ELSEVIER

International Journal of Mass Spectrometry 200 (2000) 3–25

International Journal of Mass Spectrometry

SIMION for the personal computer in reflection

David A. Dahl*

Idaho National Engineering and Environmental Laboratory, Idaho Falls, ID 83415, USA

Received 24 April 2000; accepted 2 August 2000

Abstract

This article is a reflective overview of the origins, history, and capabilities of the ion optics simulation program SIMION for the PC (versions 2.0–7.0) from the author's perspective. It provides insight into the rationale and events that contributed to the direction of the evolution and current capabilities of the program. The capabilities of version 7.0 are presented along with tests of its computational accuracy. Future developmental areas are discussed. (Int J Mass Spectrom 200 (2000) 3–25) © 2000 Elsevier Science B.V.

Keywords: Ion optics

1. Introduction

The various versions of the ion optics simulation program SIMION for the PC (versions 2.0–7.0) have gained wide usage within the mass spectrometry community since first introduced in 1986. Although the PC SIMION programs may have attained significant visibility themselves, few descriptions concerning the PC SIMION effort appear in the open literature [1]. This article serves as a reflective overview of PC SIMION's origins, developmental history, philosophy, present capabilities, and future directions from the author's perspective.

2. Origins of SIMION for the PC

The SIMION program originated in Australia in 1973. Don McGilvery created the original SIMION program while working on an undergraduate research project for Professor James Morrison [2]. The project's goal was to design a double quadrupole mass spectrometer with a common ionization volume for isotope ratio measurements. This instrument was intended to be a simple, inexpensive replacement for dedicated, multidetector, sector mass spectrometers. Unfortunately, the instrument did not perform as expected, and McGilvery created the first version of SIMION in order to simulate the ion motions in the instrument and gain insight into the problems involved. Detailed study of the source assembly optics with the various versions of the program he developed, including one with three-dimensional (3D) simulation capabilities, uncovered intractable problems with the common ionization volume.

In contrast, his SIMION effort turned out to be a great success. It played an integral role in his Ph.D. project to build a mass spectrometer for the study of gas phase molecular ions as well as in many other instrument construction projects underway within the

* Corresponding author. E-mail: dhl@inel.gov

PII S1387-3806(00)00305-5

Department of Physical Chemistry at Latrobe University during the 1970s. SIMION started out as a suite of seven FORTRAN language, DEC PDP 11-20 programs that McGilvery gradually combined over the years. New features, such as rf capabilities, were added as the need arose. McGilvery made use of SIMION's rf capabilities in collaborations with Rick Yost to study ion transmission after collision induced fragmentation in quadrupoles. These simulations helped provide the understanding needed for the triple quadrupole instrument to gain rapid acceptance.

Moreover, McGilvery freely shared the SIMION programs with the mass spectrometry community, which greatly extended its general impact. During McGilvery's postdoctoral studies with Professor McLafferty at Cornell University, during 1977–1980, SIMION was there too. Peter Todd, one of McLafferty's students, took a copy of SIMION with him to Oak Ridge National Laboratory (ORNL) in 1980 [3]. The program proved quite useful there and Hank McCowan of ORNL added features such as ion trajectory mirroring and quadratic field near-axis cylindrical symmetry ion trajectory corrections. Oak Ridge also freely shared its version of SIMION with interested researchers within the mass spectrometry community.

In 1983 Jim Delmore of the Idaho National Engineering and Environmental Laboratory (INEEL) obtained a copy of SIMION during a visit to Oak Ridge. Delmore used SIMION to greatly improve his intuition and understanding of ion trajectories within various ion source assemblies in the instruments he used. This naturally led to design improvements and the need for higher resolution simulations. During the course of this activity, Delmore had SIMION ported from its native DEC PDP environment onto a DEC VAX computer to be able to work with larger potential arrays on a faster machine. Refining (solving for potentials) on the PDP-11 without a numerical coprocessor took three to four hours for a 4000 point array. The VAX could refine a 20 000 point array in a few minutes. Consistent with McGilvery's established tradition, Delmore also shared the SIMION VAX version with other researchers.

However, access to enough CPU time and physical (nonvirtual) memory on the VAX was often difficult to obtain, causing Delmore to consider porting SIMION to an IBM AT personal computer in 1985.

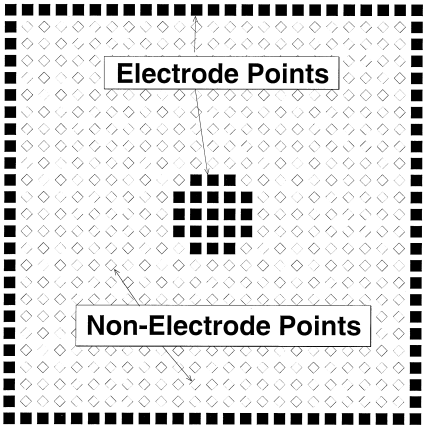

Fig. 1. Potential array. Electrode points are represented by filled squares and nonelectrode points are represented by shaded diamonds.

3. Birth of SIMION for the PC

It was at this point, in the summer of 1985, that I began working with Jim Delmore at the INEEL. My first task was getting SIMION to run on an IBM AT. Unfortunately, the IBM AT was a huge step down in speed from the VAX. However, an IBM personal computer was relatively inexpensive to buy, its CPU time was free (once you bought the computer), it could reside on a desk in the lab, and it offered the potential for a more interactive user/program environment. The objective was to create a version of SIMION that was relatively easy to use, robust, versatile, reasonably accurate, and acceptably fast on an IBM personal computer. The first challenge was to make SIMION run fast enough on the AT platform to be useable.

SIMION uses the potential array approach to estimate the electrostatic fields created by the defined electrode geometry. A potential array contains a collection of potentials arranged in a square [two-dimensional (2D)] mesh of points. The volume represented by a 2D potential array is defined by the array's symmetry assumptions (either planar or cylindrical). Selected

array points can be designated as fixed potentials to define electrode boundaries (geometry) as shown in Fig. 1. The point density of the array, relative to the electrode boundaries being defined, limits the simulation accuracy (larger arrays normally simulate fields better). The potentials of points outside of the electrodes are determined by solving the boundary value problem's Laplace equation via finite difference methods. In SIMION, this process is called refining the array. Once an array is refined, ions can be flown through its volume.

Applying finite difference methods to a boundary value problem solution (in the simplest form) involves estimating the value of each unknown point by the average of its nearest four (2D) or six (3D) neighboring points. Successive scans, or iterations, through the array of points gradually converges each point's potential to a stable solution value. The rate of convergence to a solution can be enhanced by the use of an over-relaxation technique [4] that multiplies a point's change in potential from one iteration to the next by a factor between 1 (no over-relaxation) and 2 (unstable over-relaxation). Most boundary value problems converge in the fewest number of iterations with an over-relaxation factor of approximately 1.9 (or 0.9 as used by SIMION). Finite difference methods use computer memory efficiently (e.g. one double precision word per array point) and are computationally robust. However, the time to refine an array by finite difference methods is generally proportional to n^2 where n is the number of points in the array.

3.1. Self-adjusting over-relaxation

The refining process is computationally intensive, because it involves large numbers of floating point additions and divisions (or multiplications). Speeding up the refining process was the first order of business. The version of SIMION that Delmore obtained from ORNL had a refine algorithm with lots of decision statements in the inner loops. This approach minimized the lines of code at the expense of slower refine times. The refining algorithms were unrolled into sequentially organized algorithms with no decisions in the inner loops. The code was considerably longer, but it ran dramatically faster. A study of the over-relaxation process was conducted with one dimensional models. Although over-relaxation values of around 0.9 gave the fastest convergence, the quality of the solution was often unsatisfactory, because of the solution kinks (local roughness) that high values of over-relaxation often introduced. Further study revealed that these solution kinks could be smoothed by beginning the refine with a low value (0.4) for over-relaxation that was gradually increased to 0.9 in the middle iterations of the refine and then gradually reduced back to 0.4 as the solution neared the convergence criteria. A self-adjusting over-relaxation algorithm was developed that made use of an exponential weighted historical memory factor approach that automatically varied over-relaxation in the above manner to smooth out the solution kinks while minimizing the refine iterations required. These algorithm modifications allowed an IBM AT, with a floating point coprocessor, to refine 1600 point arrays in approximately 100 s. Although this may seem dead slow by current standards it was sufficiently fast to justify a PC version of SIMION.

3.2. Fixed distance step integration

Trajectory calculations were the next order of business. The version of SIMION obtained from ORNL made use of trapezoidal rule integration with fixed time steps set by the user. In practice, the user decreased the time step until the ion trajectories did not change significantly from run to run. Although this is a very straightforward approach, applications where ions are accelerated from fractions of an eV to keV (e.g. ion guns) were a problem. Very short time steps were required for stable ion trajectories because of the high ion kinetic energies within the gun. However, these short time steps wasted huge amounts of CPU time in those regions where the ion's kinetic energy was very low (e.g. at the source). Algorithms were developed to automatically vary the time step to obtain fixed distance step integration. The user now set the number of integration steps per potential array grid unit. The default value of 10 steps per grid unit

was found to provide good ion trajectory solution convergence while minimizing CPU time.

3.3. Interactive graphics interface

The third task was to add an interactive graphics interface to PC SIMION to improve its ability to interact effectively with the user. The IBM Professional Graphics System CGI interface software was used because it followed the international integer graphics standard and provided device driver support for most of the video monitors and printers available for IBM computers. The new graphics interface only supported two of the program's functions: The first was viewing and printing trajectory images, and the second was a new function called "Modify" that allowed the user to interactively define and modify electrode geometry with the mouse. Although the graphics interface was peripheral to many of the program's functions it provided the user with an easy way to define electrode geometry and view/print ion trajectories (a big step forward).

During its development, I used PC SIMION to analyze and design the ion gun used in our SF_6 auto-neutralizing beam experiments as well as for other design simulations. These served to point out where polishing and additional features (e.g. defined magnetic fields) were needed. It is my opinion that the quality of scientific computational tools directly benefits when the creator remains a primary user, because problems are easier to see when you are being victimized by them.

