Advanced SIMION Ion Optics

Dave Dahl

Tony Appelhans

ASMS 2000

1.0 Introduction

- Objective

  Provide exposure to advanced SIMION features of:
  - array instances,
  - geometry files, and
  - user programs.

Introduction

- Format: Short Lectures (usually) followed by labs.
- Breaks at 10:15, and 2:30
- Lunch from 12-1:00 (provided)
- Bathrooms, fire exits
- Material: Course Notes, Reference Notes, SIMION 7.0 manual, CD with SIMION 7.0 and everything else on it.

Course Outline - Day 1

1. SIMION 7.0 – new features	9:15-9:30
2. Trajectories+Instance lecture	9:30-10:15
Break 10:15-10:45	
3. Trajectory + Instance Lab	10:45-11:15
4. Geometry files + lab	11:15-12:00
Lunch 12:00-1:00	
5. 2D and 3D gem files + lab	1:00-2:00
6. Strategy and tricks of the trade	2:00-2:30
Break 2:30-3:00	
7. Description of complex example	3:00-3:15
8. Creating gem files - Lab	3:15-4:30

Course Outline - Day 2

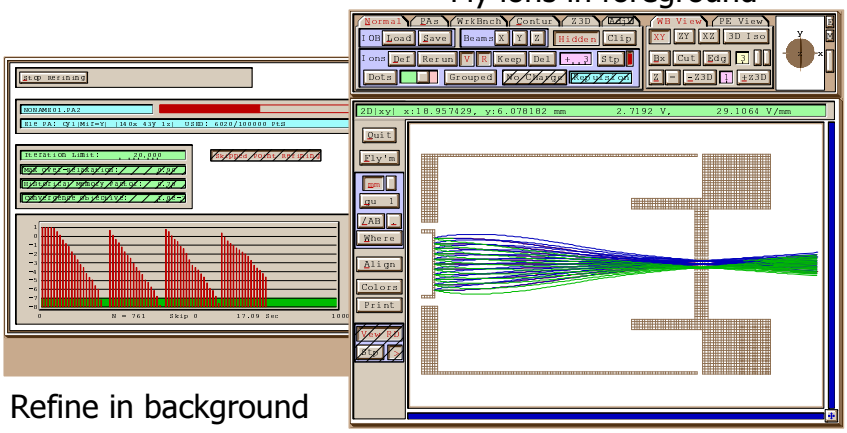
9. User Programming Intro. & Lab	9:00-9:30
10. Details Lecture & Lab	9:30-10:15
Break 10:15-10:45	
11. Lab continued	10:45-11:15
12. Strategies and tricks	11:15-12:00
Lunch 12:00-1:00	
13. Lab & Application Vignettes	1:00-2:30
Break 2:30-3:00	
14. Lab & Application Vignettes cont.	3:00-4:15
15. Graduation ceremony	4:15-4:30

2. SIMION 7.0 New Features

- Window gymnastics
  - Uses Windows display drivers and video accelerator boards
  - Activate window - cursor changes, underlines appear
  - resize for printing
  - multiple windows can be open (multiple processors)
- File loading with IOBs
  - auto-load saved/kept trajectories, voltages, scaling, data recording, etc
- Asymmetric scaling
- Where – Absolute/relative distances (x,y,z,r)
- Potential Array size
  - 50 M point arrays (500 MB)
  - address 2 GB of memory (current processor limit)

Multiple Windows

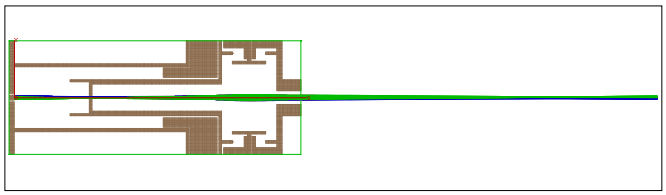
Fly ions in foreground



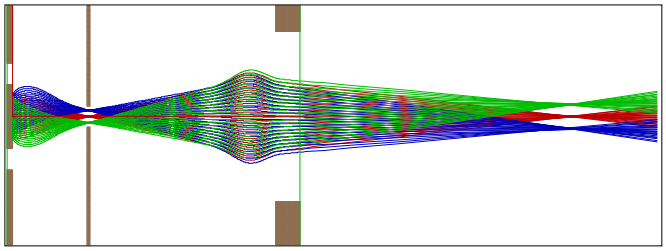
Refine in background

Asymmetric Scaling

Equal X and Y



Zoomed Y same X



SIMION 7.0 New Features

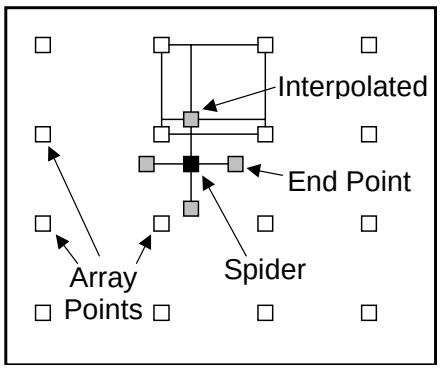
- Read/Write Arrays within User Programs
  - up to 200 arrays allowed
  - array size limited only by virtual memory
  - load/save data to ASCII files
- Additional commands in Geometry Files
  - more precise within and notin commands
- Printing and annotations
  - Any Windows compatible printer
  - output to Clipboard
  - output to Enhanced Metafiles
  - Windows Metafiles
  - Scalable True Type fonts in annotations
  - Dimensions and arrows

3.a Ion Trajectories

- How field gradients are obtained
- Constant distance step integration
- Stop length step shortening
- Binary boundary approach method
- CV of acceleration provides some vision
- How Trajectory Quality param. controls all
- Flying ions singly or in groups

Obtaining Field Gradients

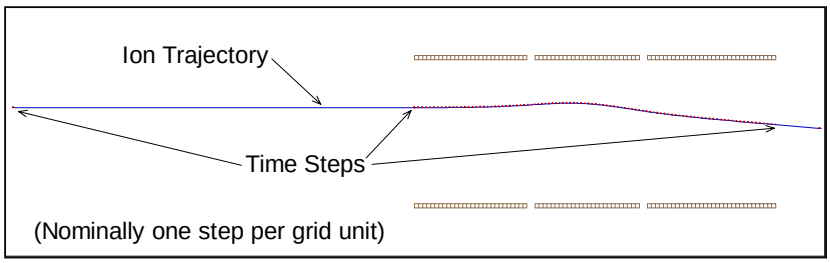
- From Arrays:



- Other: From User Programs

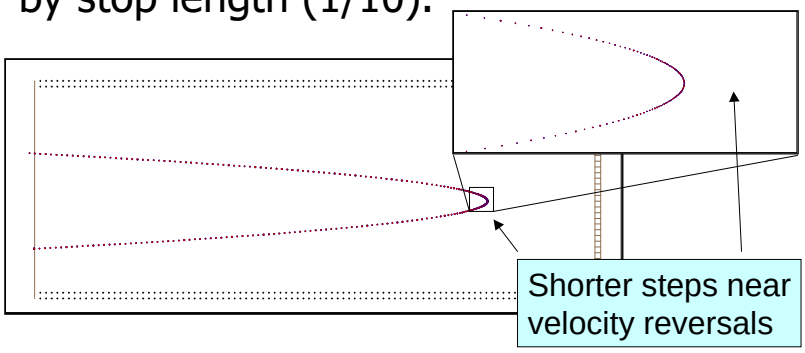
Numerical Integration

- SIMION uses a highly modified 4th order Runge-Kutta method orientated to fixed distance step integration.



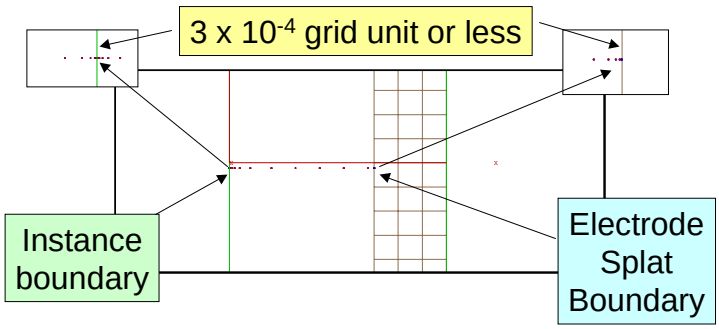
Numerical Integration

- Fixed distance step integration is modified by stop length (1/10).



Binary Boundary Approach

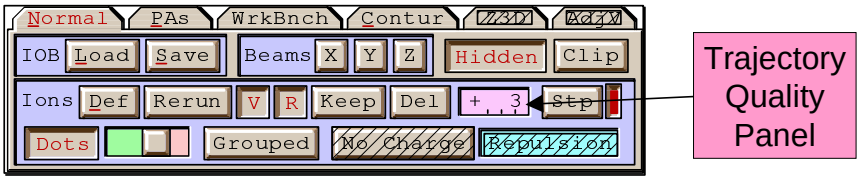
- Detect, leap back, and halve the time step is used to approach boundaries.



Binary Boundary Approach

- Detected boundaries:
  - Instance boundaries
  - Electrode (splat) boundaries
  - Velocity reversals
  - Regions of higher coefficient of variation of acceleration -- field discontinuities (grids) or time varying fields (step changes in potentials).

Adjusting Trajectory Quality



Trajectory Quality	Distance Step in gu	Minimum Step in gu	Stop Length	Binary Boundary	CV Upper Limit
-n	1/(1+n)	NA	yes	no	NA
0	1	NA	yes	no	NA
3 (default)	1	$1/(3 \times 10^4)$	yes	yes	1/3
$1 \le n \le 100$	1	$1/(n \times 10^4)$	yes	yes	1/n
$100 < n$	1/(1+(n-100))	$1/(n \times 10^4)$	yes	yes	1/n

Trajectory Quality

- Positive values of trajectory quality generally give the most accurate trajectories.
  - The default value of 3 represents a good starting compromise.
  - Higher values can improve accuracy at the cost of speed.
  - Vary positive quality to evaluate accuracy.

Trajectory Quality

- Zero or negative values of trajectory quality give the fastest computations.
  - The value of 0 provides the fastest and least accurate trajectories.
  - Increasingly negative values improve accuracy to a point.
  - Negative values are useful for flying ions in groups or when history is lost (damping).

Trajectory Quality hints

- Flying ions singly (most efficient)
  - Use default value of 3
  - Test for consistency with higher values
  - 0 or negative values have no boundary detect
- Flying ions in groups (can be very slow)
  - Use 0 or negative values for speed
  - Positive values can be VERY slow (most constrained ion slows all others down)

Ion Trajectory Calculation Lab

Introduction

These labs are designed to demonstrate the impact of the ion trajectory quality parameter on the accuracy of ion trajectories.

Location: C:\Advance SIMION Class\03. Trajectories and Instances\Trajectory Lab

DC Reflection Field Experiment

The first experiment uses linear gradient fields and an ideal grid to test the conservation of energy in linear fields with discontinuities. The fields form an electrostatic trough that acts to reflect the ion into wave-like motions.

Preparation

Load the reflections.pa# file and refine it to 1.0e-7
Remove all arrays
Load the reflections.iob file into view
Click the Fly'm buttons

Discussion

The pa with its central ideal mirror forms a reflection trough. Ions are launched with a small amount of kinetic energy in the x direction half way up one side of the trough in y. The ion forms a trajectory wave as it swings across the center grid. Energy is conserved if successive peak heights remain the same (successive ymaxes remain the same).

Note: Two red neutrals are flown to serve as a reference for peak height measurements for the tests described below.

Experiment

As a test, vary the trajectory quality to 103 or above. Now try 0 and various negative numbers. Notice that 0 or negative numbers do not conserve energy very well. Turn on data recording so that a marker is generated each time step. Notice that time steps bunch near velocity reversals and around the ideal grid discontinuity when trajectory quality is positive. This helps to better conserve the ion's energy in the calculation.

Second and Third Experiments

These tests uses a RF linear gradient fields between two plates to test the conservation of energy in linear RF fields with discontinuities. The RF field forms an electrostatic RF trough that acts to reflect the ion into wave-like motions.

RF Mirror Experiment

A sine wave RF is used between the two plates (in y). A collection of ions are given a small ke in x relatively close to one of the plates. Each ion follows a wave-like trajectory.

Preparation

Load RF Mirror.pa# and refine to 1.0e-7
Remove all PAs
Load RF Mirror.iob into View
Click the Fly'm button

Discussion

The ions are flown 1/2 cycle (by default) and then killed. The expected peak to peak dy (in mm) using a formal integration of the expected forces is compared to the dy obtained from the simulation.

If you want the ions to keep on flying set the adjustable variable:
Long_Simulation_if_1 to the value of 1.

Experiment

Note that higher values of trajectory quality improve the higher mass dy accuracies. The issue here is that these ions have small wave sizes in relation to array grid intervals. To improve accuracy you need to either use a higher density array or turn up the trajectory quality to shrink the timesteps in trajectory curvature areas.

RF Square Wave Experiment

A square wave RF is used between the two plates (in y). A collection of ions are given a small ke in x relatively close to one of the plates. Each ion follows a wave-like trajectory. The square wave will induce conservation of energy problems unless the switch edge can be detected accurately and automatically.

Preparation

Load RF Square Wave.pa# and refine to 1.0e-7
Remove all Pas
Load RF Square Wave.iob into View
Click the Fly'm button

Discussion

The ions are flown 1/2 cycle (by default) and then killed. The expected peak to peak dy (in mm) using a formal integration of the expected forces is compared to the dy obtained from the simulation.

If you want the ions to keep on flying set the adjustable variable:
Long_Simulation_if_1 to the value of 1.

Experiment

Note that 0 or negative value of trajectory quality cause considerable errors because of their fixed time step nature.

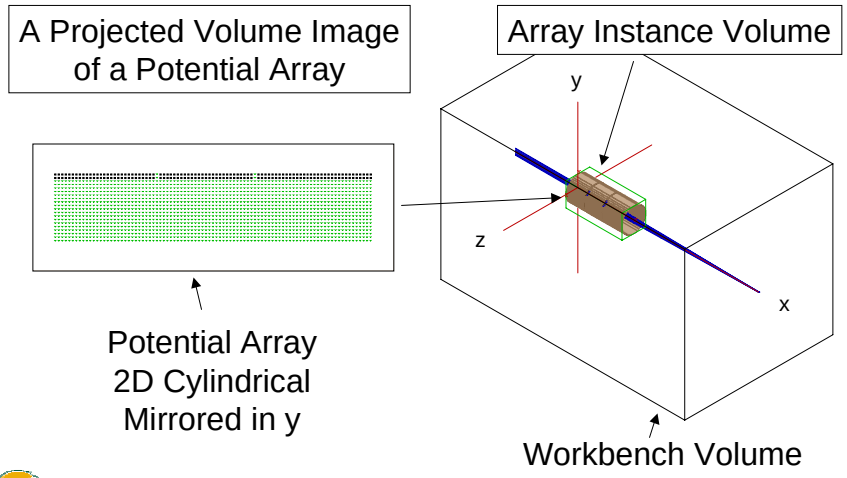
Note that higher values of trajectory quality improve the higher mass dy accuracies. The issue here is that these ions have small wave sizes in relation to array grid intervals. To improve accuracy you need to either use a higher density array or turn up the trajectory quality to shrink the timesteps in trajectory curvature areas.

Turn on data recording at each time step to verify that SIMION is indeed catching the edge of the RF square wave.

3.b Array Instances

- What they are
- How to define them
- How they interact
- How to control their priority

What is an Array Instance?



Why are Instances Useful?

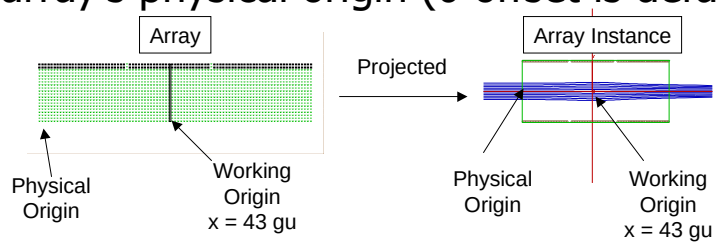
- Links PAs into the virtual workbench world
- Electrostatic and magnetic fields in same volume
- Allows higher density arrays in certain regions (e.g. sources).
- Supports simulation of beam lines with separated optics stages.

