SANDIA REPORT
SAND2009-1260
Unlimited Release
Printed February 2009

Ion Trap Simulation Tools

Benjamin R. Hamlet

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories

SAND2009-1260
Unlimited Release
Printed February 2009

Ion Trap Simulation Tools

Benjamin R. Hamlet
Next Generation Monitoring Systems
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-MS0401

Abstract

Ion traps present a potential architecture for future quantum computers. These computers are of interest due to their increased power over classical computers stemming from the superposition of states and the resulting capability to simultaneously perform many computations. This paper describes a software application used to prepare and visualize simulations of trapping and maneuvering ions in ion traps.

ACKNOWLEDGMENTS

Work described in this report was completed with funding from the Developing Key Capabilities for Quantum Computing LDRD, project number 93522 [16].

CONTENTS

FIGURES

NOMENCLATURE

BVI	Binary Vertex-Index file format used for storing polygonal meshes.
CAD	Computer Aided Drafting
CGDS	Converted GDSII file format. Created for this project.
DOE	Department of Energy
GDSII	industry standard file format for integrated circuit design files
GPC	General Polygon Clipper
GUI	Graphical User Interface
ION	Simion ion motion file format
OSG	OpenSceneGraph
PA	Simion Potential Array file
SL	Simion script file
SNL	Sandia National Laboratories
STL	Stereolithography file format
VFW	Vector Wave Form file format used by Quartus II

1. INTRODUCTION

This document contains information on the process necessary to visualize physical simulations of ion traps in Umbra [1]. The ion traps are initially described in a 2D CAD format. This geometry is triangulated and extruded into a 3D representation of the ion trap, which is then simulated with the Simion [2] physics package. Electrodes are controlled within Simion through a hardware simulation developed in Quartus II [3]. These three pieces are combined into a comprehensive, Umbra based visualization. Please refer to *Control Electrode Switching Frequency and Induced Micromotion in Trapped 9Be+* [4] for a description and results of a physical ion trap simulation related to this work.

2. GDSII LOADING, TRIANGULATION, EXTRUSION, AND FILE FORMAT CONVERSIONS

This portion of the document describes how to use the Umbra based applications developed to prepare 2D GDSII data for physical simulation using Simion. The tools are general enough, however, that they can be used for any application requiring the triangulation and extrusion of GDSII data. A variety of output formats are provided. In addition, a high level overview of the algorithms and third-party software used in the toolchain is presented.

2.1 Overview

A variety of data paths are available when using these simulation tools. Each major step of the toolchain results in the creation of an output file containing the current state of data translation. While the output of each step is meant to be used as input to the next step, these intermediate files are useful for manipulating data outside of the standard toolchain. Figure 1 provides an overview of the conversion process.

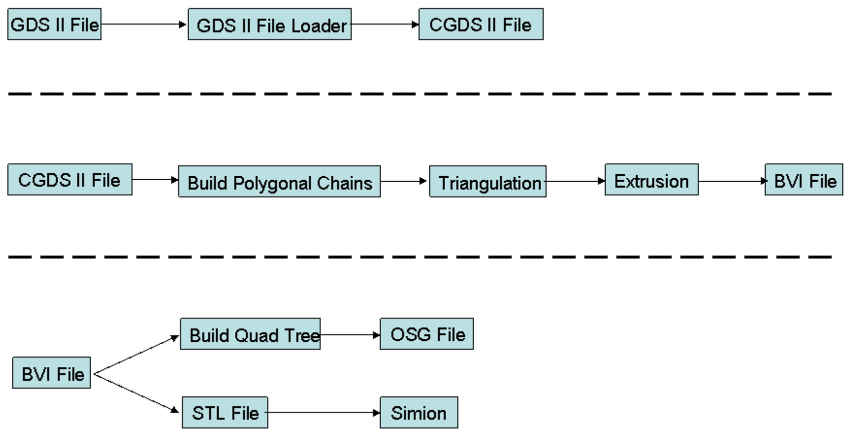


Figure 1. Overview of dataflow in the toolchain. Dashed lines represent divisions between intermediate steps of the overall toolchain.