The first PC version of SIMION was finished in early 1986, and Jim Delmore was keen on making it available to others. A manual was written and the program was named SIMION AT version 2.0. The name reflected its PC platform and the fact that it was actually my second iteration on the effort. Delmore wanted a PC SIMION poster presented at the 1986 34th ASMS Conference in Cincinnati. The poster was prepared with great reluctance, because of my conviction that there must be many equivalent or superior programs floating about within the community. Delmore said no, and history proved him right! The poster was mobbed, and we mailed upwards of 200 copies of version 2.0 during the next year. We gave the copies away in the established SIMION tradition, and it was very gratifying to see version 2.0's level of acceptance as well as have the opportunity to interact with the growing PC SIMION user community and hear their suggestions.

4. FORTRAN versions of PC SIMION

The interest in version 2.0 convinced me that an effort to further expand and improve the capabilities of PC SIMION was well justified (success had chased me down and run me over). Over the next four years, thanks to part-time support from the Department of Energy's (DOE) Basic Energy Sciences Chemistry Division (BES Chemistry), versions 3, 4, and 5 were released. These versions of PC SIMION retained the FORTRAN language of McGilvery's original SIMION, but little of its original code remained. The following briefly highlights the significant new features in each version.

4.1. Versions 3.0/3.1

Versions 3.0/3.1, released in 1987, had new algorithms to address array refining and ion trajectory calculation issues. At this point, most of the simulation time was spent changing the potentials of electrodes and then waiting for the array to re-refine before ions could be flown. The additive solution property of the Laplace equation provided the answer.

4.1.1. Fast adjust potential arrays

If a separate potential array were created, refined, and saved for the points defining each individually adjustable electrode (with all the other electrode points in the array set to zero potential), the array solutions for individual electrodes could then be combined by the appropriate scaling and summation to create the resulting composite field array. Algorithms were developed to automatically create and refine the individual adjustable electrode arrays as well as to adjust the potentials of the composite field array. The result was PC SIMION's fast adjust array

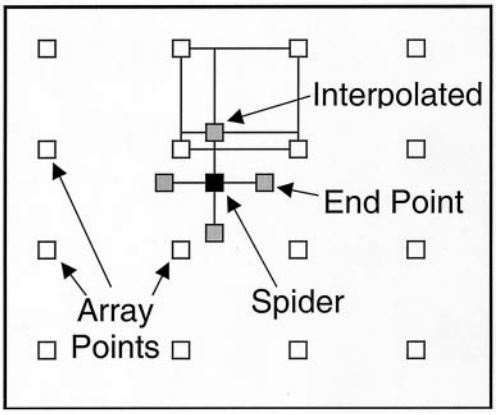

Fig. 2. Four point spider uses linear interpolation to estimate the potential of each of its four end points.

capability, and it provided a dramatic speedup. Electrode potential changes that previously required tens of minutes to re-refine now took only a few seconds to read, scale, and combine from the hard drive, once one had endured the refine time required for creating the individual electrode solution arrays.

4.1.2. Fourth order Runge-Kutta integration

Although the variable time step integration algorithm used in version 2.0 had resulted in significant time savings, ion trajectory accuracy issues remained that required a complete rethinking of the algorithms used for calculating ion trajectories. It was clear that a higher order numerical integration method was needed. A fourth order Runge-Kutta method [5] was adopted because its higher accuracy better matched the accuracy of refining and it was compatible with the use of variable time steps.

Various methods for more accurately deriving the potential gradients from a potential array were evaluated including polynomial approaches. The adopted approach made use of a 16 point (2D) copy of the array region that surrounds the ion (see Fig. 2). A four point spider of ± 0.5 grid unit extends from the ion's current location. The potential for each of the four endpoints of the spider is determined by linear interpolation using the four array points that surround it. The potential differences between horizontal and vertical spider endpoints are used to estimate the x and y potential gradients and the average of the four spider endpoint potentials estimates the potential at the ion's location. This approach proved a good match with the accuracy and order of the refined arrays, as well as being very robust even near electrodes, ideal grid discontinuities, and array boundaries (via linearly extrapolated grid point potential estimates).

Simplified (and faster) algorithms for automatically adjusting the variable integration time steps based on desired ion step length (d), current velocity (v), and acceleration (a) were adopted: (1) if the ion's acceleration and velocity are both zero: kill the ion (splat); (2) if the ion's acceleration is zero: $t_{v\neq 0} = d/v$; (3) if the ion's velocity is zero: $t_{a\neq 0} = \mathrm{sqrt}(2d/a)$; (4) if the ion's acceleration and velocity are both nonzero $t_{\mathrm{step}} = t_{v\neq 0}t_{a\neq 0}/(t_{v\neq 0} + t_{a\neq 0})$. It was found that a default value of one grid unit for the integration distance step (d) provided a good compromise between accuracy and speed with the Runge-Kutta method.

However, conservation of energy remained a problem at velocity reversals, ideal grid discontinuities, electrode boundaries, and array boundaries. Version 3.0 added a time step correction based a quantity called stop length:

$$S_{\mathrm{length}} = v^2/(2a)$$

Although stop length seldom indicates the ion's actual stopping distance (e.g. magnetic accelerations) it is a very good indicator of the rate of trajectory curvature. The time step computed above is reduced by up to a factor of 10 by stop length:

$$S_{\mathrm{length}} > 10, \qquad t_{\mathrm{step}} = t_{\mathrm{step}}$$

$$1 < S_{\mathrm{length}} < 10, \qquad t_{\mathrm{step}} = 0.1t_{\mathrm{step}}S_{\mathrm{length}}$$

$$S_{\mathrm{length}} < 1, \qquad t_{\mathrm{step}} = 0.1t_{\mathrm{step}}$$

Although the stop length approach improved conservation of energy at velocity reversals, it did not effectively address the issues created by ideal grid discontinuities, electrode boundaries, and array boundaries.

4.1.3. Binary boundary approach paradigm

The numerical integration process needed to automatically detect these problem events and use a general strategy to compensate for them (e.g. a white cane for the blind man). The binary boundary approach paradigm was found to be an excellent compensation strategy. Programs, much like many of us, can generally detect an event quite easily after it has happened (e.g. stepping off a cliff). However, unlike humans, a program can always leap back from the abyss. When a binary boundary approach is used, the program leaps back from a travesty (hitting an electrode), halves the allowed distance step and tries again. Assuming the binary boundary approach process is persistent, one can approach the boundary event (e.g. a velocity reversal) arbitrarily close without actually crossing it. However, the adoption of a minimum allowed distance step (e.g. ten thousandths of a grid unit) will ultimately force the event to occur while minimizing the introduction of energy conservation errors.

4.1.4. Coefficient of variation of acceleration controls

Although events such as hitting an electrode, entering or leaving a potential array, and velocity reversals were straightforward to detect, the detection of strong field curvatures and discontinuities (e.g. ideal grids) required calculating the coefficient of variation of acceleration (Cv) in the x, y, and z directions using the four Runge-Kutta acceleration terms. When the Cv term in any one direction exceeds a trigger value set by the user adjustable trajectory quality parameter, the binary boundary approach is invoked until the greatest Cv value reduces below the trigger level or a minimum allowed distance step forces the ion past the event threshold. Upon passing through a binary boundary approached event (e.g. an ideal grid discontinuity), the distance step is subsequently doubled each iteration until the nominal distance step is restored (e.g. one array grid unit) or another binary boundary event is detected.

PC SIMION versions 3.1–7.0 have successfully employed these strategies to obtain accurate ion trajectories with a minimum of integration time steps. Fig. 3 shows an ion's trajectory in a linear reflection field with an ideal grid creating a central electrostatic field discontinuity. The potential energy of the ion is conserved to within one part in 10^7 after 14 cycles of reflection using the default settings in SIMION 3D version 7.0.

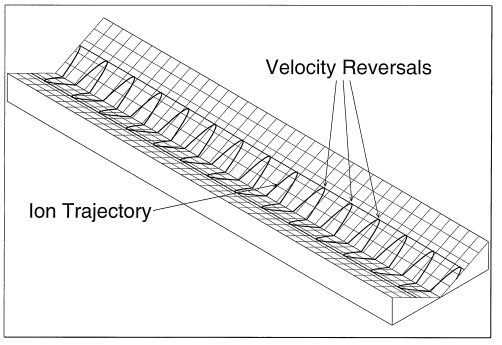

Fig. 3. Conservation of energy test in a linear reflection field made up of edge electrodes and a center ideal grid. Ion trajectory velocity reversal peak heights remain constant if energy is conserved.

4.2. Version 4.0

Although versions 2.0–3.1 proved to be quite useful, they also whetted the appetite for more capabilities. The most desirable capability would be full 3D asymmetrical simulations. However, a moderately useful $100 \times 100 \times 100$ point 3D array would require 10 MB of RAM for array storage. Unfortunately, the PCs available in 1988 were PC, AT or PS/2 class machines with 640 KB memory limitations. Thus capability enhancements in version 4.0 were limited to features that would still allow it to run in 640 KB (with extensive overlays).

4.2.1. User programming

Version 4.0 introduced the significant features of user programming and enhanced ion trajectory visualizations. User programming significantly increased the power and versatility of PC SIMION by incorporating a single pass pseudocompiler that could read user created files of programming instructions written in a HP calculator style reverse polish notation (RPN) language. These user programming instructions were

automatically incorporated in the ion trajectory calculations. PC SIMION became user extensible, allowing it to be used for applications well beyond those readily apparent to the author.

There were four classes of user program files that could be used in combination during ion trajectory simulations. The first user program type was a .FAC extension file that was used to multiply the electrostatic fields and potentials by a user defined factor. Because the user had access to parameters such as the ion's time of flight, these types of user programs could be used to create simple rf fields in much the same manner as the sinusoidal rf feature in McGilvery's original SIMION.