How Instances are Defined

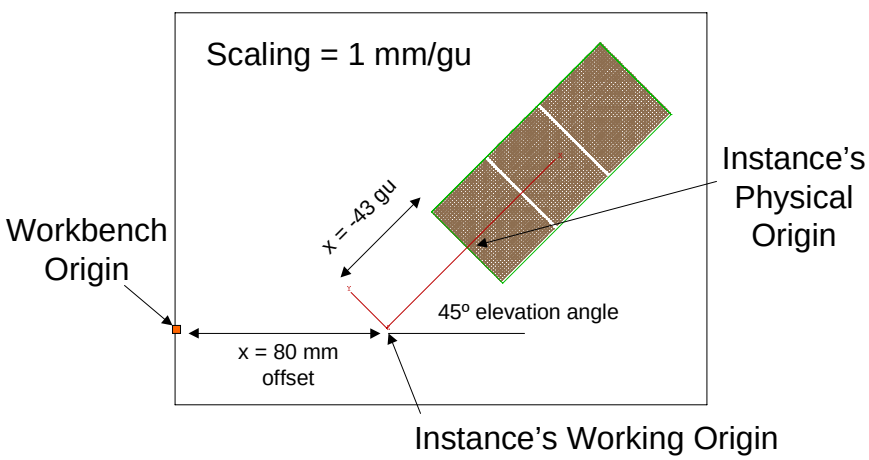
- Working Origin
- Offset of Working Origin from WB Origin
- Scaling in mm/gu
- Orientation Angles from Working Origin in
  - Azimuth
  - Elevation
  - Rotation

Working Origin

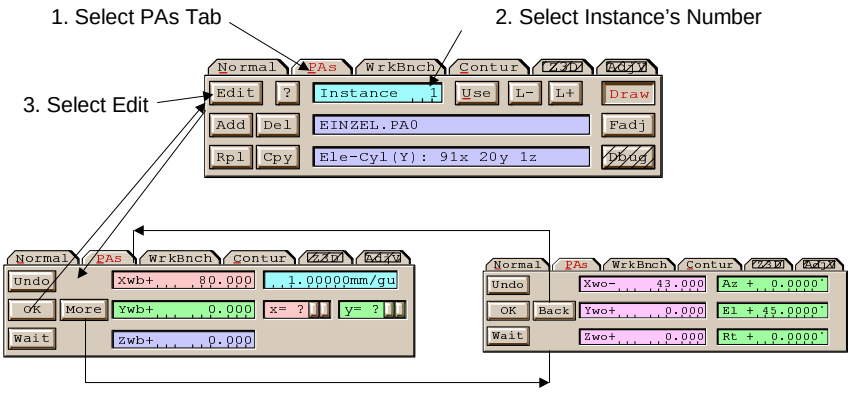
- Working Origin is the reference point for all positioning, scaling and orientations
- It is defined in terms of an offset from the array's physical origin (0 offset is default)



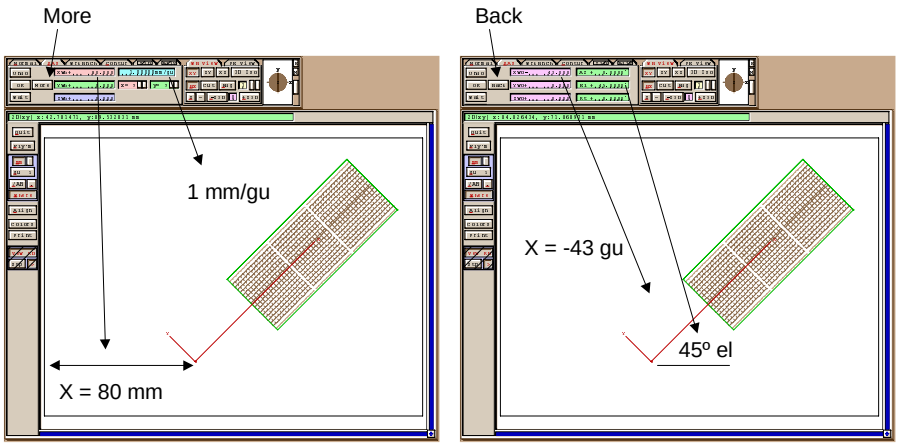
Instance Positioning, Scaling, and Orientation



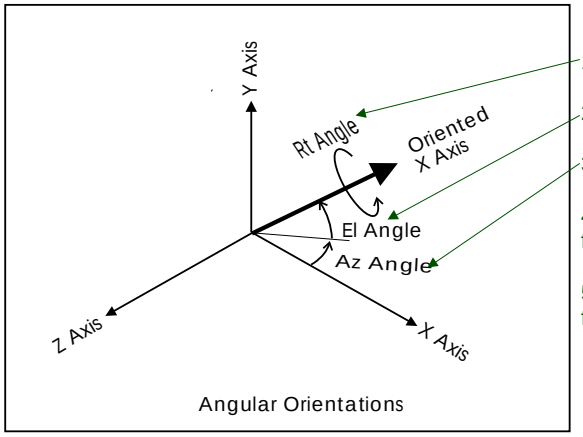
Editing an Instance



SIMION Instance Editing

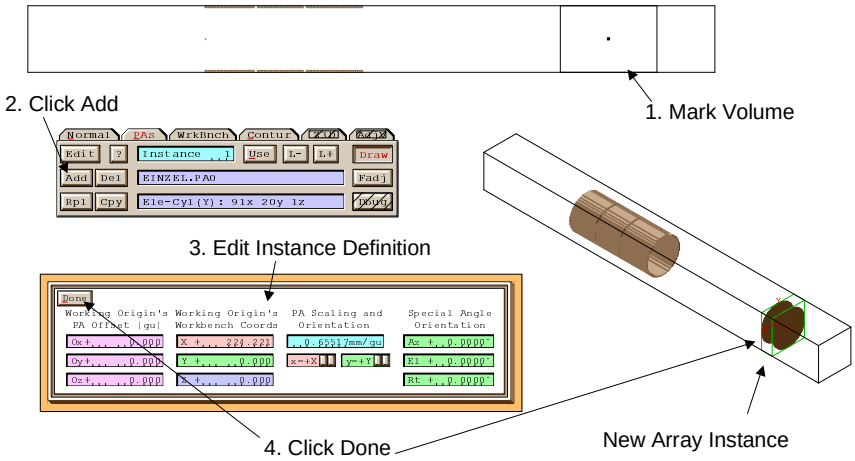


Order of Orientations and displacements



1. Rt angle
2. El angle
3. Az angle
4. Working Origin offset from array physical origin
5. Working Origin offset from workbench origin

Adding an Instance

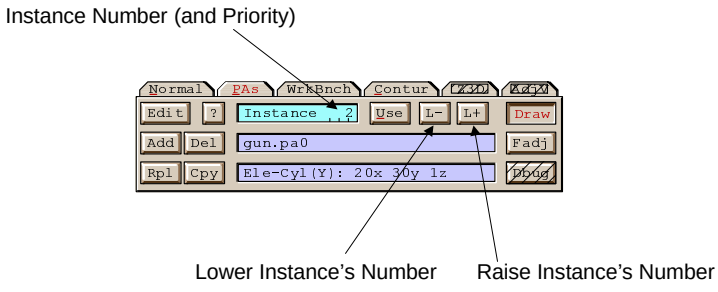


Instance Rules

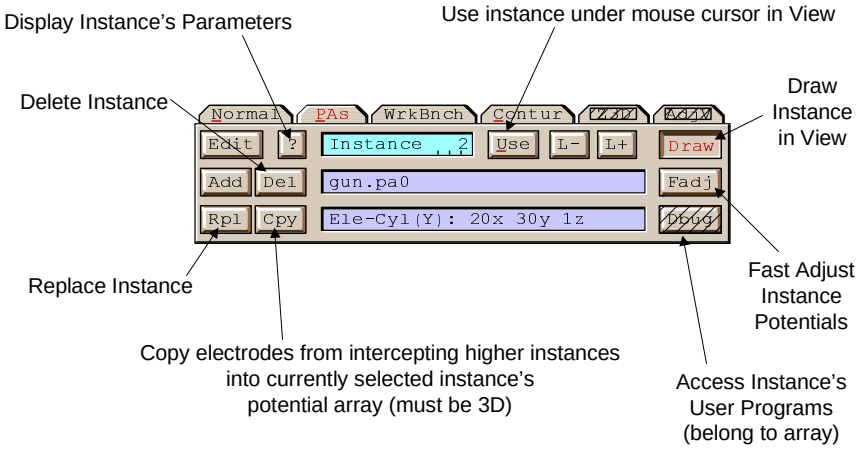
- Instance Fields are Separate
  - Each instance is an isolated island.
  - Only linear acceleration fields are assumed when ions fly between instances of differing electrostatic potentials (won't work with RF).
  - Instance Fields that overlap do NOT merge.
  - Overlapping instance volumes defer to the highest priority instance.

Instance Priority

- The instance with the highest number has the highest priority



Other Instance Options



Array Instance Labs

Introduction

These labs are designed to demonstrate important aspects of array instances.

Location: C:\Advance SIMION Class\03. Trajectories and Instances\Instance Lab

Orienting an array instance

This lab selectively orients an array instance of an einzel lens.

1. Load the **Einzel.iob** file into View.

2. Click the **3D Iso** button for an isometric view.
   Turn up display quality to 6.

3. Click the **PAs** tab.

4. Click the **?** button and examine the instance's parameters.
   Move the cursor to the upper right corner of the data display.
   Notice that the cursor switches to a double headed arrow.
   Now drag (mouse button depressed) this corner to enlarge the data display to see all parameters.
   Click the **Print** button and print a copy of these parameters.
   Notice that all instance parameters have their default values (this will change).
   Click **Vew RD** button to turn off the data display.

5. Click **Edit** and then the **More** buttons to gain access to instance edit screen number two.
   (It is suggested that you have **where** active in **mm abs** and switch views to examine changes)
   Set the Rt angle to +90 degrees (watch what happens in view for the next steps).
   Set the El angle to +45 degrees
   Set the Az angle to +30 degrees
   Set Xwo (x working origin) to –100 gu (notice that the working origin is still at the WB origin)
   Click the **Back** button to access instance edit screen number one.
   Set Ywb (y workbench offset) to 100 mm (working origin is now 100mm above the WB origin)

6. Does what you observe agree with the order of transformations described in the lecture?
   Now try orienting the instance yourself.
   Notice that SIMION always expands the workbench volume to include all instances.
   However it doesn't automatically shrink the workbench volume.
   To minimize this volume, click the **Wrkbnch** tab and click the **Min** button.
   While you're there add 1,000 mm to Xmax to expand the workspace beyond the array.

Adding a new array instance

This lab adds a second separately controllable einzel instance into the workbench volume.

1. Click the Remove All PAs from Ram button on the main menu screen to remove our prior mess.

2. Load the **Einzel.iob** file into View.

3. Now mark a region to the right of the current einzel instance for locating the new einzel instance.

4. Click the **PAs** tab and the **Add** button.
   Select the file **einzel.pb0** and depress **both** mouse buttons.
   Click the **Done** button to accept your fate.
   The new instance is number 2.
   Edit it if it needs to be moved or sized a bit.

5. Now try flying ions through both instances.
   Make sure instance 2 is centered in y and z.
   Change the scaling of instance 2 to 0.5 mm/gu.
   Fly the ions.
   Try a slight miss-alignment (offset or angular).

6. Change the potentials of each instance separately (**PAs** tab selected):
   Select the instance number (e.g. 2).
   Click the Fadj button and adjust the instance's potential.
   Fly the ions.
   Repeat the process with the other instance (e.g. 1).
   Run one instance as an acceleration and the other as a deceleration einzel.

Superimposed instances

This lab uses two superimposed instances to demonstrate instance priority and the isolation of fields within an instance.

1. Remove all PAs from ram.

2. Load the file **mirror.iob** into view and click the **Fly'm** button.
   An instance of the einzel lens has been superimposed on a tube instance with a field free internal volume and end ideal grids and electrodes that can be used as reflectors.

3. Notice that the ions that go through the einzel are focused.
   However, ions that fly near the outside of the einzel are not affected. Why? They should be.
   What does this tell you about superimposed instances.

4. Now click the PAs tab, select instance 2 (einzel.pa0), click the L- button and the Yes button.
   Re-fly the ions.
   What do you see? Explain what is causing this change. Why is this knowledge important?

5. Use data recording to show: ion number and instance number when instances change (verbose). Compare the results when the einzel.pa0 instance is 1 or 2.
   Explain. Would this be a good troubleshooting method for instance priority?

6. For a little fun, set the left and right reflector voltages to 1000 V and fly all ions grouped (data recording off).

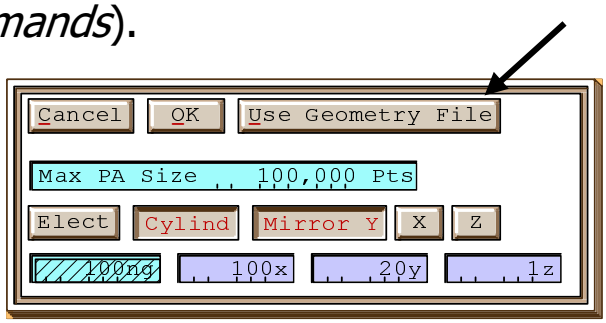
4. Geometry Files

- A geometry file is an ASCII file with the .GEM filename extension that contains the electrode/pole definitions for a potential array.
- It also may contain the definition of the potential array:

pa_define(100,20)

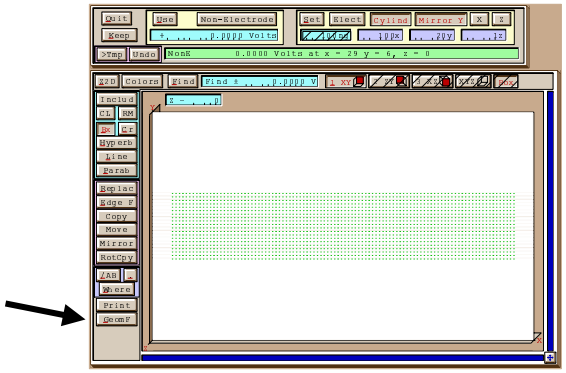
SIMION Functions that Use Geometry Files

- The New function via the Use Geometry File button (*makes use of pa_define commands*).



SIMION Functions that Use Geometry Files

- The Modify function via the GeomF button (*ignores pa_define commands for an existing array*).



How Geometry Files Work

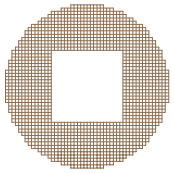
- The geometry files contain a collection of fill commands.
- Fill commands define complex areas/volumes using:
  - inclusion/exclusion commands (e.g. within and notin),
  - orientation commands (e.g. locate), and
  - basic shape commands (e.g. circle).

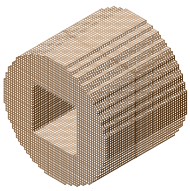
How Geometry Files Work

- Each successive fill command has increasing priority. Later fill commands override earlier fill commands.
- SIMION uses a geometry compiler to create an ordered fill search tree.
- Each point within the array is checked to see if it is affected by a fill command (from last fill toward first fill).

Example

```
e(1)                                        ; electrode of one volt
   {
        fill                                ; fill command
           {
              within{     circle(0,0,5)}    ; within a circle at 0,0  r = 5
                notin{    box(0,0,2,2)}     ; notin a box with corners of
        }                                   ; x,y:   0,0 and 2,2
    }
```





Geometry File Language

- The Language is nested - commands fit inside each other.

```
e(1)                                        ; electrode of one volt
   {
        fill                                ; fill command
           {
              within{     circle(0,0,5)}    ; within a circle at 0,0  r = 5
                notin{    box(0,0,2,2)}     ; notin a box with corners of
        }                                   ; x,y:   0,0 and 2,2
    }
```


Geometry File Language

```
e(1)                                        ; electrode of one volt
   {
        fill                                ; fill command
           {
              within{     circle(0,0,5)}    ; within a circle at 0,0  r = 5
                notin{    box(0,0,2,2)}     ; notin a box with corners of
           }                                ; x,y:   0,0 and 2,2
    }
```

- Each command has its {scope}: The region of commands that are under its influence.
- Commands have (parameters) and/or scope.
- Inline comments begin with a semicolon.

Geometry File Nesting Rules

Base Nesting Level

Fill Nesting Level

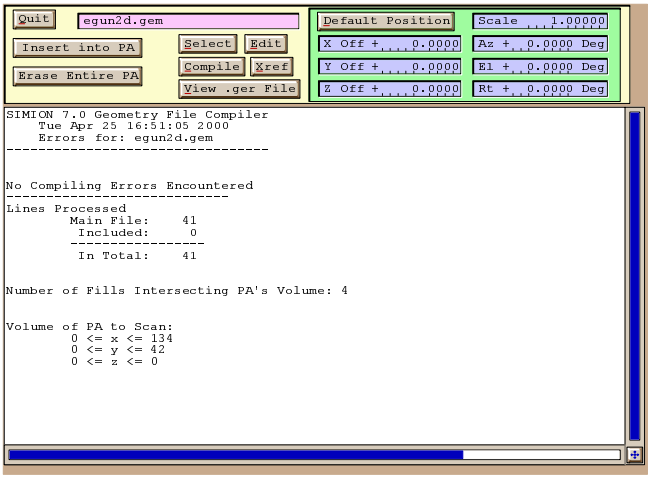
Within Nesting Level

```
e(1)                                        ; electrode of one volt
  {
        fill                                ; fill command
            {
              within{      circle(0,0,5)}   ; within a circle at 0,0  r = 5
                notin{     box(0,0,2,2)}    ; notin a box with corners of
            }                               ; x,y:   0,0 and 2,2
  }
```


Geometry File Development System

- Accessed from Modify with the GeomF button.
- Geometry files are edited, compiled and debugged.
- Potential array can be erased.
- Geometry definitions can be inserted into the potential array.
- Provides full manual control over translation, scaling and orientation via locate parameter panels (serves as the outermost Locate command).

