen will appear.

Printing Selected Object(s)

The GUI's screen is composed of a collection of objects. Objects are layered to form parent-child relationships. An object is selected by pointing to it with the cross hairs cursor and pressing the **<P>** key. If you want to want to print a group of objects point to their common parent object (*guess*) and press the **<P>** key.

The GUI will shade the selected object (*not necessarily the one you expected*) and ask if you want to print it. ***If the object is not the one you wanted click the* Cancel *button and try again.***

When the selected object(s) is(*are*) acceptable press the **Yes** button. The Annotate and Print Screen will appear.

The Annotate and Print Screen

The Annotate and Print Screen appears when any printout is requested. This screen consists of the object to be printed surrounded by a grayed screen with a collection of control buttons and annotation objects in the upper part of the screen.

Print Control Buttons

A row of buttons control access to various printing features.

Print Button

The **Print** button is used to initiate the actual printing process.

Frame Button

The **Frame** button options the drawing of a black shadow frame around the printed object.

B/W - Color Button

The **B/W - Color** button selects whether black and white or color output is sent to the printer. The effects of this button is dependent on the printer language selected (*e.g. PostScript*) and whether the pixel or vector pipeline is being used:

With Pixel Printing:

Language	Color	B/W
HP-GL *	Multi-Pen	Pen 1 Only
HP-GL/2 *	Multi-Pen	Pen 1 Only with Gray Scaling
PCL 5	Color PCL 5	Gray Scale PCL 5
PostScript	Color PostScript	Gray Scale PostScript

* Rectangular Areas Outlined only (*not filled*).

With Vector Printing:

Language	Color	B/W
HP-GL	Multi-Pen	Pen 1 Only
HP-GL/2	Multi-Pen *	Pen 1 Only with Variable Line Widths
PCL 5	Color PCL 5 *	Variable Line Widths
PostScript	Color PostScript *	Variable Line Widths

* Color Vector Line Widths = Color 15 (*white's*) Width.

Note: Line widths are set in the Print Options Screen below.

Options Button

The **Options** button accesses the Print Options Screen.

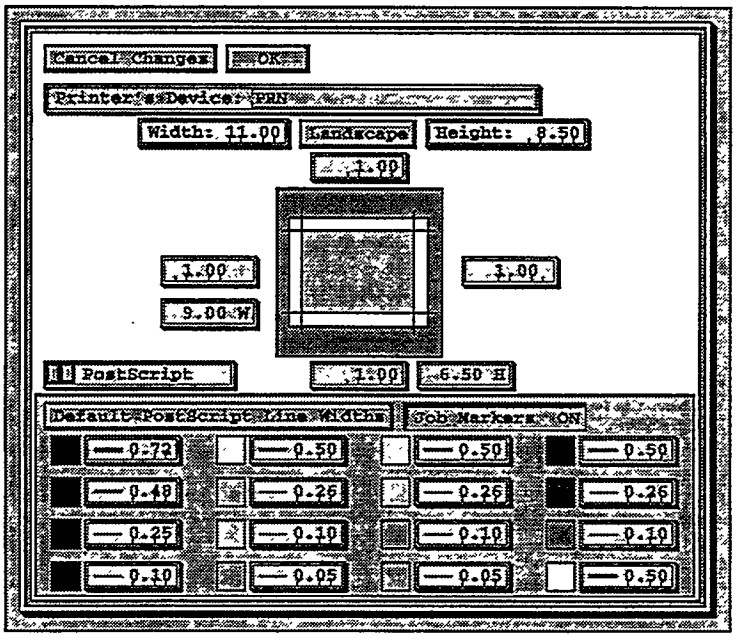
RA Button (*if visible*)

If you have used any annotations in the previous printout (***even in a <u>previous</u> program session***). The GUI will provide an **RA** button (*Restore Annotations*) to allow you to restore the previous annotations (*very handy*). If you want to restore your previous annotations (*then rip out what you don't want*), be sure to click the **RA** button ***<u>before</u>*** creating any new annotations. ***If you don't, all the previous annotations will be erased!***

Print Options Screen

The Print Options Screen is used to select device port or file, printer language, page size/margins, change colors, set line widths to substitute for colors (*B/W Vector outputs*), and more. The GUI saves printer options in the file **C:\FILES.GUI\PLOTTERS.GUI** so they can be retained between your SIMION sessions. The Print Options Screen is accessed via the **Options** button.

> ***Note:*** ***Once any option below is changed it will remain the same from session to session.***



General Print Options

The general print options apply to all printouts independent of the printer language selected.

Printer's Device

This ioline object contains the name of the currently set printing device port (*e.g.* **COM1**) or a file name if you want printer output sent to a file. Output is ***always*** appended (*added*) to the device port/file.

Page Size and Orientation Controls

Two panel objects and a button define page size (*in inches*) and orientation. Page size can be changed to reflect the larger paper size on some printers and drafting plotters. ***These controls are very useful as tools to trick your printer to work properly or produce some special effect.***

Page Margin Controls

Four panel objects are provided to define page margins. Your printed output will always be ***sized without distortion*** (*equal x and y scaling*) ***and centered*** to fit within these margins. The ***gray area*** in the page display object ***shows where your output will be placed*** on the page.

Margin settings are reasonably accurate with PostScript. However, PCL 5 devices have differences in their page alignment. ***Therefore, you may have to fiddle with the margins a bit to get images just where you want them.***

Printer Language Selector

The printer language selector object is used to select the printer language (*e.g. PostScript*). Five PostScript modes are provided along with HP-GL, HP-GL/2, and PCL 5. When you change between printer languages the lower screen will display the specific options for the selected language.

PostScript Modes

PostScript has five modes: ***PostScript, PS 1st Image, PS Next Image, PS Last Image, and PS Encapsulated***. The following describes when and how to use each PostScript mode:

PostScript

This is the normal ***one image per page*** mode (*the one you should normally use*).

PS 1st Image, PS Next Image, and PS Last Image

These three modes are provided to allow you to create multiple images on a single page. Adjust the margins ***before*** printing each image to position each in the desired page location.

PostScript uses an Opaque printing model. This means that ***later printed images cover earlier printed images*** in any areas of overlap.

It is recommended that multiple image compositions be ***output to a file***, and then copied to the printer. ***Reason:*** Most ***PostScript printers*** will automatically ***time-out*** while you're fiddling with your composition (*very irritating*).

PS Encapsulated

This mode is provided to permit you to ***export illustrations*** for inclusion in your documents (*e.g. used for all illustrations in this manual*). ***Save one image per file***. Most word processors expect a file extension (*e.g. .EPS*). ***Suggestion: Zero your margins to maximize your image size.***

> ***Note: This mode will not work properly if sent directly to your printer. It needs to be encapsulated into some other PostScript output stream.***

PostScript Options

PostScript modes support the following options:

Job Markers

The ***Job Marker*** button is provide to include a **<Ctrl D>** at the beginning and ending of each PostScript page. Most PC based PostScript devices expect to see this marker. However, other systems do not like it (*e.g. Mac*). Use whatever flavor works for you.

Note: PS Encapsulated ignores this option.

Line Width Options

When ***vector graphics*** are ***output in B/W*** with a PostScript driver, ***colors are converted into variable width black lines***. Each color has an associated line width in points. ***Panel objects***

are provided to allow you to ***adjust the widths of each color's line***. The default line width settings provide the visual effects of illumination. However, they can be adjusted as desired.

Note: The width of the color white (*15 - last color*) is used to set the width of all vector lines drawn in Color PostScript. Thus if you want to simulate fill or other effects in color vector output, change the line width for white.

HP-GL, HP-GL/2, and PCL 5 Options

These ***three drivers are all really HP-GL(/2) based*** with different assumptions concerning the target printing device. ***Therefore, all three drivers can be used with an HP 7475A plotter***. The error light may flash like crazy but a plot will emerge. Likewise, all three drivers will generate (*some form of*) output on a HP-GL/2 or PCL 5 device. The primary limitation is that for the HP-GL driver to work properly the printer must be in HP-GL/2 mode (*the ESC%-1B has been transmitted to the PCL 5 Printer*).

Both ***HP-GL and HP-GL/2 drivers assume*** the device is actually a ***pen plotter***. Thus they ***support*** options such as specifying the ***number of pens*** and the setting of ***drawing velocity*** (*for transparencies with pen based plotters*). ***B/W drawing*** is ***always*** done using ***pen 1***. Neither fills rectangular areas in pixel drawing (*because a transparency model is assumed*). ***HP-GL/2*** assumes the device can ***support variable line widths***, ***gray scaling***, and also provides ***page size definitions*** and cutting needed by drafting plotters with a roll type paper source.

PCL 5 assumes that the device is a fully capable HP-GL/2 device (*job stream controllable colors/grays and line widths*). **Note: Pen zero is considered to be an active color (*black*).**

Thus if one driver doesn't quite work try another. There are enough options to trick most HP compatible devices into working.

R90 Option Button (HP-GL/2 & PCL 5)

When this button is depressed the image is rotated 90 degrees from HP-GL/2 standards. This button is normally depressed for HP-GL/2 and PCL 5 output to LaserJets (*raised for DeskJets*). ***If your plot is twisted 90 degrees and clipped off try the opposite R90 option.***

Number of Pens (HP-GL and HP-GL/2)

This panel object sets the maximum number of pens (*from 1-16*). Set the number to that corresponding to your plotter ***or*** the maximum number of pens you actually want to use.

Drawing Velocity (HP-GL and HP-GL/2)

The drawing velocity panel object sets the pen drawing speed in ***cm/sec***. This is useful for pen plotters to control drawing quality on media like transparencies.

Line Width Options (HP-GL/2 and PCL 5)

When ***vector graphics*** is ***output in B/W*** with an HP-GL/2 or PCL 5 driver, ***colors are converted into variable width black lines***. Each color has an associated line width in mm. ***Panel objects*** are provided to allow you to ***adjust the widths of each color's line***. The default line width settings provide the visual effects of illumination. However, they can be adjusted as desired.

Note: The width of the color white (15 - last color) is used to set the width of all vector lines drawn in Color HP-GL/2 and PCL 5. Thus if you want to simulate fill or other effects in color vector output, change the line width for white.

Annotation Options - Henry Ford: "You can have any color as long as it's black"

All printed outputs from SIMION can make use of the GUI's annotator. The annotator provides an easy way to add legends, labels and lines to mark or clarify your illustration. It is not intended to serve all needs. ***However, it is quick, easy to learn, and well suited to most annotation tasks.***

If you choose to make use of the annotator be sure to annotate ***before*** you actually print. When a view is annotated (*has annotation objects on it*), the GUI will always return to this view after each printout (*instead of exiting the print menu*). ***This allows you the option of further editing your annotations without having to recreate them between printouts.***

Annotation Objects

There are three classes of annotation objects: ***Legend boxes, lines, and labels.*** Each class is represented by a sample object drawing of itself at the top of your screen.

Creating an Annotation Object

An annotation object is created by simply pointing the cursor to the class of object you want and dragging a copy of it (*with* **either** *mouse button depressed*) to the desired location on the view.

Moving an Annotation Object

To move a previously created annotation object point the cursor to it and ***drag it using the*** **right** ***mouse button.***

Deleting an Annotation Object

An annotation object is deleted if is ***dragged out of the view*** (*into the gray area or off the screen*).

Editing an Annotation Object

Each class of annotation object can be edited. ***Editing is generally done by dragging parts of the annotation object with the*** **left** ***mouse button.***

Editing Line Objects

A line object is edited by pointing the cursor to one of its ***end points*** (*a small square will surround the end point*), and then dragging that end point with the **left** ***mouse button depressed.***

Editing Legend Box Objects

A legend box object is edited by pointing the cursor to one of its four ***side lines*** (*a small square will surround the side line*), and then dragging that side line with the **left** ***mouse button depressed*** to size the legend box.

Editing Label Objects

A label object is edited by pointing to it. A small rectangular cursor will surround one of the label's **letters** (*the action point*). Now edit the label as you would any normal text. Left and right arrow, insert, delete, and etc. keys work as in other editors.

Warning: Up or down arrow keys create a blank label.

Complex Labels (e.g. subscripts)

While the annotator doesn't support complex labels directly it can be easily tricked into making them. ***For example: Just insert a space where a subscript should go, create a label that is the subscript, and then drag it into the desired location.***

The Importance of the Order of Creation for Objects

All objects are printed in the order of their creation (*oldest - first, youngest - last*). This is normally not a problem except for legend boxes. Because the printing model is ***opaque*** (*objects can't be seen through objects*), a legend box created ***after*** the label it contains ***will cover*** (*obscure*) the label when it is printed.

Moral: Create your legend boxes before creating the labels that are to be displayed upon them.

Copying An Annotation Object

To obtain a copy of an annotation objects point to it with the cursor, ***hold down the*** **<Ctrl>** ***key*** and drag off a copy with the **right** mouse button depressed.

Grouping Annotation Objects

Annotation objects can be marked and operated on as if they were part of the same group. This feature allows ***moving***, ***copying***, and ***deleting*** annotation objects ***in marked groups***.

Marking a Group of Objects

A group of objects is marked by dragging a ***rectangle to completely surround*** the desired grouping (*with* **either** *mouse button depressed*). The starting point should be outside of any actual object as should be the stopping point.

If no objects exist within the marked area the computer will beep and the rectangle will erase when the mouse button is released.

Moving a Marked Group of Objects

A marked group of objects can be moved by dragging it with ***either mouse button depressed.***

Deleting a Marked Group of Objects

A marked group of objects may be deleted by moving it into the gray area outside the view or off the page.

Copying a Marked Group of Objects

Once a group of objects has been marked it can be copied by ***holding down the*** **<Ctrl>** ***key*** and dragging off a new copy of the group with ***either mouse button depressed.***

Moving the Print and Annotate Screen

The Print and Annotate Screen is normally at the top of your screen. In some cases it may block an area you want to annotate. To toggle the Print and Annotate Screen to the bottom of your screen place the cursor in the view area (*but not over any annotation object*) and press the **<spacebar>** key. Hitting the **<spacebar>** key again will toggle the Print and Annotate Screen back to the top of the screen.

Adjusting the Size of Label Lettering

The GUI always uses printed labels that are the same size as you see on the screen. Thus the only way to adjust relative letter height is to change SIMION's screen resolution with ***Adjust Video*** buttons from the Main Menu Screen (*VESA Video BIOS Active*). VGA (*640 x 480*) gives the largest lettering and SVGA (*1280 x 1024*) gives the smallest lettering.

If you need more control over image editing consider exporting a print image file to a 3rd party graphics editor (e.g. Corel).

Restoring Annotations Via the RA Button (*if visible*)

If you have used any annotations in the previous printout. The GUI will provide an **RA** button (*Restore Annotations*) to allow you to restore the previous annotations (*very handy*). If you want to restore your previous annotations (*then rip out what you don't want*), be sure to click the **RA** button ***before*** creating any new annotations. ***If you don't, all the previous annotations will be erased!***

Using Other Programs to Edit Your Graphics

Many programs have graphics editors that can be used to edit print files created by SIMION. It is recommended that you ***use HP-GL as a transfer format*** (*one image per file*). The HP-GL files are simple and many graphics editors can read HP-GL files. HP-GL/2 and PCL 5 may also work if your graphics editor supports the advanced features of HP-GL/2. The ***last choice*** would be the ***Encapsulated PostScript*** format. ***Most PostScript compatible graphics editors can only read encapsulated PostScript files produced by specific programs*** (*SIMION will probably not be one of them*).

Hint: You may want to zero the print margins to maximize the transferred image size.

The EDY Survival Manual

Introduction

This chapter provides a short survival course in the use of EDY. EDY is an editor developed by the author and used for the development of SIMION. This program is provided in case you do not have a good way to create the ASCII files needed for user programming and creating geometry definition files.

EDY is a powerful editing program. This appendix only tries to teach you enough to start using EDY. You will probably learn a lot more from EDY itself once you get started.

What Is EDY?

EDY is a full screen ***in-memory*** editor. That means a file is loaded into the computer's memory and edited. EDY makes use of all available computer memory. ***However, if your programs are too large to fit into RAM they will not load into EDY.***

The Two 32 bit Extender Versions of EDY

Two 32 bit extender based versions of EDY are provided (*Intel and Rational*). Be sure to always use the same extender versions of EDY and SIMION together (*so they will work together*). ***The installation program automatically loads the appropriate version.***

The extender versions of EDY allocate 2 Megs of RAM for file space by default. You can change this allocation with the following DOS SET command:

SET EDY_MEM=10000000

The example above sets the RAM size to 10 Megs (*a value of 0 will option about 400k*).

The Real Mode Version of EDY

A real mode version of EDY is also included. It runs in the lower 640 K and can be called by either extender version of SIMION. Its strength is that it loads very fast. Its weakness is that it supports smaller files. ***However, it is unlikely that any user program or geometry file will ever tax its capacity.***

The real mode version is called **RMEDY.EXE** and is located in the installation directory (*e.g.* **C:\SIM6**). You can switch to this editor by getting into the installation directory and typing **SMALLEDY** and **<ENTER>** at the DOS prompt. ***If you decide to switch back, you must use the installation program to reload the proper extender version.***

Running EDY

The EDY program consists of two files: **EDY.EXE** (*program*) and **EDYSET.EDY** (*personality file*). ***Both of these files must be somewhere in the currently active search path*** (**DOS PATH** *statement*) ***for EDY to execute properly*** (*e.g.* **C:\DOS**).

Running EDY From the DOS Prompt

To start EDY from the DOS prompt simply enter **EDY** and press **<ENTER>**. If you know the file you want to edit (*e.g.* **README.DOC**) you can specify it at startup (*e.g.* **EDY README.DOC <Enter>**).

Running EDY From Within SIMION

The GUI's File Manager provides an **Edit** button in those cases when use of a file editor is allowed. To edit a file, click on its button (*leaving it depressed*) and then click the **Edit** button. To enter the editor without a file just click the **Edit** button (*without selecting a file first*).

Running Another Editor From SIMION

SIMION uses the EDY editor by default. If you prefer another editor, use the **DOS SET** command to link SIMION to your editor. The line below makes **EDLIN** the active editor (*or non-editor*):

SET GUI_EDITOR=EDLIN

Quitting EDY

To quit EDY enter the following three keystrokes: **Esc Q <Enter>.** If you have any active files in memory EDY will display them and ask you if you want them saved.

EDY'S Display Screen

Top Line

The top line of the screen contains a help and cursor data line. The four numbers on the upper right are line number, line length, cursor column, and the ASCII code for the character under the cursor.

Help Screen

The next 4-5 lines contains the help screen. Help screens in EDY are ***concurrent*** (*displayed with file image*) and ***contextual*** (*EDY knows what you're doing*). As you perform tasks EDY will automatically display the proper help screen to assist you. If you don't want the help screens press the **<F1>** key to turn them off (*you may turn help screens back on at any time by pressing the* **<F1>** *key again*).

File Windows

The file windows take up the rest of your screen. You should see a single file window at first. The file's name an type are displayed in the upper left corner of the file window. The file type is normally *(a)* for ASCII (*more of file types later*).

Bottom Status Line

The bottom screen line contains status indicators and messages. The three indicators on the lower right indicate the status of the **<Caps>**, **<Num Lock>**, and **<Insert>** keys. Any messages from EDY to you will appear in the lower left corner. You should see the word **READY** indicating that EDY is waiting for your input.

