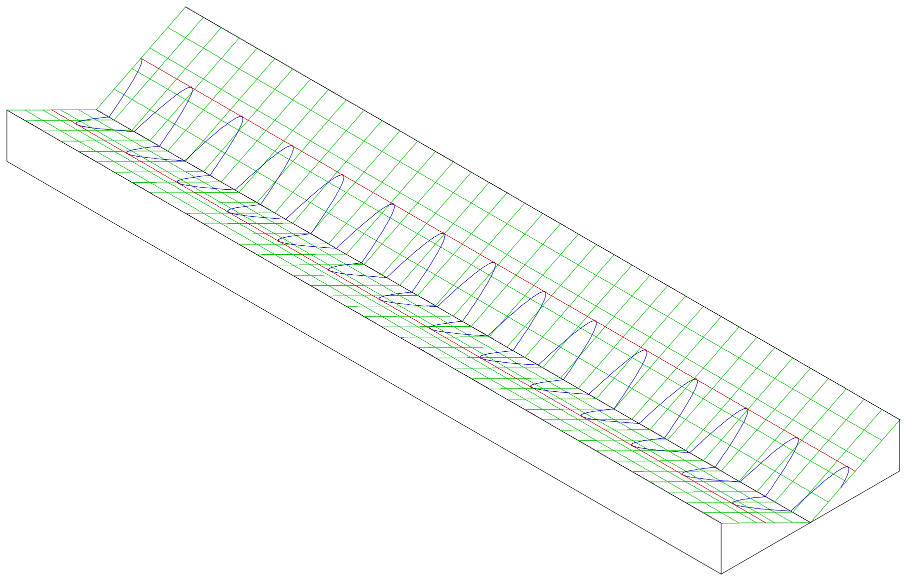
Non-Ideal Grid Ion Jump Example

Introduction

The user program example in this directory demonstrates how user programs can be applied to assist in the simulation of non-ideal grids. The primary problem with non-ideal grids is that they generally have very fine wires (e.g. 0.002” with 50 grid lines per inch). This means that the 3D array density required to model them is extremely high. Moreover, if this same density were applied to the rest of the simulation array, impossible array sizes would most likely be required. Ion jumping can provide a way out.

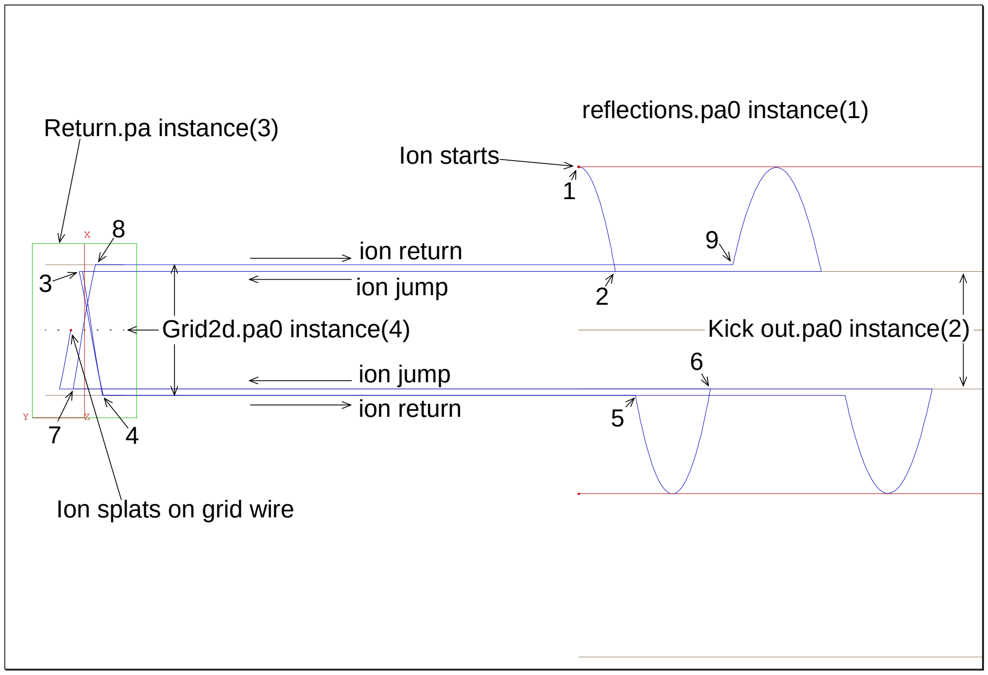
The trick

Let’s assume that a small volume around several non-ideal grid cycles could be modeled with a realistic 3D array size. Moreover, the size of the array is extended away from the grid’s surface (on both sides) to boundaries with potentials that closely match those in the same location with an ideal grid. If we could jump the ion from the main volume into the relatively small non-ideal grid volume and then jump the ion back as it exited the non-ideal grid volume, the impact of the non-ideal grid could be included in the simulation. Further, a realistic 3D array can represent the effects of an impossibly enormous 3D array by using mapping strategies that act to tile the smaller, high density array across the entire ideal grid’s surface.