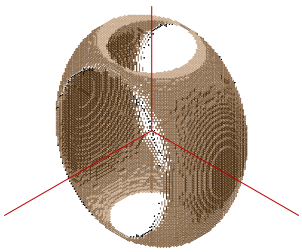
Geometry File Development System



Location, Scaling, Orientation

- Locate commands are used to locate, scale, and orient anything at any level. <u>They even can be nested within themselves:</u>

```
locate(100)
        {
        fill
                {
                within{sphere(0,0,0,50,60,40)}
                notin{sphere(0,0,0,45,55,45)}
                locate(0,0,0,1,0,0,-90)
                        {
                        notin{circle(0,0,10,5)}
                        }
                }
        }
```



Geometry File Lab

- Modifying the Trap Geometry File into a Stretched Trap
- Modifying the Trap Geometry File to Include End Cap Apertures

4.0 Introduction to Geometry Files - Lab

Files: C:\Advanced SIMION Class\0.4 Geometry Files Introduction\Lab

Modifying the Trap Geometry File into a Stretched Trap

The contents of the trap geometry file above will be modified to define a Finnigan stretched trap which will be saved as the STRETCH.GEM geometry file

No After Installation Files Need to be Created

1. Remove all PAs From RAM
2. Use New to load TRAP.GEM from directory:
   C:\Advanced SIMION Class\4.0 Geometry Files Introduction\Lab
3. Examine the array with Modify.
5. Click the GeomF button to access the geometry file development system.

6. Click the Edit button to edit the geometry file with EDY.
7. Rename the File STRETCH.GEM
   EDY keystrokes: **Esc N STRETCH.GEM** and press **Enter**
8. Now Let's save the file, exit EDY and Select and use this file as the geometry file:
   Esc Q Y and you're back in SIMION.
9. Click the Select button and click both mouse buttons on STRETCH.GEM to make it the currently selected .GEM file.
10. Use the Erase Entire PA button and then the Insert into PA button to verify that the geometry file still makes the array OK.

11. Now the task is to shift both end caps 0.76 mm away from the trap center to create a stretched trap. The nominal scaling of this potential array is 0.1 mm/grid unit. Thus we need to shift the left end cap (and its contents) 7.6 grid units to the left and the right end cap 7.6 grid units to its right.

    The best strategy is to insert **locate** commands that have the proper shifts and enclose the proper electrode definitions. This will result in the caps being shifted properly.

    Go for it!

12. Assuming you've given it a good college try and failed. Use the Select button and click both mouse buttons on the STRETCH1.GEM file. Insert the geometry from this file. It works! Look at the file with EDY to see how it was done.

Files: C:\Advanced SIMION Class\0.4 Geometry Files Introduction\Lab

Modifying the Trap Geometry File to Include End Cap Apertures

The contents of the trap geometry file will be modified to define apertures in the end caps and then saved as the HOLE.GEM geometry file.

No After Installation Files Need to be Created

1. Remove all PAs From RAM
2. Use New to load TRAP.GEM from directory:
   C:\Advanced SIMION Class\4.0 Geometry Files Introduction\Lab
   Look at it with Modify.

3. Click the GeomF button to access the geometry file development system.

4. Click the Edit button to edit the geometry file with EDY.
5. Rename the File HOLE.GEM
   EDY keystrokes: **Esc N HOLE.GEM** and press **Enter**
6. Now let's Save the file and exit EDY: **Esc Q Y** and you're back in SIMION.
7. Click the Select button and click both mouse buttons on HOLE.GEM to make it the currently selected .GEM file.
8. Use the Erase Entire PA button and then the Insert into PA button to verify that the geometry file still makes the array OK.

9. Now the task is to modify the geometry file to put a 2 mm hole through both end caps. The electrodes are all defined in units of mm. Hint the notin family of commands can be very useful for this. Notin_inside works perfectly for this assignment. A simple notin does not. Why?

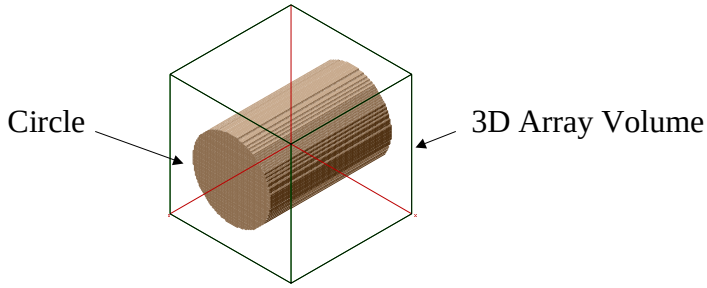
   You might explore use of both the box and circle commands to make these apertures.

   Go for it!

10. Assuming you've given it a good college try and failed. Use the Select button and click both mouse buttons on the HOLE1.GEM file. Insert the geometry from this file. It works! Look at the file with EDY to see how it was done.

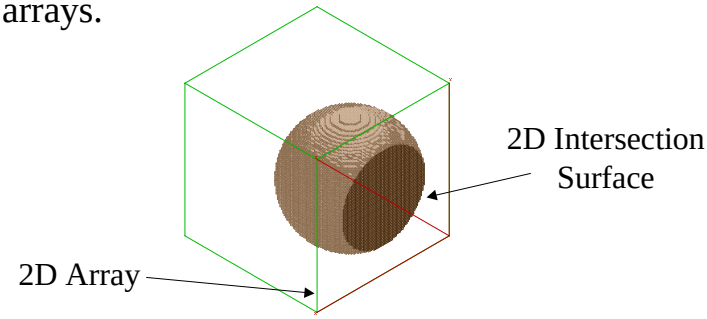
5. 2D and 3D Projection

- All 2D shape commands (e.g. box and circle) are defined in x and y and have a ± **10^6 extent in the z** direction when applied to 3D arrays.



2D and 3D Projection

- All 3D shape commands (e.g. sphere) have their 2D intersection area (z = 0) when applied to 2D arrays.



5. 2D and 3D Labs

- Explore scaling and locating
- Project a 2D geometry file into a 2D array
- Project a 2D geometry file into a 3D array
- Modify a geometry file for a 3D array

5.0 Geometry Files - Projection and Scaling of 2D and 3D Geometry files

Files: C:\Advanced SIMION Class\05. 2D and 3D Projections\2d3d lab

Inserting a Geometry File into Different Sized Arrays

The contents of a geometry file can be scaled to fit within arrays of different sizes. This lab shows how this is done

No After Installation Files Need to be Created

1. Remove all PAs From RAM
2. Use New to load intro01.gem from directory:
   Files: C:\Advanced SIMION Class\05. 2D and 3D Projections\2d3d lab
3. Examine the array with View and then Modify.
4. Click the GeomF button to access the geometry file development system.
6. Change the Scale panel from 1.0 (default) to 0.5 to re-scale the geometry definitions to the smaller potential array.
7. Click the Erase Entire PA button then click the Insert into PA button. You should now be looking at the halved scale potential array.
8. Look at the object in View.
9. Follow steps 4-8, changing the scale factor and using the x, y, and z panels to move the object around in the array. What happens if you displace it by 20 units in x?

Inserting a Geometry File into 2D and 3D arrays

The contents of a geometry file can be projected into 2D or 3D arrays, fill definitions must be consistent. This lab demonstrates typical issues.

1. Exit Modify and Remove all PAs from ram.
2. Use New to load intro01.gem from directory C:\….\2D3D lab
3. Go into Modify to access the geometry file development system screen.
4. Use Edit to access Edy, delete the " ; " comment from line 2 and insert a " ; " at the beginning of line 1.
5. Rename the file 3Dcircle.gem,(keystrokes: **Esc S 3Dcircle.gem** and press **Enter**).
6. Alt-tab back to the geometry development screen, choose quit, then quit out of Modify, and remove all PAs from Ram.

7. Use New to load the geometry file 3Dcircle.GEM.
8. Go into View. Now what do you see? Why?
9. Go into Modify, then to the geometry file development screen.
10. Use Edit to get to the 3Dcircle.gem file
11. Replace the circle(0,0, 50) command with sphere(0,0,0,50) then rename and save the file as 3Dsphere.GEM (keystrokes: **Esc N 3Dsphere.GEM, Enter, Esc S Enter**), Alt-tab back to geometry file development screen, use select to choose 3Dsphere.gem, click the Erase Entire PA button then click the Insert into PA button.
12. Look at the array in Modify (XYZ view). Now look at it in View.

13. use the locate panels to offset the object (try x=20). Click the Erase Entire PA button then click the Insert into PA button. Go into View. What happened? Why?

14. Exit View and remove all PAs from Ram.
15. Using New load the geometry file 3D.GEM. Look at the array in Modify, how is it different from what you saw in the 3Dsphere.gem file? Alt-tab to Edy and look at the 3D.GEM file. (Esc L 3D.gem Enter).
16. In the Geometry file development screen adjust the x offset by 20. Erase entire PA and Insert into PA. Look at the result in Modify and then in View. How is this different from what you saw in step 13? Why?

5.0 Geometry Files: Creating a geometry file from scratch

Files: C:\Advanced SIMION Class\05. 2D and 3D Projections\Einzel lab

Creating a geometry file for an EINZEL lens

A geometry file for an einzel lens will be created.

No After Installation Files Need to be Created

1. Remove all PAs From RAM
2. Load the EINZEL.PA# array with Load
3. Look at the electrode definitions with Modify.
4. Write down, on paper, the image of a geometry file that would create the einzel array. Try to include the proper PA_Define if you can. Seek the assistance of an instructor as necessary.
5. Remove all PAs from Ram. Use New with the Use Geometry files button to access the file manager. Click the Edit button to access EDY.
6. Name the file EINZEL.GEM with EDY (keystrokes: **Esc N EINZEL.GEM** then **ENTER)**
7. Enter the contents into the file from the keyboard.
8. Now Let's save the file (Esc, S, Enter), alt tab back to the Geometry definition screen and Select and use this file as the geometry file (Click both buttons on the EINZEL.GEM file's button).
10. If all is OK the array will be created. If not, you will be informed.
11. In any event, enter modify and see what you created.
12. Click the GeomF button to begin to change or fix your geometry definitions

13. If you want to see a successful example, look at the EINZEL1.GEM file.

6.0 Geometry File Tricks

Important Concepts

- Geometry creation is array point centered.
  - The array is scanned a point at a time.
  - Each point is tested to find the LAST fill in the list of fills that would change its value (if any).
  - If the point is within a fill's volume, it is changed to the fill's value (e.g. E(0)).
  - The process is repeated until all array points have been tested and changed as required.

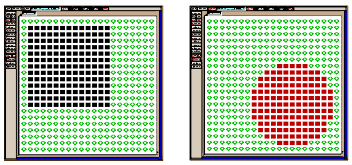
Important Concepts

- Fills tests are ALWAYS made in the reverse list order. The last fill in the list ALWAYS has the highest priority.
- Fill{inside{circle()}}
  - Fill{inside{box()}}
  - Fill{inside{sphere()}}
- The sphere fill has highest priority if more than one fill changes the array point's value.

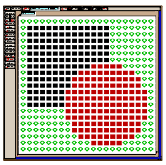
Important Concepts

- ALL fill definitions are ALWAYS projected to their final locations in array coordinates <u>BEFORE</u> the actual decision to fill is made.
  - Any overlap conflicts are resolved at the final array point locations.

As defined:

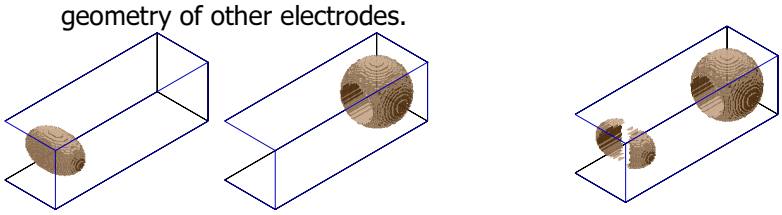


As located:



Important Concepts

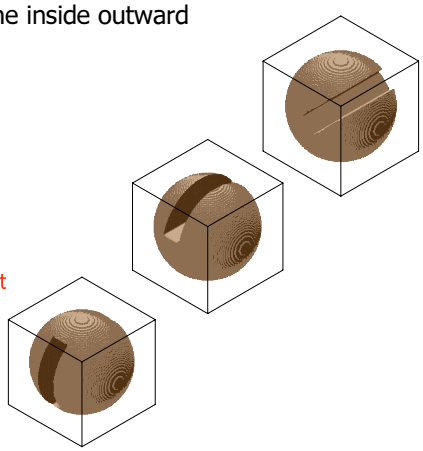
- ALL fill definitions are ALWAYS projected to their final locations in array coordinates BEFORE the actual decision to fill is made.
  - Unbounded volume fills can accidentally change the geometry of other electrodes.



Important Concepts

Nested Languages work from the inside outward

```
Locate(50,50,50,10)              ;Fifth
{
fill{within{sphere(0,0,0,5)}     ;Fourth
    Locate(0,0,0,1,0,-45)        ;Third
      {
      Locate(0,5)                ;Second
        {
        notin{cent_box(0,0,2,8)} ;First
        }
      }
    }
}
```



Important Concepts

Nested languages have nesting level rules:

```
E(1)                                  ;Base nesting level
{
  Locate(50,50,50,10)                 ;Locates work at any nesting level
  {
  fill{                               ;Fill nesting level
      within{                         ;Within nesting level
              sphere(0,0,0,5)         ;Volume function definition
              }
      Locate(0,0,0,1,0,-45)           ;Locates work at any nesting level
        {
        Locate(0,5)                   ;Locates can be nested to define translation order
          {
          notin{cent_box(0,0,2,8)}    ;Within nesting level
          }
        }
      }
  }
}
```

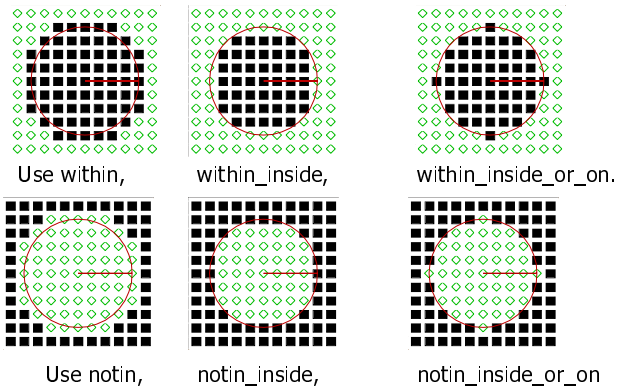

Important Concepts

Boolean language rules:

```
Fill
  {
  IF within { circle() AND box() AND polyline() } OR
     within_inside { testa AND testb } OR
     within_inside_or_on { testc }

  BUT notin { sphere() AND box3d() } OR
        notin_inside { testd AND teste } OR
        notin_inside_or_on { testf }
  }
```

Important Concepts

- Use within and notin alternatives for more precision



The Divide & Conquer Approach

- Define each entity (electrode) separately
  - Define entity in physical coords (e.g. in, mm)
  - Define the entity at the origin to simplify effort.
  - Use body and/or machining centered methods.
  - Pick the strategy that is easiest for YOU to understand.
- Then position each entity at its physical location
- Finally, scale and position the entire assembly into target array's area or volume.

The Divide & Conquer Approach

```
Locate(position and scale into array area or volume)
        {
        Locate(position entity 1 in physical volume)
                {
                Define entity 1 in physical units at origin
                }
         Locate(position entity 2 in physical volume)
                {
                Define entity 2 in physical units at origin
                }
        ; … and so on
        }
```


The Secret in Real Estate is:

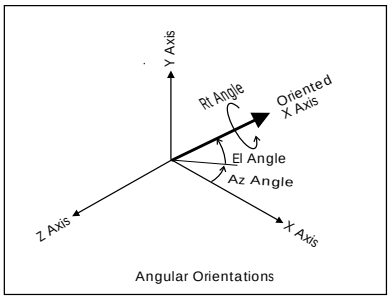
- Location
- Location
- Location

The Secret in Geometry Files is:

- Locates
- Locates
- Locates

The Locate Command

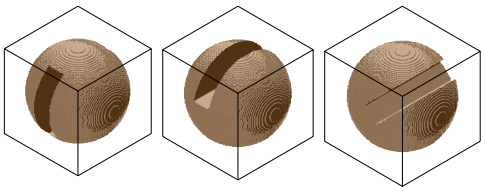
- Locate(x, y, z, scale, az, el, rt){}
  - Locate(0,0,0,1,0,0,0) defaults
- Order Transforms are Applied
  - rt
  - el
  - az
  - scale
  - x,y,z



Nested Locate Commands

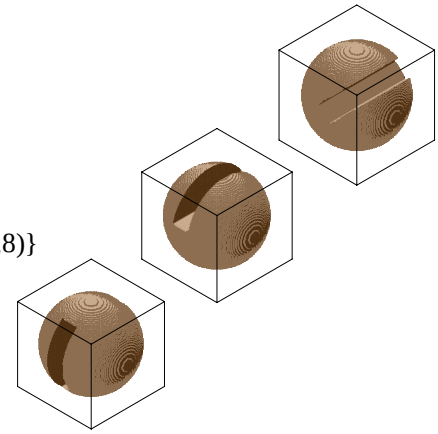
- No performance costs (locate cmds are combined)
- Insures the order that transforms are applied
- Supports progressive definitions

```
Locate(50,50,50,10){
fill{within{sphere(0,0,0,5)}
Locate(0,0,0,1,0,-45){
Locate(0,5){
notin{cent_box(0,0,2,8)}
}}}}
```



Indent Nested Commands

```
Locate(50,50,50,10)
{
fill{within{sphere(0,0,0,5)}
    Locate(0,0,0,1,0,-45)
      {
      Locate(0,5)
        {
        notin{cent_box(0,0,2,8)}
        }
      }
    }
}
```

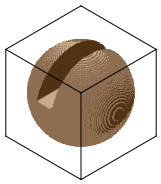


Body Verses Machining Oriented Definitions

- You can move and orient either the entity (body), the cuts (machining), or both (can be confusing).
  - The body oriented approach moves the entity and assumes that the machine is fixed.
  - The machine oriented approach moves the machining and assumes the body is fixed.
  - Moving both the entity and machine can at times simplify the process, but it can also be confusing.
- Select the approach that is easiest for YOU.