Basic Editing Procedures

The following material covers basic editing methods with EDY:

Moving the Cursor

The cursor (*flashing square*) is moved with the arrow keys. Other keys can also be used:

Home	**>**	**Beginning of line**
Ctrl Home	**>**	**First line in file**
End	**>**	**Go to end of line**
Ctrl End	**>**	**Last line in file**
Page Up	**>**	**Move one display page up**
Page Down	**>**	**Move down one display page**
Ctrl Left Arrow	**>**	**Left one word**
Ctrl Right Arrow	**>**	**Right one word**
Tab	**>**	**Right one tab stop**
Shift Tab	**>**	**Left one tab stop**

The mouse can also be used to move the cursor in any direction. Clicking ***both*** mouse buttons together toggles (*on and off*) an EDY power scroll mouse feature (*try it*).

Text Entry

Text is entered directly via the keyboard. EDY supports both ***overwrite*** and ***insert*** modes of text entry. The **<Ins>** key is used to toggle between these modes. When the **INS** screen indicator is on (*cursor is larger too*) you are in ***insert*** mode.

Creating a New Line

A new line is created by pressing the **<Enter>** key. The new (*zero length*) line will appear just ***below*** the current cursor line.

Creating Blank Lines

Blank lines are created by pressing the **<Enter>** key (*to create a new zero length line*) ***followed*** by a **<Spacebar>** key. This puts ***at least one character*** in the line. ***EDY does not like zero length lines and will automatically remove (delete) them when you move the cursor away from them.***

Deleting Text

Characters ***under*** the cursor are deleted with the **<Del>** key. Characters to the ***left*** of the cursor are deleted with the **<Backspace>** key. Areas of text (*lines and etc.*) are deleted by first marking them using the **<F2>** key ***or mouse*** and then using the **<Del>** key.

Marking Areas of Text by Using the Keyboard

EDY allows you to mark areas of text much like you mark an area in the **View** function in SIMION. First move the cursor to a corner of the area to be marked. ***If whole lines are to be marked, move to column one of the first or last line.*** Press the **<F2>** key (*If Help is active EDY will provide help screen assistance*). Now move the cursor to the diagonally opposite corner of the area to be marked. ***If whole lines are to be marked, move to column one of the other bounding line.*** Press **<F2>** again and the area is marked.

Marking with the Mouse

Areas can be marked with the mouse in roughly the same manner. Move the mouse to the starting point to mark. Press ***and hold down*** the **left** mouse button. Move the mouse to the ending point. Release the **left** mouse button and the area is marked.

Knowing How to Mark an Area

If you mark an area using its diagonals, EDY will consider the region fully bounded. However, if you mark vertically (*keep both marking points in the same column*) EDY will assume ***all text to the right*** of the marks to be marked too. Moreover, for vertical moves it will consider the lines extend out to column 32,000 (*this prevents line fragmentation on moves*).

Things You Can Do With a Marked Area

The following is a list of things you can do with a marked area:

Delete it

Press the **<Del>** key and EDY will ask you're if you sure. Press **Y** for *yes* and the marked area will be deleted.

Erase it

Press the **<->** key and EDY will ask you're if you sure. Press **Y** for *yes* and the marked area will be erased (*all non-space characters will be replaces with spaces*).

Copy it

After marking a text block move the cursor to the desired destination point. ***Pause***. Notice that a ghost image of the marked block will flash. ***This helps you position the text block.*** You now have three options:

Put Text on New Lines

Press the <*Enter*> key to create new lines for the copied text block (*existing text will shifted down to make room for the new lines*). Press **Y** to confirm the copy.

Insert Text into Existing Lines

Press the **<Ins>** key to insert the text block at the current cursor location. Existing text will be moved to the right to make room for the inserted text. EDY will ask for confirmation. Press **Y** to confirm the insertion.

Overwrite Existing Text

Press the **<+>** key to overwrite the text block at the current cursor location. Existing text under the text block will be replaced by the text block. EDY will ask for confirmation. Press **Y** to confirm the overwrite.

Moving a Text Block

A marked text block can also be moved. Press **M** to select the move capability. Now move the text block with the cursor keys or mouse. you can move horizontally or vertically. Press **M** or **<F2>** when you're through moving.

The move function works by transferring all the characters you would destroy across the text block into the space vacated when the text block moves a column or line. ***This means you can undo a mistake if you carefully retrace your steps.***

The other way to move a text block is to mark it, copy it to the new location, and then hit the **<Del>** key to delete it at its original location. ***This is the recommended way to move large blocks of text.***

Stretching a Text Block

Stretching (*enter* **S**) works much like move except the text block destroys all characters in its path and fills in the vacated space with copies of the previously adjoining characters. If boxes are marked carefully (*just one of their edges*) you can actually stretch or shrink them with the Stretch facility.

Command Entry

Commands are entered by pressing the **<Esc>** key. A help screen menu will appear to assist you in command selection. To select a command enter its first letter (*e.g.* **Q** *for Quit*). The command's help screen will then appear to help you with its options. The following discusses loading and saving files plus the DOS shell. ***Using the other commands should serve as an adventure.***

Loading a File

A file is loaded by pressing **<Esc>**, **L**, entering the file's name, and pressing **<Enter>**. If you want a directory listing press the **<F1>** key after you enter the **<L>** key. Notice also that the load command accepts other parameters beside the file's name. These parameters are separated by commas. ***Remember commas must be inserted to skip parameters.***

Saving a File

A file is saved to its existing name by the following three keystroke sequence: **<Esc> S <Enter>**. You can select another name by entering it after the **S** (*the default saving name remains unchanged*).

Renaming a File in Memory

A file in memory can be renamed with the name command: ***<Esc> N filename*** **<Enter>**. This only ***changes*** the name of the file on the window and its ***default saving name***.

Using the DOS Shell

EDY has a DOS shell (*you can execute programs within EDY*). To access the DOS shell enter the following two keystroke sequence: **<Esc> D**. A DOS prompt will now appear. You could run CHKDSK by entering **CHKDSK <Enter>**. Any DOS command is also legal. After you're through enter **<Esc>** ***twice*** to return to screen mode.

Learning About the Other Control Keys

EDY has many special keys that could prove useful. To toggle through the special keys help screens enter the **<Ctrl K>** key (*keyboard help*). If you want help on **Ctrl** or **Alt** based keys press **<Ctrl F1>** for **Ctrl** key help and **<Alt F1>** for **Alt** key help.

Arbitrary ASCII codes can be entered by holding down the **<Alt>** key while entering a three digit key code from the ***upper row*** of number keys (***not the number pad keys***) (*e.g.* **Alt 050** *enters the character code 50*).

Experiment with these keys!

Multiple Windows and Split Screens

EDY can have up to eight files in memory at once (*providing there's room*). By default, EDY starts up with two of these file windows visible. You can use the **<F7>** key to toggle between them.

Loading a Second File Into Memory

A second file can be loaded into memory by using the **<F7>** key to shift to the alternate file window and then using the Load command to load a disk file into it.

Transferring Information Between File Windows

Information (*text blocks*) are transferred between file windows by first marking the desired text block (**<F2>** key) and then using the **<F7>** key to jump to the next window. The text block can be copied in one of the three modes described above.

Dual Views and Split Screens

EDY actually maintains two views of each file on separate screens. Press the **<F3>** key to toggle between them. ***Thus you can set up two working areas in the same file.*** Moreover, it is often

nice to see both of these views at the same time. Press the **<F4>** key to toggle between split and overlapping screen view modes.

More Windows - Any Size

The **<F8>** key is used to access the window definition functions. When you start EDY all eight file windows are active but only two are visible.

Once in this mode the **<+>** and **<->** keys can be used to jump between file windows. ***If the border flashes the file window is currently invisible.*** To make a file window visible (*add it to the display chain*) press the **<Ins>** key. Likewise, to remove a file window from the display chain press the **<Del>** key.

A file window can be moved and sized too. Use the **<F8>** key to get into the window definition mode. Notice that the upper left corner of the current file window has a special flashing character. Use the arrow keys to move this character to the new location for the upper left corner. Now press **<F2>** and small window outline will appear. Adjust the size of this window with the arrow keys. Press the **<F2>** key again and the file window will move into its new size and location. ***The mouse can be used to move and size a file window in the same manner it is used to mark a text block.***

Character Graphics

The **<F6>** provides access to EDY's character graphics mode. Follow the help screen instructions and you can draw all sorts of organization charts and other worthless things.

Hint: Get into graphics mode, press **L** once (*selects single line*), move mouse to starting point, ***press and hold down*** the **left** mouse button, move the mouse (*a line is drawn*), and release the left mouse button to stop drawing. You can erase by holding ***both*** mouse buttons depressed. The **J** (*join key*) can be used to insert the proper joining character at intersections.

The **<Spacebar>** gives you access to the complete IBM character palette. Use the cursor or mouse to select the desired character. Now press **<Enter>** or the **left** mouse button to make it the current drawing character.

Note: you must exit from character graphics (press.<F6> key again) before you can mark, move, or stretch your graphics.

Macros in EDY

Macros are collections of keystrokes that EDY remembers that you can reference with a few keystrokes. You can teach EDY macros, execute macros, save macros, and recall macros.

Recalling a Macro

Be sure that the help screens are active (**<F1>** *key*). Press the **<F9>** key to activate the macro function. Notice that several macros are saved as A-J on your help screen. Press the letter **B**. Notice that Your screen now shows two file windows. Now press **<F9> A** to return to your previous single file window view. In both cases EDY copies the saved macro into the local macro space and executes it as the local macro.

Teaching a Macro

To teach EDY a local macro enter the following two keys: **<F9>** T. Now proceed with your task and EDY will learn each keystroke you enter (*up to 2000*). When you're through with the task press the **<F9>** key again to stop the teaching process. The new macro is now called the local macro. ***You may execute a local macro again by entering <F9> twice or pressing the right mouse button once.***

Saving a Local Macro

A local macro can be saved as an A-J macro. Press the keys ***<F9> S***. EDY will ask for a letter A-J. Enter the desired letter. Now enter the macro name of your choice and the macro is saved. ***Note: Macros are saved for the duration of the current session. To save macros permanently you must save the personality file (as described below).***

The MACROS.EDY File - Alt Macros

EDY also has a **MACROS.EDY** file that is installed with SIMION. It provides an advanced set of ***Alt macros*** for C code development and other tasks. This file is automatically loaded when EDY runs (*if and only if it is on the current search path*).

Viewing the List of Alt Macros

To view list of Alt macros enter **<Alt X>** then **<Alt F1>**. Press **<Esc>** to exit list.

Executing Alt Macros

To execute an Alt macro enter **<Alt X>** then **<Alt A-Z>** (*where A-Z means the letter A through Z that selects the desired macro*). The following Alt macros are explained to get you started (*for C macros: file type of t assumed and auto indents must be ON - <F5>*):

<Alt X> <Alt P>

> *Prints the current file out to any **PCL printer**. File name, date, and page numbering are provided on the listing. This is a quick way to get a nice listing of your user program and geometry files.*

<Alt X> <Alt M>

> *Creates a C function structure at the cursor's location. The cursor will be pointing at the location for entering the function's name.*

<Alt X> <Alt I>

> *Inserts an IF statement structure at the cursor's location. The cursor will be pointing at the location for entering the IF test.*
>
> *Examples of other C structures:*
>
> | | |
> |---|---|
> | **<Alt W>** | *While* |
> | **<Alt D>** | *Do While* |
> | **<Alt S>** | *Switch* |
> | **<Alt C>** | *C /* */ comment* |
>
> *The Alt macros provide an easy way to quickly create good looking C coding structures.*
>
> ***Trick: When inserting comments in C code, move cursor to first line to comment and enter <Alt X> <Alt C>. For each additional line you want to comment, just point the cursor to the line and click the right mouse button (easy and fast comments).***

Saving New Alt Macros

A local macro can also be saved as an Alt macro **<Alt A-Z>** in the same manner as described above. To see a directory of the current Alt macros press **<F9> <Alt F1>** or **<Alt X> <Alt F1>**. If you save any Alt macros, EDY automatically places them in the **MACROS.EDY** in-memory file. When you quit EDY you will be asked if you want to save the modified **MACROS.EDY** file.

Personality Files

EDY's personality can be changed and this new personality can be saved to its personality file. The personality file remembers: Screen color (**<Alt C>** *changes*), insert/overwrite mode, video driver(**<Alt V>** *changes*), file window visibility and sizes, help screens (*on or off*), default file type, and A-J macros.

To change EDY's personality: Restart EDY, change anything you want changed, and then enter the following four keystrokes: **<Esc> S ! <Enter>** to save a new personality file.

File types

EDY supports a whole collection of file formats. Basically a file is converted into a universal memory format at the time it is read into EDY according to its alleged file type (*e.g. (a) for ASCII*). Likewise it is converted back from the universal memory format to its disk file format (*when saved*) according to its currently active file type. ***You can change a file's format by loading it in as one format and saving it as another.***

The following short discussion should give you more insight:

(a) or ASCII File Type

Disk files are expected to contain lines that end with ***linefeed*** and ***carriage return*** characters. These characters are removed during file loading. Tab characters ***are*** expanded. When an ASCII file is saved ***linefeed*** and ***carriage return*** characters are appended to the end of each line. The file is ***not*** tab compressed when saved.

(t) or Tabbed ASCII

These files are ASCII files that may be tab character compressed. EDY automatically ***expands*** each tab character found using the current tab spacing for the file type. On file saving EDY first removes any trailing spaces from the end of each line and then ***tab compresses*** it using the current tab spacing for the file type.

(s) or Special Tabbed ASCII

These are just a second type of tabbed ASCII files. This allows you to have two different tab expansions available without constantly changing the tab spacing parameter (*e.g. for* **.ASM** *files*).

(b) or Binary Files

This format assumes that the file has no lines. EDY loads in arbitrarily fixed length lines. Binary files are saved back *without* line marks. ***Any disk file can be loaded as a binary file type.*** The line length parameter (*Load command*) can be used to load unmarked database records with their proper line length.

(d) or dBASE Files

This format assumes that the file is in dBASE format. EDY loads in the header and places each database record on its own line. You can delete, insert, or edit records. EDY assumes that you know what you are doing (*you can damage the database*). If you have mangled the file too much EDY will refuse to save it as a *(d)* format.

(v) or Variable Length Binary Records

This format assumes that the first two characters in each line form an 16 bit integer number that indicates the length of the line. The first two characters are removed and the rest of the line is loaded into memory. This format is useful if the file must have arbitrary (*binary*) characters and still retain a notion of variable line length. EDY's personality file **EDYSET.EDY** is of *(v)* file type.

User Programming

Introduction

SIMION 6.0 incorporates a very powerful feature called User Programs. **This feature allows you to model ion traps, quadrupoles, RF tuned devices, time-of-flight components, collision cells, and all sorts of other things.** As you become familiar with user programming, it will become apparent that you are really only limited by your imagination.

Whenever SIMION loads either a **.PA** or **.PA0** file it looks for an associated **.PRG** user program file. All such user program files will be automatically compiled and used when flying ions to modify SIMION's behavior (*whenever an ion is in an instance that uses a potential array with user programs*). These program fragments run about 2-5 times slower than direct C code modifications of SIMION (*quite fast*).

SIMION contains a User Program debugger/compiler (*accessed via the* **Test** *button on the Main Menu Screen*) to assist you in the testing, debugging, and development of your user programs. The EDY editing program supplied with SIMION (*or an editor of your choice - see EDY - Appendix H*) can be accessed from within the debugger to create/modify these user programs.

What Is a User Program ?

A user program is an ASCII file that contains one or more program segments (*like sub-programs*) written in an HP calculator based RPN language. As an ion flies within any instance tied to a potential array with active user program segments, SIMION automatically calls each program segment at the appropriate time to allow it to control how the ion flies. ***Trick: Use a crude no-field 3D array (with user programs) sized to workspace volume as instance 1 to control ions outside normal instances.***

These program segments can dynamically change fast adjust electrodes; electrostatic and magnetic fields; ion accelerations; and all sorts of other things. A user program file has the same name (*first eight characters*) as the potential array name it supports and the extension **.PRG** (*e.g.* **TEST.PRG** *is the user program file for* **TEST.PA**). *User Programs can be turned off (See Adj Var. Section).*

Program Segments Within a User Program File

A user program file is composed of an ***optional*** **Define_Data** segment ***and*** from ***one*** to ***eight*** program segments. Each included program segment has a different specific purpose and particular access and control capabilities. SIMION's user program feature is designed to allow any legal combination of user program segments to work together to provide the desired modeling. User programs are much like subroutines because they can communicate, share data, and otherwise support each other. The material below gives a very brief introduction to these program segments:

Seg Define_Data

This segment is ***always the first segment*** in any user program file (*you are not required to declare it - assumed first segment by default*). It contains the definitions for global variables (*visible to all active user program segments - in all instances that support user programs*). These variables are used for storage and communication between user programs. Two types of global variables are

supported: **Adjustable** and **Static**. **Adjustable** variables are displayed/adjusted at the beginning of each **Fly'm** (*adjustable variables can also be user accessed while ions are flying*) to allow you to change or adjust the function of your user program segments without having to edit them. **Static** variables are not user adjustable but are directly accessible to all active user programs for other control and storage functions. ***More information is provided later.***

The Eight Types of User Program Segments

There are eight types of user program segments. Each type is **only** allowed to perform certain specific functions. ***Only one type of each program segment is allowed in any one user program file.*** The following is a brief introduction to each program segment type:

Program_Segment Initialize

The **Initialize** segment is used to dynamically change an ion's initial parameters and conditions. This program segment can output messages and control looping back for another run (*e.g. useful for automatic focusing user programs*).

Program_Segment Tstep_Adjust

The **Tstep_Adjust** segment can be used to examine and possibly change the integration time step (*in microseconds*). This program segment (*if it exists*) is called just before each integration step is performed.

Program_Segment Fast_Adjust

The **Fast_Adjust** segment can be used to examine/adjust the electrode/pole potentials of fast adjust array instances (*with* **.PA0** *potential arrays*) as the ion flies. This program segment (*if it exists*) is called before any initial field determinations are made.

Program_Segment Efield_Adjust

The **Efield_Adjust** segment is used to examine/change the electrostatic field potentials and gradients calculated for each time step. ***Note: this segment type can only be used with electrostatic potential arrays.***

Program_Segment Mfield_Adjust

The **Mfield_Adjust** segment is used to examine/change the magnetic fields calculated for each time step. ***Note: this segment type can only be used with magnetic potential arrays.***

Program_Segment Accel_Adjust

The **Accel_Adjust** segment is used to examine/change the acceleration components for each time step.

Program_Segment Other_Actions

The **Other_Actions** segment is called just after each time step. It used to examine/change ion parameters like: Mass, velocity, splat, and etc. Moreover, the user can output messages and results to the data recording screen and file.

Program_Segment Terminate

The **Terminate** segment is called just after ***all ions*** have died (*splat*). It is used to examine ion parameters like: Mass, velocity, and etc. This program segment can output messages and control looping back for another run (*e.g. useful for automatic focusing user programs*).

