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.

A Brief Contextual Overview of SIMION

In Chapter 2 you were introduced to SIMION by an example of use. This brief overview is intended to create a vision of the structure of the constructs and features that combine to form SIMION. Hopefully, this overview will help you keep your bearings as you immerse yourself in the details of later chapters.

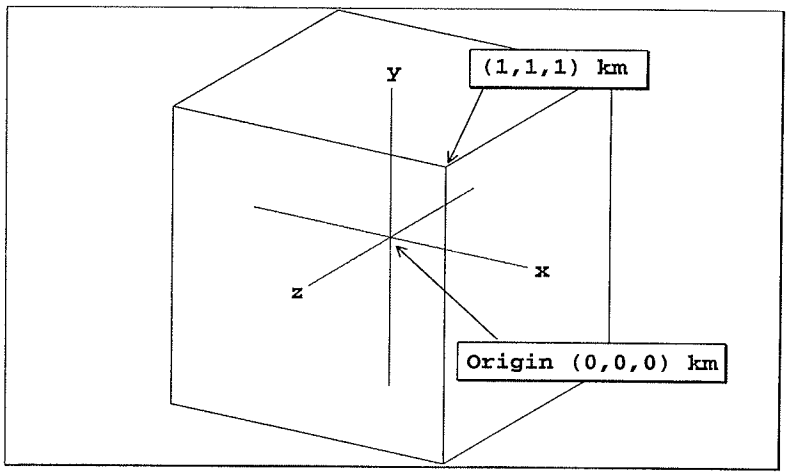

Figure 3-1 SIMION's 8 km^3 cubic universe

SIMION's Simulation Universe - a 8 km^3 Volume

SIMION's simulation universe is an imaginary 3D +/- 1 km cubic volume (*8 km^3 in volume*) with its origin (*x=0, y=0, z=0*) located in the center of the cube (*Figure 3-1 above*). The actual simulations are conducted within an ion optics workbench volume that resides within the simulation universe.

SIMION's Ion Optics Workbench

SIMION's ion optics workbench (*or just workbench*) is an imaginary 3D rectangular volume of space within the simulation universe in which the actual simulations are conducted. All features (*e.g. array instances*) must be contained within the workbench. Moreover, ions must be created within the workbench to be flown, and any ion's trajectory that crosses a workbench boundary is terminated. Thus the workbench defines the limiting volume for a simulation. You have the

option of positioning and sizing the workbench volume anywhere within SIMION's simulation universe. **This also implies that the maximum size allowed for the workbench volume is SIMION's simulation universe** (*8 km*^{3}). Workbench positions are ***always*** in mm relative to the origin of the simulation universe.

SIMION can save a workbench definition including all its array instance definitions (*discussed below*) 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 array instances are restored, all referenced potential arrays are loaded, and SIMION will ***optionally*** restore the potentials of all referenced **.PA** and **.PA0** files. SIMION can also automatically reload ion definition files (*.FLY or .ION*), data recording files (*.REC*), and kept ion trajectories that the user has elected to designate as auto-loading when the **.IOB** file was last saved.

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.

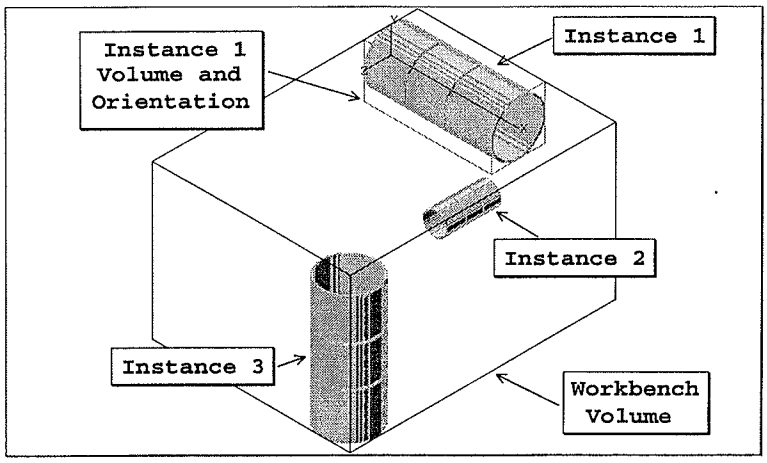

Figure 3-2 Projection of three array instances of the same potential array into the workbench volume

The Projection of Array Instances into the Workbench

An array instance is used to ***project*** (*locates, scales, and orients*) a ***single*** 3D ***image*** of a potential array into the workbench taking into account the array's symmetry and mirroring attributes. For example, a 2D cylindrical array is projected as a cylindrical 3D volume (*surface of revolution*). The array instance construct provides the coupling between potential arrays in RAM and the virtual reality of the workbench volume where ions are actually flown.