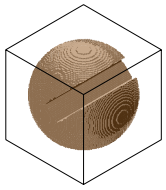
Machining Oriented Definitions

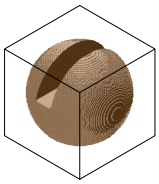
```
Locate(50,50,50,10)                ;5:  Scale and center in array
{
fill{within{sphere(0,0,0,5)}       ;4:  Define sphere at origin
   Locate(0,0,0,1,0,-45)           ;3:  Rotate slot 45 degrees in el
     {
     Locate(0,5)                   ;2:  Shift slot up 5
       {
       notin{cent_box(0,0,2,8)}    ;1:  Define slot at origin
       }
     }
   }
}
```



Body Oriented Definitions

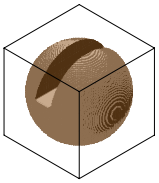
```
Locate(50,50,50,10)                ;6:  Scale and center in array
{
fill{
   Locate(0,0,0,1,0,-45)           ;5:  Rotate sphere to slot angle
     {
      Locate(0,5)                  ;4:  Shift sphere to origin
        {
         notin{cent_box(0,0,2,8)}  ;3:  Drill slot at origin
         Locate(0,-5)              ;2:  Shift sphere down 5
           {
            within{sphere(0,0,0,5)} ;1:  Define sphere at origin
           }
        }
     }
   }
}
```





Combined Approach Definition

```
Locate(50,50,50,10)                ;4:  Scale and center in array
{
fill{
   Locate(0,0,0,1,0,-45)           ;3:  Rotate sphere and slot to angle
     {
      notin{cent_box(0,5,2,8)}     ;2:  Define slot at y = 5
      within{sphere(0,0,0,5)}      ;1:  Define sphere at origin
     }
   }
}
```



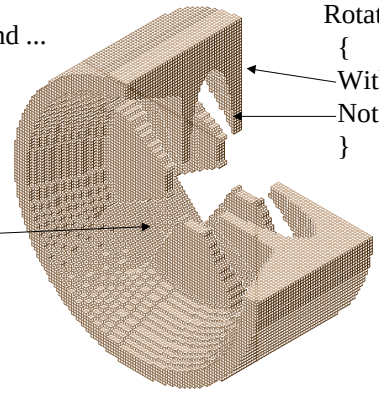
Using Notins or Erasing Fills?

- With normal Fill commands it is best to use Notins to limit or remove material.
  - Notins can act exactly like a normal erasing fill.
  - 2D commands (e.g. circle) can be used in a Notin without fear of drilling infinite holes.
- With Rotate_Fills any non-cylindrically symmetric removal will normally required a subsequent erasing fill.
  - Erasing fills (e.g. n(0)) must be volume limited.
  - 2D commands are dangerous (unlimited volume).

Using Notins and Erasing Fills with a Rotate_Fill

Then swing 90° az and ...

```
N(0)
 {
  Fill
   {
    Within{Polyline()
           Box3d()
          }
   }
  }
```



```
Rotate Fill(90)
 {
 Within{Polyline()}
 Notin{Circle()}
 }
```

The Role of Include Files

- Pros
  - Useful for individual electrode definition.
  - Handy if electrode used in multiple places in same volume or in different volume (e.g. normal and exploded views).
- Cons
  - Hides electrode definitions in different files (can be confusing).
  - Care must be used to insure maximum utility.

Recommended Method for Using Include Files

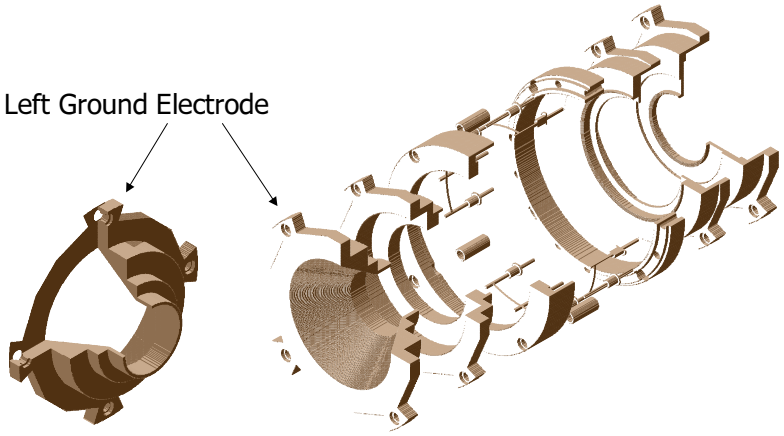
Calling File:

```
locate() ; position in physical volume
{
e(5)            ; set fill type
  {             ; call include file
   Include(draw out.gem)
  }
}
```

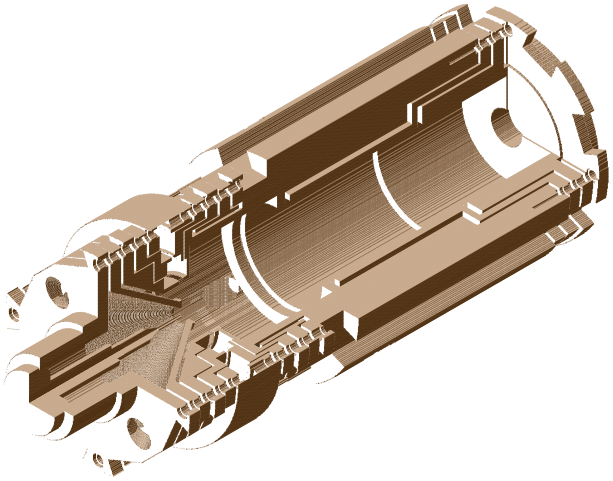
Include File: draw out.gem

```
locate()        ; place at origin
{               ; fill (within-notin)
electrode fill definitions
n(0)            ; optional erases
  {
  optional erasing fill(s)
  }
}
```

Include File Examples



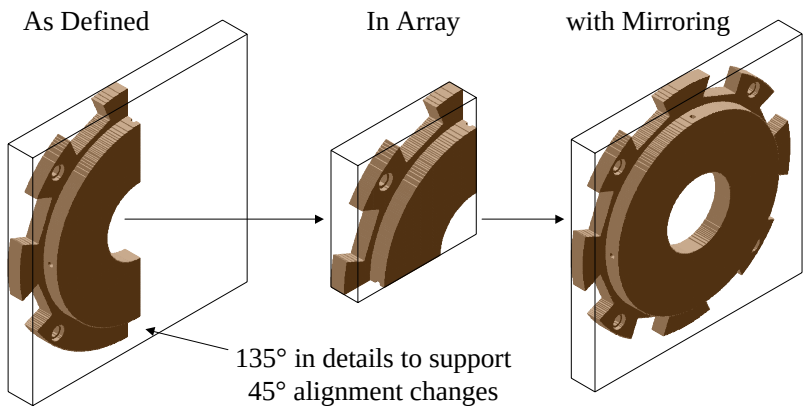
7. A Complex Example



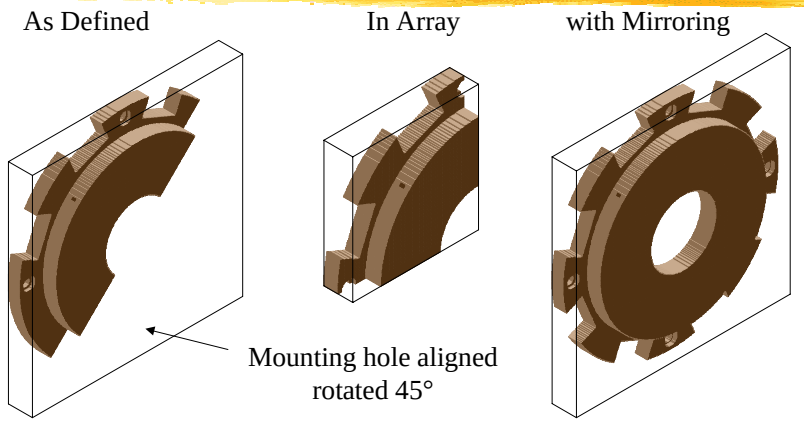
Adopted Strategy

- Do initial designs in 2D with Modify <u>then</u> 3D GEM.
- Symmetry allowed y and z mirroring so that 1/4 sized array can be used.
- Snap design dimensions to 0.010" to match maximum practical array size of 40,000,000 points (400 MB).
- Define each lens element separately at the origin and then move it to its physical location in the assembly.
- Use rotate_fills because of high cylindrical symmetry and then drill holes and slots with erasing fills.
- Design GEM for 2D/3D arrays and flexible orientation.

Buildup of Exit Ground Electrode

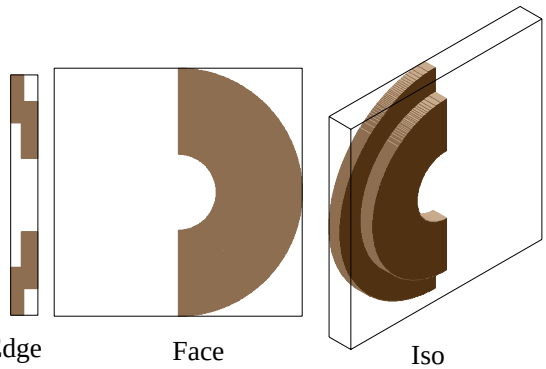


Mounting Hole Alignment



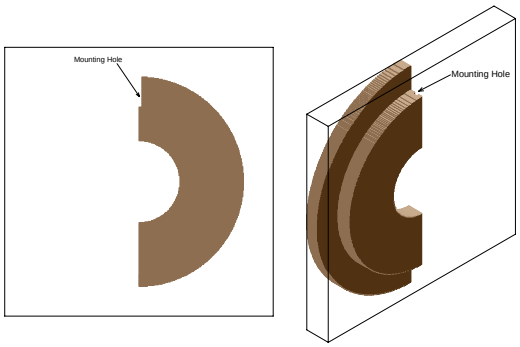
1st -- Create Electrode Blank

```
Locate(0,0,0,1,90)
  {
  rotate_fill(180)
    {
    within{polyline(x,y,
                    x,y,
                    …)
          }
    }
  }
```



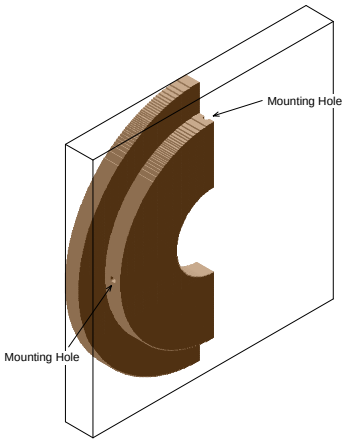
Drill 1st Quad Mounting Hole

```
N(0)       ; erasing fill
{
fill       ; community fill
  {
  Locate()  ; dummy locate
    {       ; orient drill
    Locate(0,1.4,-.31,1,0,0,-90)
      {
      Within_inside
        {  ; drill hole
        Cylinder(0,0,.001,.04,.4)
        }
      }
    }
  }
}
```



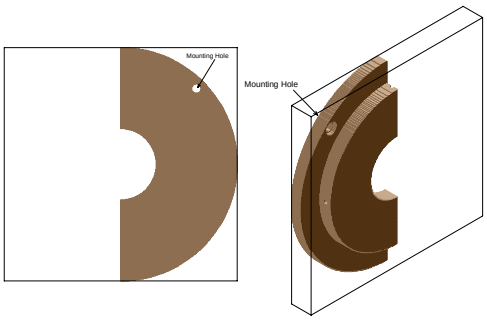
Drill 2nd Quad Mounting Hole

```
(Copy 1st and modify)
            ; swing 90 deg. in el.
Locate(0,0,0,1,0,-90)
    {       ; orient drill
    Locate(0,1.4,-.31,1,0,0,-90)
      {
      Within_inside
        {  ; drill hole
        Cylinder(0,0,.001,.04,.4)
        }
      }
    }
```



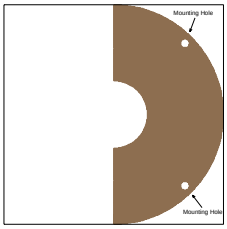
Create 1st Mounting Hole

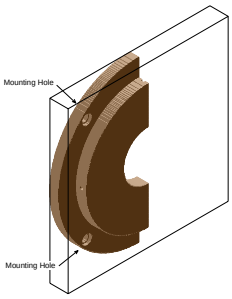
```
            ; swing into position
Locate(0,0,0,1,0,-45)
  {
  locate(0,1.65)      ; mounting circle
    {
    within_inside{cylinder(lft insul)}
    within_inside{cylinder(hole)}
    within_inside{cylinder(rht insul)}
    }
  }
```



Create 2nd Mounting Hole

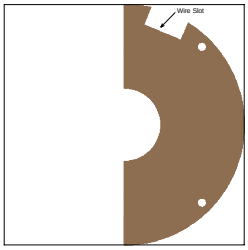
```
(Copy of 1st mounting hole)
            ; swing into position
Locate(0,0,0,1,0,-135)
  {
  locate(0,1.65)      ; mounting circle
    {
    within_inside{cylinder(lft insul)}
    within_inside{cylinder(hole)}
    within_inside{cylinder(rht insul)}
    }
  }
```

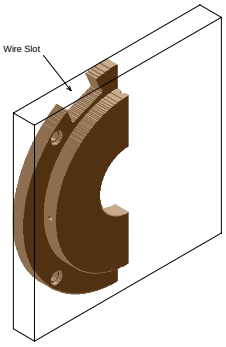




1st Wiring Slot

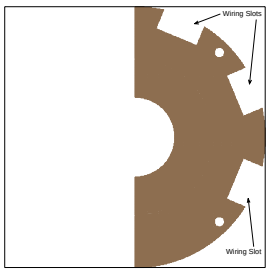
```
            ; swing -22.5 deg in el
Locate(0,0,0,1,0,-22.5)
{
Within_inside
  {         ; define 3D box
            ; on mounting
            ; circle radius
  box3d(-.3,1.5,-.201,.3,2,.001)
  }
}
```

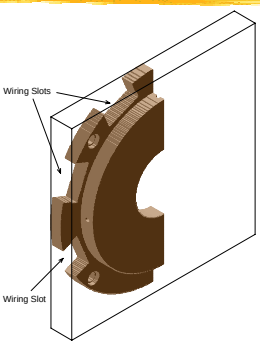




2nd and 3rd Wiring Slots

```
(copy 1st slot twice)
            ; swing -67.5 deg 2nd
Locate(0,0,0,1,0,-67.5)
{
Within_inside
  {         ; define 3D box
            ; on mounting
            ; circle radius
  box3d(-.3,1.5,-.201,.3,2,.001)
  }
}
            ; swing -112.5 deg 3rd
 Locate(0,0,0,1,0,-112.5)
{
(second copy of slot)
}
```





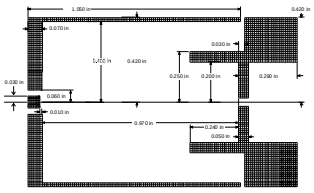
8. Geometry File Lab

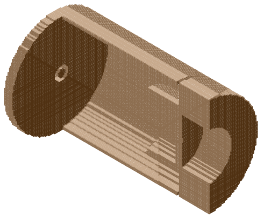
Write several geometry files:

- A 2D cylindrically symmetric design
- A 3D asymmetric design
- Additional components for the speedy

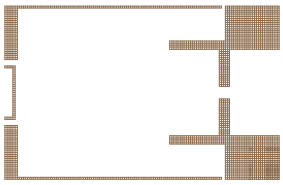
Electron Gun

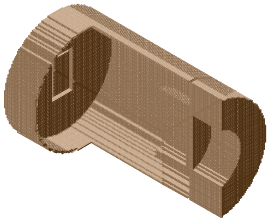
2-D Model with cylindrical source.





3-D Model with rectangular source.





8. Geometry Files – Creating 2D and 3D GEM files

Files: C:\Advanced SIMION Class\08. Creating Geometry File\Lab

Creating geometry files for an electron gun using 2D and 3D definitions

Create a gem file for the electron gun shown in Figure 8.1

No After Installation Files Need to be Created

1. Referring to Figure 8.1, create a gem file that will produce the 2D electron gun. Use real units (inches) in your geometry file and a locate command to scale the system to fit into an array of 140 x 45 x 45 grid units. If you get stuck, refer to the file egun2d.gem. Fly the ions in the file egun2d.fly with the gun set to produce electrons with 70 eV of energy. Tune the lens to focus the beam within the final aperture.