Two Examples of a SIMION User Program File

The following listings are of a simple **Accel_Adjust** program segment for adding Stokes Law viscosity effects coded in two very different styles:

A Questionable Programming Style

```
defa viscous_damping 0,seg accel_adjust rcl ion_ax_mm
rcl ion_vx_mm rcl viscous_damping * - sto ion_ax_mm
rcl ion_ay_mm rcl ion_vy_mm rcl viscous_damping * -
sto ion_ay_mm rcl ion_az_mm rcl ion_vz_mm
rcl viscous_damping * - sto ion_az_mm
```

A Suggested Programming Style

```
; This Accel_Adjust program segment adds a simple Stokes' Law Viscosity Effect
; It serves as a starting point for dabbling in ion mobility and atmospheric ion sources.
; The program segment makes use of a simple viscous damping factor and linear viscosity
; Note:  It has problems with high viscosity - see Accel  Adjust below for fixed version

;Note:  a Begin_Segment Define_Data is not required (the compiler assumes it)

        Define_Adjustable Viscous_Damping 0     ; adjustable variable Viscous_Damping
                                                ; set to 0 (no viscous damping by default)
                                                ; adjustable at the beginning of each Fly'm


Begin_Segment Accel_Adjust                      ; start of Accel_Adjust program segment


        Recall Ion_Ax_mm                        ; recall current x acceleration (mm/usec²)
        Recall Ion_Vx_mm                        ; recall current x velocity (mm/sec)
        Recall Viscous_Damping                  ; recall the viscous damping term
        Multiply                                ; multiply times x velocity
        Subtract                                ; and subtract from x acceleration
        Store Ion_Ax_mm                         ; return adjusted value to SIMION

        Recall Ion_Ay_mm                        ; recall current y acceleration (mm/usec²)
        Recall Ion_Vy_mm                        ; recall current y velocity (mm/sec)
        Recall Viscous_Damping                  ; recall the viscous damping term
        Multiply                                ; multiply times y velocity
        Subtract                                ; and subtract from y acceleration
        Store Ion_Ay_mm                         ; return adjusted value to SIMION

        Recall Ion_Az_mm                        ; recall current z acceleration (mm/usec²)
        Recall Ion_Vz_mm                        ; recall current z velocity (mm/sec)
        Recall Viscous_Damping                  ; recall the viscous damping term
        Multiply                                ; multiply times z velocity
        Subtract                                ; and subtract from z acceleration
        Store Ion_Az_mm                         ; return adjusted value to SIMION

Exit                                            ; exit to SIMION (optional statement)
```

Both of the above programs will generate the same compiled code and will run at the same speed. However, the second example will be easy to support and modify later. Remember, you have the freedom to make your programs as cryptic or verbose as you like. There are several characteristics of the language that should be apparent:

Upper and Lower Case

SIMION *ignores the case* of the statements entered. ***You may use upper and lower case freely to improve readability.***

Blank Lines and Indention's

Blank lines are ignored. Use blank lines to create good visual separation of various regions of a program. You may indent code as desired. When properly used, indention's can significantly improve code readability.

The Semicolon ; Starts an In-line Comment

In-line comments begin with a semicolon (*use a leading space or comma in front of the semicolon for separation from any preceding word*). All information after the semicolon (*including the semicolon*) is ignored by the compiler. **Unlike interpreted programs, these comments have no effect on the speed of user program execution (so use them!).**

Word Oriented Language Structure

The user programming language makes use of a word oriented structure. This means that a program is a collection of words ***separated by any number of spaces, commas, tabs, and/or lines.*** These words are analyzed to create the pseudo-machine code that SIMION actually executes. Lines have no meaning except that **all words beyond column 200 will be ignored**.

SIMION checks each word to see if it is a command, number, reserved variable, variable, or label (*in that order*).

Command Words

SIMION's language treats all command words (*e.g.* **STO**) as reserved words (**cannot be used for any other purpose - except in comments**). Note: Most commands have synonyms (*e.g.* **STO** *and* **STORE**). This allows you more freedom of expression in making your programs as cryptic or verbose as desired. Each command will be covered in detail below.

Numbers

Any word that can be converted into a number will be! Thus commands or names cannot be numbers (*e.g. 1.24e-5 is a number, 12345t is not a number*). A number appearing where the compiler expects a command will be interpreted as **Recall Constant** (*of the value of the number*).

The RPN Registers

SIMION makes use of a ten register rotary stack. Movement around the stack is automatic via the insertion and combining of numbers (*double precision - 64 bit*). The stack pointer rolls around the stack so that it always points at the most current number.

The current number is always in the x-register. The number directly above it (*preceding it*) is in the y-register, above it is the z-register, and so on.

Functions like **SIN** replace the current value of the x-register with its sine. Other functions like + add the x and y register values together and place the result in the y-register which now automatically becomes the new x-register (*a roll up*).

Entering a number or recalling a variable places the new number in the register directly below the current x-register. This register now automatically becomes the new x-register (*a roll down*).

Variable Names and Labels

Words that aren't commands and can't be converted into numbers are considered to be candidates for variable names and labels. A variable name or label follows C naming conventions. The first character must be a letter or underscore (*e.g.* _). All remaining characters must be letters, numbers, or underscores. The first **31** characters of variable names and labels are significant (*for matching purposes*). Unlike C, SIMION *ignores the case* of the variable names and labels (*it retains case for display purposes only*).

Moreover, certain names are reserved variables. These variables (*e.g.* **Ion_Time_of_Flight**) allow you to exchange information with and exert control over SIMION. Each of the reserved variables will be covered in detail below.

Unit, Orientation, and Angular Conventions

SIMION's Standard Unit Systems

The basic position/length unit is millimeters (*mm*) or grid units (*gu*) depending on the command or reserved variables. Time is measured in microseconds (*usec*). Velocity is mm/usec. Acceleration is in mm/usec^2. Magnetic fields are in Gauss. Electrostatic gradients are in volts/mm or volts/gu (*depending on the reserved variable*).

Three unit systems are used in connecting user programs with SIMION (*via Reserved variables*). The first is the currently aligned workbench coordinates (mm) and orientation(*Variables using these coordinates/orientations end with _mm*). Variables using this unit system share the locations and orientations of the ***currently aligned workbench coordinates*** (*including* **Align** *button status*).

The second unit system is the ion's current instance's PA volume coordinates (*gu*) and orientation (*these coordinates/orientations end with _gu*). This is a reversible 3D transformation from the current workbench unit system into the 3D grid unit system and orientation of the ion's current instance.

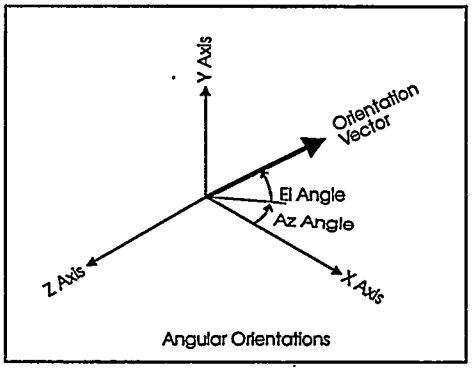
The third unit system is the ion's current instance's PA Array coordinates (*these coordinates end with _Abs_gu*). This is a ***non-reversible transformation*** from 3D PA volume coordinates to the 2D or 3D coordinates of the actual potential array. For 3D arrays x, y, and z are converted into their absolute values. For 2D planar x and y are converted to their absolute values and z is set to zero. For 2D cylindrical x is converted to its absolute value, y and z are converted into r, which is stored in y, and z is set to zero.

SIMION's Angular Conventions

Several user program commands make use of angular input and output parameters (*e.g.* **az** *and* **el**). Angular parameters are in ***degrees*** or ***radians***. Each command that makes use of angles will state whether the angles are in ***degrees*** or ***radians***.

Commands making use of azimuth and/or elevation angles follow the following conventions:

az Azimuth angle to apply when projecting the internal coordinates into external coordinates. Azimuth angle is ***degrees*** of ccw rotation about the y-axis in degrees looking down the positive y-axis toward the origin.



Angular Orientations

Azimuth of 90 degrees. Internal z-axis made parallel to external x-axis. Internal x-axis made parallel to external negative z-axis. Internal y-axis remains parallel to external y-axis (*assuming el = 0*).

el Elevation angle to apply when projecting the internal coordinates into external coordinates. Elevation angle is ***degrees*** of ccw rotation about the z-axis in degrees looking down the positive z-axis toward the origin.

Elevation of 90 degrees. Internal x-axis made parallel to external y-axis. Internal y-axis made parallel to external negative x-axis. Internal z-axis remains parallel to external z-axis (*assuming az = 0*).

Azimuth and elevation transformations are applied in the following order:

1. The elevation (*el*) transformation is applied creating an interim coordinate system.
2. The azimuth (*az*) transformation is then applied to the interim coordinate system to create the resulting coordinate system.

PROGRAMMING COMMANDS

The following is a detailed discussion of each legal user programming command including examples of use. ***Unless stated otherwise, each command is legal in any program segment.*** Command synonyms (*if any*) appear after the ***or:***

+ or: Add

Adds contents of x and y registers, puts result in y-register, and renames it as x-register (*e.g.* **1 2** *+ becomes 3 in register where 1 was originally stored*).

- or: Subtract

Subtracts x from y, puts result in y, and renames it x (*e.g.* **11 5** *- becomes 6 in register where 11 was originally stored*).

* or: Multiply

Multiplies x and y, puts result in y, and renames it x (*e.g.* **5 6 *** *becomes 30 in register where 5 was originally stored*).

/ or: Divide

Divides x into y, puts result in y, and renames it x (*e.g.* **60 10 /** *becomes 6 in register where 60 was originally stored*).

1/X or: Reciprocal_of_X

Converts the contents of the x-register to its reciprocal (*e.g.* **10.0 1/X** *becomes 0.1*).

10^X or: 10_ to_the_X

Converts the contents of the x-register to 10^x (*e.g.* **3 10^X** *becomes 1000*).

>ARR or: PA_Coords_to_Array_Coords

Converts 3D point from the current instance's Potential Array volume coordinates to actual potential **Array** coordinates (*position according to array type: 2D, Cylindrical, 3D and etc.*). On entry the x, y, and z registers are assumed to contain the point's x, y, and z **PA** volume coordinates. On exit the x, y, and z registers contain the point's actual **Array** coordinates. ***This is not a reversible transformation.*** For 3D arrays x, y, and z are converted into their absolute values. For 2D planar x and y are converted to their absolute values and z is set to zero. For 2D cylindrical x is converted to its absolute value, y and z are converted into r, which is stored in y, and z is set to zero.

>AZR or: Azimuth_Rotate

Rotates a 3D vector in the azimuth direction (***rotation around the y component axis.*** *e.g. for -90 degrees old x component becomes new z component*). On entry the y, z, and t registers are assumed to contain the vector's x, y, and z components. The x-register contains the azimuth angle of rotation (*in degrees* **ccw** *from x-axis looking down the positive y axis toward the origin*). On exit the x, y, and z registers contain the vector's rotated x, y, and z components.

>DEG or: Radians_to_Degrees

Converts value in x-register from assumed ***radians to degrees***(*e.g.* **3.1459 >DEG** *becomes 180*). The **>RAD** command performs the reverse transformation.

>ELR or: Elevation_Rotate

Rotates a 3D vector in the elevation direction (***rotation around the z component axis.*** *e.g. for +90 degrees old x component becomes new y component*). On entry the y, z, and t registers are assumed to contain the vector's x, y, and z components. The x-register contains the elevation angle of rotation (*in degrees* **ccw** *from x-axis looking down the positive z axis toward the origin*). On exit the x, y, and z registers contain the vector's rotated x, y, and z components.

>KE or: Speed_to_Kinetic_Energy

Converts from speed (*mm/usec*) to kinetic energy (*eV*). On entry the x-register is assumed to contain the ion's speed and the y-register is assumed to contain the mass of the ion (*amu*). On exit the x-register contains the ion's KE and the y-register is unchanged. ***This transform uses relativistic corrections***. The **>SPD** command performs the reverse transformation.

>P or: Rect_to_Polar

Converts from 2D rectangular to 2D polar coordinates. On entry the x-register is assumed to contain the x value and the y-register is assumed to contain the y value. On exit the x-register contains the radius and the y-register is contains the angle theta ***in degrees***. The **>R** command performs the reverse transformation.

>P3D or: Rect3D_to_Polar3D

Converts from rectangular 3D to polar 3D coordinates. On entry the x, y, and z register are assumed to contain the rectangular x, y, and z vector components. On exit the x-register contains r (*radius*), the y-register contains the azimuth angle ***in degrees***, and the z-register the elevation angle ***in degrees***. The **>R3D** command performs the reverse transformation.

>PAC or: WB_Coords_to_PA_Coords

Converts 3D point from workbench coordinates (*current alignment*) to Potential Array volume coordinates (*of current instance*). On entry the x, y, and z registers are assumed to contain the point's x, y, and z **WB** coordinates. On exit the x, y, and z registers contain the point's **PA** volume coordinates. The **>WBC** command performs the reverse transformation.

>PAO or: WB_Orient_to_PA_Orient

Converts 3D vector from workbench orientation (*current alignment*) to Potential Array volume orientation (*of current instance*). On entry the x, y, and z registers are assumed to contain the vector's x, y, and z **WB** components. On exit the x, y, and z registers contain the vector's **PA** components. ***Note: The magnitude of the vector is not changed, only its orientation.*** The **>WBO** command performs the reverse transformation.

>R or: Polar_to_Rect

Converts from 2D polar to 2D rectangular coordinates. On entry the x-register is assumed to contain r (*radius*) and the y-register is assumed to contain the angle theta ***in degrees***. On exit the x-register contains the x value and the y-register contains the y value. The **>P** command performs the reverse transformation.

>R3D or: Polar3D_to_Rect3D

Converts from polar 3D to rectangular 3D coordinates. On entry the x-register is assumed to contain r (*radius*), the y-register is assumed to contain the azimuth angle ***in degrees***, and the z-register the elevation angle ***in degrees***. On exit the x, y, and z registers contain the rectangular x, y, and z vector components. The **>P3D** command performs the reverse transformation.

>RAD or: Degrees_to_Radians

Converts value in x-register from assumed ***degrees to radians***(*e.g.* **180 >RAD** *becomes 3.1459*). The ***>DEG*** command performs the reverse transformation.

>SPD or: Kinetic_Energy_to_Speed

Converts from kinetic energy (*eV*) to ion speed (*mm/usec*). On entry the x-register is assumed to contain the ion's KE and the y-register is assumed to contain the mass of the ion (*amu*). On exit the x-register contains the ion's speed and the y-register is unchanged. ***This transform uses relativistic corrections***. The **>KE** command performs the reverse transformation.

>WBC or: PA_Coords_to_WB_Coords

Converts 3D point from the current instance's Potential Array volume coordinates to workbench coordinates (*current alignment*). On entry the x, y, and z registers are assumed to contain the point's **PA** x, y, and z volume coordinates. On exit the x, y, and z registers contain the point's **WB** Coordinates. The **>PAC** command performs the reverse transformation.

>WBO or: PA_Orient_to_WB_Orient

Converts 3D vector from the current instance's Potential Array volume orientation to workbench orientation (*current alignment*). On entry the x, y, and z registers are assumed to contain the vector's **PA** x, y, and z components. On exit the x, y, and z registers contain the vector's **WB** components. ***Note: The magnitude of the vector is not changed, only its orientation***. The **>PAO** command performs the reverse transformation.

ABS or: Absolute_Value

Converts the contents of the x-register to a positive number (*e.g.* **-2.5 ABS** *becomes 2.5*).

ACOS or: Arc_Cosine

Converts the contents of the x-register to arc cosine (***in radians***) (*e.g.* **1.0 ACOS** *becomes 0.0*).

ASIN or: Arc_Sine

Converts the contents of the x-register to arc sine (***in radians***) (*e.g.* **1.0 ASIN** *becomes 1.570796*).

ATAN or: Arc_Tangent

Converts the contents of the x-register to arc tangent (***in radians***) (*e.g.* **1.0 ATAN** *becomes 0.785398*).

BELL or: Ring_Bell

Rings computer's bell (*e.g.* **BELL** ---> *beep!!*).

CHS or: Change_Sign

Reverse the sign of the number in the x-register (*e.g.* **2.0 CHS** *becomes -2.0*).

COS **or: Cosine**

Converts the contents of the x-register (***in radians***) to cosine (*e.g.* **0.0 COS** *becomes 1.0*).

DEFA **or: Define_Adjustable**
DEFA Name Number **(e.g. DEFA Omega 1.0)**
Only Legal in Define_Data Segment

Three word command defines an ***adjustable variable*** named *Name* with an initial value of *Number* (***must be an actual number***). *Name* must not conflict with any reserved word or previously defined variable (***any type***) or label. Example: **DEFA Omega 1.0** Means define adjustable variable named Omega with initial value of 1.0. ***Can only appear in a Define_Data segment***. See discussion on variables below for more information.

DEFS **or: Define_Static**
DEFS Name Number **(e.g. DEFS Flag 0.0)**
Only Legal in Define_Data Segment

Three word command defines a ***static variable*** named *Name* with an initial value of *Number* (***must be an actual number***). *Name* must not conflict with any reserved word or previously defined variable (***any type***) or label. Example: **DEFS Flag 0.0** Means define static variable named Flag with initial value of 0.0. ***Can only appear in a Define_Data segment***. See discussion on variables below for more information.

E^X **or: E_to_the_X**

Converts the contents of the x-register to e^x (*e.g.* **1.0 E^X** *becomes 2.71828*).

ENTR **or: Enter Duplicate_X**

Rolls the register pointer down one and copies the contents of the old x-register into the new x-register (*e.g.* **3 ENTR *** *duplicates 3, squares it and places 9 in initial x-register*).

EXIT

Exits the current program segment and returns to SIMION (*e.g.* **3 X>0 EXIT ENTER** *exits program segment because 3 is greater than zero*). ***This is the default command loaded into all of program memory.***

FRAC **or: Decimal_Fraction**

Converts the contents of the x-register to its decimal fraction component (*e.g.* **2.58 FRAC** *becomes 0.58*).

GSB **or: GoSub Go_Subroutine**
GSB Label

Unconditional jump to subroutine at label named **Label. Label** name must be a legal label defined elsewhere in the ***same program segment*** by a **LBL** statement. The first encounter of a **RTN** command within the subroutine will cause execution to resume just after the calling **GSB Label** statement. **Subroutines can be nested up to 100 levels deep.** Example: **GSB Double**

means jump to the label named **Double** (*a subroutine*) and return after encountering a **RTN** statement.

GTO or: **Goto Go_To**
GTO Label

Unconditional jump to label named **Label. Label** name must be a legal label defined elsewhere in the ***same program segment*** by a **LBL** statement. Example: **GTO Entrya** means jump to the label named **Entrya**.

INT or: **Integer**

Converts the contents of the x-register to its integer component (*e.g.* **2.58 INT** *becomes 2.0*).

KEY? or: **Check_For_Key_Input**
Only Legal in Other_Actions

Checks for keyboard input. Inserts the upper case key code in x-register **after rolling pointer down one register**. If no keyboard input is available a zero is placed in the x-register. **The actual key codes generated can be found with the KEY? test button in the debugger.**

LBL or: **Label Entry Subroutine**
LBL Label

Marks a code entry point (*jump or subroutine*) with the name of Label. Label name must not conflict with any reserved word or previously defined variable (***any type***) or label. Example: **LBL Double** means an entry point called **Double.** ***Note: SIMION will only allow jumps or subroutine calls to locations within the same program segment.***

LN or: **Natural_Log**

Converts the contents of the x-register to natural logarithm (*e.g.* **2.71828 LN** *becomes 1.0*).

LOG or: **Base_10_Log**

Converts the contents of the x-register to $base_{10}$ logarithm (*e.g.* **1000 LOG** *becomes 3.0*).