SIMION allows up to 200 array instances of potential arrays to be projected into a workbench volume. Figure 3-2 (*above*) shows ***three*** array instances of the same potential array (*an einzel lens*). This demonstrates that array instances can share the same potential array, because each array instance merely projects a 3D image of a potential array. As this example demonstrates, each array instance (*or image*) can be positioned, scaled and oriented independently. However, since all three array instances project images of the same potential array the electrode potentials of each array instance are identical, and the field gradients only differ by the relative scaling factors between the array instance definitions. It is important to remember that electrostatic and magnetic fields can only exist within the projected array instance images. ***This means that the workbench regions between array instances are normally assumed to be field free (an exception for electrostatic field interpolations is discussed in Chapter 7).***

The array instance construct is very powerful. It allows you to superimpose array instances of electrostatic and magnetic arrays to create volumes containing both electrostatic and magnetic fields. Moreover, array instances can be projected inside array instances to allow more detailed fields to be defined in specific regions. Each array instance has no knowledge of any other array instance (*all are blind*). This means that the proximity of other array instances have ***absolutely no impact*** on the ***fields*** within an array 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 array instance into the equivalent locations in another array instance (*Chapters 7 and 9*). This allows you to project the impact of one array instance into another because you can **Refine** the resulting array.

Array instances can automatically be created by SIMION. In the Chapter 2 SIMION demonstration, an array instance was automatically created when the View function was entered with the **test.pa0** potential array selected. SIMION set the scaling of the array image projected by the array instance definition to 1 mm/array grid unit and located the origin of the array at the origin of the simulation universe and oriented the array instance so that the x, y, and z axis of the array were aligned with the respective x, y, and z axis of the simulation universe. The workbench volume was then minimized so that this array instance of the **test.pa0** array ***filled*** the workbench volume.

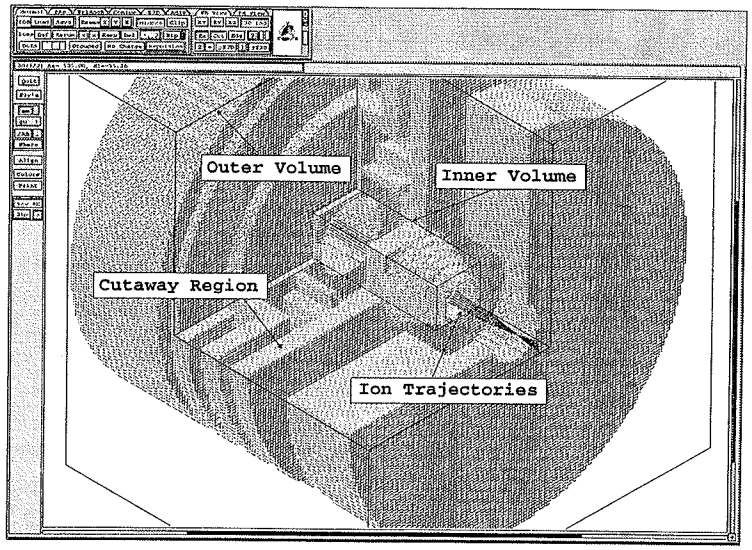

Figure 3-3 A cutaway clip of an outer 3D zoom volume to show an inner 3D zoom volume and its ion trajectories

Visualizing the Workbench with the View Function

SIMION's **View** function is used to create and visualize the workbench along with its array instances and ion trajectories. The View function is where you will spend most of your time. It has the most capability and features. The more you know about these features, the more likely you are to find using SIMION a rewarding experience (*Chapter 7*).