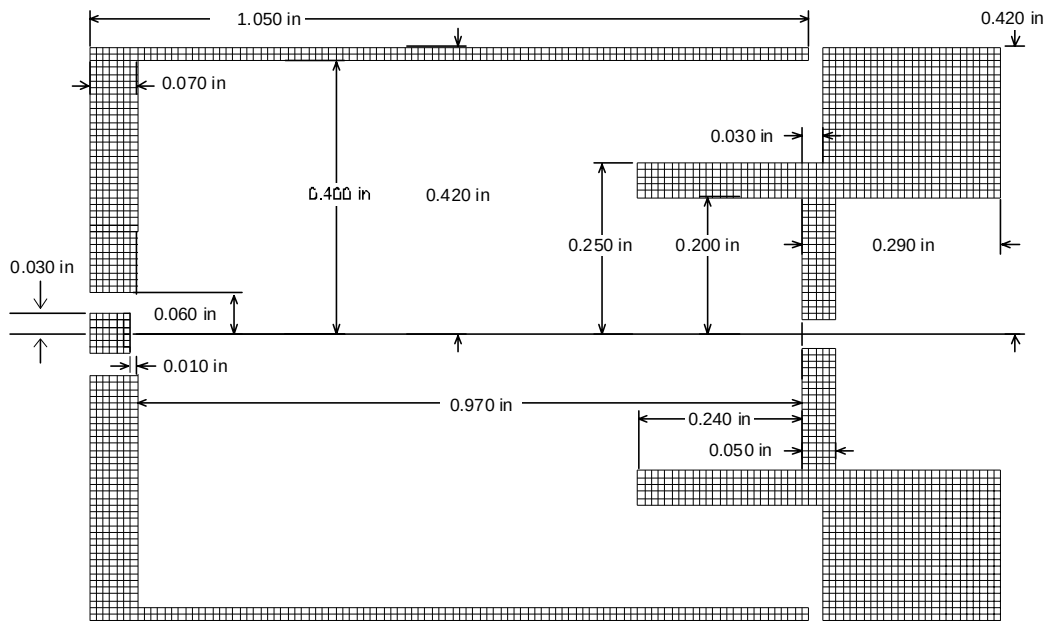


Figure 8.1 2D cylindrically symmetric electron gun.

2. Referring to Figure 8.2 and 8.3, create a gem file that will produce the 3D electron gun. Use real units (inches) in your geometry file and a locate command to scale the system to fit into an array of 140 x 45 x 45 grid units. Fly the ions in the file egun3d1.fly with the gun set to produce electrons with 70 eV of energy. If you get stuck, or to see several different approaches to solving this, refer to the files egun3d1.gem and egun3d2.gem.

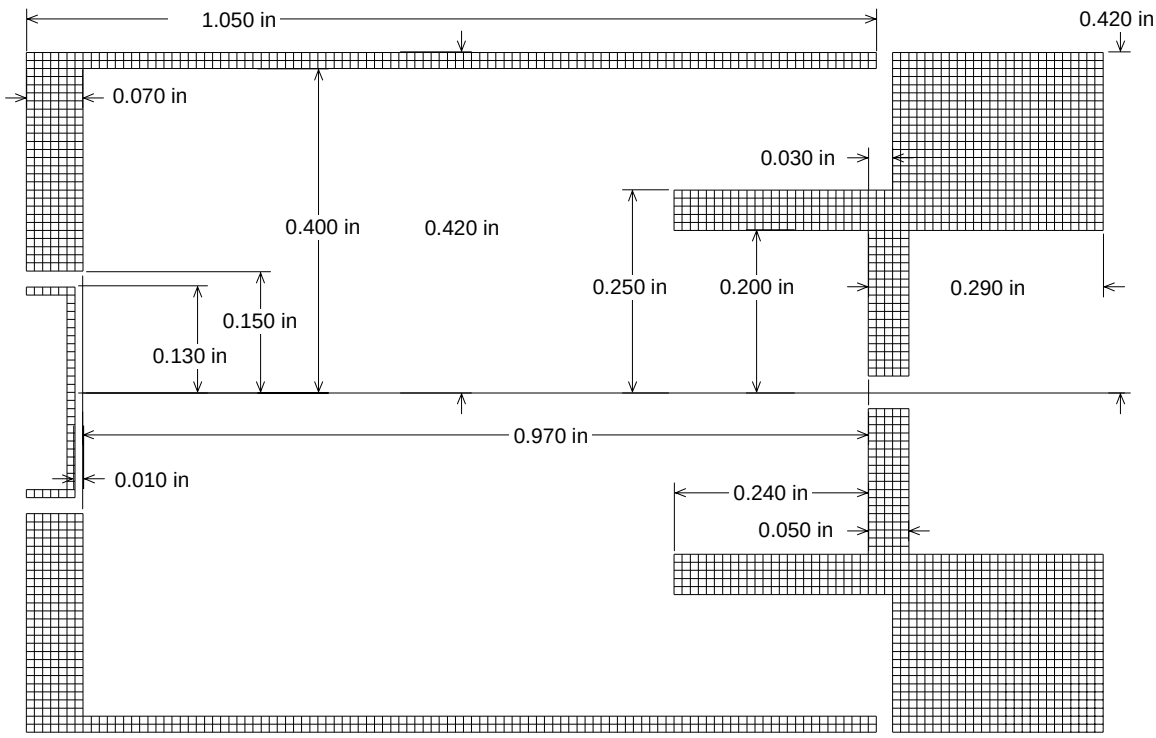


Figure 8.2 3D electron gun, symmetric about the X axis. Filament is 0.012" wide (Z in this view).

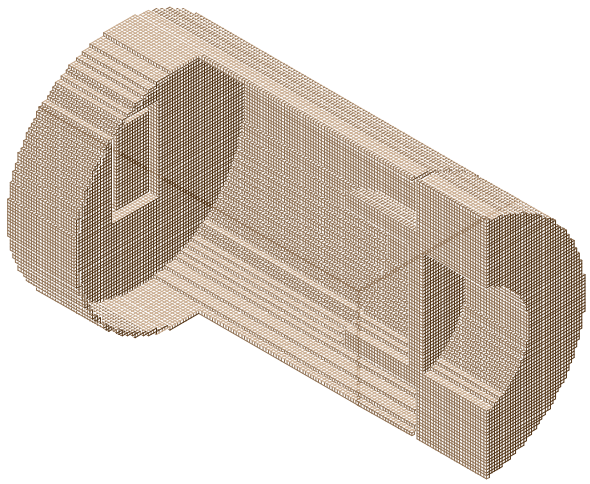


Figure 8.3 Isometric view of 3D electron gun design. Filament is 0.012" wide.

For the Speedy

3. Add a set of X-Y deflectors to the exit of the electron gun, check your design for beam distortion, how can this be minimized?

9.0 What is a User Program File ?

- A user program file is an ASCII file with the .PRG filename extension that contains the user program segments for use with a potential array (.PA or .PA0).
- It shares the name of the potential array it supports. TEST.PRG would automatically be used to support the TEST.PA array.
- User programs are associated with arrays not instances.

How SIMION Utilizes User Programs

- Each time the Fly'm button is clicked SIMION automatically compiles all user program files for any instances that have user programs attached to their associated potential array.
- If Adjustable variables are found, the user will be given a screen to view and change their initial values.
- User program segments are called only when ions are flying within the array that the user program segments support.

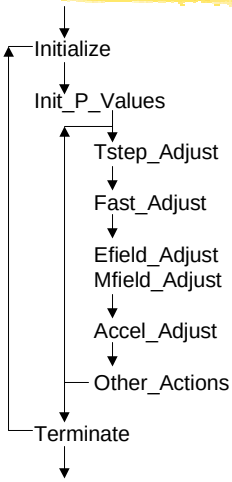
User Program Segments

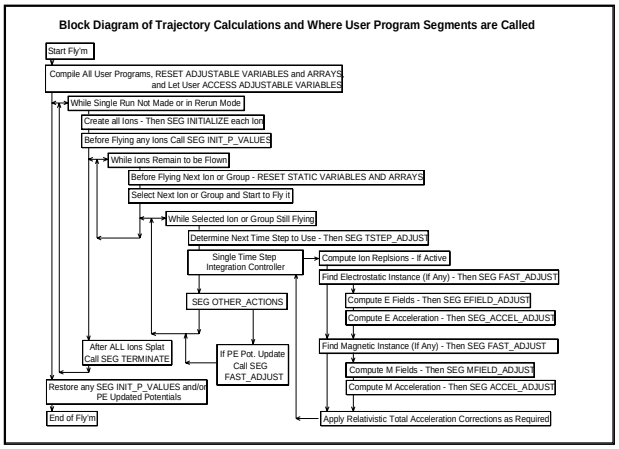
- The user program file contains user program segments.
- User program segments act like subroutines.
- SIMION calls each type of program segment at specific points in a trajectory calculation.
- Thus you must use a specific user program segment to control a specific aspect of the trajectory calculation.

The 9 Program Segments

- Initialize
- Init_P_Values
- Tstep_Adjust
- Fast_Adjust
- Efield_Adjust
- Mfield_Adjust
- Accel_Adjust
- Terminate
- Other_Actions

Integration into SIMION





Example - Good Style

```
Define_Adjustable Viscous_Damping  0          ; adjustable variable Viscous_Damping
                                              ; set to 0 (no viscous damping by default)
                                              ; adjustable at the beginning of each
Fly'm

Begin_Segment Accel_Adjust          ; start of Accel_Adjust program segment

   Recall Ion_Ax_mm                 ; recall current x acceleration (mm/usec^2)
   Recall Ion_Vx_mm                 ; recall current x velocity (mm/sec)
   Recall Viscous_Damping           ; recall the viscous damping term
   Multiply                         ; multiply times x velocity
   Subtract                         ; and subtract from x acceleration
   Store Ion_Ax_mm                  ; return adjusted value to SIMION

   Recall Ion_Ay_mm                 ; recall current y acceleration (mm/usec^2)
   Recall Ion_Vy_mm                 ; recall current y velocity (mm/sec)
   Recall Viscous_Damping           ; recall the viscous
 :
 :
 .
   Exit                             ; exit to SIMION (optional statement)
```

Bad Style - no Advantage

Runs no faster than a fully commented style.

```
defa viscous_damping 0,seg accel_adjust rcl ion_ax_mm
rcl ion_vx_mm rcl viscous_damping * - sto ion_ax_mm
rcl ion_ay_mm rcl ion_vy_mm rcl viscous_damping * -
sto ion_ay_mm rcl ion_az_mm rcl ion_vz_mm
rcl viscous_damping * - sto ion_az_mm
```


Language Rules

- Case is Ignored (upper and lower case are the same)
- Blank lines and indention's are ignored
- The Semicolon ; Starts an In-Line Comment
- The language is based on the HP RPN calculator format with a 10 register rotary stack:

  25 10 * 5 / ; means (25 * 10) / 5
- The Compiler looks for words and converts them into:
  - Commands
  - Numbers
  - Variable Names and Labels

Examples of Commands

+ or: Add

Adds contents of x and y registers, puts result in y-register, and renames it as x-register (*e.g.* 1 2 + *becomes 3 in register where 1 was originally stored*).

>KE or: Speed_to_Kinetic_Energy

Converts from speed (*mm/usec*) to kinetic energy (*eV*). On entry the x-register is assumed to contain the ion's speed and the y-register is assumed to contain the mass of the ion (*amu*). On exit the x-register contains the ion's KE and the y-register is unchanged. The >SPD command performs the reverse transformation.

Classes of Commands

- Calculations
  + - * / SIN, COS, LOG
- Transformations
  >DEG, >RAD, >KE, >P3D, >WBC
- Flow Controls
  LBL, GSB, RTN, EXIT
- Tests (do next command if <u>true</u> else skip next command)
  X=0, X<=0, X!=Y, X>Y
- Communication to User and Variables
  STO, RCL, R/S, KEY?, MESSAGE

Types of Variables

- Adjustable Variables (defined at top of program file)
  DEFA MY_VARIABLE 30.0
  Global and lasts *throughout* a Fly'm
- Static Variables (defined at top of program file)
  DEFS TIME_FLAG-2.0E4
  Global and Reset to initial value *before* each ion is flown
- Temporary Variables
  STO LUCKY
  Local and Temporary: Created with STO command. Name must be *unique* (not Adjustable, Static, or Reserved variable name)
- Reserved Variables
  Ion_Color Ion_Charge Ion_Px_mm
  Used to communicate with SIMION. Each Program Segment has specific read and write access to various reserved variables.

Session 9 - Lab on User Programs

Files: C:\Advanced SIMION Class\09. User Program Introduction\Lab

Using an Existing User Program

The best way to start programming is to steal, steal, steal.

1. Remove all PAs From RAM
2. Use New to create a new array using EINZEL.GEM from directory **C:\Advanced SIMION Class\09. User Program Introduction\Lab**
3. Save the array as EINZEL.PA# and ***then*** Refine it.
4. Remove all PAs From RAM
5. Use View to load EINZEL.IOB (EINZEL.FLY loads automatically)
6. Fly the ions and see that they focus properly.

7. Now exit View and click on the GUI file manager button
8. Select the file named **RANDOM.PRG**, click the Other button and rename the file to **EINZEL.PRG.**
9. Return to the main SIMION menu.
10. Notice that the **Test & Debug Selected PA's User Progs** button is unblocked. SIMION has recognized that there is now a user program associated with the EINZEL.PA0 potential array. Click the Test & Debug button to access the user program development system (just a quick look).
11. Click the Compile button and the program will be compiled. If you want a cross reference listing depress the Show Xref Listing button and then click the Compile button again. If you want to look at this cross reference listing, scroll the view using the window's lower right hand corner button.
12. To examine the user program file with EDY click the Edit: EINZEL.PRG button. Look at the listing. Exit EDY by the following keystrokes: **ESC Q N**.
13. Click the Compile button again to verify everything is OK. Now click the Quit key to return to the main menu.
14. Re-enter View and fly the ions. SIMION will compile the user programs and display the adjustable variables. Click the E Fmt button to switch from E to F format display. Now fly the ions with the default settings. What a mess!
15. You can turn off the user programs for a single run. Click Fly'm and then Click the On button to turn the user program off, and then fly the ions.
16. Re-Fly the ions with various adjustable variable settings and see what happens. Are the results reasonable?
17. Perhaps you would like to change the starting values for the adjustable variables in the user program. Click the PAs tab and then the Debug button to access the user program development screen. Now click the Edit button to access the user program file with EDY. Change the starting values of the adjustable variables. Now save the file and exit from EDY: **ESC Q Y**. Click the Compile button to check for compile errors. Now click the Quit button to return to View. Re-Fly the ions to confirm that the changes in initial value have been made successfully

Adding a User Program Segment

User programs generally develop by evolution. In this can we would like to exert some user control from the keyboard. The object is to create a marker every time a keyboard key is pressed. This is really a very simple task.

1. Click the PAs tab and then the Debug button to access the user program development screen. Now click the Edit button to access the user program file with EDY.
2. Add the following two lines at the bottom of the file:

   seg other_actions
   key? x!=0 mark

   The first line defines the start of an Other_Actions program segment. The second line has three commands:

key?	Tests for keyboard input and returns a non-zero key code if you hit a key
x!=0	Tests the return value from key? for a non-zero value (you hit a key) If the value is non-zero the next instruction will be executed (do if true) else it will be skipped and the program segment will exit without doing anything.
mark	This command creates a single marker. It uses the currently selected marker color and can be used as a trigger event for data recording. The mark command is executed if the x!=0 test is true.

3. Now save the changes and exit EDY: ESC Q Y. Test compile your efforts. If you have an error fix it with EDY. Keep trying until it compiles OK. Now click the Quit button to return to View.
4. Fly the ions as dots with the speed slider to the center to slow the ions down a bit. Hit the spacebar and watch for markers. Can you create markers anywhere? What is the problem? ***Remember, an array's user program will only be called when ions are actually within its instance's array volume***.

Additional Challenges

1. Modify the user program to output the ion's x, y, and z position in workbench coordinates (mm) when you hit a key. Hint: The **MESSAGE**, **LBL**, **GTO**, and **EXIT** commands might be helpful.
2. Change this program to output x, y, and z in 3D array coordinates and then in absolute array coordinates.
3. For a real challenge calculate and display the ion's KE when you hit a key. Hint: The **>P3D** and **>KE** transforms might be helpful.

10.0 Arrays in User Programming

- Both adjustable and static arrays (variables) are now supported via user programs.
- Up to 200 (total) unique adjustable and static arrays allowed for all user programs.
- Array data can be stored-to and read-from ASCII files. Maximum of Save and Load commands for all user programs is 200.

Array Format and Size

- All arrays are one dimensional. Index mapping is the responsibility of the user.

$$\begin{pmatrix} e_1 \\ e_2 \\ e_3 \\ \cdot \\ \cdot \\ e_n \end{pmatrix}$$

- Each element (e_n) is a double precision floating point number (8 bytes).
- Index starts at 1 (not 0). Size is heap limited.

Defining Arrays

- Adjustable arrays:

  Command: ADEFA or Array_Define_Adjustable

  ADEFA Name Size ; "filename"

  Name = name of array
  Size = number of array elements
  "filename" = file containing initialization values (optional)

  ADEFA my_array 100 ; "energy.dat"

Defining Arrays

- Static arrays:

  Command: ADEFS or Array_Define_Static

  ADEFS Name Size ; "filename"

  Name = name of array
  Size = number of array elements
  "filename" = file containing initialization values (optional)

  ADEFS my_array 100 ; "energy.dat"

Addressing Arrays

- Array elements are called with ARCL and stored with ASTO commands.

  When using ASTO, the *value* is held in the y register, the index in the x register:

```
7                    ; value in y register
50 ASTO my_array     ; placed at index 50 in my_array
                     ; stack rolled up, value (7) now
                       in x register
```

  When using ARCL, the index is placed in the x register and ARCL replaces this index with the *value*.