The second user program type was the .ELE extension file that allowed the user to adjust the potentials of fast adjust electrodes as the ions flew. SIMION dynamically fast adjusted the 16 point copy of the array region around the ion using potentials passed back by the .ELE user program. Memory swapping algorithms were incorporated to automatically load the appropriate regions of the needed potential array solution file images from disk into RAM to speedup the process as much as possible. The .ELE user programs added real power to PC SIMION, because fast adjustable electrode potentials could be varied independently from within a user program. This allowed PC SIMION to be used in time-of-flight (TOF) and simplified Fourier transform mass spectrometry (FTMS) simulations.

The third user program type was the .AR? extension file that allowed the user to explicitly define the electrostatic and/or magnetic fields in up to ten user defined regions of the potential array. The .AR? user programs allowed arbitrary analytical expressions to be used to define electrostatic and magnetic fields in ion trajectory simulations.

The fourth user program type was the .XX? extension file that allowed the user to control the ion's position, velocity, and fate within up to ten user defined regions of the potential array. These user program types were most useful for killing ions, forcing neutralization or fragmentation, and other similar stunts.

When combined, these user programs served to allow PC SIMION to attack whole new classes of ion simulation problems (e.g. TOFs, bunchers, quadrupoles, ion traps, and simple FTMS simulations). Also, user programs allowed the user to inject much more influence and imagination into the ion trajectory simulations, providing the appropriate levels of knowledge and ambition were applied.

4.2.2. Enhanced trajectory visualizations

The ability to simulate a broader range of problems added new challenges to visualizing the results. Both isometric 3D and *zy* views were added to the traditional *xy* trajectory view. This allowed complex 3D ion motions (e.g. ion motions in a quadruple) to be visualized, significantly adding to PC SIMION's capacity for improving insight and understanding.

However, it is my belief that the potential energy surface display was the most significant visualization feature added to version 4.0. Seeing an ion's trajectory is one thing. Understanding it is quite another. This gulf between ion trajectory visualization and understanding has served to give ion optics an aura of mystery for many (myself included). Contours of the potential field can be helpful, but contour maps are not the natural way we look at the world, and thus often prove more confusing than enlightening. Ion motions in electrostatic fields behave much the same way (though not identically) as golf ball motions on sloping surfaces. In the early days of vacuum tube design, rubber sheets were stretched over electrode shapes set to heights that represented their potentials. If the slopes were relatively mild, the trajectories of the balls rolled on the rubber sheet physical model could simulate ion/electron trajectories fairly accurately. The concept of the potential energy surface is really the reverse transformation, because the calculated ion trajectories are projected on a 3D surface that represents the rubber sheet model of the potential array.

Suddenly the "why" of the ion's trajectory becomes apparent (see Fig. 4). The upper illustration shows a 2D (*xy*) view of ion trajectories through an einzel lens. Note that the contours are not very helpful in understanding why the decel/accel einzel lens is focusing the ions. However, the potential energy

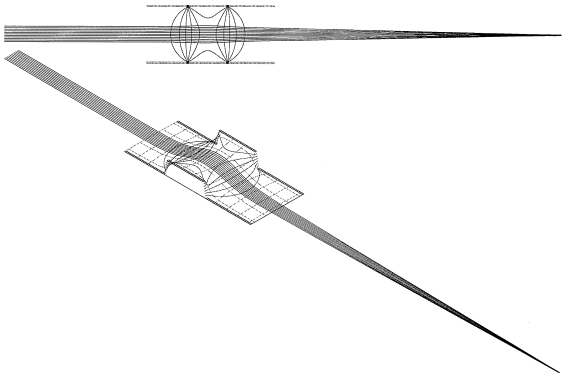

Fig. 4. Comparison of a 2D trajectory view (top) with a potential energy surface view (bottom).

surface view (below) shows that the center electrode creates a saddle retarding field. Ions entering the einzel lens decelerate in a weak diverging field that gradually becomes a stronger converging field as they approach their lowest velocity in the center of the lens. As the ions continue, they are accelerated in a strongly converging field that gradually changes to a weak diverging field as they exit the lens. The net effect is a converging focus, because the ions have spent the most time in a converging field. It is now more apparent why the ion trajectories have focused, because the trajectories have been displayed in a framework more consistent with how our minds see and interpret other roughly equivalent physical phenomena. However, the real benefit of potential energy surfaces is that the gain in intuitive understanding helps the user more knowledgeably predict the effects of changes in potentials or electrode geometry. For many of us, potential energy surface displays have removed some of the mystery from ion optics.

4.3. Version 5.0

The advent of the Intel 386 processor provided IBM compatible PCs with true 32 bit addressing capabilities. It was now possible, in principle, to support million point arrays assuming that the computer had 16MB of RAM (then very expensive) and a 32 bit DOS extender to allow program access to this memory. Version 5.0, created in 1989, was the first tentative step toward a 3D asymmetrical version of PC SIMION. It was a FORTRAN based program that supported 2D arrays of up to one million points.

Unfortunately, it also served to illustrate emphatically that the refine times of classical finite difference algorithms are proportional to the number of array points squared (the n^2 limitation). Although this version was never formally released, it did have a number of heroic volunteer users who were desperate for larger 2D arrays. Refine times for million point arrays often were measured in terms of days to weeks. Clearly other refining approaches were needed to make a 3D asymmetrical version viable on anything short of a supercomputer.

5. C versions of PC SIMION

My personal vision for PC SIMION's future revolved around creating a highly interactive program that could project 3D images (instances) of several 2D and/or 3D electrostatic and magnetic potential arrays into an arbitrary 3D volume allowing, in principle, the simulation of entire instruments. Version 5.0 was clearly a misstep toward the goal of an asymmetrical 3D PC SIMION, but it did serve to point out additional limitations beyond the impact of n^2 potential array refine times for very large potential arrays. Although versions 2.0–5.0 had functioned acceptably with a CGS integer graphics standard, accurately displaying ion trajectories and potential arrays within volumes with scales that might vary from microns to kilometers clearly would require a floating point vector graphics capability. Additionally, FORTRAN by its nature did not provide the flexibility (recursion), constructs (pointers) or the dynamic memory allocation capabilities required to support interactive user interfaces. The C programming language, in contrast, was much better suited to the diverse computational and interface requirements that the envisioned SIMION 3D would require.

5.1. PC SIMION's graphics interface development

Fortuitously, in early 1990, our group needed to create a unique data system to support a new pulsed

extraction quadrupole secondary ion mass spectrometry (SIMS) instrument [6]. This provided the opportunity for me to concentrate on a different but potentially related problem of user interfaces while allowing thoughts on the n^2 refining problem to slowly percolate. After looking at how the then available 16 bit Windows GUI (graphics user interface) might support our data system needs, it became apparent that creating a more optimal GUI for the SIMS data system that also could be used in future versions of PC SIMION would be highly desirable.

5.1.1. Floating point vector graphics

The subsequently developed GUI used a floating point vector based graphics GDI (graphics development interface) as opposed to Windows integer pixel based GDI. An integer pixel graphics image is defined in terms of pixels (dots) that are incrementally separated. A floating point vector graphics image is defined in terms of vectors (line segments). Both approaches have their merits. Integer pixel graphics is naturally more oriented to the organization of video displays. Floating point vector graphics is more oriented to the high resolution capabilities of graphics devices such as flatbed plotters and laser printers using languages such as PostScript, HPGL, HPGL/2, and PCL5 to process floating point vectors at the device level for high accuracy hard copy. A floating point vector strategy was adopted, because images defined by floating point vectors are scalable without loss of accuracy and can be easily converted into the pixel format of video displays.

The new GUI's floating point vector GDI used a reduced instruction set configuration (RISC) to speed development and improve portability. Drawing commands were limited to a small set of relatively high level calls that could, in principle, support video accelerators (e.g. dot, line, rectangle, translucent rectangle, and label). Unlike Windows, commands such as color and drawing mode are consistently applied to each drawing command.

5.1.2. Graphics device driver issues

All graphical environments must face the issue of device drivers. In the GUI's case, there was the need for video display and printer support. The IBM Professional Graphics Toolkit CGI package used in the FORTRAN versions of PC SIMION made use of a specific device driver for each video card and printer. This approach requires lots of device drivers and development time (often by the peripheral manufacturers themselves). What was needed was a way to create more universal video and printer device drivers.

It turned out that many video cards used similar addressing strategies based on the selected display resolution. The trick was to determine the precise addressing strategy required for each resolution supported by a particular video board. Fortunately, VESA, the Video Electronics Standards Association had recognized this problem and promoted the use of a VESA bios standard among its membership. The VESA bios allowed programs to interrogate a video card about its available resolutions and addressing strategies. This allowed the development of a single video device driver for the GUI that automatically recognized and supported the higher resolutions of video cards having the VESA bios extensions.

The issue with printer output was more complex, because there were a huge number of printers, each with its own native language and/or special software features. After reflecting on the problem I decided to write generalized printer language drivers instead of printer device drivers. Individual printer language driver support was provided for PostScript, HPGL, HPGL/2, and PCL5. These generalized printer language drivers could support many plotters and laser printers. Moreover, these drivers output floating point numbers directly to the printer or plotter for maximum hard copy quality. Unfortunately, this solution excluded inexpensive raster printers to the chagrin of many PC SIMION 6.0 users (an issue rectified in SIMION 7.0).

5.1.3. Graphic object oriented GUI

The adopted GUI architecture uses the notion of layers of graphical objects. Graphical objects are parent–child related based on their layering (from top to bottom). An object's parent is the first object lying immediately below it that entirely contains it. The

screen object acts as the ancestor for all other graphical objects. When an object is redrawn all of its descendants are automatically drawn in layered order from back to front. Likewise, when an object is deleted, all of its descendant objects are also deleted, and its parent object and its remaining descendents are subsequently redrawn.