Strategy

A 2D ion reflection simulation is used as an example because it is relatively simple and serves to illustrate several points. In this simulation, an ideal grid at ground potential is positioned midway between two parallel plates with potentials of 100 volts. An ion injected near one plate with an initial trajectory parallel to it will oscillate as it moves between the plates as shown in the figure above.



In order to include the effects of a non-ideal grid three additional arrays were defined. One was a small size but high density model of a region of a non-ideal grid, and the other two were simply boundary regions that controlled the kick-out (kick_out.pa0) of the ions from the ideal grid array into the high density grid array and back (return.pa). The kick-out array modeled the central region of the equivalent volume surrounding the ideal grid (**kick out.pa0 – figure above**). The kick-out array's instance was set to a higher number (higher priority) than the base 2D array's instance (**reflections.pa0**). When an ion enters the kick out instance it is automatically jumped via its user programs into the volume of the non-ideal grid's instance (**Grid2d.pa0** in this case). Moreover, the kick out instance's array mimicked the base array's field perfectly to allow the use of an adjustable variable flag (**_Ideal_grid_if_not_zero**) that signals when to skip the ion jumping and just simulate an ideal grid.

The non-ideal grid's volume (**Grid2d.pa0**) is within a larger volume of a dummy array instance (**return.pa**) that has a lower instance number (lower priority) than the non-ideal grid's instance. Thus as the ion exits the non_ideal grid's instance it always enters the instance of **return.pa**. User programs tied to this instance jump the ion back to the base array using the same xshifts and zshifts calculated in the kick out instance. The ion finds itself in the reflection instance (base array) and resumes its trajectory until it encounters the kick out instance again. At which point the whole process is repeated. You can follow this activity on the figure above by following the numbered steps 1-9. Note that the ion in the figure eventually splats on a grid wire. The grid (grid2d.pa0) is a 90% pass grid and the ion was killed on the fourth pass through the grid.

The other_actions segment in kick out.prg

The user program kick out.prg contains an other actions segment that controls ion jump out. If the user adjustable variable **_Ideal_grid_if_not_zero** is set to a non-zero value the ion jump out is aborted and the array instance simulates an ideal grid.

Otherwise the other_actions segment calculates the required x and z shift parameters required to jump the ion into the non-ideal grid's volume. These values are stored in the static variables **xshift_mm** and **zshift_mm**. The ion is jumped by subtracting these shifts from its current values for ion_px_mm and ion_pz_mm and saving the results to these variables to update the ion's position.

The calculated x and z shifts act to tile the non-ideal grid properly across the entire ideal grid. This gives exactly the same result as if an enormous non-ideal grid were used. The z shift is calculated in the following manner:

```
Rcl ion_pz_mm           ;get ion's current workbench coords in mm
25.4 /                  ;convert to inches
0.040 /                 ;convert into array interval steps (0.040") of two grid spans
nint                    ;get the nearest integer to this value of steps
0.040 *                 ;convert back to inches
25.4 *                  ;convert back to mm
sto zshift_mm           ;store in z shift static variable
```

The x shift parameter is calculated the same way except to add a 70.5 mm additional offset to account for the x position shift to the non-ideal grid instance.

The other_actions segment in return.prg

The kick out instance uses the x and z shifts to jump the ion into the grid2d.pa0 instance. If the ion does not hit a grid wire it will eventually emerge out of this instance and immediately find itself in the return.pa instance. At this point the other_actions segment in return.prg adds the x and z shifts back to the ion's current position to jump the ion back into the reflections.pa0 instance.