Lifetime of Array Variables

- Adjustable array variables:
  - Are initialized to zero (or from a file) before any Initialize program segments are called.
  - Values are retained throughout the period of ion flying. Globally visible.
- Static array variables:
  - Are initialized to zero (or from a file) immediately before each ion (or group) begins to fly.
  - Values are retained only until the end of the flight. Globally visible.

Array Initialization Options

- All array elements (adjustable and static) are first initialized to zero as described previously.
- A file containing array initialization data can be loaded after the array has been pre-zeroed.

  ALOAD *array_name* ; "filename"

  (the name of the file comes after the semicolon and is inside quotation marks)

Array Initialization Options

- SIMION will read only the number of values needed to fill the array (defined by *size* in ADEF).
- If there are fewer values in the file than the defined array size, the remaining values in the array will be zero.
- Files are free format ASCII, numbers separated by spaces or commas, blank lines allowed, the ";" recognized as start of an inline comment. Maximum line length 200 characters.
- Errors in file format are trapped and flagged.

Array File Saving Options

- ASAVE or Array_Save command.

  ASAVE my_array ; "filename"

  Saves contents of array my_array to file named "filename"

- If "filename" already exists, contents will be destroyed and new data inserted (not appended).
- Format of file is ASCII with 5 coma-separated numbers per line.

  1,2,3,4,5

  3,5,7,6,8

Array Summary

Array Types:

- ADEFA Defines Adjustable Array
- ADEFS Defines Static Array
- ALOAD Load array with values from an ASCII file
- ASAVE Save array contents to an ASCII file
- ASTO Store a number into an array element
- ARCL Recall a number from an array element

The Init_P_Values Segment

New User Program Segment

- Used to initialize an entire fast-adjust array (e.g. .PA0) before flying any ions.
- Used in place of fast_adjust to increase speed (when appropriate).
  - Best for flying large number of ions in small arrays.
  - Only applicable when fields *do not change* during ion flying.

The Init_P_Values Segment

- Unlike all other program segments, ions do not have to be in the instance for Init_P_Values to be called.
- This means ion and instance context have no meaning within an Init_P_Values segment.
- Can access only these reserved variables:
  - Adj_Elect00 - Adj_Electro30
  - Adj_Pole00 - Adj_Pole30

The Init_P_Values Segment

- No instance related coordinate transformations allowed (>WBC etc).
- Can retain changed potentials at end of fly'm using the reserved variable:

  Retain_Changed_Potentials

  by saving a non-zero value to this variable within any Terminate segment.

Init_P_Values Example

```
Defa Tune_Voltage  100          ; tuning voltage default
Seg_Init_P_Values               ; beginning of segment
        RCL Tune_Voltage        ; get tuning voltage
        STO Adj_Elect01         ; store in fast adjust electrode 1
        Exit
Seg_Terminate                           ; beginning of segment
        1 STO Retain_Changed_Potentials ; set variable to non-zero
                                        ; to save ending potential.
        Exit
```

Session 10 - Lab on User Programs

Files: C:\Advanced SIMION Class\10. User Program Introduction\Lab

Modifying an Existing User Program

The best way to start programming is to steal, steal, steal.

1. Remove all PAs From RAM
2. From View load the file tune.iob from directory
   C:\Advanced SIMION Class\10. User Program Introduction\Lab
3. Fly the ions. Note the tuning voltage displayed in the data window. Compare this with the voltage on the middle electrode in the view screen. Why are they different?
4. Click the PAs tab and then the Dbug button to get to the User Program file screen.
5. Click edit to see the user program in EDY. Find the Terminate Segment and using the Retain_Changed_Potentials variable modify this segment so that the last potential used is kept and displayed in the view screen. Save your changes (Exc, S, Y) then alt tab back to the user program screen and compile the altered program. If it compiles, quit back to SIMION and fly the ions again.
6. Now compare the voltage displayed in the data window with that in the view window for the middle electrode. They should not agree. Why?
7. Now fly the ions on a Potential Energy surface. Compare the voltage displayed in the data window with that in the view window for the middle electrode. They should agree. Why?
8. Get to the ion definition screen and increase the number of ions being flown to 15 for each group.
9. Fly the ions with a goal of 1 e-5 and time how long it takes to complete all trajectories. (It will abort after 20 iterations). Write down the time.
10. Get back to the User Program screen, and then into the user program in EDY.
11. Modify the program to use the Init_P_Values segment instead of Fast_Adjust to set the voltage. (Hint, put ; at the beginning of the lines in the Fast_adjust segment that change the voltages, and add a Segment Init_P_Values just below it, adjusting the same electrode.)
12. Save your changes, compile the program and fly the ions using the same goal value (1 e-5). Time how long this version takes to run through 20 iterations. Why is it faster?
13. Use manual fast adjust to set the middle electrode to 300 volts. Now fly the ions in a 2D (xy) view using Init_P_Values. Compare the voltage displayed in the data window with that in the view window for the middle electrode. They should agree. Why?

    Explain what is going on! Hint: Under what conditions does the potential of the entire array change under program control?

11.0 User Programming Tricks

- Important concepts
- Useful overall strategies
- When to use various user program segments

User Program Files

- A user program file contains one or more program segments (functions).
- Each user program file is associated with one and only one potential array.
- Each potential array can be associated with one and only one user program file.
- User program files share the name of their array (e.g. einzel.pa0 <> einzel.prg).

Legal Program Segments

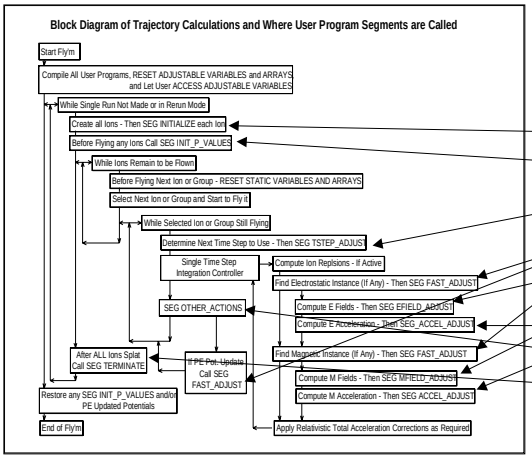
Only one segment of each type can be in a user program file

Define_Data	Defines Global variables and arrays
Init_P_Values	Pre-sets fast adjust potentials (new)
Initialize	Initializes ion's starting parameters
Tstep_Adjust	Controls time step size used
Fast_Adjust	Fast adjusts potentials as ions fly
Efield_Adjust	Adjusts electrostatic fields
Mfield_Adjust	Adjusts magnetic fields
Accel_Adjust	Adjusts ion accelerations
Other_Actions	Post tstep control of ion's parameters
Terminate	Post flight analysis and rerun control

Program Segment Access Rules

- Each program segment is ONLY accessed at specified points in the trajectory computation cycle.
- The ion MUST be in an instance that references a potential array that has a user program file containing the program segment (e.g. Initialize). The ONLY exception is the Init_P_Values segment.

Program Segments and Calculations



Segment Access Paradigm

- Reserved Variables used for communication
- The propose and modify paradigm:
  - 1. SIMION calculates: proposed time step
  - 2. SIMION calls Tstep_Adjust (as per rules) passing value in ion_time_step
  - 3. Tstep_Adjust examines and/or modifies the value in ion_time_step
- Reserved Variable r/w access is controlled

Reserved Variables

- Reserved Variables used for communication between SIMION and program segments

Variable Name	Use	Units	Read Access	Write Access
Adj_Elect00 to Adj_Elect30	Fast Adj Electrode' Voltages	Volts	Fast_Adjust Init_P_Values	Fast_Adjust Init_P_Values
Adj_Pole00 to Adj_Pole30	Fast Adj Pole' Mag Potentials	Mags	Fast_Adjust Init_P_Values	Fast_Adjust Init_P_Values
Ion_Ax_mm Ion_Ay_mm Ion_Az_mm	Ion's current Acceleration (WB Coordinates)	mm/micro sec^2 (WB Orientation)	Accel_Adjust Other_Actions Terminate	Accel_Adjust
Ion_BfieldX_gu Ion_BfieldY_gu Ion_BfieldZ_gu	Magnetic Field at Ion's Location (PA's Orientation)	Gauss (PA's Orientation)	Mfield_Adjust	Mfield_Adjust
Ion_BfieldX_mm Ion_BfieldY_mm Ion_BfieldZ_mm	Magnetic Field at Ion's Location (WB Orientation)	Gauss (WB Orientation)	Accel_Adjust Other_Actions Terminate	None
Ion_Charge	Ion's current charge	in units of elementary charge	Any Prog Seg except Init_P_Values	Initialize Other_Actions
Ion_Color	Color of Ion	0-15	Initialize Other_Actions	Initialize Other_Actions
Ion_DvoltsX_gu Ion_DvoltsY_gu Ion_DvoltsZ_gu	Voltage Gradient at Ion's Location (PA's Orientation)	Volts/grid unit (PA's Orientation)	Efield_Adjust	Efield_Adjust
Ion_DvoltsX_mm Ion_DvoltsY_mm Ion_DvoltsZ_mm	Voltage Gradient at Ion's Location (WB Orientation)	Volts/mm (WB Orientation)	Mfield_Adjust Accel_Adjust Other_Actions Terminate	None
Ion_Instance	Current Instance	1- max instance	Any Prog Seg except Init_P_Values	None
Ion_Mass	Ion's current mass	amu	Any Prog Seg except Init_P_Values	Initialize Other_Actions
Ion_mm_Per_Grid_Unit	Min Current Scaling'	mm/grid unit	Any Prog Seg except Init_P_Values	None
Ion_Number	Ion's Number	1- max ion	Any Prog Seg except Init_P_Values	None

Variable Name	Use	Units	Read Access	Write Access
Ion_Px_Abs_gu Ion_Py_Abs_gu Ion_Pz_Abs_gu	Ion's current PA Array Coordinates	grid units (PA's Abs Coordinates)	Any Prog Seg except Init_P_Values	None
Ion_Px_gu Ion_Py_gu Ion_Pz_gu	Ion's current (PA's Coordinates)	grid units (PA's Coordinates)	Any Prog Seg except Init_P_Values	None
Ion_Px_mm Ion_Py_mm Ion_Pz_mm	Ion's current Worldanch Coordinates	mm (WB Coordinates)	Any Prog Seg except Init_P_Values	Initialize Other_Actions
Ion_Splat	Ion Status Flag'	Flying = 0 Not Flying != 0	Initialize Other_Actions	Initialize Other_Actions
Ion_Time_of_Birth	Ion's Birth Time	micro seconds	Any Prog Seg except Init_P_Values	Initialize Other_Actions
Ion_Time_of_Flight	Ion's current TOF'	micro seconds	Any Prog Seg except Init_P_Values	Other_Actions
Ion_Time_Step	Ion's Time Step'	micro seconds	All but Initialize and Init_P_Values	Tstep_Adjust
Ion_Volts	Electrostatic Pot at Ion's Location	Volts	Efield_Adjust Mfield_Adjust Accel_Adjust Other_Actions Terminate	Efield_Adjust
Ion_Vx_mm Ion_Vy_mm Ion_Vz_mm	Ion's current Velocity (WB Coordinates)	mm/micro sec (WB Orientation)	Any Prog Seg except Init_P_Values	Initialize Other_Actions
Rerun_Flym	Rerun Flym Flag'	NO = 0 YES = 1	Initialize Other_Actions Terminate	Initialize Other_Actions Terminate
Trajectory_Image_Control	Controls Trajectory Image Viewing and Recording'	Value View Retain 0 YES YES 1 YES NO 2 NO YES 3 NO NO	Initialize Other_Actions Terminate	Initialize Other_Actions Terminate
Retain_Changed_Potentials	Controls restoring of changed potentials at end of Fly'm*	NO = 0 (default) YES = 1	None	Terminate
Update_PE_Surface	New PE Surface'	YES != 0	None	Other_Actions

Temporary Variables

- Visibility: Visible only in user program segment that defines it
- Lifetime: Exists only during a reference to the program segment
- User Adjustability: No
- Conserves Changes: No
- Error Trapping: Yes (for sto without rcl)

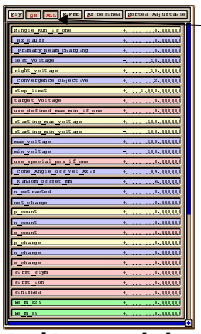
Static Variables & Arrays

- Visibility: Globally visible to most user programs except: Initialize, Init_P_values, and Terminate.
- Lifetime: Reset before each ion or group flight.
- User Adjustability: No
- Conserves Changes: No

Adjustable Variables & Arrays

- Visibility: Globally visible to all user programs
- Lifetime: Values conserved throughout entire fly'm (including reruns).
- User Adjustable:
  - Before ion flying starts
  - During ion flying (via AdjV tabbed screen)
- Conserves Changes: YES (before fly'm)

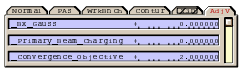
Controlling Variable Visibility

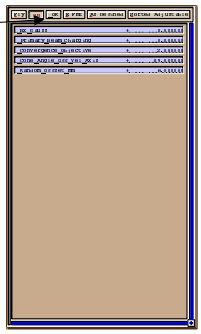


All
Adjustable
Variables

_OK
Leading
Underscore
Adjustable
Variables

AdjV runtime Screen





- Adjustable Variables
  - Use leading underscore in names to see (e.g. _Left_Electrode_Voltage)

Controlling Variable Visibility

- Use Adjustable & Static Arrays for Flags

Adefa aflags 20 ;"defaults.dat" adjustable flags -- initialized
Adefs sflag 500 ; static array flag -- zeroed for each ion

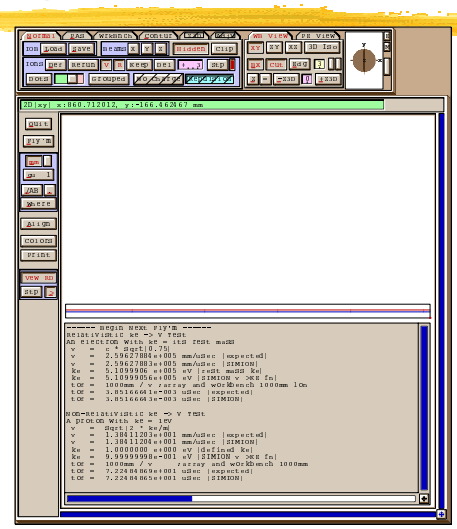
1 arcl aflags ; simple recall (x = aflags[1])
0 1 sto aflags ; simple store (aflags[1] = 0)

defa flag 0 ;flag shown at start (unless _OK underscore)

rcl flag
1 sto flag

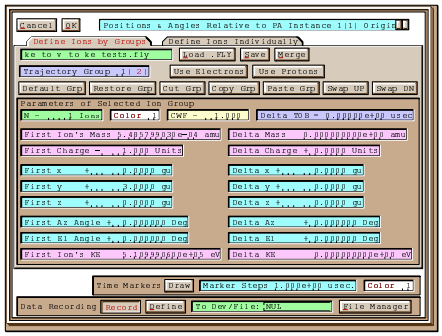
Creating Integrated Demos

1. Ke to Velocity
   (Test Demo)

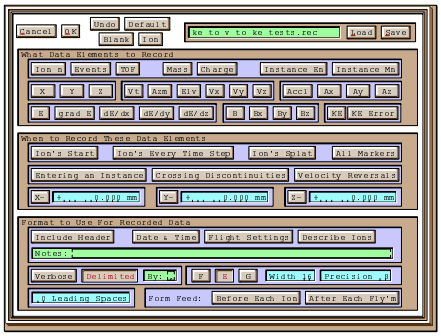


Creating Integrated Demos

1. Ke to Velocity (Test Demo)



Ion Definitions



Data Recording

Creating Integrated Demos

1. Ke to Velocity (Test Demo)

```
seg other_actions

rcl ion_splat x=0 exit                          ;exit if ion hasn't splatted
1 rcl ion_mass x>y goto ions   ;jump if proton
                                                ;else output relativistic electron data
        message ;Relativistic ke -> v Test
        message ;An electron with ke = its rest mass
                                                ;relativistic velocity from
                                                ; 2 = 1/(sqrt(1 - v*v/c*c))
        message ; v   =  c * sqrt(0.75)
                                                ; given c = 2.99792458e+005 mm/usec
        message ; v   =  2.59627884e+005 mm/usec (expected)
        rcl ion_vx_mm                           ; velocity at splat
        message ; v   = # mm/usec (SIMION)


                                                ; rest mass of electron
                                                ; and starting ke
        message ; ke  =  5.1099906 e+005 eV (rest mass ke)
        rcl ion_mass                            ; get mass of electron
        rcl ion_vx_mm                           ; get its exit velocity
        >ke                                     ; convert back to ke as check
        message ; ke  = # eV (SIMION v >KE fn)

                                                ;equation for const velocity tof
        message ; tof =  1000mm / v ;array and workbench 1000mm long

                                                ;tof assuming expected velocity
        message ; tof =  3.85166641e-003 usec (expected)
        rcl ion_time_of_flight                  ;tof computed by SIMION
        message ; tof = # usec (SIMION)
        message ;
        exit
```

```
lbl ions                                        ;proton data output
message ;Non-Relativistic ke -> v Test
message ;A proton with ke = 1eV
message ; v   =  sqrt(2 * ke/m)
                ; v   =  sqrt(2.0 * 1.60217733e-19/1.6726231e-27)
                ;where                  1eV = 1.60217733e-19 J
                ;               proton's mass = 1.6726231e-27 kg
message ; v   =  1.38411203e+001 mm/usec (expected)
rcl ion_vx_mm                                   ; velocity at splat
message ; v   = # mm/usec (SIMION)
                                                ; defined initial ke
message ; ke  =  1.0000000 e+000 eV (defined ke)
rcl ion_mass                                    ; get mass of proton
rcl ion_vx_mm                                   ; get its exit velocity
>ke                                             ; convert back to ke as check
message ; ke  = # eV (SIMION v >KE fn)

                                                ;equation for const velocity tof
message ; tof =  1000mm / v   ;array and workbench 1000mm long
                                                ;tof assuming expected velocity
message ; tof =  7.22484869e+001 usec (expected)
rcl ion_time_of_flight                          ;tof computed by SIMION
message ; tof = # usec (SIMION)
exit
```


Creating Integrated Demos

- Recommended Strategies
  - Use auto-loading FLY (or ION) and REC files
  - Control displayed number format with REC file
  - Use self-evident message output if possible via Initialize, Other_Actions, Terminate, and/or data recording (REC)
  - Hide Adjustable Variables from user (e.g. _Voltage (shown), flag (hidden), or via arrays)

Randomizing Ions

- Use Initialize Segment to Randomize Ions

_Random Directory

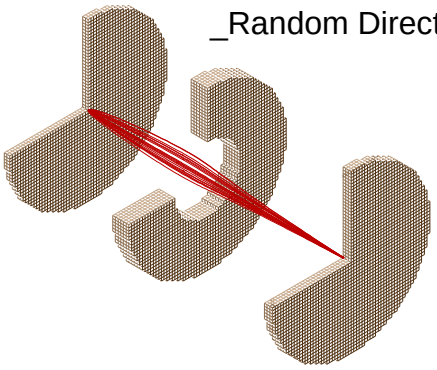
Initialize

Input Ions

Group of ions with identical starting parameters.

