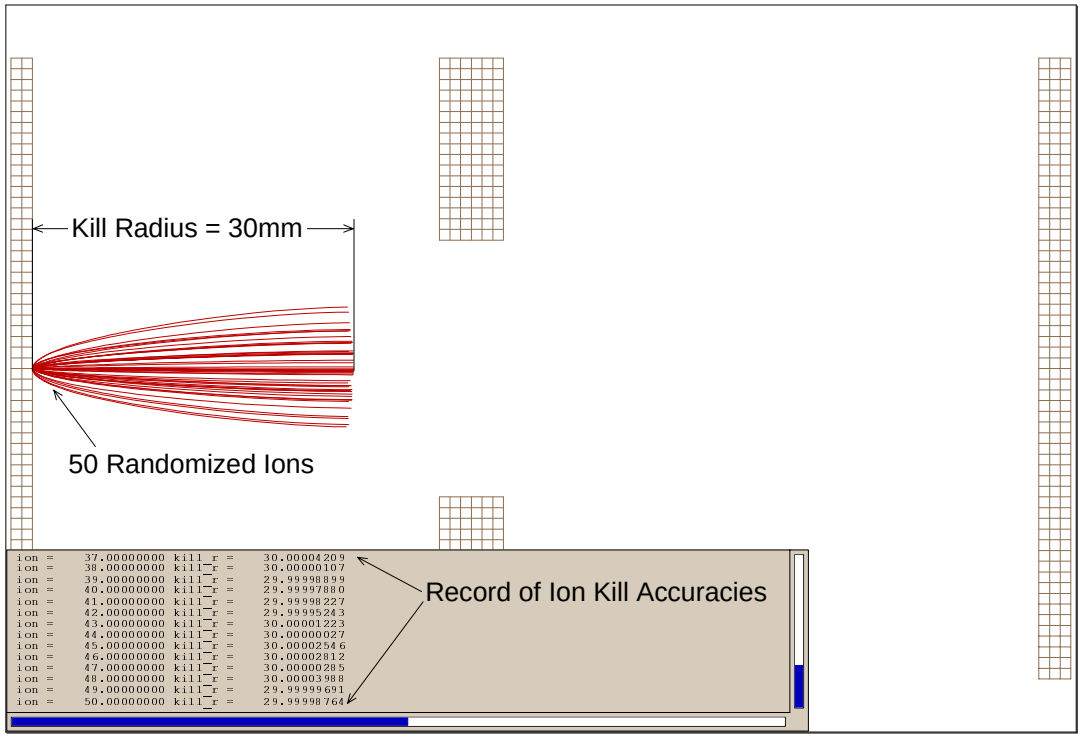
Boundary Approach Example

Introduction

The user program example in this directory demonstrates how to accurately approach a boundary. Time boundaries can be approached exactly using the tstep_adjust program segment as demonstrated in the Critical Timing directory examples. However, things are not quite so simple if you need to approach a physical position boundary (e.g. 3D radius distance from ion's origin shown in the figure below) or a velocity boundary or what have you.



Although this example demonstrates how to approach a 3D radius boundary (position boundary), the basic concepts presented here can be extended to most any other boundary problem. Unlike time step boundaries (which can be nailed exactly), other boundaries need to be approached via some strategy. The approach used in this example is SIMION's normal binary boundary approach strategy. However, most of the concepts presented can be applied with other (perhaps faster) methods that you might devise or use.

Strategy

The basic boundary approach strategy is to use the other_actions segment to control the whole process and use the tstep_adjust segment to selectively control time steps as directed by the other_actions segment.

Functions of the other_actions segment

The scheme is to have the other_action segment monitor the ion's location. As long as the ion's new location is inside of the kill radius minus the user defined error range, the other_action segment merely saves a copy of the ion's current 3D location and velocity numbers as well as its time of flight.

If the ion's position is contained within the kill radius band the ion is marked for killing on the next time step so that the current trajectory line segment will be drawn.

If the ion's position exceeds the outer limit of the kill radius, the ion's position, velocity, and time of flight are restored to the previously saved values. The most recent time step is also saved for use by tstep_adjust. A flag is set for halving the saved time step.

Function of the tstep_adjust segment

The tstep_adjust segment checks the halving flag and exits if it is not set. However, if the halving flag is set the time step is halved and saved as the new time step for the ion, and the halving flag is reset to avoid uncontrolled looping lockups.

Thus the ion will approach the kill zone. If it jumps across it the user program segments will automatically shift it back in time, halve the time step and try again. This process will continue until the ion lands squarely within the kill zone at which point it will be killed.

Adding the excitement of arrays

In order that this example wouldn't be too boring, arrays are used to track up to 1000 ions simultaneously. This serves to demonstrate the usefulness of arrays. If we were only going to fly one ion at a time arrays wouldn't necessary. However, the arrays allow the ions to be flown grouped, because each ion's prior time step conditions can be saved separately.

Running the example

Preparation: Load and refine the array **Boundary Kill.pa#**, and remove all PAs from ram. Now load the file **Boundary Kill.iob** into view and click the **Fly'm** button.

You can adjust the kill radius as well as the kill band. Fly the ions singly or in groups to see that array storage keeps it all together.

This example serves as a good starting and looting point for creating code for almost any boundary approach problem. Have fun.