All active graphical objects are maintained as data structures (holding values and pointers to values, strings, and functions) in a double linked heap list to facilitate their dynamic object creation and deletion. Unlike Windows, which uses a messaging paradigm, the GUI makes use of a recursive input poling and object commanding paradigm. Communication with objects (e.g. a draw yourself directive) is very efficient. To draw a graphical object and its descendants, a pointer to its structure is passed to the GUI's draw function. The draw function then gets a pointer from the object's structure to the object's actual drawing function and calls this function passing it the pointer to the object's structure. After the object's redraw is complete, the draw function proceeds to find and redraw each of the object's descendants in layer order using the double linked heap list. In contrast, a message paradigm as used in Windows, sends messages to each child window in the appropriate order and waits for the message to be processed, redrawing to occur, and the message conformation before proceeding. Although the message paradigm is very flexible, it often leads to a form of communication chaos when a large number of child windows (objects) are involved (a bedlam of messaging). The GUI takes a much more regimented and efficient approach by being able to identify and call any selected object function by directly using the appropriate function pointer stored in the object's structure.

This graphical object oriented approach has proven efficient and powerful. Object structures contain a collection of function pointers to selectively call when entering and leaving the object, to draw and position the object's cursor, to draw the object itself, and to access the actions that the object supports, to name a few. Other pointers in the structure are used to reference the object's specific and general help screens. The object's structure also holds values that define its size, location, color, and other features. Thus when the Print button in the View function is clicked in PC SIMION 6.0 or 7.0, the GUI's general print function is passed a pointer to the graphical object that is actually displaying the current ion trajectory view inside the window, and the print function automatically creates the additional objects that support printer selection, output controls, user annotations, and the actual printing process. Incidentally, printing makes use of a dual output port approach in which the object is told to redraw itself and its vector drawing commands are automatically sent to both the designated printer and the video display (to indicate printing progress).

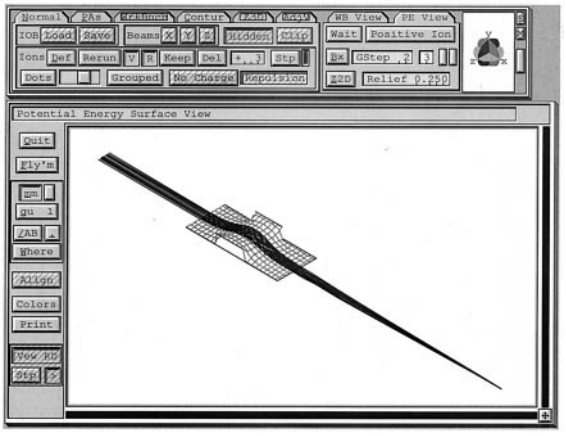

Fig. 5. PC SIMION 7.0 View window with potential energy surface of an Einzel lens.

5.1.4. Application to data systems

Over the years, many GUI based data systems have been developed for our group's quadrupole and ion trap SIMS instruments. They have proven to be robust, flexible, and easy to use. The flexible graphical object nature of the GUI has allowed the creation of specialized graphic objects such as panel objects (for numerical entry) and single button window objects (for simultaneous *xy* direction window scrolling) that enhanced the interactive nature of the SIMS instrument data systems (Fig. 5). Moreover, the use of graphical object structures containing function pointers supports dynamic object inheritance capabilities.

Thus the spherical view control object (shown in Fig. 5) is first created as a simple cover surface object that has its function pointers subsequently changed to reference functions tailored to support the required display and controls needed for spherical view control and display.

5.2. Version 6.0

By the fall of 1991 the effort to develop the GUI and the first pulsed extraction SIMS data system had largely come to closure, and my focus returned to PC SIMION and its n^2 refining problem. Version 3.0 era experience had demonstrated that when an array was refined, doubled in size, and refined again successively, the total time spent in the array refining process was proportional to the number of points in the final array's size. This strategy became the recommended approach in PC SIMION manuals to avoid the penalties of n^2 refine times. What was needed was a way to successively halve the array's visible points down to a size that refines quickly and then double and refine the array's number of visible points successively until the full array was visible and refined.

Although this approach appeared relatively straightforward, it was not the same as doubling and refining a potential array successively. In the latter case, the electrode geometry of the array remains unchanged by successive doubling. However, in the former case, controlled point visibility via halving leads to point invisibility. If some of the invisible points are electrode points, the refining process does not see them, and their contributions to the potential field remain unaccounted for until these points become visible as the number of skipped points is reduced. A simple point skipping approach that ignores this, generally suffers enough refine time losses when the skipped electrode points regain visibility to fully offset any expected time savings.

5.2.1. Skipped point refining

An effort was initiated to develop a more time efficient refining algorithm that could include the effects of invisible skipped electrode points early in the solution process. This effort led to the development of the skipped point refining algorithm. Skipped point refine times are roughly proportional to n for arrays that are relatively close to square or cubic. Fig. 6 compares how the refine times increase (on a 400 MHz Pentium II) as the size of the potential array shown in Fig. 1 is successively doubled. It demonstrates how the skipped point refining algorithm can dramatically improve refine times as array size increases.

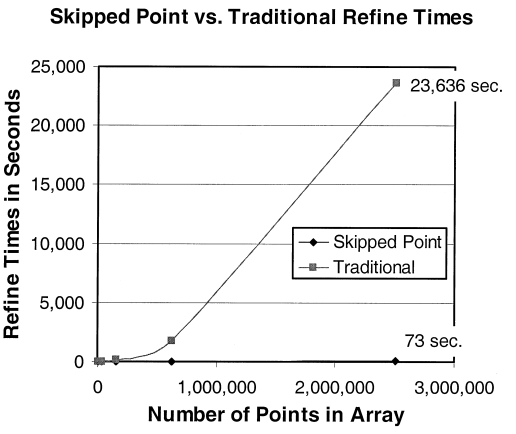

Fig. 6. Example of the difference in refine times between skipped point refining and the over-relaxation methods used in earlier PC SIMION versions.

Skipped point refining makes use of powers of two point skipping, a rather direct approach that becomes very complex in practice. This approach initially refines the smallest practical array size by skipping points (point skipping $= 2^{\text{skip level}}$), estimates the values of intermediate points, doubles the array density, and then refines again. The process continues until no points are being skipped (the final refine). The skipped point algorithm attacks the invisible electrode point problem by scanning for and flagging any skipped electrodes at the beginning of each level of skipped point refining. The refining algorithm now knows which points have one or more invisible electrode points near them. These special points are refined by including the effects of the invisible electrode points by a normalized inverse distance weighting function that converges properly for linear gradients. This approach, combined with special compensations for irregular shifts in array boundaries

as skipping changes, makes for a very complex refining algorithm, but one that is dramatically faster for large arrays than conventional finite difference methods. Skipped point refining methods permitted version 6.0 to support array sizes up to 10 million points. This was a good match with personal computer capabilities in 1995.

5.2.2. Ion optics workbench

Although skipped point refining was the critical first step in making a 3D version of PC SIMION possible, many other issues remained to be resolved. To successfully simulate more complex 3D problems, including complete instruments, the notion of an ion optics workbench volume was adopted. The maximum size of this simulation volume was limited to ± 1 km (8 km^3), because double precision floating point numbers on Intel compatible personal computers have approximately 15 digits of precision, and 1 km to 1 μm was about the maximum workable range (10^9) that would still conserve acceptable integration accuracy for ion trajectories.

The ion optics workbench volume holds virtual 3D images of potential arrays that are orientated, scaled, positioned, and projected as array instances within the volume. 2D arrays are converted to 3D images according to their symmetry. Thus a 2D cylindrical array becomes a 3D cylindrical volume of revolution when projected into the workbench volume. Unlike the prior 2D versions (2.0–5.0), the 3D versions of PC SIMION (6.0–7.0) can project up to 200 3D images of potential arrays into the ion optics workbench simulation volume.

The primary challenge faced by this approach was providing algorithms to visualize 3D array images and ion trajectories in ways that were highly interactive (even when ions were flying), flexible (2D and 3D volume zooms), and user intuitive. For example, one would like to be able to easily cut away portions of array images to see details and ion trajectories inside (see Fig. 7). Traditional computer graphics polygon rendering methods normally require powerful graphics engines to be highly interactive. Moreover, these methods do not naturally support the interactive cutting away of portions of images to see inside. This led to the development of a set of isometric graphics algorithms to achieve the desired interactive 3D viewing within the personal computer performance envelope.

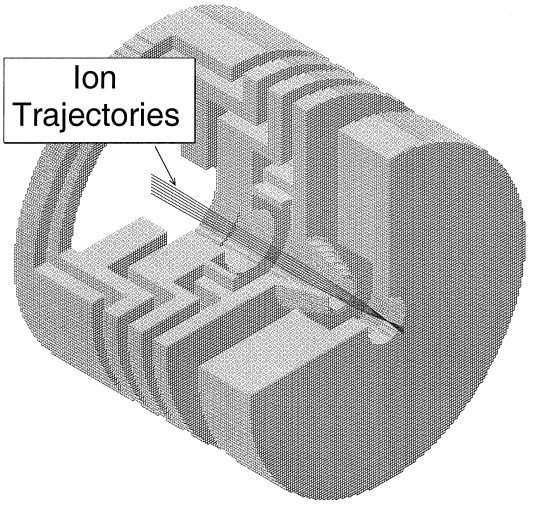

Fig. 7. Example of using the cutaway viewing capabilities to view ion trajectory focusing within an ion extraction lens assembly.