Safety margins in y

If you study the jumps in the figure above, you will see that the ion jumps progressively outward in absolute y (the vertical center is y = 0). If the size of the kick out instance and the non-ideal grid were identical, there is the risk of one bit positioning errors landing the ions where you didn't want them to go (in the wrong instance). This has happened for a formerly well behaved simulation when a new SIMION 7.0 Beta from a different compiler was run with it. In all probability some of the intermediate calculation numbers remained on the 80 bit floating point stack for a few extra cycles instead of being rounded off and being stored back to 64 bit numbers in ram between calculations. The resulting 1 or 2 bit difference was enough to introduce chaos. ***Moral: Make sure your jumps always land inside (not on the edge of) where you want to go. Ignore this at your peril!***

Running the non-ideal grid simulations

The following information guides you through the six simulations provided.

Initial preparation

The following files must be loaded and refined before any simulations can be run: **reflections.pa#** and **kick out.pa#**.

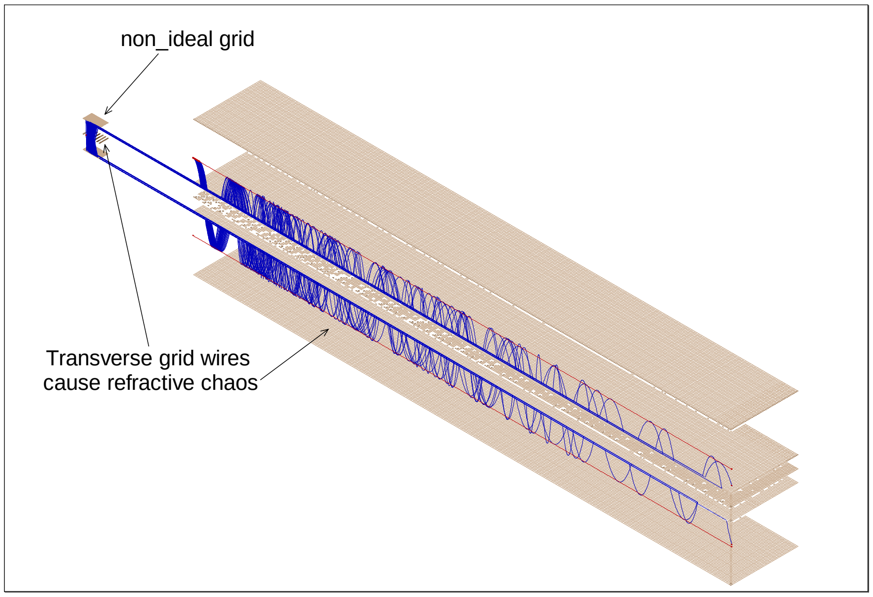
Running ideal grid reflections simulation

Remove all PAs from ram and load the **reflections.iob** file into view. Click the **fly'm** button and the ideal grid reflection simulation shown in the first figure will be run.

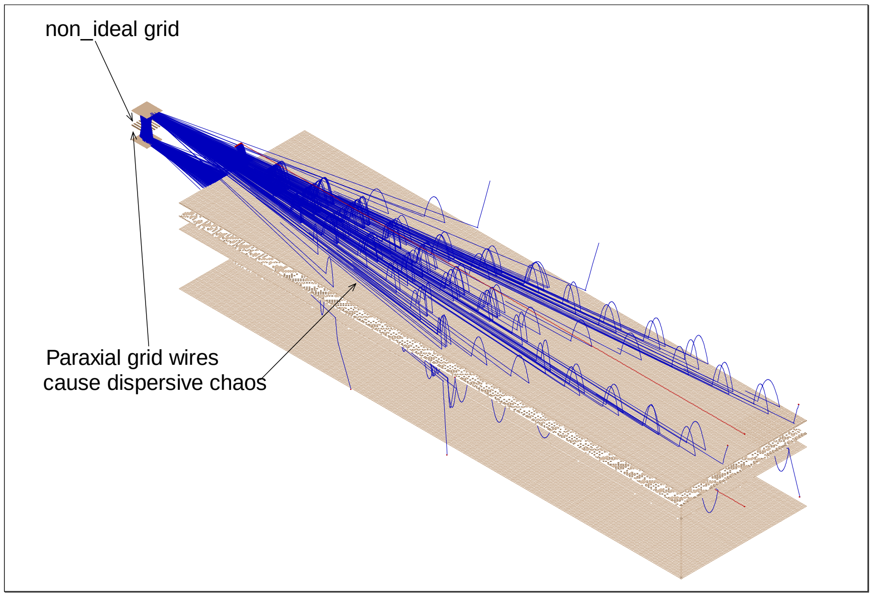
The non-ideal 2D grid reflections simulation

Initial preparation: Use the new function with the grid2d.gem file to create the non-ideal 2D grid array. Save the array as grid2d.pa# and refine it.