Adjustable Variables

Percent_Energy_Variation
Cone_Angle_Off_Vel_Axis



Randomizing Ions

- Initialize Segment Strategy

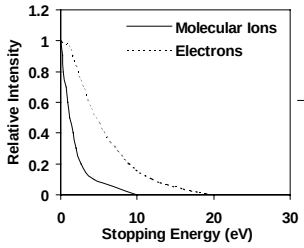
Get polar v components	Get ion's velocity components from SIMION Convert to 3D polar and save speed plus az and el angles
Compute KE Randomize KE Convert to v	get ion_mass, recall speed, convert to ke, and save ke use rand and percent_energy_variation to randomize ke get ion_mass, recall new ke, convert to v, and save speed
Create random conical pattern in vertical Convert to 3D Rectangular Swing back to az = el = 0	assume ion flying vertical randomize el off vertical by cone angle randomize az to +- 180 degrees convert to rectangular 3D coords rotate vector -90 in elevation (restores a starting point)
Rotate entry el Rotate entry az Pass V Rect. to SIMION	rotate vector initial el angle rotate vector initial az angle pass ion's velocity components back to SIMION

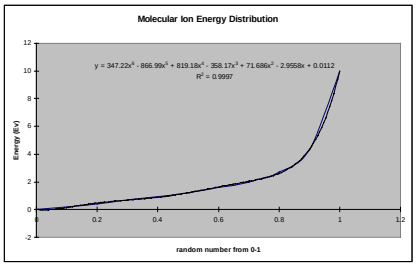
Initialize for Random Ions

```
;-----------------------------------------------------------------
; this user program randomly changes the initial ke and direction of ions
; energy is randomly changed +- Percent_Energy_Variation * ke
; ions are emitted randomly within a cone of revolution around the
; ion's defined velocity direction axis
; the full angle of the cone is +- Cone_Angle_Off_Vel_Axis
; (e.g. 90.0 is full hemisphere, 180 is a full sphere)
;-----------------------------------------------------------------

;--------- you can use it with your own lenses without modification ----------
;          (just rename user program file using your pa's name)
;------ Note: you can also modify the emission distributions as desired ------
defa Percent_Energy_Variation     50    ; (+- 50%) random energy variation
defa Cone_Angle_Off_Vel_Axis      90    ; (+- 90 deg) cone angle hemisphere


seg initialize                          ; initialize ion's velocity and direction
     ;------------------- get ion's initial velocity components -------------
     rcl ion_vz_mm                      ; get ion's specified velocity components
     rcl ion_vy_mm
     rcl ion_vx_mm

     ;------------------- convert to 3d polar coords -------------
     >p3d                               ; convert to polar 3d

     ;------------------- save polar coord values ----------------
     sto speed rlup                     ; store ion's speed
     sto az_angle rlup                  ; store ion's az angle
     sto el_angle                       ; store ion's el angle

     ;------------------- make sure Percent_Energy_Variation is legal -------------
                                        ; force 0 <= Percent_Energy_Variation <= 100
     rcl Percent_Energy_Variation abs
     100 x>y rlup sto Percent_Energy_Variation

     ;------------------- make sure Cone_Angle_Off_Vel_Axis is legal -------------
                                        ; force 0 <= Cone_Angle_Off_Vel_Axis <= 180
     rcl Cone_Angle_Off_Vel_Axis abs
     180 x>y rlup sto Cone_Angle_Off_Vel_Axis

     ; ---------------------- calculate ion's defined ke -------------
     rcl ion_mass                       ; get ion's mass
     rcl speed                          ; recall its total speed
     >ke                                ; convert speed to kinetic energy
     sto kinetic_energy                 ; save ion's defined kinetic energy

     ; ---------------------- compute new randomized ke -------------
                                        ; convert from percent to fraction
     rcl Percent_Energy_Variation 100 /
     sto del_energy 2 * rand *          ; fac = 2 * del_energy * rand
     rcl del_energy - 1 +               ; fac += 1 - del_energy
     rcl kinetic_energy *               ; new ke = fac * ke

     ; ---------------------- convert new ke to new speed ------------
     rcl ion_mass                       ; recall ion mass
     x<>y                               ; swap x any y
     >spd                               ; convert to speed
     sto speed                          ; save new speed

     ;-- compute randomized el angle change 90 +- Cone_Angle_Off_Vel_Axis -------
     ;-------- we assume elevation of 90 degrees for mean ----------
     ;-------- so cone can be generated via rotating az +- 90 -------
                                        ; (2 * Cone_Angle_Off_Vel_Axis * rand)
     2 rcl Cone_Angle_Off_Vel_Axis * rand *
                                        ; - Cone_Angle_Off_Vel_Axis + 90
     rcl Cone_Angle_Off_Vel_Axis - 90 +

     ;-------------- compute randomized az angle change -------------
     ;-------- this gives 360 effective because of +- elevation angels ---
     180 rand * 90 -                    ;          +- 90 randomized az

     ;---------------------- recall new ion speed -------------------
     rcl speed                          ; recall new speed

     ;-------- at this point x = speed, y = az, z = el -------------
     ;------------ convert to rectangular velocity components ---------
     >r3d                               ; convert polar 3d to rect 3d

     ;------------ el rotate back to from 90 vertical -------------
     -90 >elr

     ;------------ el rotate back to starting elevation -------------
     rcl el_angle >elr

     ;------------ az rotate back to starting azimuth -------------
     rcl az_angle >azr

     ;------------ update ion's velocity components with new values --------
     sto ion_vx_mm                      ; return vx
     rlup
     sto ion_vy_mm                      ; return vy
     rlup
     sto ion_vz_mm                      ; return vz

     ;---------------------- done ----------------------------------------
```


Another Randomizing Example

- Monte Carlo example
  - Secondary ion energies (fitting)





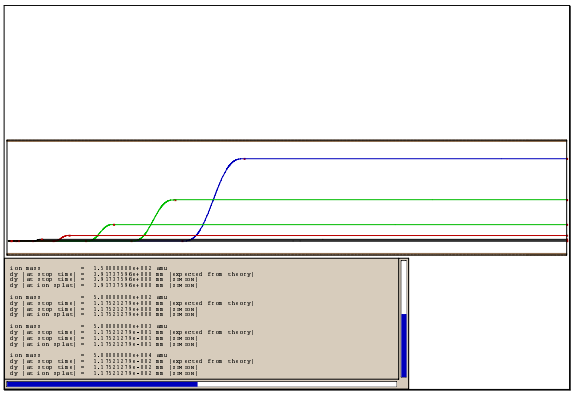
Another Randomizing Example

- Monte Carlo example cont.
  - Secondary ion energies (coding)

```
;molecular energies
rand sto x
          ;get random number x (0-1)
0.0112
          ; +0.0112
2.9558 rcl x * -                                  ; -2.9558x
71.686 rcl x        rcl x * sto xpower * +        ; +71.686x^2
358.17 rcl xpower rcl x * sto xpower * -          ; -358.17x^3
819.18 rcl xpower rcl x * sto xpower * +          ; +819.18x^4
866.99 rcl xpower rcl x * sto xpower * -          ; -866.99x^5
347.22 rcl xpower rcl x * sto xpower * +          ; +347.22x^6
abs
```


Controling Times and Time Steps

- Critical Timing Example



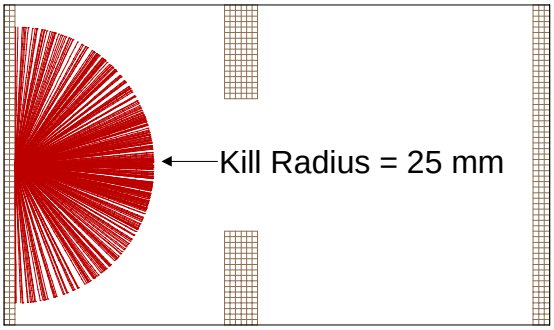
Controling Times and Time Steps

- Strategy: Tstep_Adjust via Ion_Time_Step

```
If TOF >= left  then right test
If TOF + TS < left then exit
ion_time_step = left - TOF
exit

LBL right test
If TOF >= right then exit
If TS > MAX then TS = MAX
If TOF + TS < right then exit
ion_time_step = right - TOF
exit
```


Binary Boundary Approach

- Use Tstep_Adjust and Other_Actions



Binary Boundary Approach

- Strategy

```
Other_Actions Control

If boundary crossed
        if not too close
        restore prior ion state
        save time_step
        set half step flag
else
        save ion's state
        (Px,Py,Pz,
        Vx,Vy,Vz,TOF)
```

```
Tstep Control

If half step flag not set
        exit
else
        get saved time_step
        halve it
        save to ion_time_step
        clear half step flag
```


Fast Adjusting Potentials

- Strategy: Fast_Adjust or Init_P_Values
  - Time varying fields: Fast_Adjust (only)
    - Tstep_Adjust may be needed for step size control
  - Static fields: Fast_Adjust or Init_P_Values
    - Fast_Adjust for large arrays and smaller numbers of ions.
    - Init_P_Values for smaller arrays and larger numbers of ions.
    - Conduct time trials (very useful).

Specifying Known Fields

- Strategy: Mfield_Adjust or Efield_Adjust
  - Fixed Magnetic fields (Mfield_Adjust)
    - Use field-free magnetic array
    - 0 sto BfieldZ_gu ;zero Bz
      sto BfieldY_gu ;zero By
      rcl _Bfield_gauss ;get Bx from Adj. Var.
      sto BfieldX_gu ;save user defined Bx

Specifying Known Fields

- Strategy: Mfield_Adjust or Efield_Adjust
  - Measured Magnetic fields (Mfield_Adjust)
    - Use field-free magnetic array
    - Adefa Bfield 10000 ;"mfield.dat" measured B
    - Get ion's gu coords (abs or 3D)
      Use to get interpolated B fields at point
      Pass Bx, By, and Bz fields back to SIMION
      (see measured magnetic fields example)

Specifying Known Fields

- Strategy: Mfield_Adjust or Efield_Adjust
  - Analytic Electrostatic fields (Efield_Adjust)

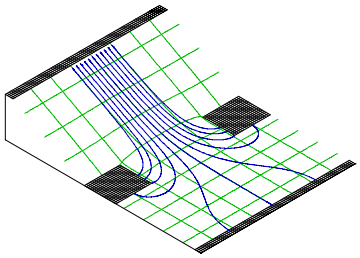
	seg efield_adjust	;quadrupolar field
Selectively use	3 rcl ion_number x<=y exit	;use refined field <= 3
Compute and store potential	rcl ion_px_mm entr *	; x^2
	rcl ion_py_mm entr * -	; (x^2 - y^2)
	10 / sto ion_volts	; (x^2 - y^2)/10
Compute and store x gradient	rcl ion_px_mm 0.2 *	; volts/mm x gradient
	rcl ion_mm_per_grid_unit *	; volts/gu x gradient
	sto ion_dvoltsx_gu	; pass to SIMION
Compute and store y gradient	rcl ion_py_mm -0.2 *	; volts/mm y gradient
	rcl ion_mm_per_grid_unit *	; volts/gu y gradient
	sto ion_dvoltsy_gu	; pass to SIMION
Zero z gradient	0 sto ion_dvoltsz_gu	; no field in z
	exit	

Modifying Ion Accelerations

- Strategy: Use Accel_Adjust

```
seg accel_adjust                ; beginning of accel_adjust segment

   rcl ion_time_step x=0 exit   ; exit if zero time step
   rcl linear_damping x=0 exit  ; exit if damping set to zero
   abs sto damping              ; force damping term to be positive
   * sto tterm                  ; compute and save number of time constants
   chs e^x 1 x><y -             ; (1 - e^(-(t * damping)))
   rcl tterm / sto factor       ; factor = (1 - e^(-(t * damping)))/(t * damping)

   rcl ion_ax_mm                ; recall ax acceleration
   rcl ion_vx_mm                ; recall vx velocity
   rcl damping * -              ; multiply times damping and sub from ax
   rcl factor *                 ; multiply times factor
   sto ion_ax_mm                ; store as new ax acceleration

   rcl ion_ay_mm                ; recall ay acceleration
   rcl ion_vy_mm                ; recall vy velocity
   rcl damping * -              ; multiply times damping and sub from ay
   rcl factor *                 ; multiply times factor
   sto ion_ay_mm                ; store as new ay acceleration

   rcl ion_az_mm                ; recall az acceleration
   rcl ion_vz_mm                ; recall vz velocity
   rcl damping * -              ; multiply times damping and sub from az
   rcl factor *                 ; multiply times factor
   sto ion_az_mm                ; store as new az acceleration
```

Drag Demo



Arbitrary Ion Accelerations

- Strategy: Use Accel_Adjust
  - Can define arbitrary field forces (e.g. $1/r^5$)
  - Can define arbitrary field directions
    - Always normal to ion's current velocity
    - Always parallel to ion's current velocity
    - Moving points of attraction
    - Random points of attraction
    - Whatever you can imagine

Controlling Ion's Fate

- Strategy: Other_Actions
  - Position
  - Velocity
  - Mass
  - Charge
  - Color
  - TOF
  - Splat

User Programming's

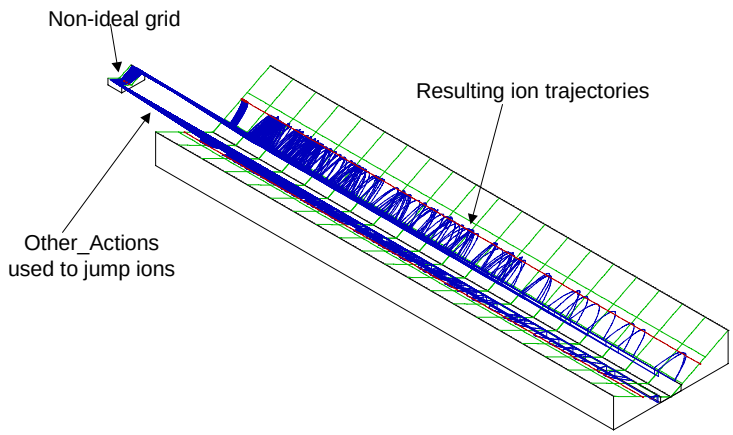
Master Controller

(Most Powerful Program Segment)

Controlling Ion's Fate

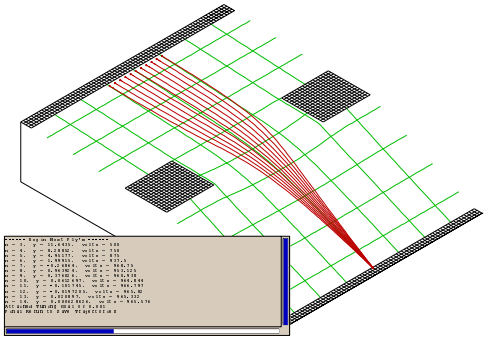
- Strategy: Other_Actions
  - Mass and/or charge
  - Energy and/or position
  - Boundary approaches
  - Collision simulations
  - Simulations involving jumps
  - Selective ion marking (e.g. via color & marks)
  - Selective ion killing