1. Step 1 of the process converts a GDS II file into a “Converted GDS II” (CGDS) file. A CGDS file is a flattened, ASCII representation of the original GDS II file.
2. Step 2 converts a CGDSII file into a triangulated and extruded 3D mesh. A series of text input files are used to control the process, including the extrusion amounts and filters to select which portions of the geometry should be triangulated and extruded. The result is stored in a “Binary Vertex-Index” (BVI) file.
3. Step 3 takes a BVI file and converts it into an OSG file for use with Umbra, into an STL (Stereo lithography) file for use with Simion, or both.

These steps are discussed in detail in the following sections.

2.2 Step 1: GDSII to CGDS

GDSII is a compact two dimensional CAD format. A GDSII file is organized into layers, and each layer composed of various structures. In particular, file sizes are kept to a minimum through the use of “structure references” and “array references.” These allow a particular piece of geometry to be defined once and then replicated at many different positions and orientations. For instance, a designer can draw a circle structure one time and then replicate the circle through commands that perform actions such as “copy this circle 20 times in the horizontal direction and 15 times in the vertical direction starting at position (10, 10).” This saves space in the GDSII file, but such references must be replaced with the actual geometry before the triangulation process. This replacement is called “flattening” the GDSII file.

Step 1 of the toolchain is to load a GDSII file, flatten the file, and then save the results to CGDS files. One or more CGDS files are created for each layer in the GDSII file, depending on input parameters to the conversion function. This step makes use of an open source application called Boolean [5]. Boolean is used to load and flatten the GDSII file, and output driver was written that saves the result to a CGDS file. The geometry stored in the CGDS files have units of mm.

2.2 Step 2: CGDS to BVI

Step 2 contains most of the work for preparing the GDSII geometry for simulation in Simion. This step involves the triangulation and extrusion processes necessary for generating 3D geometry from the original 2D data. The first step in this process involves loading a CGDS file and building polygonal chains out of the data. Figure 2 helps explain this process. As the figure shows, a polygonal chain is the geometry that results from the union of two or more overlapping polygons. The term chain is used because an iterative algorithm is used which grows the chain by adding on links (additional polygons) that somehow overlap the existing chain. A chain emerges wherein many polygons (which may have a high degree of spatial separation) are linked together through intermediate polygonal links. Polygonal chains are built of the maximum size possible given the input data, so the result after building polygonal chains is a set of mutually disjoint (non-overlapping) polygons.

The polygonal chains are built to eliminate overlaps in the original GDSII data. If the overlapping regions were not eliminated, the triangulation process (and the resulting 3D data) would contain overlapping triangles. Such triangles produce visual artifacts and, perhaps more importantly, are not allowed by the STL format required by Simion. The boolean operations performed during this process were carried out in part by the open source General Polygon Clipper (GPC) [6] package.