Remove all Pas from ram and load the **reflections 2Da.iob** file into view. Click the fly'm button and observe the ion trajectories. The 2D non-ideal grid has transverse grid wires. These wires tend to pseudo-randomly refract the ions in the x direction killing the pattern in about one cycle (see figure below).



Remove all Pas from ram and load the **reflections 2Db.iob** file into view. Click the fly'm button and observe the ion trajectories. The 2D non-ideal grid has grid wires running in the x direction. These wires tend to pseudo-randomly refract the ions in the z direction dispersing the ions in the volume between the plates (see figure below).



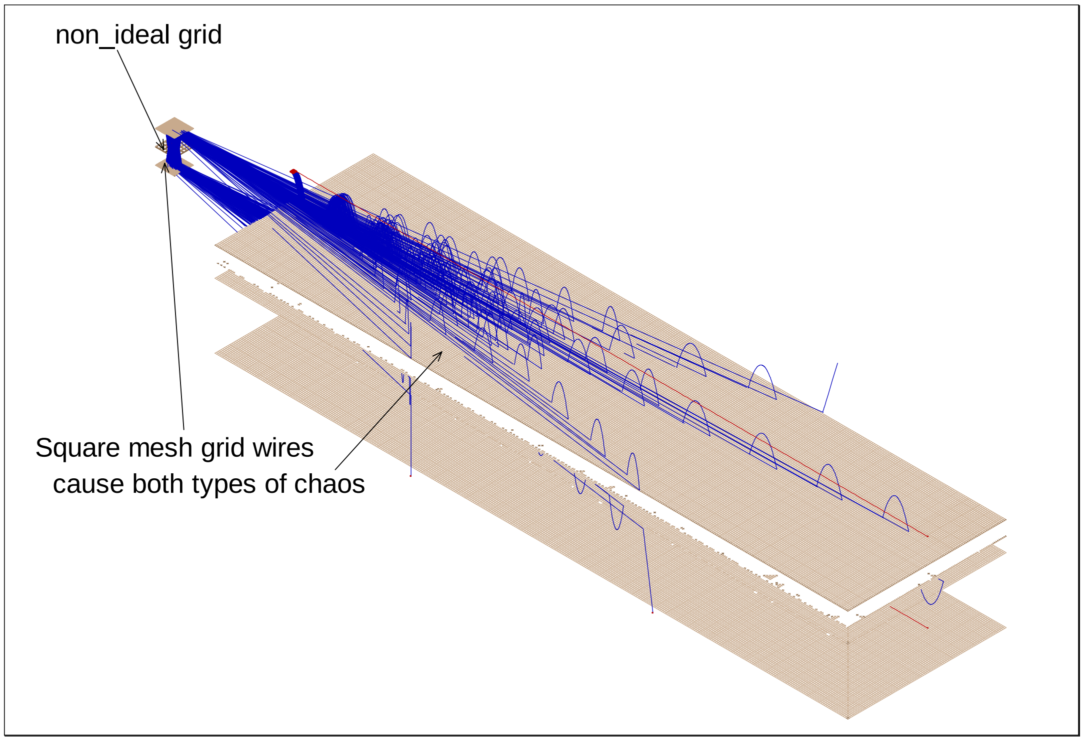
The 3D non-ideal grid simulations

For the 3D simulations equivalent to the two 2D versions demonstrated above you will need to use New and the **Grid3D1.gem** file to create a 3D array with grid wires in only one direction. Save the array as **Grid3D1.pa#** and then refine it (this could take awhile).

Now remove all PAs from ram. The files **Reflections 3Da.iob** and **Reflections 3Db.iob** perform the same simulations as their equivalent 2D version described and illustrated above except with 3D arrays. **Moral: Use 2D arrays whenever you can**.

The last 3D simulation uses a square mesh non-ideal grid. You will need to use New and the **Grid3D.gem** file to create a 3D array with grid wires in both directions. Save the array as **Grid3d.pa#** and then refine it (all you have is time).

Now remove all PAs from ram. Load the file **Reflections 3D.iob** into view and click fly'm. Note that square mesh non-ideal grids combine both forms of chaos shown above (see figure below).


non_ideal grid
Square mesh grid wires
cause both types of chaos