MARK or: **Mark_All_Ions**
Only Legal in Other_Actions Segment

Sets markers for all ions. Useful for making visual marks and as an event to trigger data recording. Note: Forces drawing of current line segment. Handy for forcing clean changes in ion color and etc.

MESS or: **Message**
MESS ; X reg = # and Y reg = #
Only Legal in Initialize, Other_Actions, & Terminate

Displays the following (*same line*) comment (***50 chars max*** *- without the ;*) as a data record line. Useful for user prompts and recording data. **Note: Each # character is replaced by a register value** (***the first left-to-right # is x reg value the second is y reg value and so on***). In the example

above (*assuming x = 23 and y =125.3*) the output would be: **X reg = 23 and Y reg = 125.3.** *The actual format used for the # numbers is the same as currently defined for data recording.*

NOP

The NOP command does nothing. It can be used to fill space or kill time (*e.g.* **NOP NOP NOP NOP KEY?** *kills a little time before a key test*).

R/S or: Run/Stop
Only Legal in Initialize, Other_Actions & Terminate

Program halts execution, informs user on the display, and requests a keystroke to resume. The upper case key code for the key entered will be in the new (*rolled down by one*) x-register when execution resumes. **The actual key codes generated can be found with the KEY? test button in the debugger.**

RAND or: Random_Number

Rolls the register pointer down one and inserts a random number between 0 and 1 into the new x-register.

RCL or: Recall
RCL Name

Rolls the register pointer down one and inserts the value of the variable **Name** in the new x-register. **Name** must be a currently active variable name that the user program segment is allowed to reference (*e.g.* ***RCL TEMP1*** *recalls value of variable named* ***TEMP1***). See more information below concerning: Adjustable, static, reserved, and temporary variables.

RLDN or: Roll_Register_Pointer_Down

Rolls the register pointer down by one (*or registers up by one* ***HP convention***). The current y-register was the prior x-register (*e.g.* **5 10 RLUP RLDN** *has 10 in the new x-register*).

RLUP or: Roll_Register_Pointer_Up

Rolls the register pointer up by one (*or registers down by one* ***HP convention***). The current x-register was the prior y-register (*e.g.* **5 10 RLUP** *has 5 in the new x-register*).

RTN or: Return Return_From_Subroutine

Returns to statement after the calling **GSB** if in a subroutine else returns to SIMION from called user program segment. Note: Use **EXIT** to force return to SIMION from program segment.

SEED or: Random_Seed

Uses the current contents of the x-register as a new seed for the random number generator. The x-register is unchanged.

SEG or: **Begin_Segment**
SEG Name

Begins a new data definition or program segment. **Name** must be one of the following: ***Define_Data, Initialize, Fast_Adjust, Efield_Adjust, Mfield_Adjust, Accel_Adjust, Other_Actions,*** or ***Terminate****(e.g. SEG Efield_Adjust starts the efield adjust program segment)*. The SEG command automatically inserts a leading **EXIT** command to force exiting of any preceding program segment. See discussion of program segments below for more details.

SIN or: **Sine**

Converts the contents of the x-register (***in radians***) to its sine (*e.g.* **1.570796 SIN** *becomes 1.0*).

SQRT or: **Square_Root**

Converts the contents of the x-register to its square root (*e.g.* **81 SQRT** *becomes 9.0*).

STO or: **Store**
STO Name

Stores the current contents of the x-register into the variable named **Name**. **Name** must be a currently active variable name that the user program segment is allowed to reference. **If Name does not exist and it is not illegal, the compiler will create a temporary variable named Name** (*e.g.* **STO TEMP1** *store x-register value to existing variable* **TEMP1** *or to a created temporary variable of that name*). See more information below concerning: Adjustable, static, reserved, and temporary variables.

TAN or: **Tangent**

Converts the contents of the x-register (***in radians***) to its tangent (*e.g.* **1.0 TAN** *becomes 1.557408*).

X><Y or: **X<>Y XY_Swap Swap_XY**

Exchanges the values in the current x and y registers (*e.g.* **1 2 X><Y** *puts 2 in y-register and 1 in x-register*).

X=0 or: **If_X_EQ_0 If_X_Equals_0**
X!=0 or: **If_X_NE_0 If_X_Not_Equal_0**
X<0 or: **If_X_LT_0 If_X_Less_Than_0**
X<=0 or: **If_X_LE_0 If_X_Less_Than_Or_Equal_0**
X>0 or: **If_X_GT_0 If_X_Greater_Than_0**
X>=0 or: **If_X_GE_0 If_X_Greater_Than_Or_Equal_0**

The six test commands above compare the x-register value to zero. If the selected test is true the next instruction following the test will be executed (**do if true**). Else the next instruction will be skipped (*e.g.* **5 X>=0 GSB MORE RTN** *results in calling subroutine* **MORE** *because 5 is greater or equal to 0*).

X=Y	or: If_X_EQ_Y If_X_Equals_Y
X!=Y	or: If_X_NE_Y If_X_Not_Equal_Y
X<Y	or: If_X_LT_Y If_X_Less_Than_Y
X<=Y	or: If_X_LE_Y If_X_Less_Than_Or_Equal_Y
X>Y	or: If_X_GT_Y If_X_Greater_Than_Y
X>=Y	or: If_X_GE_Y If_X_Greater_Than_Or_Equal_Y

The six test commands above compare the x-register value to the y-register. If the selected test is true the next instruction following the test will be executed (**do if true**). Else the next instruction will be skipped (*e.g.* **3 5 X<Y GSB MORE RTN** *results in executing RTN because 5 is not less than 3*).

User Adjustable Variables

These are variables defined by three word **DEFA NAME VALUE** statements placed in the **Define_Data** segment (*at the top of a user program file*). Adjustable variables are read/write global variables that SIMION allows you to assign new values to before (***and during***) each **Fly'm**.

Adjustable variables are **globally visible**. **Thus ALL user program segments in ALL user program files that define an adjustable variable of the same name will actually reference the same adjustable variable.** The initial value of the adjustable variable will be that defined in the ***first*** user program file ***compiled*** that defined the adjustable variable (*compiled in instance order*).

Whenever, adjustable variables are active SIMION will display their names and current values (*in an adjustable panel window*) just before beginning a trajectory computation. You can change any adjustable variable's value before flying the ions. **Any changes at this point will be retained for the next Fly'm**. *Note:* Changes to adjustable variables made **while** ions are flying *(by you or user programs*) will be forgotten when the **Fly'm** terminates *(this allows program segments to communicate across rerun flights without permanently changing adjustable variables).* Adjustable variables are reset to program defined values when a new **.IOB** file is loaded or the user programs are otherwise reset (*e.g. leaving and re-entering* **View**, *using the debugger, and etc.*).

Note: The adjustable variable panel window has an ***ON/OFF*** button for user programs. The ***ON/OFF*** button allows you to turn user programs off during the current (*and only the current*) **Fly'm**. This is handy for applications when you may not want to use user programs in certain runs (*e.g. tuning*). **Hint: You can always define a dummy adjustable variable to gain access to the ON/OFF button.**

SIMION also has an **AdjV** tab on the ***top of the View Screen*** that will be ***unblocked whenever adjustable variables are active*** during a **Fly'm**. Clicking this tab will give you access to adjustable variables (*via a panel screen*) while ions are flying (*selection slider provided if more than three variables to display*). By default, SIMION displays the adjustable variables it encountered when compiling your user programs. ***However***, if any adjustable variable ***begins with an underscore*** (*e.g.* **_Damping**) ***only*** the ***adjustable variables compiled with leading underscores will be displayed*** (*allowing you to select your control variables*). ***Avoid*** displaying ***adjustable variables that the user programs write to*** (*change*), because ***SIMION will not display these changes***.

Static Variables

These are variables defined by the three word **DEFS NAME VALUE** statements placed in the Define_Data segment (*at the top of a user program file*). Static variables are read/write global

variables. **Static variables are very useful for record keeping within a user program segment and for communications between two or more active user program segments.**

Static variables are **globally visible**. **Thus ALL user program segments in ALL user program files that define a static variable of the same name will actually reference the same static variable.** The initial value of the static variable will be that defined in the ***first*** user program file ***compiled*** that defined the static variable (*compiled in instance order*).

Unlike temporary variables the values of the static variables ***are not forgotten*** between calls to the user program segment. **Moreover, SIMION resets each static variable to its specified initial value at the beginning of each individual or grouped ion trajectory calculation (*flight or rerun*).**

Temporary Variables

Whenever the compiler encounters a **STO** statement with a variable name that doesn't match any currently defined variable (**of any type**) it automatically creates a temporary variable for the value. Temporary variables only retain their values during the execution of the program segment. **When the user program segment returns to SIMION all is forgotten.**

Temporary variables **must be defined** via a **STO** statement before they can be accessed. **Thus a RCL to an undefined variable (*assumed temporary*) will result in a fatal compiler error.**

Further, creating a temporary variable via a **STO** statement that is not referenced later via a **RCL** statement will ***also*** result in fatal compiler error. This prevents a miss-spelled reserved variable **STO** being made a temporary variable (*a mistake*) and thus introducing a hard-to-find program bug.

RESERVED VARIABLES AND THEIR FUNCTIONS

User programs communicate to SIMION through special reserved variables. These reserved variables can be read (*user program can input them with a* **RCL** *statement*) or written (*user program can output to them with a* **STO** *statement*). SIMION limits the read/write access to reserved variables by program segment. This keeps you from exerting control at a bad time (***however***, *you are free to control things badly*). A table of reserved variables appears below:

There are three unit systems used with reserved variables. The first is the currently aligned workbench coordinates/orientation(*Variables using these coordinates/orientations end with _mm*). Variables using this unit system share the locations and orientations of the ***currently aligned workbench coordinates*** (*including Align button status*).

The second unit system is the ion's current instance's PA volume coordinates/orientation (*these coordinates/orientations end with _gu*). This is a reversible 3D transformation from the current workbench unit system into the 3D grid unit system and orientation of the ion's current instance.

The third unit system is the ion's current instance's PA Array coordinates (*these coordinates end with _Abs_gu*). This is a ***non-reversible transformation*** from 3D PA volume coordinates to the 2D or 3D coordinates of the actual potential array. For 3D arrays x, y, and z are converted into their absolute values. For 2D planar x and y are converted to their absolute values and z is set to zero. For 2D cylindrical x is converted to its absolute value, y and z are converted into r, which is stored in y, and z is set to zero.

Name	Use	Units	Read Access	Write Access
Adj_Elect01 to Adj_Elect30	Fast Adj Electrode Voltages	Volts	Fast_Adjust	Fast_Adjust
Adj_Pole01 to Adj_Pole30	Fast Adj Pole Mag Potentials	Mags	Fast_Adjust	Fast_Adjust
Ion_Ax_mm Ion_Ay_mm Ion_Az_mm	Ion's current Acceleration (WB Coordinates)	mm/micro sec^2 (WB Orientation)	Accel_Adjust Other_Actions Terminate	Accel_Adjust
Ion_BfieldX_gu Ion_BfieldY_gu Ion_BfieldZ_gu	Magnetic Field at Ion's Location (PA's Orientation)	Gauss (PA's Orientation)	Mfield_Adjust	Mfield_Adjust
Ion_BfieldX_mm Ion_BfieldY_mm Ion_BfieldZ_mm	Magnetic Field at Ion's Location (WB Orientation)	Gauss (WB Orientation)	Accel_Adjust Other_Actions Terminate	None
Ion_Charge	Ion's current charge	in units of elementary charge	Any Prog Seg	Initialize Other_Actions
Ion_Color	Color of Ion	0-15	Initialize Other_Actions	Initialize Other_Actions
Ion_DvoltsX_gu Ion_DvoltsY_gu Ion_DvoltsZ_gu	Voltage Gradient at Ion's Location (PA's Orientation)	Volts/grid unit (PA's Orientation)	Efield_Adjust	Efield_Adjust
Ion_DvoltsX_mm Ion_DvoltsY_mm Ion_DvoltsZ_mm	Voltage Gradient at Ion's Location (WB Orientation)	Volts/mm (WB Orientation)	Mfield_Adjust Accel_Adjust Other_Actions Terminate	None
Ion_Instance	Current Instance	1- max instance	Any Prog Seg	None
Ion_Mass	Ion's current mass	amu	Any Prog Seg	Initialize Other_Actions
Ion_mm_Per_Grid_Unit	Min Current Scaling[1]	mm/grid unit	Any Prog Seg	None
Ion_Number	Ion's Number	1- max ion	Any Prog Seg	None
Ion_Px_Abs_gu Ion_Py_Abs_gu Ion_Pz_Abs_gu	Ion's current PA Array Coordinates	grid units (PA's Abs Coordinates)	Any Prog Seg	None
Ion_Px_gu Ion_Py_gu Ion_Pz_gu	Ion's current (PA's Coordinates)	grid units (PA's Coordinates)	Any Prog Seg	None
Ion_Px_mm Ion_Py_mm Ion_Pz_mm	Ion's current Workbench Coordinates	mm (WB Coordinates)	Any Prog Seg	Initialize Other_Actions
Ion_Splat	Ion Status Flag[2]	Flying = 0 Not Flying != 0	Initialize Other_Actions	Initialize Other_Actions
Ion_Time_of_Birth	Ion's Birth Time	micro seconds	Any Prog Seg	Initialize Other_Actions
Ion_Time_of_Flight	Ion's current TOF[3]	micro seconds	Any Prog Seg	Other_Actions
Ion_Time_Step	Ion's Time Step[4]	micro seconds	All but Initialize	Tstep_Adjust
Ion_Volts	Electrostatic Pot at Ion's Location	Volts	Efield_Adjust Mfield_Adjust Accel_Adjust Other_Actions Terminate	Efield_Adjust
Ion_Vx_mm Ion_Vy_mm Ion_Vz_mm	Ion's current Velocity (WB Coordinates)	mm/micro sec (WB Orientation)	Any Prog Seg	Initialize Other_Actions
Rerun_Flym	Rerun Flym Flag	NO = 0 YES = 1	Initialize Terminate	Initialize Terminate
Update_PE_Surface	New PE Surface[5]	YES != 0	None	Other_Actions

Notes

1 The value in the **Ion_mm_Per_Grid_Unit** reserved variable is ***normally*** the scaling of the ion's current instance. However, if the ion is currently in ***both*** electrostatic and magnetic potential array instances, the value stored is the smaller mm_per_grid_unit scaling (*e.g. for .25 and .30: .25 would be stored in variable*).

2 The **Ion_Splat** reserved variable has several possible SIMION generated values:

0	keep-on-trucking
-1	Hit electrode
-2	Dead-in-water (*ion not moving and no forces on it*)
-3	Outside workbench
-4	Ion killed (*used in beam repulsion*)

The **Other_Actions** segment can be used to monitor and change an ion's fate.

3 The value stored in **Ion_Time_of_Flight** depends on the program segment called:

Initialize	Start time of next step
Tstep_Adjust	Start time of next step
Fast_Adjust	Start time of current test time step (*varies - Runge-Kutta*)
Efield_Adjust	Start time of current test time step (*varies - Runge-Kutta*)
Mfield_Adjust	Start time of current test time step (*varies - Runge-Kutta*)
Accel_Adjust	Start time of current test time step (*varies - Runge-Kutta*)
Other_Actions	Stop time of current step (*start time of following step*)
Terminate	Stop time of last step

4 The value stored in **Ion_Time_Step** depends on the program segment called:

Tstep_Adjust	Full Time step for next step
Fast_Adjust	Test time step (*varies - Runge-Kutta*)
Efield_Adjust	Test time step (*varies - Runge-Kutta*)
Mfield_Adjust	Test time step (*varies - Runge-Kutta*)
Accel_Adjust	Test time step (*varies - Runge-Kutta*)
Other_Actions	Time step of current step
Terminate	Last time step tried

5 Storing a ***non-zero value*** in **Update_PE_Surface** (*e.g. 1*) requests a PE surface update (*can only be called from* **Other_Actions** *segment*). If there is a **Fast_Adjust** ***program segment active*** and ***PE view mode is currently active***, the **Fast_Adjust** program segment will be called after **Other_Action** exits. The fast adjust potentials it returns will be used to fast adjust the entire potential array and its PE surface will be redrawn. **This is an excellent way to make PE surfaces undulate**. See user program demos for examples of how to use effectively: **_BUNCHER**, **_RFDEMO**, **_TRAP**, and **_TUNE**.

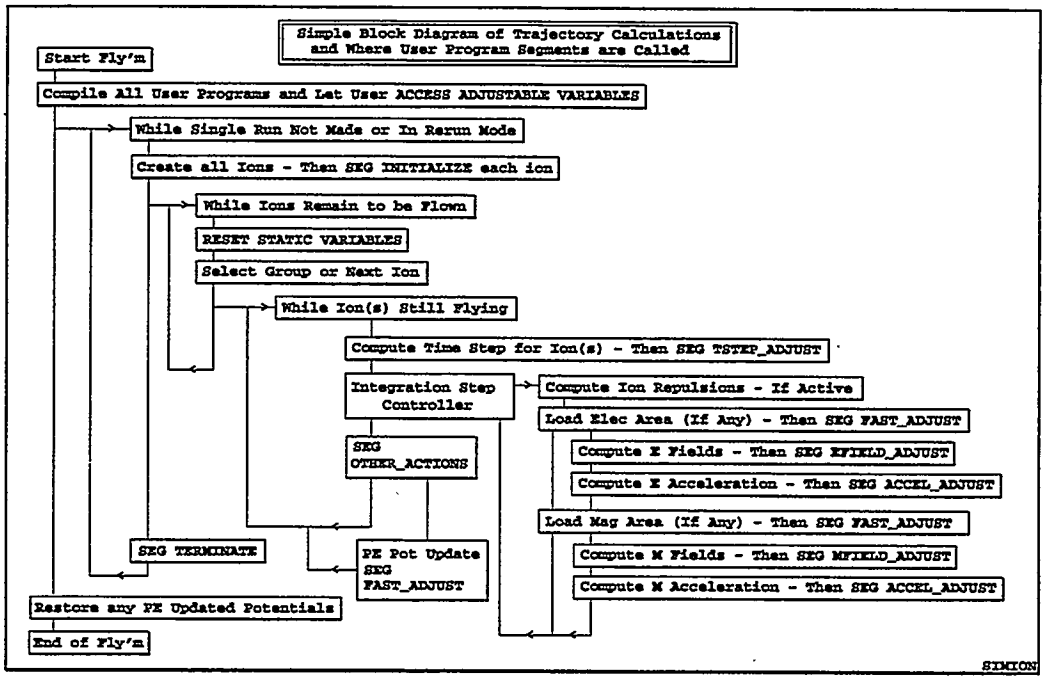
Numerical Constants

Whenever the compiler encounters a number when it is looking for a command it will assume it is a constant to be loaded into the x-register (*via* **RCL** *constant*). In order to save space, it keeps an array of unique numerical constants. Multiple references to the same number will automatically be translated into multiple references to the same numerical constant.

Compiler Limits

The debugging compiler outputs a compilation summary of the resources used. The following memory limits apply to user programs:

1. Program memory is limited to 5,000 commands *per user program file* (*e.g.* **.PRG** *file*).
2. The maximum number of **unique** adjustable variables for *all active user program files* is 200.
3. The maximum number of **unique** static variables for *all active user program files* is 200.
4. The maximum number of **unique** numerical constants *per user program file* is 200.
5. The maximum number of **unique** messages for *per user program file* is 100 (*50 character*).
6. The maximum number of **unique** temporary variables in a *single user program segment* is 200.
7. The maximum number of **unique** entry point labels (*e.g.* **LBL LOOP**) in a *single user program segment* is 200.