5.2.3. Visualization methods

The isometric graphics algorithms take advantage of the 3D cubic mesh nature of projected potential array images to obtain fast and efficient hidden line removal. Assuming that the image of an array is integrally aligned with the workbench coordinate system (each array axis is parallel to a workbench coordinate axis), array points are naturally positioned in bore lines (deeper points are directly inline with the points above them). This is true for all 2D views and also for the 8 possible diagonal isometric views. To find the highest visible electrode/pole point in a bore line, one need only start at the highest possible array point in the chosen bore line (closest to the observer) and descend down the bore line until the first electrode/pole point is encountered. When the first visible electrode/pole point is encountered a quick check is made for its connections to other adjacent electrode/pole points. This information is then used to rapidly create full screen visualizations as illustrated in Fig. 7 (drawing time $\cong$ 1 s–400 MHz personal computer). A

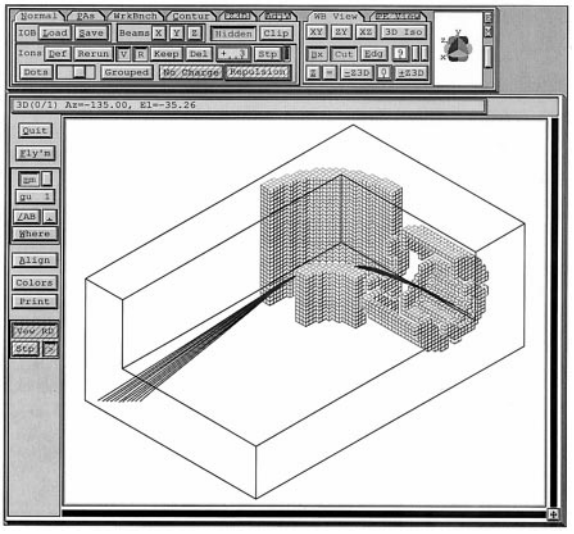

Fig. 8. Illustration of cutaway clipping via the use of the single window button isometric pointing capabilities (lower right corner).

sampling-style bore line algorithm is employed when the array instance is not integrally aligned with the workspace.

This approach facilitates the creation of cutaway views by allowing each bore line search to start at the front surface of the appropriate cutaway plane. Moreover, the option of skipping (not testing) one or more adjacent bore lines allows the adjustment of image quality so that a simpler (less detailed and faster drawn) image of an array can be automatically drawn as the user zooms away from an array's image, or when the user adjusts the drawing quality manually. The bore line strategy also facilitates hidden line removal of ion trajectories.

In order to take full advantage of these visualization capabilities version 6.0 was developed as a 32 bit MSDOS extended memory GUI application that would run in both MSDOS and Windows environments. Many of the GUI's graphical objects, such as panel objects and single button window objects, provide more interactive capabilities for programs similar to SIMION than the roughly equivalent Windows controls.

For example, in Fig. 8, SIMION's view screen shows ion trajectories in a cutaway view of a 3D potential array. At the lower right corner of the window is a small button. This button can be used to scroll the view in the *x* or *y* direction by placing the cursor on the button, holding down a mouse button, and moving the mouse in the *x* or *y* direction. Advanced features such as 3D isometric mouse pointing are performed in much the same manner as *x* or *y* scrolling. However, unlike 2D scrolling, when the mouse is moved in one of the three isometric directions the cutaway plane for that direction of motion automatically moves, tracking the mouse's motion. This provides a quick interactive way to obtain or adjust cutaway views.

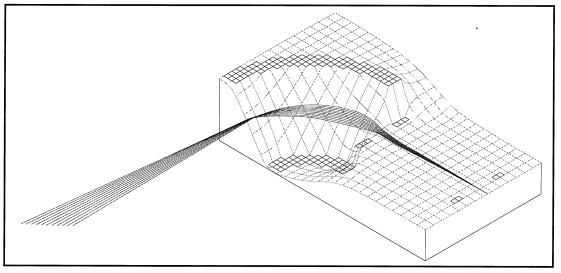

Fig. 9. Potential energy surface view of ion trajectories in Fig. 8.

As discussed previously, PC SIMION's visualization software uses floating point vector graphics to permit a wide range of workbench volumes from a cubic micron to a cubic kilometer to be viewed full screen. This and the strategy of nested 3D volumes is employed to allow the user to easily define a bi-directional path of zoom volumes to obtain the desired inner volume of the workbench for study. 3D zoom volumes can be viewed in isometric, 2D surface, or as potential energy views of selected 2D surfaces (see Fig. 9). Potential energy views remain a very powerful component of PC SIMION, because they provide highly intuitive representations of how electrostatic fields act to create the ion trajectories.

5.2.4. Geometry definition language

A solids modeling language and associated compiler were developed to support the complex geometry definition requirements of 3D arrays that went beyond the capabilities of the interactive array definition/modification function (Modify) within PC SIMION. The solids modeling language employs a nested structure of commands that define the fill

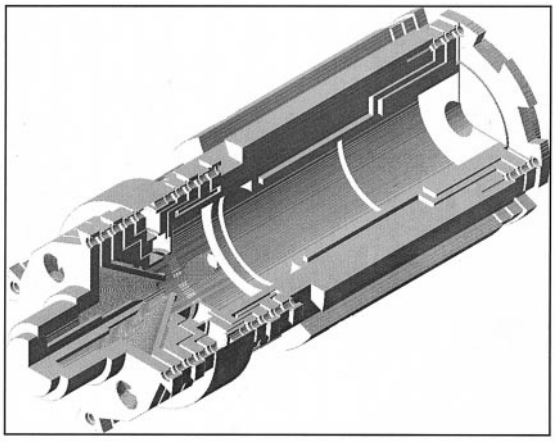

Fig. 10. Cutaway view of a 35 million point 3D array that simulates the extraction optics for a high temperature SIMS instrument.

volumes or volumes of revolution in terms of collections of inclusion and exclusion volumes defined by Boolean combinations of volume primitives (e.g. circles, spheres, parabolas, points and multiple line segments).

Files containing these geometry definitions are converted by a recursive compiler into a 3D heap structure that projects the solid geometry definitions into potential array geometry. The recursive nature of the compiler and the 3D heap structure it generates allows arbitrarily complex (virtual memory limited) geometry files to be processed. Geometry file debugging is facilitated via extensive error checking and interactive display of the generated potential array geometry.

The user can selectively map the same geometry definitions into either 2D or 3D arrays using different array grid densities as modeling requirements dictate. Fig. 10 shows a cutaway view of a complex high temperature SIMS extraction lens generated on a 0.010 in. grid spacing (a 35 million point 3D array using version 7.0).

5.2.5. Magnetic potential arrays

The ability to project images of multiple potential arrays into the workbench volume provided a relatively painless opportunity to extend magnetic simulation capabilities by adding support for magnetic potential arrays. Magnetic potential arrays make use of the same array defining and refining capabilities as electrostatic arrays. The units of magnetic potential are defined to be gauss $\times$ grid units (or Mags). Mags, a contrived unit for magnetic potential, is very convenient because its gradient in array coordinates is gauss. This means the magnetic fields in magnetic arrays are independent of array scaling in workbench coordinates (useful).

However, magnetic potential arrays are not as simple as electrostatic arrays. Although conductors conserve electrostatic potential along their surfaces, it is not very likely that a truly constant magnetic potential will be maintained along a pole face because of magnetic circuit geometry and permeability issues. Although magnetic potential arrays clearly do not make PC SIMION a magnetic circuits program, they can accurately calculate magnetic fields if the values of magnetic potential are defined accurately along the surfaces of the magnetic poles. On the other hand, if the magnetic field is known, the user programming capability described below can be used with analytical expressions for defining magnetic fields in the projected volume of a magnetic array or the array variables capability of version 7.0 can be used to project interpolated magnetic field estimates from measured data.

5.2.6. Instance order

Ion trajectories are calculated in workbench coordinate space where 3D images (or array instances) of electrostatic and/or magnetic potential arrays have been projected. Array instance data is maintained in a list, and PC SIMION uses the array instance's order in the list to resolve overlapped instance conflicts by always looking upward from the end of the list and using the first electrostatic and magnetic instances encountered that currently contain the ion. Ions can be flown separately (as in previous versions) or simultaneously in groups with ion trajectories displayed as either lines or moving dots. Flying ions in groups often serves to help visualize and understand the ion's relative motions and interactions.

5.2.7. Charge repulsion algorithms

Although PC SIMION does not support Poisson style space-charge calculations, its capability to fly groups of ions simultaneously has enabled the implementation of algorithms for interactively estimating the onset of charge repulsion effects via three methods. Each of these methods allows the user to change the charge, factor, or current (depending on method) as the ions fly to help determine when the onset of charge repulsion effects starts impacting the ion trajectories.

The first method is called Beam repulsion (for use with ion beams) in which each ion is considered to represent an infinite line of charge (a $1/r$ effect). The total user specified beam current is allocated to each ion's line of charge in the group according to charge weighting factor parameters. Ions are flown using space coherent integration. This is accomplished by using ion number one (located in the center of the beam) as leader of the pack. At each time step for ion number one, a plane is computed that contains the ion and is normal to its velocity. This plane is then used to control the time steps of all other ions so that they will fall within the plane to preserve the line charge nature of the beam repulsion simulation.

The second method is called coulombic (or ion cloud) repulsion (a $1/r^2$ effect). Each ion is allocated a portion of the total user specified charge (in coulombs) according to charge weighting factor parameters. Ions are flown using time coherent integration (the normal method for groups of ions).

Factor [or separated ion(s)] repulsion is the third method (a $1/r^2$ effect). Each ion acts as a multiple (or factor ≥ 1) of its charge allocated via charge weighting factor parameters. Ions are flown using time coherent integration as in coulombic repulsion above.

In each charge repulsion estimation method, an ion represents a collection or cloud of charged particles. 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. PC SIMION compensates for this problem by using radius factors 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. PC SIMION updates this value at 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 PC SIMION always uses 3.0×10^{-11} mm (10 times the classical electron radius) for the effective cloud diameter.

5.2.8. User programming capabilities

Perhaps the most powerful capability within PC SIMION is user programming. User programming allows the user to write subroutines that are uniquely associated with a designated potential array in a RPN calculator style language and have PC SIMION automatically compile and incorporate these subroutines into the ion flying process as ions pass through the projected image of the potential array. A user programming capability was first made available in version 4.0 of the program. However, its implementation was limited, and the whole notion of user programming was revisited in version 6.0.