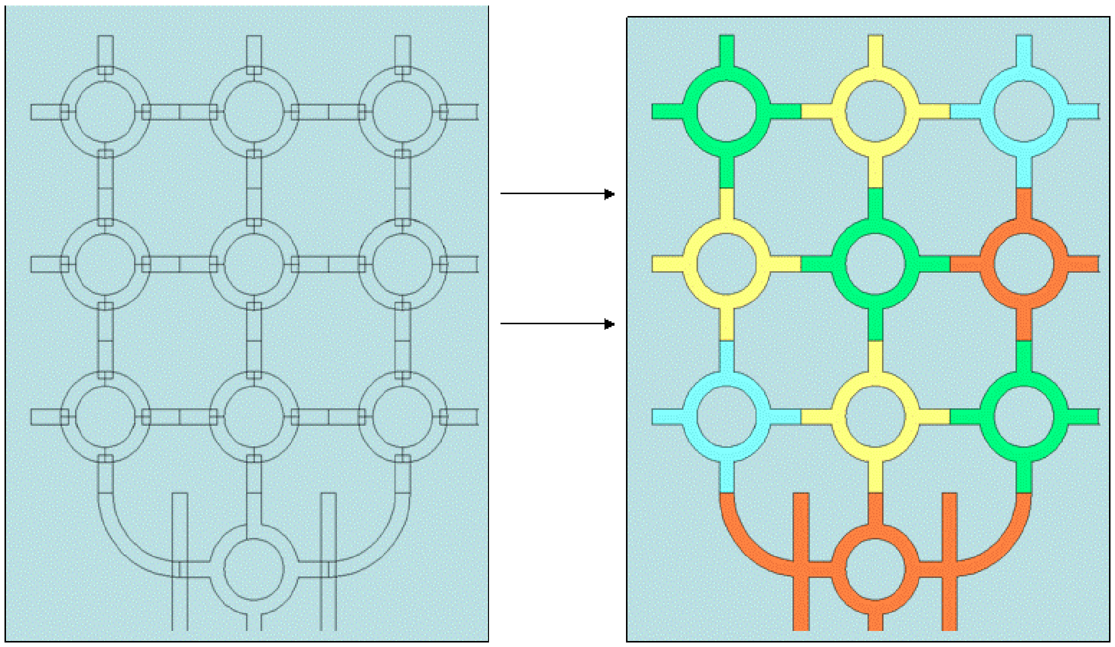


Figure 2. Handling overlapping polygons. The image on the left shows a detail of a CGDS file. Note the overlapping polygons. The image on the right shows the same data after building polygonal chains; the chains have been colored to show their boundaries.

The next step after creating polygonal chains is triangulating those chains. This is done with a constrained Delaunay triangulation. This means the triangulation process takes as input a series of vertices and line segments (the polygonal chains) and triangulates the data with the restriction that each of the original line segments are edges in the triangulation. In general, this will not result in a true Delaunay triangulation. Due to this, and to the desire to create high quality meshes, the toolchain contains an option to create "visual quality" and "simulation quality" meshes. Both meshes will look the same when drawn as solids. However, the simulation quality setting will take longer to run and generate larger files as it uses a "Constrained Conforming Delaunay Triangulation" with the additional restrictions of limits set on the minimum angle allowed per triangle and on the maximum allowable area per triangle. Such a triangulation is Delaunay and tries to create triangles that are as close to equilateral as possible (due to the angle constraint). The area constraint is for simulation packages that will incorrectly distribute values (such as voltages across the surface of an electrode) if triangles are of radically different sizes. As it turns out, the Simion package does not have these constraints and so a "visual quality" mesh is acceptable when using that package. Packages implementing Delaunay triangulation are widely available on the Internet, including an implementation in OpenSceneGraph [7]. This implementation was found to be slow and occasionally buggy, so this toolchain uses a well-known implementation developed by Jonathan Shewchuk [8].

The third step in creating a 3D mesh from the CGDS file is extruding the geometry. This step takes geometry from triangulated 2D data into a 3D mesh. This is performed by creating a copy of the triangulated 2D data, adding an extrusion amount to the vertical component of each vertex, and connecting the top and bottom portions with sidewalls. The sidewalls are created by finding the outside edges of triangles on the perimeters of the triangulated polygonal chains. If the result is a “visual quality” mesh then each sidewall is composed of two triangles. If the result is a “simulation quality” mesh then each sidewall is composed a series of triangles conforming to a minimum angle constraint. In addition, the STL format required by Simion specifies that adjacent triangles must share an entire edge. This means that a vertex of one triangle may not lie on the edge of another triangle; the vertex must be coincident with a vertex of the other triangle. Because of this constraint, all sidewalls on a solid object are subdivided at the same level as the most highly subdivided sidewall.