Details of User Program Segments

User program segments (*if defined*) are called like subroutines from within SIMION. The illustration above contains a simple flow diagram of ion trajectory calculations. The diagram shows where user programs are compiled, adjustable variables changed, static variables reset, and the various user program segments called. Take the time now to carefully study the general flow of events. It is important that you clearly understand where and when each user program segment is called if you expect to make creative use of user programming within SIMION.

Each user program segment is implemented as a monitor/modifier to the normal course of events. That is, SIMION calculates the next time step to use for an ion *and then* calls the **Tstep_Adjust** program segment (*if it exists*). The **Tstep_Adjust** program segment can then monitor and perhaps change the proposed time step. This gives you complete freedom to monitor, modify, or substitute your own values for important items like: Time step; electrostatic and magnetic fields; accelerations; and even ion mass, charge, and etc.

The reserved variables **Ion_Time_of_Flight** and **Ion_Time_Step** take on different values depending on what program segment they are referenced from. The **Fast_Adjust**, **Efield_Adjust**, **Mfield_Adjust**, and **Accel_Adjust** program segments are called multiple times during a Runge-Kutta integration step. The value of **Ion_Time_of_Flight** is the TOF in microseconds at the *start* of the specific Runge-Kutta term's step. The **Ion_Time_Step** is the specific Runge-Kutta term's time step. You can use these values to get the desired TOF for your uses. ***For example, if you need the middle***

TOF of the term's time step load the Ion_Time_of_Flight and add half the Ion_Time_Step to it. See the notes on reserved variables above for more information.

The Initialize Program Segment

The **Initialize** program segment (*if active*) is called after each ion has been initialized for the next **Fly'm**. At this point you have the option of changing the following parameters: **Ion_Charge, Ion_Color, Ion_Mass,** Ion's WB position, **Ion_Splat, Ion_Time_of_Birth,** Ion's WB velocity, and **Rerun_Flym. Initialize** segment can output **Message** and **R/S** commands. It is useful for supporting the rerunning of trajectories under program control (*looping back from the* **Terminate** *segment via the* **Rerun_Flym** *reserved variable*). ***Use adjustable variables to communicate between reruns.***

```
Defa Mass 100                          ;Mass to use set by user and Terminate

Seg Initialize                         ; beginning of segment
      RCL Ion_Vz_mm                    ; recall starting velocity
      RCL Ion_Vy_mm
      RCL Ion_Vx_mm
      >P3D                             ; convert to polar 3d
      RCL Ion_Mass                     ; recall initial mass
      x><y >KE                         ; get ion's ke
      x><y RLUP RCL Mass STO Ion_Mass  ; substitute current mass
      x><y >SPD x><y RLUP              ; convert back to speed
      >R3D                             ; get new velocity with same ke & new mass
      STO Ion_Vx_mm RLUP               ; save new starting velocity
      STO Ion_Vy_mm RLUP
      STO Ion_Vz_mm
```

The Tstep_Adjust Program Segment

The **Tstep_Adjust** program segment is used to monitor/adjust the **Ion_Time_Step** (*in micro seconds*) to use with the selected ion. ***Note: The time step is really a requested time step.*** Other circumstances like other ions (*grouped ion flying*), delayed time-of-birth, binary boundary approaches of the ion(s) can result in a ***shorter*** time step being used. However, if you write your program segment to be persistent you can hit exact time-of-flight points. The example below serves as an illustration:

```
Defa Start_Time 100                    ;starting time to use

Seg Tstep_Adjust                       ; beginning of segment
      RCL Start_Time
      RCL Ion_time_of_Flight
      X>=Y EXIT                        ; exit if ion at or beyond start time
      RCL Ion_Time_Step +
      X<=Y EXIT                        ; exit if time step less or equal to that needed
      RCL Start_Time
      RCL Ion_time_of_Flight -
      STO Ion_Time_Step                ; else shorten time step to what is just right
```

The Fast_Adjust Program Segment

The **Fast_Adjust** program segment is only legal with **.PAO** fast adjust potential arrays. This program segment allows you to fast adjust the potentials of adjustable electrodes or poles as the ion flies. This is very useful for all sorts of simulations. **Note: Only change via a Fast_Adjust**

program segment those electrode potentials you need to change as the ions fly (*faster and saves RAM*).

In order to execute as fast as possible, SIMION only loads a copy of each ***needed*** fast adjust electrode solution PA file (*because the* **Fast_Adjust** *program segment changes their electrode's/pole's potential*) into memory (*e.g.* **.PA1**). If your computer doesn't have enough RAM (*RAM needed = 8 bytes * array size * electrodes stored*) virtual memory will be used (*only if active*). ***If not enough memory is available (real or virtual) SIMION will abort the trajectory run (you may have to increase your virtual size).*** Once these files have been loaded SIMION will try to avoid reloading them (*potentially a slow process*).

The example shows a simple **Fast_Adjust** program segment (*Hint: Use* **Tstep_Adjust** *segment above to create a precise turn-on time*):

```
Defa Start_Time 100                     ; starting time to use
Defa AC_Voltage 500                     ; Voltage to use
Defa Omega 25                           ; angular frequency

Seg Fast_Adjust                         ; beginning of segment
        0.0 STO voltage                 ; assume zero volts
        RCL Ion_time_of_Flight
        RCL Start_Time
        X>Y goto done                   ; skip ac if start time > TOF
        - RCL Omega * SIN               ; sin(omega * (TOF - Start_Time))
        RCL AC_Voltage * STO voltage    ; ac voltage to use
LBL done
        RCL voltage
        STO Adj_Elect01                 ;adjust electrode number one
```

The Efield_Adjust Program Segment

The **Efield_Adjust** program segment can be used to monitor and set the **Ion_Volts, Ion_Dvoltsx_gu, Ion_DvoltsY_gu,** and **Ion_DvoltsZ_gu** electrostatic potential and fields. ***Note: This user program segment can only be used with an electrostatic potential array.*** Even though your potential array may be 2D, the potentials and fields supplied/required are the full 3D fields produced by your array as projected as a 3D object in the workbench. The field gradients are in volts per grid unit and are PA oriented (*to simplify your task*). SIMION scales and orients these fields into workbench coordinates after program segment exit.

```
Defa Start_Time                         ; time to start field
Defa AC_Voltage 500                     ; voltage to use
Defa Omega 25                           ; angular frequency

Seg Efield_Adjust                       ; beginning of segment
        RCL Ion_Time_of_Flight          ; test to see that field is on
        RCL Start_Time -
        x>0 goto running                ; jump to running if field is on
        0 STO Ion_Volts                 ; return zero field
        STO Ion_DvoltsX_gu
        STO Ion_DvoltsY_gu
        STO Ion_DvoltsZ_gu
        EXIT
lbl running
        RCL Omega * SIN                 ; sin(omega * (TOF - Start_Time))
        RCL AC_Voltage *                ; ac voltage to use
        STO Ion_DvoltsZ_gu              ; set gradient in z
        RCL Ion_Pz_gu *                 ; compute voltage
```

```
STO Ion_Volts                    ; save voltage
0 STO Ion_DvoltsX_gu             ; set x and y field components to zero
STO Ion_DvoltsY_gu
```

The Mfield_Adjust Program Segment

The **Mfield_Adjust** program segment can be used to monitor and set the **Ion_BfieldX_gu, Ion_BfieldY_gu,** and **Ion_BfieldZ_gu** magnetic fields (*gauss*). ***Note: This user program segment can only be used with a magnetic potential array.*** Even though your potential array may be 2D, the potentials and fields supplied/required are the full 3D fields produced by your array as projected as a 3D object in the workbench. The field gradients are in gauss and are PA oriented (*to simplify your task*). SIMION orients these magnetic fields into workbench orientation after program segment exit.

```
Defa Start_Time                  ; time to start field
Defa Gauss 5000                  ; magnetic field to use

Seg Mfield_Adjust                ; beginning of segment
    0 STO Ion_BfieldX_gu         ; assume field is off
    STO Ion_BfieldY_gu
    STO Ion_BfieldZ_gu
    RCL Ion_Time_of_Flight       ; test to see that field is on
    RCL Start_Time
    x>y EXIT                     ; field is off
    RCL Gauss STO Ion_BfieldZ_gu ; create field in z direction
```

The Accel_Adjust Program Segment

The **Accel_Adjust** program segment can be called by electrostatic and magnetic potential array instances. In either case, SIMION computes the electrostatic or magnetic accelerations (*as appropriate*) and ***adds these components to the total acceleration*** and then calls the **Accel_Adjust** program segment. The program segment can then input and modify the ***total acceleration at this point*** as appropriate.

From the flow diagram (*above*) note the ***order of the total acceleration calculation***:

1. Ion Repulsion Accelerations (*if any*)
2. Electrostatic Accelerations(*if any*) are added
3. Magnetic Accelerations(*if any*) are added

Trick 1: To isolate the electrostatic acceleration in electrostatic PA's **Accel_Adjust** segment ***when charge repulsion is active*** (*total acceleration contains both at that time*), create an **Efield_Adjust** segment in the **.PRG** file that saves the acceleration components to three static variables (*total acceleration contains only ion repulsion accelerations at that point*). Now in the **Accel_Adjust** segment recall the current total acceleration and subtract the ion repulsion acceleration from it (*stored in the static variables*). A similar trick could be used to isolate magnetic components from the total acceleration in a magnetic PA's user programs.

Trick 2: If you want to add viscosity effects using the integration stabilization ***below*** it should be applied to the true total acceleration. The proper location for the **Accel_Adjust** segment would be in the electrostatic PA's user program file if only electrostatic fields are active or in the magnetic PA's user program file if magnetic fields are active (*too or only*).

An Improved Accel_Adjust Segment for Viscosity

The problem with viscosity is that it has an exponential transient decay of acceleration. When the damping term is very small (***large time constant***) the Runge-Kutta integration works just fine. However, when the damping is high (***small time constant***) the Runge-Kutta integration can overestimate the acceleration badly. SIMION's CV self protection (*when quality > 0*) detects this problem and chops the time step. While the computation is salvaged in this manner the time step can be so small that it may take 30 minutes or longer to fly a single ion.

The trick is to give the Runge-Kutta system what it really wants: ***An estimate of the average acceleration within the test time step***. It can be shown that the average acceleration can be computed by multiplying the initial total acceleration including viscosity (*at the beginning of the test time step*) by the following factor:

factor = (1 - e^(-tstep * damping))/(tstep * damping)

```
; This fixed Accel_Adjust program segment adds a Stokes' Law Viscosity Effect
; It is an example of the external problem fix (in the honored NASA Hubble tradition)
; It also demonstrates the real power of SIMION's user programming

Define_Adjustable Viscous_Damping 0          ; adjustable variable Viscous_Damping

Begin_Segment Accel_Adjust                   ; start of accel_adjust program segment
        Recall Ion_Time_Step x=0 Exit        ; exit if zero time step
        Recall Damping x=0 Exit              ; exit if zero damping
        * Store nt                           ; number of time constants in step
        chs e^x 1 x><y -                     ; (1 - e^(-nt))
        Recall nt / Store factor             ; divide by nt store as factor

        Recall Ion_Ax_mm                     ; recall current x acceleration (mm/usec^2)
        Recall Ion_Vx_mm                     ; recall current x velocity (mm/sec)
        Recall Viscous_Damping               ; recall the viscous damping term
        Multiply                             ; multiply times x velocity
        Subtract                             ; and subtract from x acceleration
        Recall factor *                      ; multiply by exponential averaging factor
        Store Ion_Ax_mm                      ; return adjusted value to SIMION

        Recall Ion_Ay_mm                     ; recall current y acceleration (mm/usec^2)
        Recall Ion_Vy_mm                     ; recall current y velocity (mm/sec)
        Recall Viscous_Damping               ; recall the viscous damping term
        Multiply                             ; multiply times y velocity
        Subtract                             ; and subtract from y acceleration
        Recall factor *                      ; multiply by exponential averaging factor
        Store Ion_Ay_mm                      ; return adjusted value to SIMION

        Recall Ion_Az_mm                     ; recall current z acceleration (mm/usec^2)
        Recall Ion_Vz_mm                     ; recall current z velocity (mm/sec)
        Recall Viscous_Damping               ; recall the viscous damping term
        Multiply                             ; multiply times z velocity
        Subtract                             ; and subtract from z acceleration
        Recall factor *                      ; multiply by exponential averaging factor
        Store Ion_Az_mm                      ; return adjusted value to SIMION
Exit                                         ; exit to SIMION (optional statement)
```

The Other_Actions Program Segment

The **Other_Actions** program segment (*if active*) is called after each ion's time step. At this point you have the option of changing the following parameters: **Ion_Charge, Ion_Color, Ion_Mass,** Ion's WB position, **Ion_Splat, Ion_Time_of_Birth,** and Ion's WB velocity. This program is most useful for changing ion parameters (*e.g. mass or charge*) during its flight.

Note: You can also perform binary boundary approaches (*of all types*) with **Other_Actions** and **Tstep_Adjust** segments working together. Each time the **Other_Actions** is called it could look for a boundary. If the boundary was crossed (*e.g. velocity*) it would restore ion's parameters at the end of last step (*stored in static variables by* **Initialize** *and* **Other_Actions** *just before they exit*) and flag a halving of the time step to the **Tstep_Adjust** segment. ***Of course there would have to be some minimum time step looping exit limit or the program would lock up.***

```
; For use with Tstep_Adjust segment above to neutralize ions at Start_Time
Defa Start_Time 100                          ; time to make change

Seg Other_Actions                            ; beginning of segment
      RCL Start_Time                         ; time to make change
      RCL Ion_Time_of_Flight                 ; ion's TOF
      x!= 0 EXIT                             ; exit if not at switch point
      RCL Ion_PZ_mm                          ; get ion's position
      RCL Ion_PY_mm
      RCL Ion_PX_mm
      RCL Ion_Number                         ; get ion's number
                                             ; output ion data at neutralization point
      MESS ; Ion # Neutralized at: x = #, y = #, z = #
      BEEP                                   ; sound a beep too
      1 STO Ion_Color                        ; switch ion's color to red
      0 STO Ion_Charge                       ; neutralize ion
      mark                                   ; mark event
```

The Terminate Program Segment

The **Terminate** program segment (*if active*) is called ***after*** all ions have stopped flying in the current flying cycle. At this point you have the option of changing the **Rerun_Flym** reserved variable. Setting this variable to 1 depresses the **Rerun** button and the ions are re-flown. Setting **Rerun_Flym** to 0 turns off the **Rerun** button and the current **Fly'm** is terminated. Note: If the **Rerun_Flym** variable is not changed, rerunning retains its current status (***that of the* Rerun *button***). ***Note: Use Adjustable variables to communicate between reruns.***

Important: When the **Rerun** Button is depressed (*by you or* **Rerun_Flym**) ion trajectories are not saved (*remembered for redrawing*). ***Trick***: To save the trajectories in the last run clear the **Rerun_Flym** variable with the Initialize segment ***at the start of the last run. Also: Data recording to a file*** is blocked ***if and only if*** the **Rerun** button is depressed ***before*** the **Fly'm** button is clicked.

```
; For use with Initialize segment above as a looping demo
Defa Mass 100                                ; Mass to use set by user and Terminate
Defa Del_Mass 1                              ; delta mass between runs
Defa N_Runs 10                               ; number of runs to make

Seg Terminate                                ; beginning of segment
      RCL Ion_Number
      1 X!=Y EXIT                            ; exit if not ion number one
      1 STO Rerun_Flym                       ; set rerun by default
      RCL Mass RCL Del_Mass + STO Mass       ; Mass += Del_Mass
```

```
RCL N_Runs 1 - STO N_Runs      ; dec n_runs by one
x>0 EXIT                       ; rerun if runs remain
0 STO Rerun_Flym               ; terminate Fly'm after this run
```

User Program Demos

There are several user program demonstration sub directories. ***The files in these subdirectories were shipped compressed.*** To maximize compression of .PA files all non-electrode points were set to zero. ***Thus you must load, refine and save specific arrays before you can execute the demos successfully.*** The **README.DOC** files in each of these sub directories contain instructions on what to do.

The user program directories are:

_BUNCHER:	Ion bunching demo
_DRAG:	Simple Stoke's Law viscosity demo
_ICRCELL:	Full 3D ICR Cell with external ion injection modeled
_QUAD:	Quadrupole demo with 3D modeling of entrance and exit regions
_RANDOM:	Random ion generator demo
_RFDEMO:	Simple demos of various ways to generate RF effects
_TRAP:	Ion Trap Demo with damping, ion repulsion, and undulating PE surfaces
_TUNE:	Auto-focusing demo for a simple lens

Creating and Testing User Programs

User program files can be created with EDY as a simple ASCII file. The actual creation of a user program file is normally done outside SIMION. However, you can click the GUI File Manager button on the Main Menu Screen and then click the **Edit** button in the file manager to gain direct access to EDY from within SIMION. ***Remember that all user program files must have the same name as their potential array and a .PRG extension. Moreover, all user program files must be in the same project directory as the potential arrays they support.***

Running Another Editor From Within SIMION

The Debugger uses the EDY editor by default. If you prefer another editor, use the **DOS SET** command to link SIMION to your editor (*works for many editors*). The line below makes **EDLIN** the active editor (*or non-editor*):

SET GUI_EDITOR=EDLIN

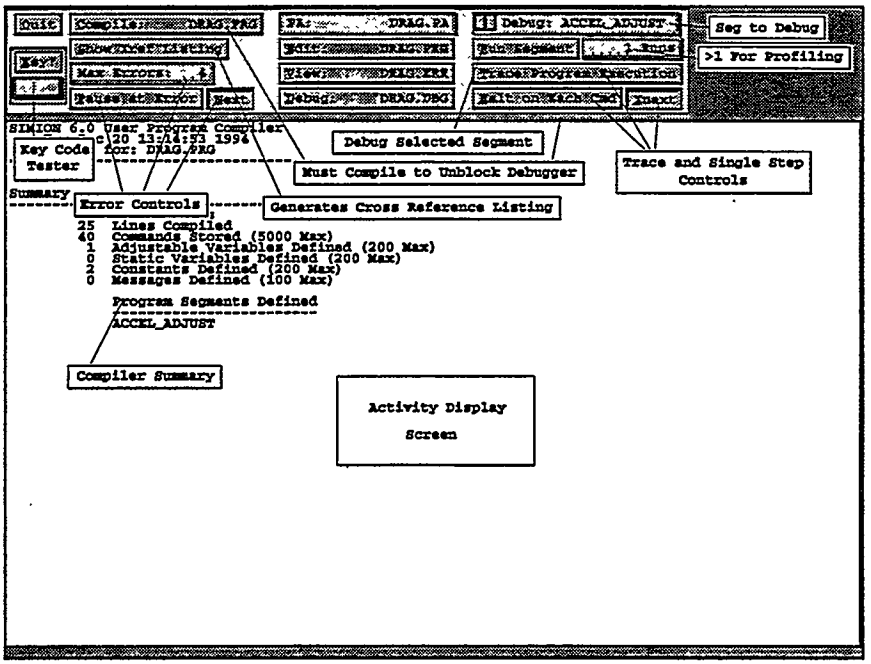
Testing User Programs with SIMION's Debugger

Whenever SIMION loads a potential array it automatically compiles any associated user program file. To test or debug the user program segments associated with a particular potential array, click the potential array's button within the active PA window on the Main Menu Screen. If the selected potential array has a user program file the **Test & Debug** button will be unblocked. To test or debug these user program segments click the **Test & Debug** button.