Versions 6.0–7.0 user programs are defined in terms of a collection of user written program segments that are contained within .PRG user program files. Each .PRG file shares the name of the potential array it supports. User program segments within a .PRG file are selectively called by PC SIMION whenever an ion is within the projected instance of the .PRG file's associated potential array. PC SIMION calculates a parameter (e.g. the length of the next time step) and then passes this information, as well as other ion state parameters, to a program segment of a given name (e.g. Tstep_Adjust—if it exists in the .PRG file for the potential array the ion is currently within). The program segment then can examine the calculated parameter (e.g. time step) and adjust its value as required. Thus, PC SIMION now calculates something first and then passes the result to a user program segment (if it exists) for examination and modification as appropriate. The seven allowed program segments

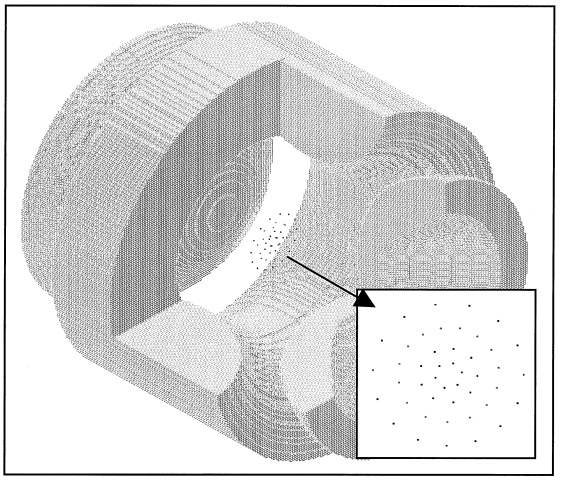

Fig. 11. User program simulation of ion crystal pattern formation processes in an ion trap with vacuum pressures in the viscous region. Ion crystal pattern shown in lower right. The pattern shown contains ions of 100, 200, and 300 u. The two inner shells contain the lightest ions, and the outer shell contains the heaviest ions.

within a .PRG file can be used to monitor and control: ion initial conditions, fast adjustable potentials, potentials and gradients, integration time step, ion accelerations, and the ion's state including position, velocity, mass, charge, color, death, and etc.

The expanded user program paradigm has added considerable power and flexibility to PC SIMION. The ability to vary potentials while ions are flying permits simulations of the time varying potentials found in ion traps, quadrupoles, TOF, and FTMS mass spectrometers as examples. Issues such as viscous or collisional effects can be simulated by including the appropriate models in a user program segment. User programs can also control the updating of potential energy surfaces to graphically illustrate the undulating nature of rf or other time varying fields. Fig. 11 demonstrates the observed ion mass shelling and crystal patterns formed in an ion trap [7] when mutual ion repulsion and viscous damping pressures are simulated. Our group has employed user programmed Monte Carlo simulations to analyze and develop self-stabilizing charge compensation methods [8]. Impacts of chemical effects such as fragmentation and neutralization can also be simulated. The implementation of user programming is sufficiently general that PC SIMION is now capable of modeling the motions of arbitrary objects in arbitrarily user defined fields with user defined acceleration characteristics associated (or unassociated) with these fields.

6. Version 7.0

Version 6.0 was released in 1995 at about the same time as Windows 95. Although version 6.0 was a 32 bit extended memory MSDOS program, it was fully functional with the beta versions of Windows 95 and NT. However, Microsoft drew a line in the sand with the actual release versions of Windows 95 and NT by reducing DPMI (DOS protected mode interface) virtual memory from 2 GB to 64 MB. This was a severe constraint for larger potential arrays, and it became clear that a Windows compatible version would be needed. However, the intense effort required to create version 6.0 had taken its toll, relegating the Windows compatible version effort to a background activity.

6.1. GUI porting alternatives

At this juncture there were two alternative conversion strategies: Convert PC SIMION to a fully Win32 compliant program with the classical Windows look and feel, or port PC SIMION's GUI into the Win32 environment and retain version 6.0's look and feel. On balance, it appeared it would be easier and better to port version 6.0's GUI into the Win32 environment to retain its features such as keyboard accessible buttons, numerical panel objects, and one button windows with 3D pointing capabilities than to totally rewrite PC SIMION in a full Windows paradigm.

The porting effort proved challenging because of the many paradigm differences between the two GUIs and the nonsymmetrical and inconsistent nature of the Win32 GDI. Personal consternation aside, the effort proved successful, and version 7.0 retains version 6.0's GUI while running as a native Win32 application in the Windows NT/9x environment. The retained 6.0 GUI interface is in some ways superior to Windows for applications similar to SIMION (as discussed previously). However, adopting the Win32

platform permits access to useful features such as Windows printers, clipboards, metafiles, TrueType fonts for annotations, long file names, and video accelerator cards. The integration of PC SIMION into the Windows environment appears to mesh advantages from each environment.

6.2. Larger arrays

Version 7.0, as in previous PC SIMION versions, strives to couple its capability enhancements with the capabilities of the current PC hardware and software. Thus 7.0 can support 50 million point arrays (requiring 500 MB of RAM) versus 6.0's maximum of 10 million points. Virtual memory support has increased to 2 GB, and access to physical RAM has increased from 64 MB (version 6.0) to whatever is currently installed in the computer (version 7.0). These increases are consistent with the current availability of PCs with more than 512 MB of RAM and hard drives of over 10 GB capacity.

6.3. Ease of use and visualization enhancements

Features were added to help make the program more useful and seamless to the user. Version 7.0 can load (or save) ion optics bench definitions, associated potential arrays, saved ion trajectory images, ion definitions and data recording files together to help simplify simulation setups. Cursor position can be displayed in relative and absolute coordinates. The annotator supports TrueType fonts and additional capabilities such as arrows and dimensions.

The options in 7.0 for preferentially controlling the display of ion trajectories versus recorded data have been expanded to more efficiently support those simulations where recorded data are more desirable than visualized ion trajectories (e.g. external ion injection into ion trap simulations).

An asymmetrically scaled 2D display capability has been added to aid in visualizing ion focusing regions in long beam lines. The two illustrations in Fig. 12 demonstrate how asymmetrical scaling helped elucidate the beam focusing characteristics of our group's PC SIMION designed univoltage ion gun [9].

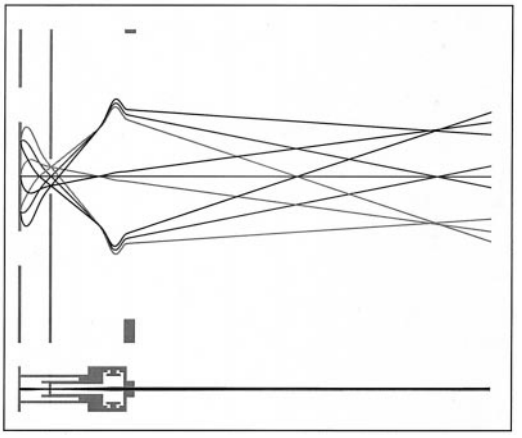

Fig. 12. Asymmetrical x and y scaling visualization option. The top panel shows how increased magnification in y scaling can help trajectory visualization. The lower panel shows an equal x and y scaling view.

The y scale in the upper illustration has been greatly magnified to help resolve the ion gun's focusing characteristics.

6.4. Functional improvements

Potential array capabilities have been expanded to allow fast proportional scaling of the potentials of groups of electrode/pole points and fast voltage adjustment of other groups of points in the same potential array. This allows proportional scaling adjustment of electrode/pole points that create gradient fields (e.g. TOF reflectrons) while retaining the ability to define individual fast adjust electrodes in the same potential array.

Geometry files have received several new commands (e.g. notin_inside and notin_inside_or_on) that assist in defining more precise boundaries. These commands can help to preserve dimensional accuracy better when electrode geometry definitions are projected into arrays of different sizes.

User programming adds support for arrays and data files that can be used to initialize and save array data. A new user program segment has been added (Init_P_Values) that can be used to fast adjust and/or fast scale selected array potentials just before ions are flown. This feature is more efficient than the Fast_Ad-

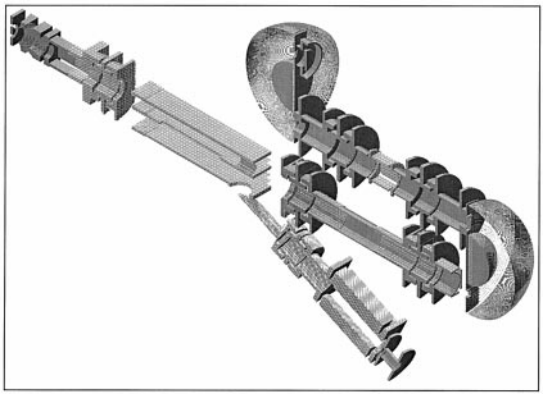

Fig. 13. PC SIMION 7.0 Simulation of Argonne National Laboratory's SARISA (surface analysis by resonant ionization of sputtered atoms) instrument.

just program segment when these potentials do not change while ions are actually flying. Additional commands and reserved variables increase the span of control the user can exert on what is displayed, what is retained, and how certain calculations are performed.

Data striations in the random number generator used in PC SIMION versions (4.0–6.0) for certain user programming applications were detected by Richard Morrison of Monash University, Australia. Version 7.0 incorporates a new random-walk pseudorandom generator that has passed extensive tests for randomness [10].

7. Complex instrument simulation example

PC SIMION versions 6.0 and 7.0 provide a quantum improvement over previous PC SIMION versions for complex simulations. Fig. 13 shows a SIMION 3D 7.0 cutaway view of the secondary beam line of Argonne National Laboratory's (ANL) SARISA (surface analysis by resonant ionization of sputtered atoms) instrument. Igor Veryovkin et al. [11] of ANL have used a version of SIMION 3D 7.0 to model the entire instrument. The goal of the ANL effort is to create a more highly optimized instrument. The simulation includes the entire instrument (both primary and secondary beam lines) modeled using 21 array instances that make extensive use of array variables in user programs to dynamically control electrode potentials as the motions of ions and ionization of neutrals are simulated. The array memory requirement for this simulation is approximately 440 MB of RAM, and a 400 MHz Pentium II PC with 512 MB of RAM can run entire simulations interactively in physical RAM. This example serves to illustrate how the capabilities of PC SIMION have progressed from its beginnings in 1986 as version 2.0.