This phase of the conversion process takes two separate input files. The first describes the amount of extrusion to perform for each layer. The extrusion amount is in the same units as that of the CGDS file; the default being mm. The second describes one or more filters to apply to the CGDS data. A filter is a polygonal region used to clip out geometry – any polygon that has one or more vertices outside the region defined by the filter polygons is clipped. Filters can be used to isolate specific portions of geometry, making them useful for selecting the portion of a design needed for simulation. They can also be used to limit the size of output mesh files by splitting a CGDS file into several regions.

Finally, the triangulation and extrusion process is complete and the results are written to a “Binary Vertex-Index” (BVI) file. This is a binary file format that contains a list of each vertex used in the mesh (without repetition), a list of indices that specify how to create triangles out of the vertex data, and a list of triangle normals. BVI files are necessary for the third step in the conversion process, which will convert geometry into formats used by Umbra and Simion. Figure 3 shows a portion of a triangulated and extruded GDSII file similar to the one seen above.

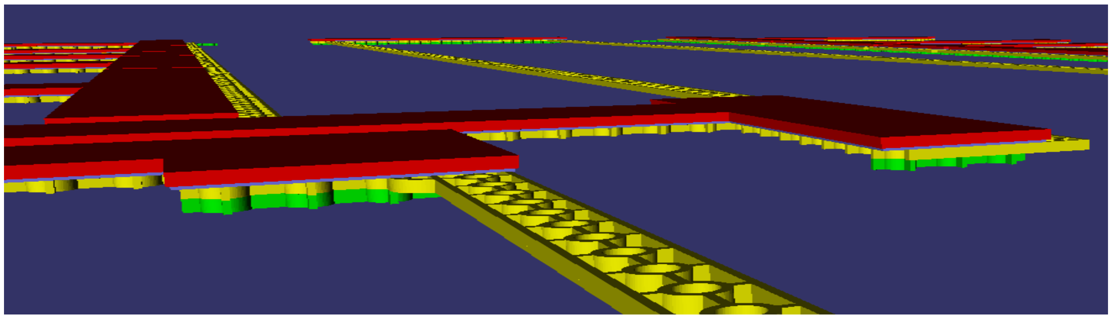


Figure 3. Results of the triangulation and extrusion process applied to a GDSII file.

2.3 Step 3: BVI to STL and BVI to OSG

The main reason to create a BVI file as the output of the triangulation and extrusion process is to allow the 3D data to be easily converted to formats needed by various applications. The two formats of immediate concern are the OSG format, a simple 3D file format native to OpenSceneGraph (and thus Umbra) and the Stereo Lithography (STL) format that Simion uses.

The OSG format contains vertex data, index data to form triangles from those vertices, normal data, and color data. Figure 3 above actually shows an OSG file as seen in Umbra.

The STL [9] format is composed of a set of unordered triangle facets. Each facet includes three vertices and a normal. It is a simple format meant to describe sold geometry to rapid prototyping machines. This application space is why the restrictions described above (triangles must not non-overlap, adjacent triangles must have coincident edges) are placed on valid STL files. Geometry not following these restrictions is likely to contain holes. Such holes are also undesirable in a physical simulation, so the STL restrictions are appropriate for this toolchain. Essentially, valid STL geometry should describe "water tight" solids. Figure 4 shows an STL file as viewed in Simion, both before and after the simulation's refinement process.