The Notion of 3D Zoom Volumes

Selective visualization is the name of the game, and SIMION bases all its visualizations on the construct of 3D zoom volumes and their surfaces. In much the same way that 2D zoom areas select and magnify sub-areas within a larger 2D view, 3D zoom volumes are used to select and magnify 3D volumes within the workbench volume. SIMION supports the concept of multiple nested 2D and 3D zooms. This means that once zoomed into a particular 3D zoom

volume, an inner volume can then be marked and zoomed into. The current series of zooms are also remembered (*2D and 3D*) to allow you the freedom of backing out a zoom level or two and then zooming back in at will.

Figure 3-3 (*above*) illustrates the notion of multiple nested 3D zoom volumes. It also illustrates the use of a View option to automatically cutaway portions of the current 3D zoom volume in order to see the next inner 3D zoom volume in the display chain. The point here is that View has visualization tools that allow you to quickly and easily zoom into what you want to see and many options to help you see it your way (*e.g. selective array instance drawing, cutaway clipping, edge only drawing, adjustable drawing quality, potential energy views, and more*). SIMION supports advanced features like 3D pointing and cutaway clipping by using 2D mouse motions (*via 3D isometric pointing*).

3D Front Surfaces Used for 2D Views

2D views are generally (*though not always*) based on the visible front surface of the current 3D zoom volume. Each 3D zoom volume has six boundary surfaces that can each selectively become the visible front surface depending on the ***2D*** direction the ***3D*** volume is viewed from (*via the Orientation Sphere object in View*). Features like potential energy surfaces and contouring generally use the fields on the current 3D zoom volume's front surface to generate their outputs. Details about all this and more are to be found in Chapter 7.

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 the Ions to Fly

SIMION defines each ion by the parameters that specify its initial condition: Mass, charge, starting position, kinetic energy, initial direction of motion, time of birth (*for delayed starts*), color, and etc. The ion's starting position and direction of motion can be defined either in the units (*mm*) and orientation of the current workbench volume or in terms of the units (*grid units*) and orientation of a selected array instance to tailor ion definitions to the problem at hand (*Chapter 8*).

Defining Ions in Groups of Similar Ions

Ions can either 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 in the demos*). ***You should use this ion definition method whenever possible.***

Defining Ions Individually

In some cases each ion is totally unrelated to any other ion. These ions are best defined by individual definition methods (*via Ion by Ion Definitions*). SIMION saves these individual ion definitions in **.ION** files (*ASCII file format*). The ASCII file format ***also*** allows you to create your own ion definitions outside SIMION (*via editor or other program*) and use them within

SIMION (*Appendix D - for file format*). ***This is the most flexible (non-dynamic) ion definition method.***

Defining Ions Dynamically

Moreover, you also have the option of defining ions dynamically via the user program feature discussed in Appendix I. This advanced and very powerful approach can be used to create random collections of ions that satisfy user defined probability distributions (*e.g. ion starting positions, energies, and/or angles of emission*).

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.

An Ion's Perception of the Workbench

Each ion flying through the workbench volume continually looks about to see where it is. The ion compares its location in workbench coordinates (*mm*) with the locations of all array instances to determine which electrostatic and/or magnetic array instance volumes, ***if any***, it is currently within.

If the ion determines that is currently within more than one electrostatic array instance or within more than one magnetic array instance it ***always resolves*** the conflict by selecting to use the fields of the electrostatic and/or magnetic array instance(*s*) with the highest priority (*e.g. highest array instance list number*). The ordering of array instances allows overlapping array instances of the same type (*e.g. electrostatic*) to be selectively visible to the ions in the desired order (*Chapter 7*).

Moreover, even when an ion is flying between array instances it looks forward and backward along its trajectory path line to see if the line intercepts both a leaving and entering electrostatic array instance. If it does, the intercept potentials are calculated and the corresponding linear gradient is applied along the ion's current trajectory line (*Chapter 7*).

The Interactive Nature of Ion Flying

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 array instance, or whatever*). Chances are that SIMION will cooperate.