8. Some tests of PC SIMION's accuracy

In the final analysis a simulation tool's value is determined by how accurately it simulates reality. Many issues contribute to simulation accuracy including: capabilities of the tool, how the problem was simulated, and the user's knowledge and understanding of the issues involved.

The following material discusses how accurately PC SIMION version 7.0 simulates a small collection of problems where the correct answers are known from first principles. Version 7.0 includes these tests in directories with its other simulation demos. The physical constants used in version 6.0 and 7.0 are from the 74th edition of the *CRC Handbook of Chemistry and Physics* [12]. Values of these physical constants (e.g. *c*, electron mass, and elementary charge) are defined to 8 or 9 significant place accuracy.

The ranges of fractional accuracy (abs((simulation − expected)/expected)) determined from these tests are summarized in Table 1. The span of fractional accuracy is due to the collection of parameters being tested as well as the impact of varying the trajectory quality parameter, array density, and in some cases the use of analytical expressions for known fields. The descriptions of each series of tests contain the assumptions and the range of ways the simulations were conducted. Moreover, these tests clearly illustrate how simulation details can impact accuracy.

Table 1
Summary of PC SIMION accuracy tests, where the fractional error = abs[(simulated − expected)/expected]

Test	Description	Fractional error range
9.1	Conversions between kinetic energy and velocity	10^{-8}–10^{-9}
9.2	Ion acceleration in linear electrostatic potential gradients	10^{-7}–10^{-9}
9.3	Conservation of energy in discontinuous linear gradient reflection fields	10^{-7}–10^{-8}
9.4	Magnetic and electrostatic radii of refraction	10^{-6}–10^{-8}
9.5	rf conservation of energy	10^{-5}–10^{-8}
9.6	Cylindrical electrode field orbits	10^{-4}–10^{-7}
9.7	Ion motions in a static quadrupole field	10^{-4}–10^{-9}

8.1. Conversions between kinetic energy and velocity

The first simple test involves flying an electron with a relativistic kinetic energy equal to its rest mass and a proton with a nonrelativistic kinetic energy of 1 eV through one meter of field free space. The test evaluates the conversion of initial kinetic energy into relativistic (Table 2) and nonrelativistic (Table 3) velocities, measuring the time of flight through a field-free meter of space, and then converting the velocities back to kinetic energies at the one meter exit point (splat). Each table compares the simulation values with the values expected from first principles. Agreement is within one part in the eighth significant place.

8.2. Ion acceleration in linear electrostatic potential gradients

The second test series covers ion accelerations in electrostatic potential arrays with linear potential gradient fields. The first test is a nonrelativistic cross acceleration of a 100 u ion. The ion has an initial kinetic energy of 20 eV in the $-y$ direction. It is accelerated 50 mm in the $+x$ direction by a 1 V/mm gradient electrostatic field. The ion's final velocity in the x direction (9.822 693 45 mm/μs) is within one part in the eighth significant place from the x velocity expected [$vx_{\text{expected}} = \text{sqrt}(2a_{\text{expected}}s)$, where a_{expected} = force/mass value and s = 50 mm]. The final kinetic energy from vx and vy (6.999 999 91 $\times$ 10^1 eV) is within one part in the eighth significant place of its expected value.

The next test is linear relativistic acceleration of an electron (initially at rest) across a potential difference equal to one electron rest mass (501 999.06 eV). The final velocity (2.596 278 84 $\times$ 10^5 mm/μs) is within one part in the ninth significant place of the expected value. A cross relativistic acceleration test uses three electrons with initial kinetic energies in $-y$ of 1, 3, and 7 electron rest masses and accelerates them by one electron rest mass (501 999.06 eV) in the x direction. The electrons' calculated final kinetic energies (1.021 99812 $\times$ 10^6, 2.043 996 19 $\times$ 10^6, *and* 4.087 992 00 $\times$ 10^6 *eV*) and speeds are all better than one part in the seventh place of the expected values.

8.3. Conservation of energy in discontinuous linear gradient reflection fields

A linear potential gradient reflection field is used to test conservation of energy with velocity reversals and crossing ideal grid discontinuities (Fig. 3). An ion started with a given kinetic energy in x halfway up a linear trough field in y (y_{start} = 25 mm) oscillates along the trough formed by two electrodes (set to

Table 2
Tests of relativistic conversions between kinetic energy and velocity

Relativistic electron tests	From simulation	Expected from first principles	Fractional error
Kinetic energy (KE) to velocity (KE = 5.109 990 6 $\times$ 10^5 eV)	2.596 278 83 $\times$ 10^5 mm/μs	2.596 278 84 $\times$ 10^5 mm/μs	4 $\times$ 10^{-9}
Time of flight (d = 1000 mm)	3.851 666 43 $\times$ 10^{-3} μs	3.851 666 41 $\times$ 10^{-3} μs	5 $\times$ 10^{-9}
Convert velocity to KE at splat	5.109 990 56 $\times$ 10^5 eV	5.109 990 6 $\times$ 10^5 eV	8 $\times$ 10^{-9}

Table 3
Tests of nonrelativistic conversions between kinetic energy and velocity

Nonrelativistic proton tests	From simulation	Expected from first principles	Fractional error
Convert KE to velocity (KE = 1.0 eV)	$1.384\,112\,04 \times 10^{1}$ mm/μs	$1.384\,112\,03 \times 10^{1}$ mm/μs	7×10^{-9}
Time of flight (d = 1000 mm)	$7.224\,848\,65 \times 10^{1}$ μs	$7.224\,848\,69 \times 10^{1}$ μs	6×10^{-9}
Velocity to KE at splat	$9.999\,999\,98 \times 10^{-1}$ eV	1.0 eV	2×10^{-9}

equal potentials) and a center ideal grid (grounded). After 14 cycles the ion y max ($y_{\text{cycle 14}} = 2.499\,999\,3 \times 10^{1}$ mm) is within one part in the seventh significant place of its starting y location (25 mm), and its time of flight is within one part in the eighth significant place of the expected value.

8.4. Magnetic and electrostatic radii of refraction

Ions with velocities normal to a static magnetic field have circular trajectories. The magnetic radius of refraction is: $r_n = mv/(B_n e)$. Where r_n is the magnetic radius of refraction, m is the ion's relativistic mass, v is its velocity, B_n is the magnetic field intensity normal to v, and e is the charge on the ion. Orbital diameters and orbital frequencies are calculated for a 10 000 G field normal to velocity of three test ions: A relativistic electron with one rest mass of kinetic energy, a proton with 1.0 keV of kinetic energy, and a 100 u ion with 100 eV of kinetic energy. The orbital diameters (5.904 597 779, 9.138 793 39, and $2.879\,482\,21 \times 10^{1}$ mm) and the orbital frequencies ($1.399\,623\,27 \times 10^{10}$, $1.524\,516\,52 \times 10^{7}$, and $1.535\,611\,33 \times 10^{5}$ s^{-1}) are all within one part in the sixth significant place of their expected values. Increasing the trajectory quality parameter from 3 to 103 improves the accuracy to better than one part in the eighth significant place.

The electrostatic radius of refraction is: $r_n = mv^2/(-E_n e)$. Where r_n is the electrostatic radius of refraction, m is the ion's relativistic mass, v is its velocity, E_n is the electrostatic field intensity normal to v, and e is the charge on the ion. This test employs a trick. A user program automatically applies a 40 000 V/mm gradient normal to the ion's current velocity vector (would that we could create such fields). Four test ions are used: a relativistic electron with one rest mass of kinetic energy, a 1 u negative ion with an electron's rest mass of kinetic energy, a 100 u ion with a kinetic energy of 40 000 eV, and a 10 000 u ion with a kinetic energy of 40 000 eV.

The orbital diameter estimates ($3.832\,497\,52 \times 10^{1}$, $5.108\,590\,17 \times 10^{1}$, 3.999 999 16, and 4.000 000 00 mm) and the orbital frequency estimates ($2.156\,351\,86 \times 10^{9}$, $6.184\,806\,07 \times 10^{7}$, $2.210\,882\,55 \times 10^{7}$, and $2.210\,882\,74 \times 10^{6}$ s^{-1}) are all better than 1 part in the sixth significant place of their expected values. Increasing the trajectory quality parameter from 3 to 103 improves the accuracy to better than one part in the eighth significant place.

8.5. rf conservation of energy

In this series of tests a 100 V rf cosine wave and an 80 V rf square wave of a frequency of 140 000 Hz are impressed between two parallel plate electrodes 80 mm apart in y (e.g. $V_{p1} = 100 \cos wt$ and $V_{p2} = -100 \cos wt$). If an ion materializes near one of the plates flying in the x direction at the instant that the rf cosine wave's potential peaks or in the center of the rf square wave's peak at a potential that attracts the ion to the opposite plate (e.g. $t = 0$), the ion will follow a periodic trajectory between the plates (providing the ion's excursion distance does not cause it to hit one of the plates).

For the rf cosine wave: $a = k(\cos wt)/m$, $v = k(\sin wt)/(mw)$, and $s = -k(\cos wt)/(mw^2)$. Where a is acceleration, v = velocity, s = displacement, m = mass, w = angular frequency, and k includes the potential and scaling factors. The expected quarter cycle ($wt = \pi/2$) displacement is $r = k/(mw^2)$, and the expected peak to peak value is $Dy = 2r$. The

simulation test uses a collection of ions with masses ranging from 10 to 50 000 u. *Dy* (peak to peak) for each of these ions vary from one part in the fifth significant place for the 50 000 u ion to one part in the eighth significant place for the 10 u ion. If the value of the trajectory quality parameter is increased from 3 to 103 all estimates are better than one part in the eighth significant place when compared to expected values.