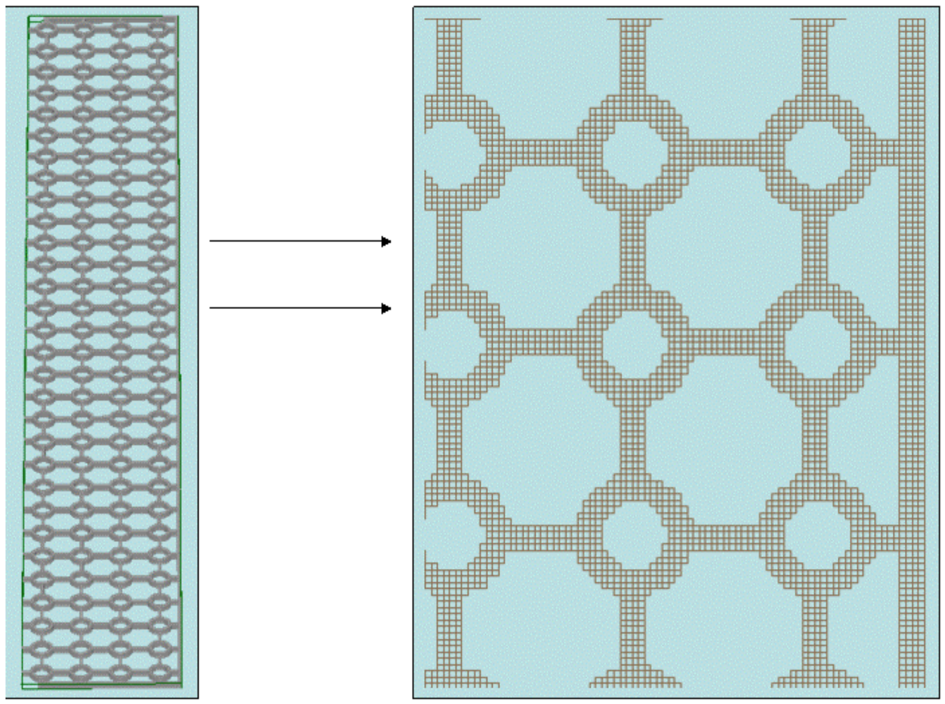


Figure 4. Ion trap geometry in Simion. The image on the left shows an STL file in Simion, on the right is a detail of the same image as used by the simulation. Note that the 3D structure has been approximated by a series of cubes.

2.6 Using the Toolchain

This section will describe the Umbra commands necessary to run the toolchain described above. The various open source applications described, such as Boolean, Triangle, and GPC, have been integrated into Umbra, streamlining the toolchain. A single command can be used to convert from a GDSII file into 3D OSG and STL files. The command is described first, followed by instructions on how to independently run various portions of the toolchain.

2.6.1 General Notes on the Umbra Command Line

The commands described here are run through Umbra's command line. This command line is actually a TCL interpreter [10], so any TCL commands or scripts can also be run on this command line.

Each step of the toolchain is compiled into an Umbra library called qc. To load this library, type

```
> umb::load qc
```

on the command line. From this point on, any command that is part of qc will have a prefix of `qc::`

Along the same lines, most of the functions users will call are member functions on objects of a class called `qc::CADExtrusion` To create an object of this command, type the following on the command line:

```
> qc::CADExtrusion objectName
```

Where “objectName” can be replaced with anything that makes sense.

Finally, the default path for the command line is the Umbra `bin` directory. Thus, any relative paths found below (those that start with “`../`”) start from the Umbra bin directory. Therefore, “`../et/qc`” is actually `Umbra/et/qc`.

2.6.2 Converting from GDSII to STL, OSG, or Both

As mentioned above, a single command can be used to convert from GDSII to STL or OSG. The format of the command is:

```
> qc::loadGDS gdsFile filterFile extrusionAmountsFile qualitySetting \
              outputFormat outputDir maxPolygonssPerFile
```

This is an example of that command:

```
> qc::loadGDS ../et/qc/Planar_Module2.gds \
../ET/qc/filters/electrodeTight.txt \
../et/qc/planarModule2_extrusionAmts.txt 0 3 \
../et/qc/planar_module2_all_data_filtered/ 15000000
```

Please note that the ‘\’ characters should not be typed, they are listed to show the continuation of the previous line. Thus, the above command should be typed onto one line on the Umbra command line.