Data Recording

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

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

User Programs – The Sky is the Limit

The most powerful ion flying 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, record user defined data records, 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.***

Potential Arrays and How to Create, Refine, and Use Them

All the constructs and capabilities discussed above are designed to effectively exploit the potential array. Electrostatic and magnetic potential arrays serve as the building blocks for SIMION's electrostatic and magnetic field simulations. Potential arrays define 2D areas and 3D volumes in terms of groups of equally spaced points (*square or cubic meshes*). Electrode/pole geometry is defined by designating selected groups of points as electrodes/poles of assigned potentials (*setting the boundary conditions*). The key to SIMION is knowing how to define potential arrays and their associated electrode/pole geometry to provide accurate ion trajectory simulations (*Chapters 4, 5, 6, and 7*).

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 (*discussed in the housekeeping section below*). Otherwise, you may be forced save the arrays, erase the ***entire*** PA memory, and reload the arrays 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*), geometry files (*Appendix J*), or by your own programs using SIMION's potential array file format (*Appendix D*). It is suggested that you take the time to ***really*** learn how to use **Modify** effectively.

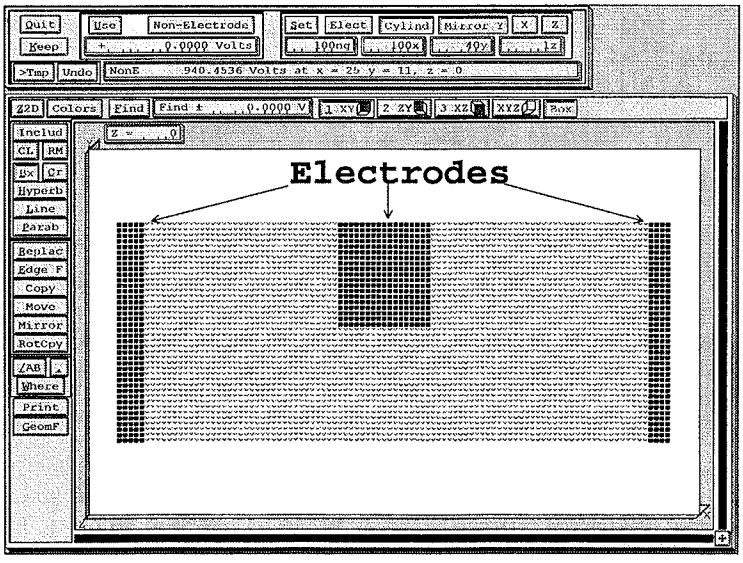

Figure 3-4 Defining electrode geometry via the Modify function

The Modify Function (Chapter 5)

SIMION's **Modify** function provides a very direct way to define array geometry via a simple point and click drawing interface (*Figure 3-4*). Features are provided to support quick creation of electrode shapes from rectangles, circles (*or ellipses*), multi-line boundaries, parabolas, and hyperbolas. Editing functions like selective marking, finding, moving, copying, and mirroring help speed the geometry definition process.

The **Modify** function is most appropriate for the quick definition of 2 dimensional potential arrays. Moreover, it has tools to support the definition of relatively simple 3 dimensional potential array geometry too.

Geometry Files (Appendix J)

Geometry files provide a 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 (either a new array or an existing array in **Modify***). 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 can be easily scaled so that a geometry file can be made to work with any size array. This means you can get more detailed electrode/pole surface definitions by inserting geometry file definitions into a larger array (*handy*).

It is important to remember that a small mistake with a geometry file can be quickly corrected and the geometry reconstructed automatically by SIMION. However, with **Modify**, a small mistake can sometimes take a significant amount of time to correct. If the geometry is complex (*2D or 3D*), geometry files are generally the best and most time efficient approach.

External Potential Array File Creation

Appendix D contains the format used for SIMION's potential arrays. In some cases it may be to your advantage to write a program (*or a subroutine*) that directly creates potential array files. Perhaps you know the field explicitly, and merely need to define it in a way that SIMION can use. External array creation may be the best approach.

Defining Fields with User Programs (Appendix I)

However, if the field can be defined by one or more analytical functions you may want to take advantage of the user programming features (*Appendix I*) to directly (*and precisely*) define these fields with the analytical functions with the aid of a ***dummy*** potential array.