For the rf square wave: $s = at^2/2 = -kt^2/m$. Where a = acceleration, t = time, s = displacement, m = mass, and k includes the potential and scaling factors. The expected quarter cycle displacement is $r = kt_{1/4}^2/m$, and the expected peak to peak value is $Dy = 2r$. In order to obtain the proper *Dy* estimates the trajectory algorithms must automatically detect the square wave transition to initiate the binary boundary approach. The simulation test uses a collection of ions with masses ranging from 10 to 50 000 u. The accuracy of the *Dy* estimate (peak to peak) for these ions vary from one part in the sixth significant place for the 50 000 u ion to one part in the eighth significant place for the 10 u ion. If the value of the trajectory quality parameter is increased from 3 to 103 all estimates are better than one part in the eighth significant place when compared to expected values.

The reason that raising the value of the trajectory quality parameter improved the *Dy* estimates for higher mass ions is that values above 100 for the trajectory quality parameter shorten the distance time step and also reduce the binary boundary approach limits. Since the high mass ions move fewer grid units in *y*, a higher value for the trajectory quality parameter increases the number of time steps used and forces tighter control of velocity reversals.

8.6. Cylindrical electrode field orbits

The next series of tests involve the simulation of an ion orbiting between two cylindrical electrodes. The electrostatic potential between the electrodes is: $V_r = V_{r1} + k \ln (r/r1)$. Where V_r is the field potential at r, V_{r1} is potential at $r1$, $r1$ is the radius of the inner electrode, r is the desired radius value, and k is scaled to give the potential of the outer cylindrical electrode when $r = r2${the inner radius of the outer electrode: $k = [\ln (r2/r1)]/(V_{r2} - V_{r1})\}$. The test involves launching a nonrelativistic ion on an orbital trajectory from midpoint (r = 300 mm) between the two cylindrical electrodes. The ion's initial kinetic energy equals that required for a perfect circular orbit in an ideal analytical field. After 1/2 orbit the ion's current orbital radius is compared to its starting radius.

The first series of tests make use of 2D planar arrays with mirroring in x and y to simulate the cylindrical electrodes. A 2D planar array of 103 × 103 points in x and y gives a 1/2 orbit radius estimate of 300.138 221 mm (300 mm expected) that is within 1 part in the fourth significant place. Using a larger 2D planar array of 821 × 821 points in x and y gives a 1/2 orbit radius estimate of 300.015 655 9 mm (approximately 1 part in the fifth significant place).

The second series of tests make use of 2D cylindrical arrays with y mirroring (surfaces of revolution) to simulate the cylindrically concentric field. A 2D cylindrical array of 26x by 103y points gives a 1/2 orbit radius of 299.988 169 mm (one part in the fifth significant place). A really enormous 2D cylindrical array of 801x × 3281y points gives a 1/2 orbit radius of 299.999 959 mm (or one part in the seventh significant place).

A third test makes use of a user program to provide the analytically correct field gradients into the trajectory computations. The resulting 1/2 orbit radius of 300.000 035 mm is accurate to within one part in the seventh significant place of the expected value.

These tests point out that larger arrays model fields more accurately. Moreover, choosing an array that matches the problem's symmetry can also help improve field estimates. However, user programs can offer higher quality ion trajectory estimates when the field is known analytically.

8.7. Ion motions in a static quadrupole field

The final series of tests simulate flying ions within a static quadrupolar field. The electrostatic potential of a quadrupolar field is: $V_{xy} = V_{\text{xpole}}(x^2 - y^2)/r_0^2$. Where V_{xy} is the potential at point x, y, V_{xpole} is the potential of the x direction hyperbolic electrodes, x

Table 4
Expected vs. simulation values of x_{max} at velocity reversals

x at velocity reversal	Ion one's x_{max} (mm)	Ion two's x_{max} (mm)	Ion three's x_{max} (mm)
Expected values from first principles	70.710 678 12	50.000 000 0	25.000 000 0
From first simulation	70.698 767 30	49.991 027 7	24.995 445 7
Fractional error	2×10^{-4}	2×10^{-4}	2×10^{-4}

and y are the offsets from the center of the quadrupole, and r_0 equals radial offset of the hyperbolic electrodes from the center of the field. The tests involve flying three ions of varying initial kinetic energies from the center of the quadrupole up an electrostatic hill toward the positive x electrode. The tests measure the x_{max} when the ion's velocity reverses, and the time of flight to velocity reversal (the 1/4 cycle time).

The first simulation used a large 2D planar potential array of 1200x and 1200y points with mirroring in x and y. Table 4 compares the expected x_{max} values to those obtained from the first simulation. These estimates were within two parts in the fourth significant place. Because the ion motions in a static quadrupolar field are sinusoidal, the expected 1/4 cycle times should be the same for all ions: 6.193 482 464 μs. Table 5 compares the expected $t_{1/4}$ times to those obtained from the first simulation.

In a second simulation, user programs were employed to provide the analytically correct fields to the trajectory algorithms. In this case the x_{max} estimates were within one part in the ninth significant place and the estimated 1/4 cycle times were within one part in the eighth significant place.

9. Reality of simulations

It has been my personal experience that simulations can only be as good as the understanding that goes into them (e.g. garbage in garbage out). Tools such as PC SIMION should not be used in blind faith. Just because a few ions manage to get through does not mean the design is acceptable.

Questions similar to the following should be asked when evaluating a simulation effort. Do the range of initial conditions of the test ions accurately represent the range of expected initial conditions for the actual ions? Do ion trajectories change significantly when higher values of the trajectory quality parameter are specified, when potential array sizes are increased (doubled), or when arrays are refined to a lower error value? Does reverse flying ions back from their termination point demonstrate a comfortable level of internal consistency? Does the simulation of analytically known fields via user programs yield significantly different results from using refined potential arrays?

It is important to understand the physics of the problem and the implications of the chosen simulation strategy. Do not assume the simulation tool thinks for

Table 5
Expected vs. simulation values for time to velocity reversals ($t_{1/4}$ = 1/4 cycle times)

$t_{1/4}$ at velocity reversal	Ion one's $t_{1/4}$ (μs)	Ion two's $t_{1/4}$ (μs)	Ion three's $t_{1/4}$ (μs)
Expected values from first principles	6.193 482 46	6.193 482 46	6.193 482 46
From first simulation	6.192 522 79	6.192 388 39	6.192 354 83
Fractional error	2×10^{-4}	2×10^{-4}	2×10^{-4}

you! Although the results of the simulations may be very exciting, wait until the design is built and tested successfully before declaring victory. Always be suspicious!

10. Future directions for PC SIMION

PC SIMION has always been a part-time one-man effort with the attendant irregular rates of progress. Thus the PC SIMION effort must be viewed as a work in progress with each successive version being the next chapter in the saga. Unfortunately, one's horizon expands with each new version, as does the list of desirable new capabilities. Three general areas are currently on the list to be addressed in future versions of PC SIMION.

The first area of interest is in expanding the user's ability to program PC SIMION to perform more complex tasks. This will probably manifest itself in some form of job control language with hooks and other program extensions that allow tasks such as defining array geometry, refining, and ion flying to be more automated and integrated. If successful, a future version of PC SIMION could be able to perform tasks such as optimizing electrode geometry under user programmed control.

The second is in adding simulation methods to the program to improve its modeling of complex electrode shapes. Version 6.0 was designed to support, at least in principle, multiple modeling methods. The modeling of field emission sources and the higher accuracy modeling of complex electrode shapes will require the addition of some form of a flexible mesh style modeling method that hopefully can be fully integrated into the program's current visualization and operating paradigms.

The third would be to add full Poisson space charge simulation capability to a future version of PC SIMION. Accurate modeling of space charge effects is very dependent on using truly representative collections of simulation ions and on accurate field determinations in the ions' low kinetic energy regions (e.g. near emission surfaces). Thus this effort will probably wait until significant accomplishments have been made in the first two areas.

11. Conclusions

PC SIMION has certainly changed over its 15 years of development (or obsession). However, the experience has been challenging and a lot of fun (particularly versions 6.0 and 7.0). My only personal disappointment has been that later versions of PC SIMION have been commercialized instead of freely shared with the community in the tradition of Don McGilvery's SIMION. However, it is rewarding to see one's efforts widely used by others to advantage in their endeavors.

References

[1] D.A. Dahl, J.E. Delmore, A.D. Appelhans, Rev. Sci. Instrum. 61 (1990) 607.
[2] D.C. McGilvery, Proceedings of the 46th ASMS Conference on Mass Spectrometry and Allied Topics, May 31–June 4, 1998, Orlando, Florida.
[3] P.J. Todd, private communication (2000).
[4] G.E. Forsythe, W.R. Wasow, Finite-Difference Methods for Partial Differential Equations, Wiley, New York, 1960.
[5] R.W. Hornbeck, Numerical Methods, Quantum, New York, 1975.
[6] A.D. Appelhans, D.A. Dahl, J.E. Delmore, Anal. Chem. 62 (1990) 1679.
[7] R.F. Wuerker, H. Shelton, R.V. Langmuir, J. Appl. Phys. 30 (1959) 342.
[8] D.A. Dahl, A.D. Appelhans, Int. J. Mass Spectrom. Ion Processes 178 (1998) 187.
[9] D.A. Dahl, A.D. Appelhans, Int. J. Mass Spectrom. Ion Processes 189 (1999) 39.
[10] D.A. Dahl, C.L. Atwood, R.A. La Violette, Appl. Math. Model. 24 (2000) 771–778.
[11] Proceedings of the Twelfth International Conference on Secondary Ion Mass Spectrometry, SIMS XII, Sept. 5–10, 1999, University Catholique de Louvain, Brussels, Belgium, Elsevier, New York, 2000, p. 237.
[12] Handbook of Chemistry and Physics, D.R. Lide (Ed.), 74th ed. CRC Press, Boca Raton, FL, 1993.