Note: There is also a way to access the debugger from within the **View** function. Click the **PAs** tab, select the desired instance, and then click the **Debug** button. ***Access to the* Debug *button will be blocked if the instance's PA has no .PRG file or ions are currently flying.***

Getting the Lay of the Land

The debugging compiler screen is composed of a collection of control objects above an activity display screen (*see illustration below*). There are four groups of objects: Key code support, compiler controls, file access, and debugging controls.



Key Code Support

Note: This feature is used to determine the key codes that **KEY?** and **R/S** commands put in the x-register. The **Key?** button and the display object below it provide key code support. To determine the key code of any keyboard key, click the **Key?** button and then press the desired key (*or key combination, e.g.* **<Ctrl A>**). The key code will appear in the display object.

Compiler Controls

The next column of objects to the right are the compiler controls. They are used to test compile the current **.PRG** file. The compiler outputs to a **.ERR** file. This file is kept if there are errors or if the **Show Xref Listing** button is depressed.

The **Compile** button is used to start the test compiler. **Note: You must test compile successfully (*no errors*) before access to the debugger will be unblocked.**

The **Show Xref Listing** button is used to generate a cross reference listing in the **.ERR** file. This is useful to see how your user program was compiled. It also provides the code addresses so you can locate the problem in your source when run-time errors are generated.

The last three objects are used for compilation error control. The panel object is used to set the maximum number of errors allowed before test compilation aborts. Setting the value to one aborts compilation on the first error. You can then correct the source file (**Edit** *button*) and try again. The two remaining buttons are used to activate a pause at each error and the compile to the next error options.

File Access

The center column of objects contain three buttons that allow you to access specific files with EDY. The **Edit** button accesses the current **.PRG** file. When you return from editing the source file (**.PRG**) SIMION assumes that you probably changed something, and automatically blocks access to the debugger until you successfully test compile again.

The **View** button is used to edit the **.ERR** file. **This file is only kept (*stored*) if there is an error or the Show Xref Listing button was depressed during a test compile.**

The **Debug** button is used to edit the **.DBG** debugger output file. Each time the debugger runs it creates a **.DBG** file containing the debugger's output.

Debugging Controls

The right column of objects are used for running the debugger. ***These controls are blocked until a successful test compilation has been made.*** The selector object (*on top*) allows you to select the program segment to debug.

The **Run Segment** button is used to run the debugger on the selected program segment. When the debugger executes it compiles the selected segment; allows you to set the values of all adjustable, static, and reserved variables *used*; and then runs the program segment.

The type of run is controlled by the remaining four control objects. The **Runs** panel object accesses the run-time profiler if more than one run is requested. This allows you to determine how fast your user program segment executes. Be sure to use a large enough number of runs to get a relatively accurate set of estimates. This feature is very useful if you need to optimize the performance of a user program segment.

The **Trace Program Execution** button puts the debugger in trace mode. Each command that executes produces a line of trace information containing the command, its address, function, and the contents of the lowest four registers (*x, y, z, and t*).

If the **Halt on Each Cmd** button is depressed the debugger will run in trace mode and halt after each command is executed. Click the **Xnext** button to execute the next command. These controls are useful for single step execution of a program segment.

Runtime Errors

The user program run-time system is designed to catch most execution errors (*e.g. dividing by zero*). When a run-time error is detected (*whether in the debugger or when flying ions*) the run-time system will halt the execution, display the type of error and it command's address, and abort any further execution of user program segments (*e.g. abort the* **Fly'm**). ***Use a cross-reference listing to find the location of the error in your source.***

Endless Loop Lockups

The user program run-time system is designed to allow you to exit an accidental endless loop lockup. If you are flying ions just hit the **Esc** key and the loop will be exited and the **Fly'm** aborted.

If you loop lock in the debugger you can also hit the **Esc** key. ***However, there is also a more useful approach.*** Click on the **Halt on Each Cmd** button (***YES,*** *the mouse works in the debugger even in a locked loop*). The debugger will instantly switch to single step trace mode. Now click the **Xnext** button to step through the execution and see just where and under what conditions the program segment is loop locked. When you've found it, just hit the **Esc** key to stop the debugger.

Geometry Files

Introduction

SIMION provides two methods for defining the geometry of electrode/pole points in a potential array: The **Modify** function and/or geometry files. **Modify** allows the user to interactively create, modify, and view electrode/pole point geometry. Geometry files are typically used for complex 3D geometry and/or an any geometry definitions that may need to be scaled (*e.g. doubled without introducing the jags*).

Geometry files can either be used in conjunction with **Modify** or as a stand-alone method for geometry definition within **New**. The **Modify** function has a geometry file development system within it. **Modify**'s geometry file development system provides a quick way to write, test, and modify geometry files.

Note: This is an advanced SIMION feature. It requires that you learn a geometry definition language. However, the effort you spend learning to create geometry files will add powerful capabilities to your SIMION bag of tricks.

What Is a Geometry File ?

A geometry file is an ASCII file with a **.GEM** extension (*e.g.* **TEST.GEM**) that contains a series of fill (*and other*) instructions. Geometry instructions are similar in structure to C language functions. However, unlike C language functions, geometry instructions are nested (*somewhat like in PASCAL*) to enhance their power.

How Does SIMION Process a .GEM File?

The geometry compiler in SIMION reads the selected geometry file and converts its **Fill** (*and other*) instructions into a decision list in RAM. SIMION then uses this decision list to determine the fate of each point in the target potential array. ***This is done one point at a time.***

SIMION takes the coordinates of the point and ***looks from the last* Fill *defined toward the first* Fill *defined*** in the decision list until it encounters the first **Fill**'s volume that contains the point. When and if this **Fill** is encountered the point is changed to the **Fill**'s value and the search stops for that point.

The effect is the same as if each **Fill** were applied to the potential array as it was encountered while reading the geometry file (*not an efficient approach*). ***Each point will have the type and value of the last* Fill *in the geometry file that changed it. This is a very important concept to remember!***

Two Examples of a SIMION .GEM File

Below are two listings of **.GEM** files that create a **101x, 101y, 101z** 3D potential array and insert a hollow one volt electrode point ellipsoid in the middle. Note: These geometry files are coded in two very different (*though legal*) styles. It is suggested that you study these simple examples carefully. They provide an introduction to the world of SIMION geometry files:

A Questionable Geometry Definition Style

```
pa_define(101,101,101,p,n)locate(50,50,50){e(1){fill{within{sphere(,,,45,25,45)}notin{sphere(,,,40,20,40)
}}}}
```

A Suggested Geometry Definition Style

```
; This geometry file creates a 101,101,101 planar non-mirrored 3d PA and
; inserts a hollow ellipsoid in the middle

; Define PA to create

PA_Define(101,101,101,planar,non-mirrored)

; Define ellipsoid

Locate(50,50,50,1,0,0,0)                    ; locate geometry origin in center of PA
    {
    Electrode(1)                            ; electrode of one volt
        {
        Fill                                ; solid fill
            {
            Within{Sphere(0,0,0,45,25,45)}  ; within ellipsoid centered at geometry origin
                                            ; rx = 45, ry = 25, rz = 45
            Notin{Sphere(0,0,0,40,20,40)}   ; not in ellipsoid centered at geometry origin
                                            ; rx = 40, ry = 20, rz = 40 (hollow out ellipsoid)
            }
        }
    }
```

A Quick Demo of Geometry Files

Do the following if you want a demonstration of geometry files. Start SIMION from the **\SIM6** directory. Click the **New** button. Click the **Use Geometry Definitions** button. Click on the **GEOMETRY** directory (*below* **\SIM6**). Point to the **DEMO.GEM** file button and click ***both*** mouse buttons. **DEMO.GEM** is the example above. After the PA is created and the geometry is inserted, look at it with the **View** function. Cut the ellipsoid in half to verify that it is hollow.

Now exit **View** and enter the **Modify** function. Click the **GeomF** button (*bottom button on left edge*) to access the geometry development facility. Click the **View** button (*accesses EDY*) to view the geometry file. Exit EDY (**<ESC> Q N**). Now erase the potential array by clicking the **Erase Entire PA** button. Click the **Quit** button to return to **Modify** and verify that the array is erased. Click the **GeomF** button to reenter the geometry development facility. Now click the **Insert into PA** button and the geometry will be reinserted. ***The above example should serve to wet your appetite for the material below.***

Geometry Language Rules

Both of the above geometry definitions will generate the same potential array and electrode geometry and will run at the same speed. However, the second example will be easy to support and modify later. Remember, you have the freedom to make your geometry definitions as cryptic or verbose as you like. There are several characteristics of the language that should be apparent:

Upper and Lower Case

SIMION *ignores the case* of the geometry instructions entered. ***You may use upper and lower case freely to improve readability.***

Blank Lines and Indention's

Blank lines are ignored. Use blank lines to create good visual separation of various regions of geometry definitions. You may indent as desired. When properly used, indention can significantly improve instruction readability (*particularly nested geometry language instructions*).

The Semicolon ; Starts an In-line Comment

In-line comments begin with a semicolon. All information after the semicolon (*including the semicolon*) is ignored by the geometry compiler (*to the end of the current line*). **These comments have no effect on the speed of geometry files (so use them!).**

Line length Limits

The geometry compiler ***ignores*** characters ***beyond column 200*** in all lines.

Instruction Oriented Language Structure

Geometry files are composed of a ***nested*** collection of geometry instructions. Examples of the three formats used for geometry instructions are discussed below:

Type 1: Sphere(,,,45,25,45)
Type 2: Fill{ }
Type 3: Locate(50,50,50){ }

The Instruction Name - Required: All Types

All geometry instructions begin with their name (*e.g.* **Sphere**). SIMION supports the use of synonyms for many of its geometry instructions. For example: The "**e**" is a synonym of "**electrode**". The specific discussions of each instruction give its synonyms (*if any*).

Parameter List ( ) - Required: Types 1 & 3

A parameter list (*when required*) immediately follows the instruction name and is delimited by a left and right parenthesis (*e.g. ( )*). One or more parameters (*as required by the specific instruction*) are entered within the parenthesis delimiters. Parameters can be numbers ***or*** words (*depending on the instruction*) and are ***separated by commas and/or spaces*** (*e.g. (1,2,3) same as (1 2 3)*).

Most parameters have a default value. That is, if you skip them, SIMION will assume a value for them. For example: **Circle()** is equivalent to **Circle(0,0,10)**. You may use commas to skip (*use default values for*) one or more parameters. For example: **Circle(10,,30)** is equivalent to **Circle(10,0,30)**.

The specific discussions of each instruction give its parameter requirements (*if any*) along with default parameter values.

The Instruction Scope { } - Required: Types 2 & 3

Many instructions have what is known as a *scope* (*range of effect*). ***The scope of an instruction is limited to instructions that appear within its wavy brackets (e.g. { }).*** These scope brackets ***always*** appear immediately after the instruction name (*type 2*) or after the required parameter list (*type 3*).

Scope is a very important concept in geometry files because it is used for ***nesting*** instructions (***instructions inside of instructions***). It delimits the range of effect or bounds of an instruction. The following instruction segment serves as an example:

```
electrode(100)                  ; use electrode of 100 volts
    {
    fill{....}                  ; fill something with 100 volt electrode points
    electrode(200)              ; use electrode of 200 volts
        {
        fill{...}               ; fill something with 200 volt electrode points
        }
    fill{...}                   ; fill something with 100 volt electrode points
    }
fill{...}                       ; fill something with 0 V electrode points (default)
```

In the example above the first **Fill** uses electrode points of 100 volts. The second **Fill** uses electrode points of 200 volts because it is inside the scope of a 200 volt electrode definition. The third **Fill** uses electrode points of 100 volts because it is in the scope of the 100 volt electrode points. The fourth **Fill** is using the default value of 0 volt electrode (*assuming that this is not an included geometry file*).

Classes of Instructions

There are several classes of instructions. The following discusses each class of instruction and gives the names of the instructions in that class:

PA Definition Class

Instructions: PA_Define()

Defines potential array to create if **.GEM** called from **New** function. ***This must always be the first instruction in .GEM file when defined*** (*optional instruction*).

Include Class

Instructions: Include()

The Include Class instruction allows a **.GEM** file to reference another **.GEM** file. Thus you can keep component definitions in separate **.GEM** files and have a base **.GEM** file reference and insert these components in their desired locations.

Point Definition Class

Instructions: **Electrode(){}**
Non_Electrode(){}
Pole(){} *synonym for Electrode*
Non_Pole(){} *synonym for Non_Electrode*

The Point Definition Class of instructions define the point type and potential to use in fills within their scope. ***Note: Pole and Electrode are synonyms.***

Fill Class

Instructions: **Fill{}**
Edge_Fill{}
Rotate_Fill(){}
Rotate_Edge_Fill(){}

Fill Class instructions define the four types of fills supported in geometry files: **Fill** is used for full ***defined*** volume fills, **Edge_Fill** is used to change only the boundary points of the ***defined*** volume. **Rotate_Fill** is used to create a fill volume by rotating a surface of revolution through an angle. **Rotate_Edge_Fill** is used to change only the boundary points of a fill volume produced by rotating a surface of revolution through an angle.

The scope of these instructions contains the volume inclusion and exclusion instructions (*e.g.* **Within**s *and* **Notin**s*: below*) to be used to determine the volume they act upon. To be acted on by a fill instruction (*any of the four types*) a point ***must be within a defined inclusion volume*** (*e.g.* **Within**) ***and not within any exclusion volume*** (*e.g.* **Notin**).

Within Class

Instructions: **Within{}**
Notin{}

Within Class instructions always appear within the scope of a Fill Class instruction. They contain one or more Test Class instructions (*e.g.* **Circle()**). Each Test Class instruction within a Within Class instruction is tested and the results are ***logically AND***ed. Thus in the following statement group: **within{sphere(0,0,0,25)sphere(20,0,0,25)}** only those points that fall within ***both*** sphere definitions will be considered to be within the inclusion volume.

Within is used to define inclusion volumes. **Notin** is used to define exclusion volumes. More than one **Within** or **Notin** can appear within the scope of any Fill Class instruction. If more than one of each appears their results are ***logically OR***ed. Thus a ***point*** (*to be acted on by a fill*) ***must be*** <u>***within at least one***</u> ***of the*** **Within**s ***and*** <u>***not within any***</u> ***of the*** **Notin**s.

Test Class

Instructions: **Box(), Centered_Box(), Corner_Box()**
Box3d(), Centered_Box3d(), Corner_Box3d(),
Circle(), Cylinder(), Sphere()
Parabola(),Hyperbola()
Points(), Points3D(), Polyline()

Test Class instructions can only appear within the scope of a Within Class instruction (*e.g.* **Within** *or* **Notin**). they test to see if the point in question is within their volume. If it is, they return a logical ***TRUE***. If more than one Test Class instruction appears within a Within Class instruction (*e.g. within*) their results are ***logically AND***ed. ***All tests must return TRUE in a Within Class instruction for the result to be considered TRUE.***

Note that test class instructions contain both 2D (*e.g.* **Circle()**) and 3D (*e.g.* **Sphere()**) instructions. All 2D instructions are defined in terms of an xy plane (*e.g. z = 0*). However all 2D instructions are assumed to extend to plus or minus 10^6 in the z axis direction so they can be used with 3D arrays. Any 3D instruction can also be used in a 2D array. It will test TRUE in the areas where it intersects the z = 0 plane (*xy plane*).

Of course all this becomes much more interesting (*complex*) when one sprinkles Location Class instructions within the scopes of a geometry file (*below*). ***Note: Test Class instructions have minimal definition options because Locate Class instruction(s) can always be used to more precisely define them.***

Location Class

Instructions: **Locate(){}**
Project(){} **synonym for locate**

Location Class instructions are used to locate (*e.g. project*) the geometry axis within their scope (*by location, scaling and orientation*) onto the geometry axis outside their scope. These are the ***workhorse instructions of geometry files*** because they can appear within any scope in any geometry file. ***Most errors result from the misuse of* Locate *instructions***.

As a contrived example: Let's assume we want a cylindrical tube (*with elliptical hole*) with ends cut at a 45 degree angle (*forming elliptical ends*). Further, we want the resulting tube pointed down and centered on the x-axis. The following instruction fragment will do this:

```
locate(0,0,0,1,-90)                ; swing cylinder -90 degrees (cw) to center on x-axis
    {
    fill                           ; volume fill with current point and type
        {
        within                     ; include volume
            {
            circle(0,0,10)         ; radius 10 tube with infinite z extent
            locate(0,0,0,1,45)     ; swing box volume ccw 45 degrees
                {
                box(0,-100,40,100) ; box of x width of 40 (infinite z extent)
                }                  ; use large values of y so circle limits volume in y
            }
        notin{circle(0,0,8,4)}     ; bores 8 rx by 4 ry elliptical hole in circular rod
        }
    }
```

To understand what goes on here (*without yet knowing instruction details*) ***one must work from the inside out when dealing with* Locates** (*projects*). Thus the box must be thought of as rotated *ccw* 45 degrees in azimuth in order to compute its intersection volume with the circular rod of the circle instruction. The result is then rotated *cw* 90 degrees in azimuth to center it along the x-axis beyond the outer locate. ***Wasn't that fun!***

Instruction Nesting Rules

Geometry instruction classes have rules concerning when and where they can appear in a geometry file. This is called instruction nesting rules. These nesting rules make use of the notion of nesting levels. ***The geometry compiler enforces these rules!***

A geometry file has three nesting levels: ***Base Nesting Level, Fill Nesting Level, and Within Nesting Level.*** A geometry file is at ***Base Nesting Level*** whenever instructions are ***outside the scope of a Fill Class instruction.*** The level is at ***Fill Nesting Level*** whenever instructions are ***inside the immediate scope of any Fill Class instruction.*** Finally, the level is at ***Within Nesting Level*** whenever instructions are ***inside the immediate scope of any Within Class instruction.*** The sample geometry code fragment below serves to demonstrate nesting levels:

```
e(...)
    {                                          ;at Base Nesting Level
    locate(...)
        {                                      ;at Base Nesting Level
        fill
            {                                  ;at Fill Nesting Level
            locate(...)
                {                              ;at Fill Nesting Level
                within
                    {                          ;at Within Nesting Level
                    circle(...)
                    locate(...){box(...)}
                    }                          ;return to Fill Nesting Level
                }                              ;at Fill Nesting Level
            notin
                {                              ;at Within Nesting Level
                hyperbola(...)
                }                              ;return to Fill Nesting Level
            }                                  ;return to Base Nesting Level
        }                                      ;at Base Nesting Level
    }                                          ;at Base Nesting Level
```

The following gives the classes of commands that can appear within each of the three nesting levels:

Classes Legal in Base Nesting Level

Point Definition Class
Include Class
Fill Class
Location Class

Classes Legal in Fill Nesting Level

Within Class
Location Class

Classes Legal in Within Nesting Level

Test Class
Location Class

Note:

1. Location Class instructions (*e.g.* **Locate**) are ***legal at any nesting level.***
2. Also, a PA Definition Class instruction (*e.g.* **PA_Define**) must always be the first base level instruction in a **.GEM** file ***if defined.***

Geometry Instructions

The following is a detailed discussion of each legal geometry instruction. Instruction synonyms (*if any*) appear after the ***or:***

Box	or: Box2D Format: box(xmin, ymin, xmax, ymax) Default Values: box(0,0,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 2D box (*e.g. on current xy plane: z = 0*). Parameters define the 2D min. and max. corner locations of the box. The box extends plus or minus 10^6 in z. Returns ***TRUE*** if point within its bounds.

box(-5,,50,60)

Defines 2D box with lower left corner at -5x, 0y and upper right corner at 50x, 60y. Zmin is -10^6 and zmax is 10^6.