Controlling Ion's Fate



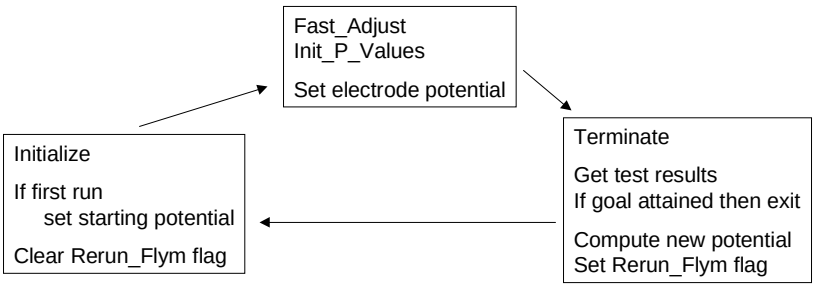
Controlling Successive Runs

- Strategy: Terminate and Initialize
  - Via the Rerun_Flym reserved variable



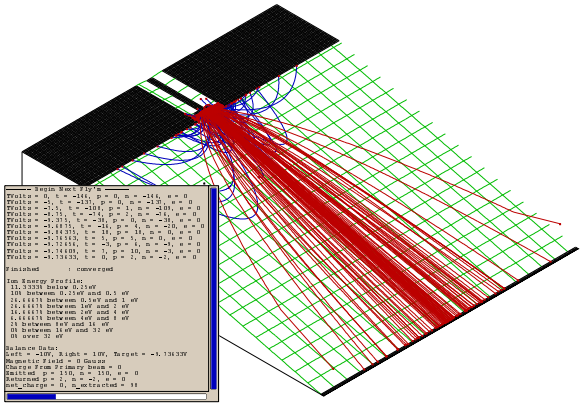
Controlling Successive Runs

- Strategy: Terminate and Initialize



Complex Multi-Run Simulations

- Monte Carlo -- Self-Charge Stabilization



Afternoon User Programming Lab

- Self-directed lab. Do what interests you.
- Options:
  - Form groups to work on a common interest
  - Explore user program demos in reference
  - Continue to explore a previous lab
  - Attack the instructors

12.0 User Programming Lab

- Self-directed lab. Do what interests you.
- Options:
  - Form groups to work on a common interest
  - Explore user program demos in reference
  - Continue to explore a previous lab
  - Attack the instructors

Demo Topics in Reference

- Critical Timing
- Boundary Approach
- Measured Magnetic Fields
- Symmetry Jump
- Non-Ideal Grids
- Monte Carlo

Short Strategy Discussions

- As Time and Interest Permits
  - Measured Magnetic Fields
  - Symmetry Jump
  - Non-Ideal Grids
  - Monte Carlo

12.a Measured Magnetic Fields

- Problem: How to simulate measured magnet fields with SIMION
- Solution: Save the magnetic field data in an ASCII file, load the data file into a user program array, and use Mfield_Adjust to define the fields

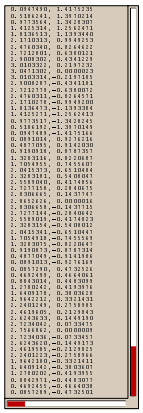
Tricks Used

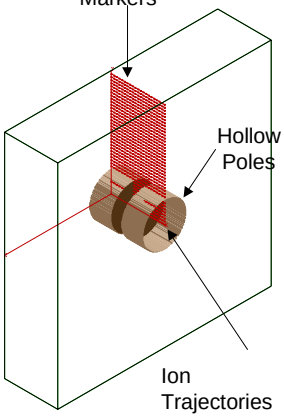
- Potential Array data used to create data file
- Data file loaded into user program array
- Mfield_Adjust uses linear interpolation of array data to define magnetic fields to SIMION

Potential Array to Data File

- Fly neutrals on array intervals
- Use hollow electrodes (edge filled)
- Use time markers and data recording

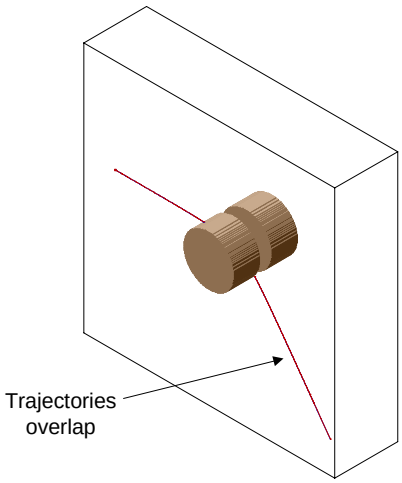
Bx and Br saved to file





Comparison Simulations

- Fly first ion in potential array's Field
- Fly second ion in data file's field
- Trajectories are almost identical



For More Information

- See the Measured Magnetic Fields demo in the user program reference directories
- Useful strategies
  - Ways to create data file from array data
  - Use of data files and user program arrays to simulate fields
    - 2D structure array addressing
    - Linear interpolation of fields
    - Conversion of 2D fields into 3D fields

12.b Ion Jumping Tricks

- Ion jumping tricks can be very useful
  - Jumping between symmetries
  - Non-ideal grid simulations
  - ?
- Other_Actions program segments are used to jump the ions.

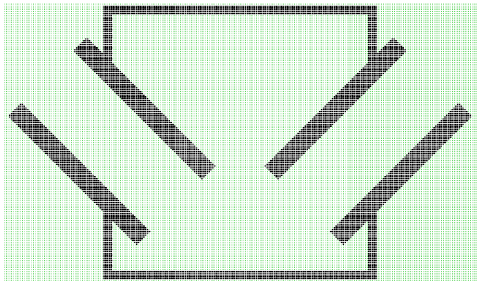
Symmetry Jump Example

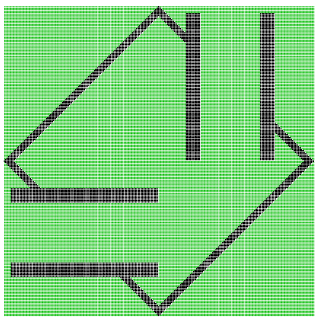
- Problem: Non-ninety degree housing symmetry causes unwanted jags near some surfaces.
- Solution: Create two or more housings with array orientations appropriate for portions of flight.

  Then jump ions between housings.

Create Housings with GEM

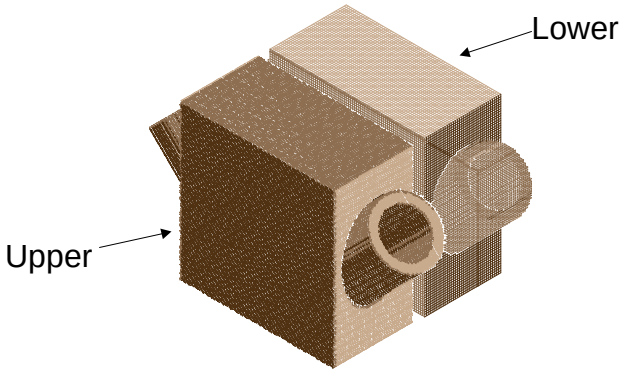
- Each housing is orientated differently





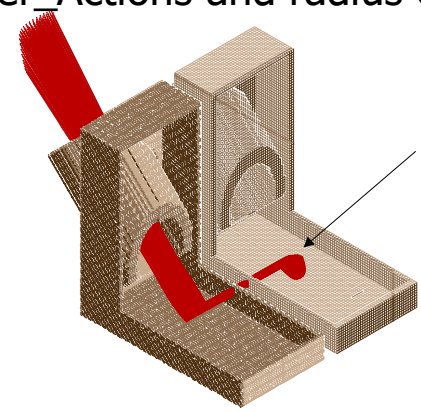
Project Housings

- Shift Housings in z and orient properly



Jump Ions Between Housings

- Use Other_Actions and radius criteria



For Detailed Information

- See Symmetry Jump Demo in user program reference section.

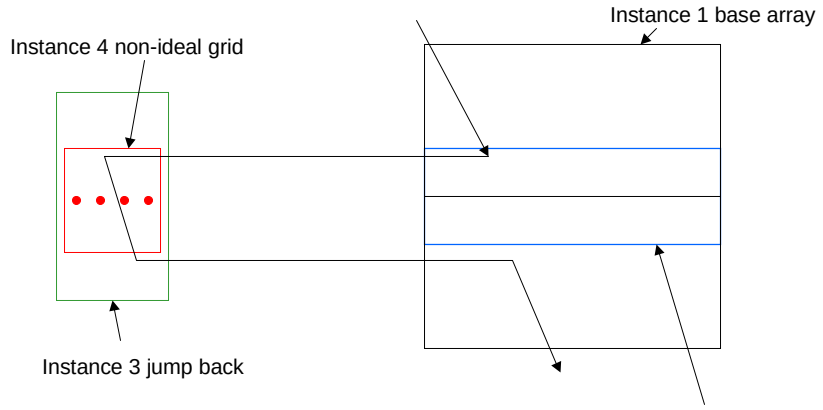
12.c Non-Ideal Grids

- Problem: Non-Ideal grid simulations can require extremely large arrays
  - Grid wires are usually very small and closely spaced
  - Remaining volume usually very large relative to grid wire dimensions
- Solution: Use ideal grids in main array and jump ions into and out of a non-ideal grid repeating volume to simulate the non-ideal grid effects.

The Tricks

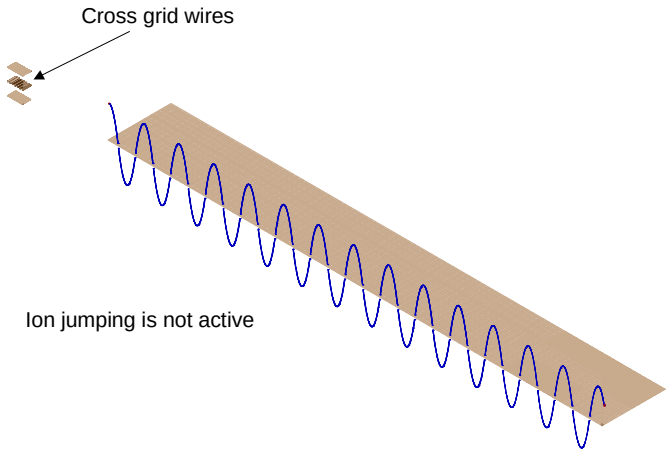
- Use a small repeating non-ideal grid pattern to simulate an arbitrarily large non-ideal grid.
- It is important that the boundary conditions match at jump in and out points
- Jump instances are used to control action (field free instances with other_actions)

The Use of Jump Instances

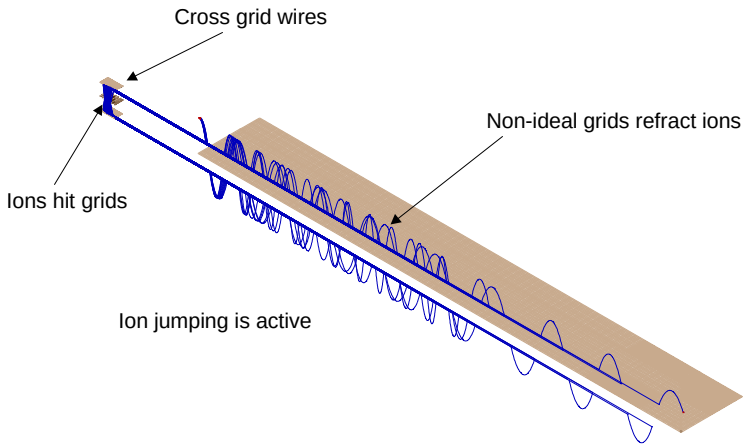


IMPORTANT: Size instances to avoid jumping onto edges

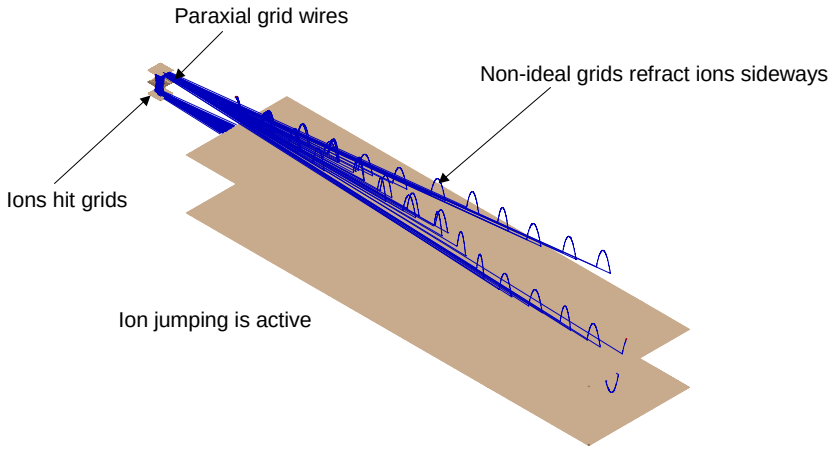
Ideal Grid Trajectories



Non-Ideal Grid Trajectories



Non-Ideal Grid Trajectories



For More Information

- See Non-Ideal grid demo in user program reference section
- Demo includes 3D square grid simulations also

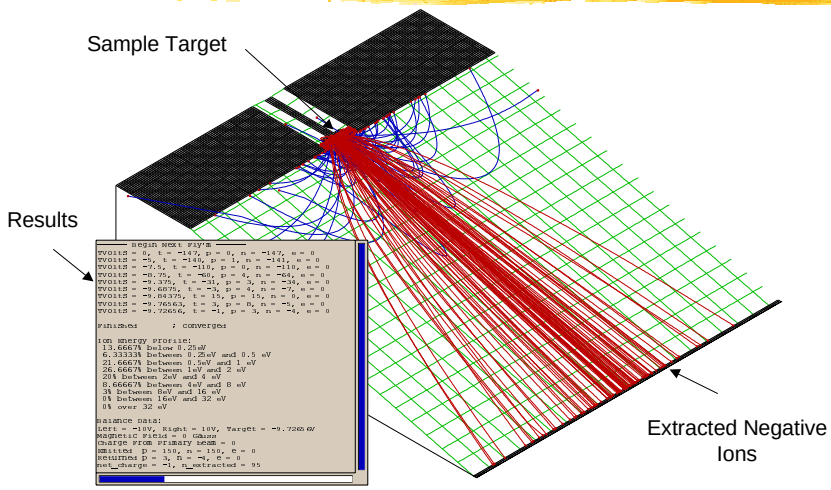
12.d Monte Carlo Simulations

- Problem: How to simulate and understand self-charge stabilization (electrostatic self-servos for sample charge control)
- Solution: Use SIMION with complex user programs and Monte Carlo techniques to investigate the nature of charge balance points.

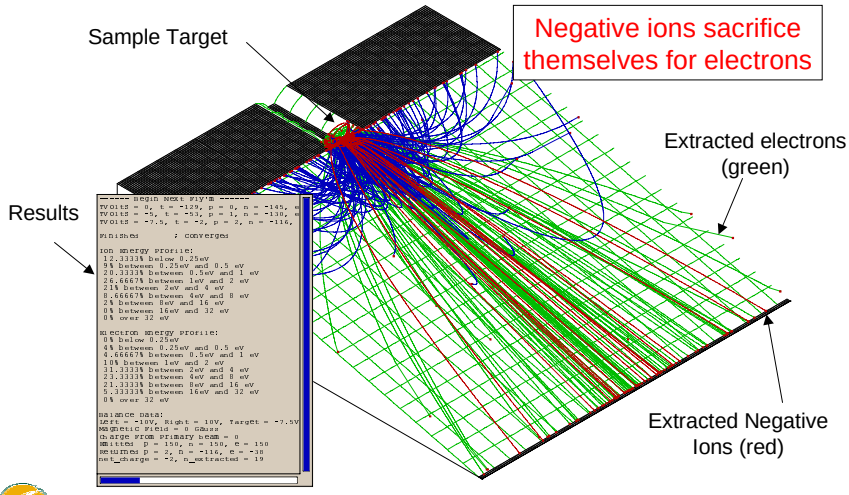
Tricks Used

- 6th order power functions used to simulate secondary ions and electron ke
- Simulation keeps track of all charge entering, leaving, and returning
- Target's potential adjusted for charging direction on each successive rerun (binary)
- Simulation terminates when charge balance criteria are met

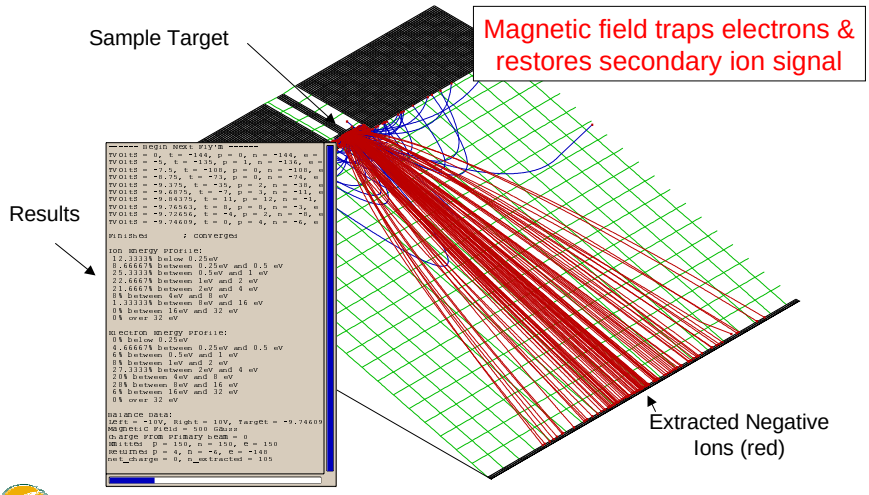
Example of Charge Balance



Charge Balance (electrons)



Charge Balance (electrons) with Magnetic Field Suppression



For More Information

- See the Monte Carlo demo in the user program reference directories
- Useful strategies
  - Complex control
  - Secondary ion and electron energy distribution
  - Data recording and display
  - Magnetic field inclusion