The Two Types of Potential Arrays

There are two basic classes of potential arrays the Basic potential array and the Fast Adjust Definition array. Each has its value depending on the application. It is important that you recognize when and how to use these array types (*Chapters 2 and 6*).

The Basic Potential Array (.PA)

In the Basic potential array the actual electrode/pole potentials are formally defined (*e.g. fixed valued*) 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 ***only*** change the potential of a selected electrode/pole you must change the potentials of ***all*** of the

points that define the geometry of the electrode/pole with either **Modify** (*Chapter 5*) or by changing a geometry file (*Appendix J*), and then **Refine** the array (*Chapter 6*).

This Basic potential array is designated as such to SIMION by saving it with a **.PA** extension. The Basic potential arrays (***.PA***) are 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 basic potential arrays (**.PA**) would be for simple solid electrode beam stops.

The Fast Adjust Definition Array (.PA#)

In Fast Adjust Definition arrays, the geometry of up to ***30*** electrodes/poles with individually adjustable potentials can be defined. In this case the points defining the geometry of a particular adjustable electrode/pole are all of a designated potential (*electrode one points = 1.0 volt*). Thus the integer potential values of ***1 to 30*** are reserved for designating adjustable electrodes/poles. Electrode/pole points that do not have these reserved potentials are treated as proportionally adjustable potentials as in the **.PA** array above (*Chapter 6*).

Fast Adjust Definition arrays are designated as such to SIMION by saving them with a **.PA#** extension. When SIMION **Refines** a Fast Adjust Definition array (***.PA#***), it automatically creates and refines separate solution arrays for each fast adjustable electrode/pole as well as creating and refining a single separate array for the proportionally adjustable potential points too.

You should normally consider using **.PA#** arrays for your geometry definitions, because they offer the most flexibility and power. They even allow you to adjust electrode potentials dynamically via the user program function as the ions fly (*Appendix I*).

Refining Potential Arrays

Once the electrode/pole geometry has been defined for a potential array the next task is solving for the potentials of the points that are not electrodes or poles. In SIMION this process is called refining (*accomplished by various finite difference methods*). Once an array has been refined (*solved*) it can then be fast adjusted to the desired electrode potentials to obtain the potentials for the non-electrodes/non-pole points.

Chapter 6 contains an extensive discussion of the separate refining processes applied to Basic potential arrays (***.PA***) and Fast Adjust Definition arrays (***.PA#***) as well as the associated fast scaling or fast adjust options. ***If you change the geometry or the defined potentials of electrode/pole points in an array it must be refined again to obtain the correct field values for the points that are not electrodes or poles.***

Housekeeping Issues

The following material covers general housekeeping issues you should be aware of to make effective use of SIMION. ***Please take the time to read and understand the following material before proceeding.***

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 use virtual memory (*your disk drive – much slower*). If you get an out of memory error, read the material in Appendix B on how to increase your virtual memory allocation (*This is automatically done in 95, but must be explicitly done in NT*). ***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 (*a large common block of 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.

It is possible that the above heap defragmentation approach may not always work because the operating system doesn't always consolidate returned RAM properly (*thank Bill*). This will manifest itself in a refusal to allocate array space that you know it should be able to. The only recourse is to save your files, exit SIMION, and then restart.

Be Temperate in Your Array Sizing

The bottom line here is not to be too greedy. Keep the size of your arrays reasonable. Remember that a ***10 million point potential array requires 100 Mb of RAM*** (*or virtual memory*). 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 (*assuming you have changed them*) 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 potential array files they reference (*e.g.* ***.PA*** *and* ***.PA0***). 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 referenced potential array file (e.g.*** **.PA** ***and*** **.PA0***) must always be in the project directory for this to work properly.*

File Compatibility With Earlier SIMION Versions

All SIMION 6.0 file formats are upward compatible with SIMION 7.0. However, some file formats are not downward compatible from 7.0 to 6.0 (*e.g.* ***.ION***). See Appendix D for the details.

In general, ***no*** file formats from the earlier SIMION versions ***2.0 to 5.0*** are compatible with SIMION 7.0. SIMION will however convert potential arrays with either **.PA** or **.PA#** extensions to 7.0 format.