Box3D	or: Format: box3d(xmin, ymin, zmin, xmax, ymax, zmax) Default Values: box3d(0,0,0,10,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 3D box. Parameters define the 3D min. and max. corner locations of the box. Returns ***TRUE*** if point within its bounds.

box(-5,,-10,50,60,100)

Defines 3D box with lower left corner at -5x, 0y, -10z and upper right corner at 50x, 60y, 100z.

Centered_Box	or: Centered_Box2D, Cent_Box, Cent_Box2D Format: centered_box(xc, yc, xw, yw) Default Values: centered_box(0,0,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 2D centered box (*e.g. on current xy plane*). Parameters define the center point and dimensions of the box. The centered box extends plus or minus 10^6 in z. Returns ***TRUE*** if point within its bounds.

centered_box(-5,,50,60)

Defines 2D centered box with center at -5x, 0y and dimensions of 50x wide, 60y high. Zmin is -10^6 and zmax is 10^6.

Centered_Box3D	or: Cent_Box3D Format: centered_box3d(xc, yc, zc, xw, yw, zw) Default Values: centered_box3d(0,0,0,10,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 3D centered box. Parameters define the center point and dimensions of the box. Returns *TRUE* if point within its bounds.

centered_box(-5,,-10,50,60,100)

Defines 3D centered box with center at -5x, 0y, -10z and dimensions of 50x wide, 60y high, 100z deep.

Circle	or: Ellipse Format: circle(xc, yc, rx, ry) Default Values: circle(0,0,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 2D circle or ellipse (*e.g. on current xy plane*). Parameters define the center and radii. The 2D circle extends plus or minus 10^6 in z. ***Note: If ry is defaulted, a circle of radius rx will be drawn.*** Returns *TRUE* if point within its bounds.

circle(15,20,30,10)

Defines and ellipse centered 15x, 20y with radii of 30 rx, 10 ry. Zmin is -10^6 and zmax is 10^6.

Corner_Box	or: Corner_Box2D, Corn_Box, Corn_Box2D Format: corner_box(xmin, ymin, xw, yw) Default Values: corner_box(0,0,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 2D corner box (*e.g. on current xy plane*). Parameters define the min. corner point and dimensions of the box. The corner box extends plus or minus 10^6 in z. Returns *TRUE* if point within its bounds.

corner_box(-5,,50,60)

Defines 2D corner box with lower left corner at -5x, 0y and dimensions of 50x wide, 60y high. Zmin is -10^6 and zmax is 10^6.

Corner_Box3D	or: Corn_Box3D Format: corner_box3d(xmin, ymin, zmin, xw, yw, zw) Default Values: corner_box3d(0,0,0,10,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 3D corner box. Parameters define the min. corner point and dimensions of the box. Returns *TRUE* if point within its bounds.

corner_box3d(-5,,-10,50,60,100)

Defines 3D corner box with lower left corner at -5x, 0y, -10z and dimensions of 50x wide, 60y high, 100z deep.

Cylinder	or: Format: cylinder(xc, yc, zc, rx, ry, length) Default Values: cylinder(0,0,0,10,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 3D circular or elliptical cylinder. Parameters define the center at one end, radii, and length. ***Length is always converted to its absolute value and extends in the minus z axis direction. Note: If ry is defaulted, a cylinder of radius rx will be drawn.*** Returns *TRUE* if point within its bounds.

cylinder(15,20,30,10,,50)

Defines a circular cylinder with one end centered at 15x, 20y, 30z with a radius of 10 and length of 50 (*e.g. extending from z = 30 to z = -20*).

Edge_Fill	or: Edge_Fill_Volume Format: edge_fill { } Default Values: NA Class: Fill Class When Legal: Base Nesting Level

Defines a ***volume edge fill*** using the currently active point type and potential. It raises the nesting level to Fill Nesting Level within its scope (*e.g. { }*). ***Its scope must contain at least one* Within *or* Notin *instruction.*** If no **Within** instruction is supplied, the point is assumed within, subject to rejection by **Notin** tests. ***There is no limit to the number of* Withins *or* Notins *that can appear inside the scope of a Fill Class instruction.***

The **Edge_Fill** instruction fills only the boundary points (*edge*) of the equivalent fill volume. Thus if a normal fill would create a solid sphere, an **Edge_Fill** would create a spherical shell. This has the same function as the **Edge** command in **Modify**.

```
edge_fill
    {
    within{sphere(0,0,0,50,20,50)}
    notin{circle(0,0,20,10)}
    }
```

Creates an ellipsoid shell with an elliptical shell passing through its center in the z direction.

Electrode	or: E, P, Elect, Pole, Electrode_Points, Pole_Points Format: electrode(potential) { } Default Values: electrode(0) { } Class: Point Definition Class When Legal: Base Nesting Level

Defines fill point type of electrode or pole and its associated potential to use within the instruction's scope (*e.g. {}*).

```
electrode(1)
    {
    fill{...}                    ; Electrode/Pole points of one volt used for fill
    }
```

Fill	or: Fill_Volume Format: fill { } Default Values: NA Class: Fill Class When Legal: Base Nesting Level

Defines a ***full volume fill*** using the currently active point type and potential. It raises the nesting level to Fill Nesting Level within its scope (*e.g. { }*). ***Its scope must contain at least one within or notin instruction.*** If no within instruction is supplied, the point is assumed within, subject to rejection by notin tests. ***There is no limit to the number of withins or notins that can appear inside the scope of a fill class instruction.***

Each within or notin instruction defines a separate volume (*through the use of tests: e.g.* **circle**()). Points are checked against these tests to determine if the various **Withins** and/or **Notins** are ***TRUE. A point is considered to be within the fill if it is inside <u>at least one</u>*** **Within** ***volume and <u>not inside any</u>*** **Notin** ***volumes***. The results of multiple **Withins** and/or **Notins** are ***OR***ed with their own kind to satisfy the above test.

```
fill
    {
    within{circle(0,0,20)}
    within{circle(100,20,20)}
    notin{circle(0,0,10)}
    notin{circle(100,20,10)}
    }
```

Fills two hollow circles (*2D potential array*) or two hollow tubes (*3D potential array*). The tubes are centered at 0x, 0y and 100x, 20y. Both have an outside radius of 20 and an inside radius of 10. This of course assumes that there are no locates outside of the fill and thus the units are in potential array grid units.

Hyperbola	or: Format: hyperbola(xc, yc, rx, ry) Default Values: hyperbola(0,0,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 2D hyperbola ***in the y-axis direction only*** (*e.g. on current xy plane*). Parameters define the center and focus radial offset of vertices in x and y. The 2D hyperbola extends plus or minus 10^6 in z. Returns ***TRUE*** if point within its bounds.

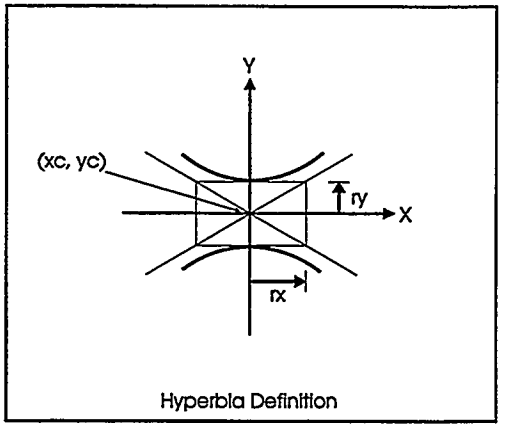
Function used: $(y - yc)^2/ry^2 - (x - xc)^2/rx^2 = 1$

hyperbola(50,0,10,20)

Defines hyperbola extending in y directions with center at 50x, 0y with radius to x vertices of 10 and y vertices of 20. ***Hint: Use* Locate *instruction to orient hyperbolas in x direction when desired:***

locate(50,0,0,,,-90){hyperbola(0,0,20,10)}

Example above defines matching x direction hyperbola to y direction hyperbola defined above it. Zmin is -10^6 and zmax is 10^6.



Hyperbla Definition

Include	or: Include_File Format: Include(Filename) Default Values: None Class: Include Class When Legal: Base Nesting Level

Includes (*inserts*) geometry instructions from the referenced **.GEM** file (*Filename: e.g.* **test.gem**) at the point of the include instruction's location in the referencing geometry file. Included geometry files can reference other geometry files via includes. The geometry compiler limits include nesting to 15 levels deep to protect against accidental **Include** recursion (*an include file directly or indirectly calling itself*).

The Filename cannot be defaulted and must be a legal geometry file name (*e.g.* **test.gem**). The **.GEM** file extension will automatically be added (*e.g.* **TEST** *changed to* **TEST.GEM**) when needed.

A suggested use for **Include** files involves components. You can define a component in an include **.GEM** file and then reference it from a calling **.GEM** file:

```
;lens1.gem file image
locate(,,,,-90)                    ;swing lens to align with x-axis
    {                              ;centered on origin
    fill                           ;simple volume fill
        {
        within{cylinder(0,0,-2,50,,4)}   ;cylinder with r = 50 centered at origin
        notin{circle(0,0,5)}             ;define aperture in lens of r = 5
        }
    }
;end of lens1.gem file image
```

```
;fragment of referencing geometry file
locate(10){ e(1){ include(lens1) } }      ;locate lens1 at 10x with ele points of 1 volt
locate(35){ e(2){ include(lens1) } }      ;locate lens1 at 35x with ele points of 2 volts
locate(48){ e(3){ include(lens1) } }      ;locate lens1 at 48x with ele points of 3 volts
```

The above example works only if lens1 is defined in some useful orientation for the calling locates to use (*as in the example above*). Moreover, it is often useful for the point type to be specified in the referencing **.GEM** file as opposed to the included **.GEM** file (*as shown above*).

Locate	or: Project, Project_it, Transform Format: locate(x,y,z,scale,az,el,rt) { } Default Values: locate(0,0,0,1,0,0,0) { } Class: Location Class When Legal: Any Nesting Level

***Locates** (projects) **geometry coordinates within its scope** (internal geometry coordinates) **into the geometry coordinates active <u>just outside</u> its scope** (external geometry coordinates)* using the defined transformation parameters:

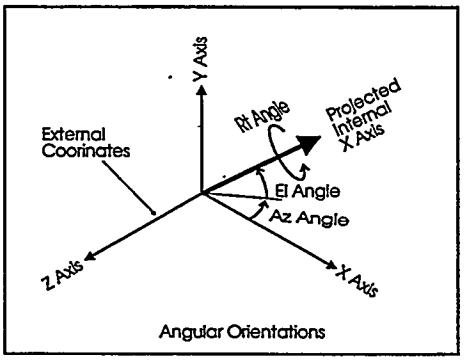
x,y,z — Amount to offset the geometry origin when translating it from internal to external coordinates.

locate(10,20,30) { }: Projects the internal geometry origin to **10x, 20y, 30z** in external geometry coordinates.

scale — Scaling factor to use when translating internal coordinates to the equivalent external coordinates.

locate(,,,2) { }: Scales internal coordinates by a **factor of two** to translate them into the external coordinates.

az — Azimuth angle to apply when projecting the internal coordinates into external coordinates. Azimuth angle is ***degrees*** of ***ccw*** rotation about the ***y-axis*** looking down the positive ***y-axis*** toward the ***origin***.



locate(,,,,90) { }: Azimuth angle of 90 degrees. Internal z-axis made parallel to external x-axis. Internal x-axis made parallel to external negative z-axis. Internal y-axis remains parallel to external y-axis (*assuming el = rt = 0*).

el — Elevation angle to apply when projecting the internal coordinates into external coordinates. Elevation angle is ***degrees*** of ***ccw*** rotation about the ***z-axis*** looking down the positive ***z-axis*** toward the ***origin*** .

locate(,,,,,90) { }: Elevation angle of 90 degrees. Internal x-axis made parallel to external y-axis. Internal y-axis made parallel to external negative x-axis. Internal z-axis remains parallel to external z-axis (*assuming az = rt = 0*).

rt Rotation angle to apply when projecting the internal coordinates into external coordinates. Rotation angle is *degrees* of *ccw* rotation about the ***x-axis*** looking down the positive ***x-axis*** toward the ***origin***.

locate(,,,,,90) { }: Rotation angle of 90 degrees. Internal y-axis made parallel to external z-axis. Internal z-axis made parallel to external negative y-axis. Internal x-axis remains parallel to external x-axis (*assuming az = el = 0*).

Order Transforms are Applied

The above transformations are applied in the following order (*via a 3D transfer matrix*) :

1. The rotation (**rt**) transformation is applied first, creating a new interim coordinate system.
2. The elevation (**el**) transformation is applied next to the interim coordinate system creating the next interim coordinate system.
3. The azimuth (**az**) transformation is then applied to the interim coordinate system creating the next interim coordinate system.
4. The scaling (**scale**) transformation is then applied.
5. Finally the origin offset (**x,y,z**) transformation is applied.

How Locates are Actually Used by SIMION

SIMION converts each locate instruction into a 3D transfer matrix. Transfer matrices of nested locate instructions are multiplied to obtain the aggregate 3D transfer matrix to translate test instruction coordinates into potential array coordinates. The inverse of this aggregate 3D transfer matrix is used in translating potential array coordinates into the current test instruction coordinates.

Example of Nested Locate Instructions

Sometimes it is easier to visualize a transformation (*not get lost*) by using nested locates: ***Remember: Always apply nested locates from the innermost locate working outward toward the outermost locate!***

```
;single locate twists cylinder along x-axis and then elevates it 30 degrees
Locate(0,0,0,1,-90,0,30)              ;First: rt ccw 30 degrees then az cw 90 degrees
    {
    fill{within{cylinder(0,0,0,20,,50)}}
    }

;double locate twists cylinder along x-axis and then elevates it 30 degrees
Locate(,,,,,30)                       ;Then: el ccw 30 degree to elevate toward y-axis
    {
    Locate(,,,,-90)                   ;First: az cw 90 degree to align with x-axis
        {
        fill{within{cylinder(0,0,0,20,,50)}}
        }
    }
```

Non_Electrode	or: N, Non_E, Non_P, Non_Pole, Non_Electrode_Points, Non_Pole_Points Format: non_electrode(potential) { } Default Values: non_electrode(0) { } Class: Point Definition Class When Legal: Base Nesting Level

Defines fill point type of non_electrode or non_pole and its associated potential to use within the instruction's scope.

```
non_electrode()
    {
    fill{...}                          ;fill with Non_Electrode points of zero volts
    }
```

Notin	or: Format: notin { } Default Values: NA Class: Within Class When Legal: Fill Nesting Level

Must be called from within at the Fill Nesting Level (*within a Fill Class instruction:* **Fill**). Holds one or more Test Class instructions within its scope that define a volume (*3D potential array*) or area (*2D potential array*). ***There is no limit to the number of Test Class instructions that can appear within the scope of a Within Class Instruction.***

If a potential array point is contained within the intersection volume (*or area*) of the tests contained in its scope, a ***TRUE*** is returned. Thus, all the test class instructions (*e.g.* **circle()**) inside the scope of a Within Class instruction (*e.g.* **within{}**) must return ***TRUE*** for the Within Class instruction to return ***TRUE*** to the Fill Class instruction.

All that is required for a point to be considered ***NOT*** within a fill's volume is that ***at least one*** **Notin** instruction returns a ***TRUE***.

```
fill
    {                                  ;within right half of ellipsoid
    within{sphere(0,0,0,50,30,50) box(0,-30,50,30)}
                                       ;notin inner ellipsoid circle combo
    notin{sphere(0,0,0,45,25,45) circle(0,0,25)}
    }
```

The example above creates the right half of an ellipsoid ($x > 0$) centered 0x, 0y, 0z and outer shell of 50rx, 30ry, 50rz with an inner ellipsoid circle removed from it (*a complex shape*).

PA_Define	or: Format: pa_define(nx, ny, nz, Sym, Mirror, Type, ng) Default Values: pa_define(100,20,1,Cyl,Y,Elect,100) Class: PA Define Class When Legal: When First Instructions in File

Defines potential array to create if .GEM called by New function. Instruction is ignored when called from within **Modify**. ***However, its parameters are always checked for errors***. If it exists, it must always be the ***first*** command in the .GEM file (*optional command*).

nx The x dimension of the potential array. ***Must always be 3 or greater.***

ny The y dimension of the potential array. ***Must always be 3 or greater.***

nz The z dimension of the potential array. ***Must always be 1 or greater.*** A value of 1 defines a 2D array. Values greater than one define a 3D array.

Sym The symmetry of the array: Cylindrical or Planar. The compiler just checks the ***first letter***. Thus "**C**" or "**P**" are sufficient. ***Note: 3D arrays must always have planar symmetry***.

Mirror The array mirroring can be: **None, X, Y, Z, XY, YZ, XZ, XZY**. The compiler just scans the string for "**X**", "**Y**", and "**Z**". If none of these three characters are found a mirroring of **None** is assumed. Legal mirroring varies by array symmetry and if 2D or 3D:

All 3D arrays:	All mirroring options are legal
Planar 2D arrays:	All mirroring ***except*** z are legal
Cylindrical 2D arrays:	y mirroring is required, x is legal, ***z is illegal***

Type The array type: Electrostatic or Magnetic. The compiler just checks the ***first letter***. Thus "**E**" or "**M**" are sufficient.

ng The magnetic scaling parameter (*discussed elsewhere in main manual*). Must always be 1 or greater.

Example: PA_Define(101,51,71,p,yz,m,30)

The array defined above has dimensions of 101nx, 51ny, 71nz. It is 3D planar with mirroring in y and z. The array's type is magnetic with a ng scaling value of 30.

Parabola	or: Format: parabola(xv, yv, focus_offset) Default Values: parabola(0,0,10) Class: Test Class When Legal: Within Nesting Level

Defines 2D y direction parabola (*e.g. on current xy plane: z = 0*). Parameters define the vertex and focus_offset in y. Both positive and negative numbers for focus_offset are legal (*0 is illegal*). The 2D parabola extends plus or minus 10^6 in z. Returns ***TRUE*** if point within its bounds.

Function used: $(y - yv) = (x - xv)^2/(4 * focus\_offset)$

parabola(15,20,30)

Defines parabola in positive y direction with vertex at 15x, 20y with focus_offset of 30. Hint: Use **Locate** instruction to rotate parabola to x-axis when desired. Zmin is -10^6 and zmax is 10^6.

Points	or: Points2D Format: points(x,y, x,y, ...) Default Values: None Class: Test Class When Legal: Within Nesting Level

Defines a collection of 2D (*e.g. on current xy plane: z = 0*) points (un*connected points*) . Parameters define the x,y coordinates of each point. ***Up to 100 x,y points can be defined in the parameter list.*** Each 2D point extends plus or minus 10^6 in z. Returns ***TRUE*** if point within its bounds of one of the defined points.

```
fill                                        ;solid fill with current point def
   {
   locate(50,50)                            ;shift center to 50x, 50y
      {
      within{points(   -15,-15              ;create three points at ends of triangle
                       15,-15
                       0,15)}
      }
   }
```

Defines three points at the ends of triangle. This produces three lines in a 3D array. Note the use of **Locate** to make the definition easier. Zmin is -10^6 and zmax is 10^6.

SIMION tries to map each point into a point (*2D*) or line (*3D*) in the potential array. However, odd scaling and orientations can result in up to *four* PA points being changed instead of one. **It is recommended that points be defined in potential array coordinates to avoid this issue.**

Points3D	or: Format: points3d(x,y,z x,y,z ...) Default Values: None Class: Test Class When Legal: Within Nesting Level

Defines a collection of 3D points (un*connected points*) . Parameters define the x,y,z coordinates of each point. ***Up to 65 x,y,z points can be defined in the parameter list.*** Returns ***TRUE*** if potential array point is within bounds of one of the defined points.

```
fill                                        ;solid fill with current point def
   {
   locate(50,50)                            ;shift center to 50x, 50y
      {
      within{points3d(-15,-15,0             ;create three 3D points at ends of triangle
                      15,-15,0
                      0,15,0)}
      }
   }
```

Defines three points at the ends of triangle on the z = 0 plane. Note the use of **Locate** to make the definition easier.

SIMION tries to map each point into a point in the potential array. However, odd scaling and orientations can result in up to six potential array points being changed instead of one. **It is recommended that points be defined in potential array coordinates to avoid this issue.**

Polyline	or: Format: polyline(x,y, x,y, ...) Default Values: None Class: Test Class When Legal: Within Nesting Level

Defines 2D (*e.g. on current xy plane: z = 0*) polyline (*connected line segments*) . Parameters define the x,y endpoints of the connected line segments. ***Up to 100 x,y pairs can be defined in the parameter list.*** SIMION assumes the polyline defines a 2D closed area (*automatically forcing closure with extra line segment if required*). The 2D polyline closed area extends plus or minus 10^6 in z. Returns ***TRUE*** if point within its bounds.

```
fill                                        ;solid fill with current point def
    {
    locate(50,50)                           ;shift center to 50x, 50y
        {
        within{ellipse(0,0,40,30)}          ;ellipse 40rx, 30ry
        notin{polyline(  -15,-15            ;create triangular hole
                         15,-15
                         0,15)}
        }
    }
```

Defines an ellipse with a triangular hole in it. SIMION automatically closes the triangle polyline. Note the use of **Locate** to make the definition easier. This produces a tube in a 3D array. Zmin is -10^6 and zmax is 10^6.

Remember this is an area (*volume*) fill test. You can backtrack along a polyline to attempt grids. However, odd angles and scaling may result in a pretty shabby grid definition. ***It is recommended that you not backtrack, but rather define a volume, edge fill it, and later (below) erase the unwanted edge boundary portions with non-electrode fills (see example in*** **Rotate_Edge_Fill** ***below).***

Rotate_Edge_Fill	or: Rotate_Edge_Fill_Volume Format: rotate_edge_fill(Angle_of_Revolution) { } Default Values: rotate_edge_fill(360) { } Class: Fill Class When Legal: Base Nesting Level

Defines a ***volume of revolution edge fill*** using the currently active point type and potential. It raises the nesting level to Fill Nesting Level within its scope (*e.g. { }*). ***Its scope must contain at least one*** **Within** ***or*** **Notin** ***instruction.*** If no **Within** instruction is supplied, the point is assumed within, subject to rejection by **Notin** tests. ***There is no limit to the number of*** **Within**s ***or*** **Notin**s ***that can appear inside the scope of a Fill Class instruction.***

Each **Within** or **Notin** instruction defines a separate ***area of intersection with upper half of the xy-plane*** (*z = 0 and y >= 0*) through the use of tests (*e.g.* **circle**()). The coordinate system used is

the coordinate system's scope that **Rotate_Edge_Fill** instruction appears in. **Locate** *instructions appearing within the scope of the* **Rotate_Edge_Fill** *do not change this test coordinate system. They merely change where the tests may intersect its xy plane.*

This fill area is then rotated *ccw* **Angle_of_Revolution** degrees around the *x-axis* looking down the positive **x axis** toward the **origin** (*same as rt angle in* **Locate**). *All potential array points that fall on the EDGE of this volume of revolution will be changed to the currently active type and potential.*

The **.GEM** file fragment below uses a **Rotate_Edge_Fill** to create a parabolic grid. *It is important that you take the time to understand this instruction fragment if you are to successfully use* **Rotate_Edge_Fill** *properly.*

```
;fragment below makes a 180 degree parabolic mirror grid surface

pa_define(101,101,101,p,n)                    ;101, 101, 101 3D planar non-mirror

locate(10,50,50)                              ;locate point of rotation for grid center
   {
   e(1)                                       ;use one volt electrode points
      {
      rotate_edge_fill(180)                   ;180 degree rotate fill
         {                                    ;parabolic mirror in x-axis direction
         within{locate(,,,,,-90){parabola(0,0,10)} box(0,0,50,100)}
         }
      }
   n(0)                                       ;zero volt non-electrode to erase
      {
      fill                                    ;erase edge electrodes cut planes
         {                                    ;erase edge electrodes in z = 0 plane
         within{locate(,,,,,-90){parabola(0,0,10)} box3d(-0.5,-1000,0.5,1000,1000,0)}
                                              ;erase edge electrodes in x = 50 plane
         within{box3d(49.5,-1000,-1000,50.5,1000,1000)}
         }
      }
   }
```

Creates a 180 degree parabolic grid in the x-axis direction in a 3D array. Note: The use of a **Locate** instruction to rotate the parabola into an x-axis parabola. The second **fill** (*volume fill* - **Fill**) is used to erase edge electrode points the z = 0 and x = 50 cut planes. This results in a half a parabolic grid with no grids in the cut planes.

Rotate_Fill	or: Rotate_Fill_Volume Format: rotate_fill(Angle_of_Revolution) { } Default Values: rotate_fill(360) { } Class: Fill Class When Legal: Base Nesting Level

Defines a *volume of revolution fill* using the currently active point type and potential. It raises the nesting level to Fill Nesting Level within its scope (*e.g. { }*). *Its scope must contain at least one* **Within** *or* **Notin** *instruction.* If no **Within** instruction is supplied, the point is assumed within, subject to rejection by **Notin** tests. *There is no limit to the number of* **Within**s *or* **Notin**s *that can appear inside the scope of a Fill Class instruction.*

Each **Within** or **Notin** instruction defines a separate *area of intersection with upper half of the xy-plane* (*z = 0 and y > 0*) through the use of tests (*e.g.* **circle**()). The coordinate system used is

the coordinate system's scope that **Rotate_Fill** instruction appears in. **Locate** ***instructions appearing within the scope of the*** **Rotate_Fill** ***do not change this test coordinate system.*** They merely change where the tests may intersect its xy plane.

This fill area is then rotated ***ccw*** **Angle_of_Revolution** degrees around the ***x-axis*** looking down the positive ***x-axis*** toward the ***origin*** (*same as rt angle in* **Locate**). All potential array points that fall within this volume of revolution will be changed to the currently active type and potential.

The **.GEM** file fragment below uses two rotate_fills to create the internals for a spherical ESA. ***It is important that you take the time to understand this instruction fragment if you are to successfully use*** **Rotate_Fill** ***properly.***

```
;fragment below makes inner workings of spherical ESA

pa_define(101,101,101,p,n)                  ;101, 101, 101 3D planar non-mirror

locate(50,10,10)                            ;locate point of rotation for ESA
    {
    e(1)                                    ;use one volt electrode points
        {
        rotate_fill(90)                     ;90 degree rotate fill
            {                               ;inner spherical surface
            within{circle(0,0,40) centered_box(0,0,70,200)}
            }
        }
    e(2)                                    ;use two volt electrode points
        {
        rotate_fill(90)                     ;90 degree rotate fill
            {                               ;outer spherical surface
            within{centered_box(0,0,70,140)}  ;90 degree solid cylinder
            notin{circle(0,0,60)}           ;with spherical inner surface
            }
        }

    }
```

Creates a 90 degree spherical ESA in a 3D array. The inner electrode is one volt with a spherical radius of 40 and a width of 70. The outer electrode is two volts with a spherical radius of 60 and a width of 70. This might serve as starting point for a real **.PA#** definition file for a 90 degree spherical ESA.

Sphere	or: Ellipsoid Format: sphere(xc, yc, zc, rx, ry, rz) Default Values: sphere(0,0,0,10,10,10) Class: Test Class When Legal: Within Nesting Level

Defines 3D sphere or ellipsoid. Parameters define the center and radii. ***Note: If ry is defaulted, ry will be set to rx. Likewise, if rz is defaulted, rz will be set to ry.*** Returns ***TRUE*** if point within its bounds.

sphere(15,20,30,45,20,45)

Defines and ellipsoid centered 15x, 20y, 30z with radii of 45rx, 20ry, 45rz.

Within	or: Format: within { } Default Values: NA Class: Within Class When Legal: Fill Nesting Level

Must be called from inside the Fill Nesting Level (*within a Fill Class instruction:* **Fill**). Holds one or more Test Class instructions within its scope that define a volume (*3D potential array*) or area (*2D potential array*). ***There is no limit to the number of Test Class instructions that can appear within the scope of a Within Class instruction.***

If a potential array point is contained inside the intersection volume (*or area*) of the tests in its scope a ***TRUE*** is returned. Thus, all the test class instructions (*e.g.* **circle()**) inside the scope of a Within Class instruction (*e.g.* **within{}**) must return ***TRUE*** for the Within Class instruction to return ***TRUE*** to the Fill Class instruction.

All that is required for a point to be considered within a fill's volume is that *at least one* **Within** instruction returns a ***TRUE***. This within designation will be revoked if *at least one* **Notin** instruction returns a ***TRUE***.

```
fill
   {                                    ;within right half of ellipsoid
   within{sphere(0,0,0,50,30,50) box(0,-30,50,30)}
   notin{sphere(0,0,0,45,25,45)}        ;notin inner ellipsoid
   }
```

The example above creates the right half of a ellipsoid shell ($x > 0$) centered 0x, 0y, 0z and outer shell of 50rx, 30ry, 50rz with an inner shell of 45rx, 25ry, 45rz.

Developing, Testing, and Using Geometry Files

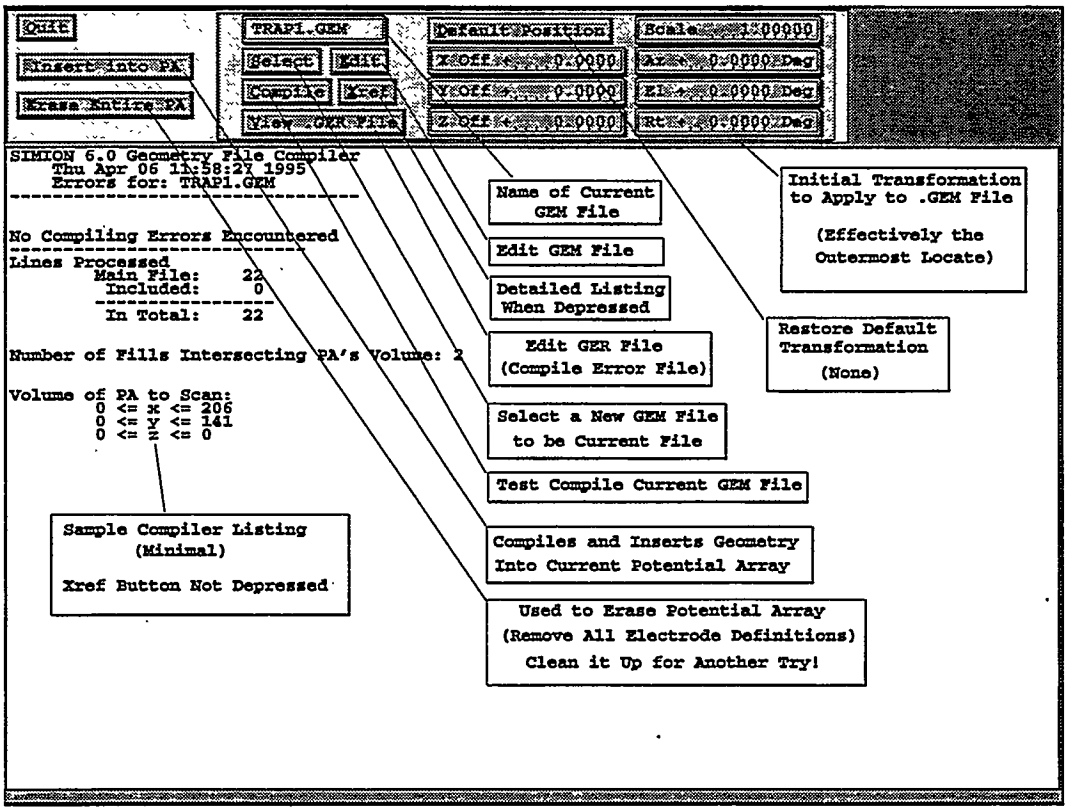
SIMION provides geometry file development tools within the **Modify** function. Geometry files can be utilized by either **Modify** or **New** to define electrode geometry.

Geometry Examples Provided With SIMION

There is a collection of geometry file examples in the **GEOMETRY** subdirectory (*below the* **\SIM6** *subdirectory*). This directory contains a collection of trivial and non-trivial examples. Examples include: ESAs, quads, traps, lenses, and ICR cells. The **NEW** function can be used directly on most of these **.GEM** files to quickly see what they create. It is intended that they should serve to get you started on your geometry file adventures. ***Be sure to scan the* README.DOC *file in the subdirectory for any directions and cautions.***

Accessing the Modify Geometry Development Tools

The geometry file development tools are accessed in **Modify** via the **GeomF** button (*on lower left edge of* **Modify** *screen*). If there is no currently active **.GEM** file, SIMION will ask you to select one via an automatic call to the GUI File Manager. Assuming you're trying this for the first time, switch to the **GEOMETRY** directory and select **TRAP1.GEM** (*click both mouse buttons when on its file name button*). The Geometry development screen shown below will appear:



An Introduction to the GEM Development Tools

The illustration above shows the Geometry Development Screen. It is used to edit, test compile and insert geometry definitions into the currently active potential array. Geometry development tools include a geometry compiler, Status Screen, .GEM file access buttons (*using the EDY editor by default*), positioning controls, and geometry insertion and erasure controls.

Running Another Editor From SIMION

The geometry development tools access the **EDY** editor by default for geometry file inspection and modification. If you prefer another editor, us the **DOS SET** command to link SIMION to your editor. The line below makes **EDLIN** the active editor (*or non-editor*):

SET GUI_EDITOR=EDLIN

The Geometry File Development Cycle

The geometry file development cycle involves the following steps:

1. Create a new geometry file.
2. Test compile and edit until there are no compiler errors.
3. Insert the geometry into the potential array and examine it.
4. If there are geometry errors, edit geometry file, erase potential array, and repeat from step 2.

Creating a New Geometry File

Start the geometry file creation process by using the **New** function to create a blank potential array to use with the geometry definitions (*you may want to remove all PAs from RAM first to reduce the clutter*).

Next, enter **Modify** and click on the **GeomF** button.

- If there is no currently active geometry file, the GUI File Manager will appear. Click on the **Edit** button to create a new file.
- If there is a currently active geometry file clicking the **GeomF** button will access the geometry file development screen. Click the **Select** button to access the GUI File Manager and then click the **Edit** button to create a new file.

You may now use the editor (*EDY by default*) to type in your geometry file. Be sure to use indention to define your nesting levels (*as in the examples above*). Indention of nested instructions makes them much easier to read and understand later. Be sure to use comments to improve geometry instruction readability.

The next step is to name and save your file. In EDY, naming is done by the **<ESC> N filename <ENTER>** key sequence. Now save the file by the **<ESC> S <ENTER>** key sequence. Finally, quit EDY (**<ESC> Q N**) and select the new **.GEM** file (*in the GUI File Manager*) by clicking both mouse buttons on its file name button.

Performing a Test Compile

To test compile a geometry file, click the **Compile** button. If the geometry file compiles properly (*along with all of its include files*) a compiler summary will appear as in the status screen above.

If the compiler finds an error it will beep and display a message to tell you what it didn't like. Errors are also written to the **.GER** file (*e.g.* **TEST.GER** *is error file for* **TEST.GEM**). ***Click the* View .GER File *button to access the* .GER *file.***

There are times when a more complete listing may help you to understand what is going on with the compiler. This is done by depressing the **Xref** button ***before*** clicking the **Compile** button. The compiler then generates a complete cross-reference listing to the Status Screen ***and* .GER *file*** (*even if there are no errors*). The cross-reference listing is useful in understanding how the compiler interpreted the various geometry instructions.

Editing the Current .GEM file

Normally the error message on the Status Screen is enough to point out your error. To edit the current **.GEM** file click the **Edit** button. The editor will be called (*EDY by default*), allowing you to the fix the error, re-save the file, quit the editor, and recompile.

Erasing the Potential Array

The **Erase Entire PA** button is provided to allow you to remove ***all*** electrode/pole definitions from the potential array before inserting geometry definitions into it. **When an array is erased, all points are converted to non-electrode (*pole*) points of zero volts (*Mags*).** Use this button between successive geometry insertion attempts to remove the clutter of the past.

Beware! This erases everything in the potential array - not just what was inserted by the geometry file.

Inserting Geometry Definitions Into a PA

The **Insert into PA** button is used to actually insert geometry definitions into the current potential array. When this button is clicked the geometry compiler compiles the selected **.GEM** file (*and all its include files*). If there are no errors the geometry definitions are then *added* to the potential array. Note: Your potential array is not erased first (*that is your responsibility*). ***Geometry definitions are added to any that may already be in the potential array***. A progress bar at the bottom of the screen is provided to assure you SIMION has not died.

Using the Initial Transformation

While your **.GEM** file will probably contain one or more **Locate** instructions, you may still want final control over just where these geometry definitions actually are placed in your potential array. SIMION provides panel objects that can be used to define this initial transformation. You may think of the initial transformation as the outermost **Locate** instruction (*the entire* **.GEM** *file is within its scope*). Normally (*by default or if the* **Default Position** *button is clicked*) this transformation does nothing (*unit matrix*).

As an example, let's say that that we want to double the potential array twice and then insert the geometry into this expanded array. We would exit **Modify**, **Double** the array twice, re-enter **Modify**, click the **GeomF** button, *erase* the potential array, set the *scale* to 4.0 (*to compensate for array doubling twice*), and insert the geometry into the potential array. The main advantage of doing things this way (*instead of simply doubling the array*) is that the resulting geometry is smoother (*none of the* **Double** *caused jags*).

Accessing Geometry Files Via the New Function

When you click on the **New** button (*on the Main Menu Screen*) or try to **Modify** an empty potential array (*automatically invoking the* **New** *function*), the potential array definition screen has a **Use Geometry File** button. If you click this button, the GUI File Manager will ask you to select a geometry file to be automatically inserted in the new potential array.

If you have a **PA_Define** instruction as the *first* instruction in the selected **.GEM** file, SIMION will use it (*in lieu of the* **New** *definition screen values*) to create the desired potential array.

Note: When inserting geometry files from the **New** Function, the initial transformation will *always be defaulted to no transformation* (*unit matrix*). ***Thus your*** .GEM ***file must contain all the transformations required to place the geometry properly in the target potential array***.

If you have a geometry compiler error, you must use the geometry file development tools in **Modify** (*via the* **GeomF** *button*) to find and fix the problem(*s*).