The command breaks down as follows:

- `qc::loadGDS` is the command being run. It is a global function (not a member function of a class) and is essentially a wrapper around the various objects and commands necessary to convert a GDSII file into an STL or OSG file.
- `../et/qc/Planar_Module2.gds` is the original GDSII file being converted.
- `../ET/qc/filters/electrodeTight.txt` is a file describing the polygonal filter used to select the portion of the GDSII file to triangulate and extrude.
- `../et/qc/planarModule2_extrusionAmts.txt` is a file describing the amount to extrude each layer of the GDSII file. As mentioned above, extrusion amounts are in the same units as the CGDS file and default to mm.

- `0` means “create a visual quality mesh.” This can be replaced with a 1 which would mean “create a simulation quality mesh.” As mentioned above, a “visual quality” mesh can be used with Simion.
- `3` means “generate both OSG and STL files.” This can be replaced with a `0` to mean “only generate BVI files,” a `1` to mean “only generate OSG files”, or a `2` to mean “only generate STL files.”
- `../et/qc/planar_module2_all_data_filtered/` is the output directory. This is the base directory for the output files, and will end up containing many subdirectories. These are further explained below.
- `15000000` is the maximum number of polygons that can be stored in a CGDS file. If a layer of the GDS file contains more than this many polygons, then the layer will be split into more than one CGDS file.

Note, however, that the final two parameters are optional. If they aren’t specified, the defaults are `C:/home` for the output directory and `10,000,000` for the maximum number of polygons per CGDS file.

The format of the various input files are discussed in Appendix A.

As mentioned, the base output directory will contain several different files and subdirectories after the conversion process completes. In the base output directory is a series of `.cgds` files. There will be one or more of these for each layer in the original GDS file (depending on the maximum number of polygons parameter discussed above). This is also a `.tcl` file and a subdirectory with names similar to those of the CGDS files. The TCL file is used during the conversion process, and the subdirectories contain the converted data. The `.txt` file named `[original GDS file]_layers.txt` contains a list of the layers in the GDSII file that have been converted into CGDS files.

Contained within subdirectories holding converted data are several more subdirectories. Of particular importance is the `bvi/` directory. It is here that the `.bvi .osg` and `.stl` files generated by the toolchain are found. There is also a single `[cgds_filename]_filenames.txt` file that contains a list of all of the CGDS files that were created for a particular layer from the GDSII file. If the maximum number of polygons parameter to the `qc::loadGDS` command was smaller than the number of polygons in one of the layers, then several CGDS files will have been created. This file will contain a list of all of those files, and is used to insure all of the `.bvi`, `.osg`, and `.stl` files corresponding to that layer are generated in the same directory. All the other files and directories are used internally and do not contain any output data.

2.6.3 Converting from BVI to OSG or STL

If the qc::loadGDS command has already been used to convert a GDSII file into BVI files but it is preferred for the data to be in OSG or STL format, a series of TCL commands are provided to perform these conversions. This will save time over converting directly from the GDSII file.

The conversion from BVI to OSG is straightforward. The commands required are listed below, followed by a description of what they do.

```
> umb::load qc
> qc::CADExtrusion cad
> cad connect ir_scene scene
> cad loadBVI ../et/qc/Planar_Module2_data/ridges0_data/bvi/ridges0.bvi
```

The overall result of this series of command is the creation of a rdiges0.osg file in the same directory as the original ridges0.bvi file. The OSG file is then drawn in the Umbra display window.

As discussed earlier, the first line loads the QC toolchain library. The second line creates an instance called "`cad`" of the `qc::CADExtrusion` class. The third line (an example of the Umbra "connectors" paradigm) tells the object `cad` to use the default Umbra display for all graphical operations. Finally, the fourth line actually performs the conversion. Simply send the command a BVI filename as a parameter, and the QC library will load the BVI file, convert it to OSG, save the OSG to disk, and display the results. As an intermediate step, the data is stored in a quad tree within the OSG file to speed up the drawing process.

The conversion from BVI to STL is almost identical to the conversion from BVI to OSG. The only difference is the fourth line, which should be changed to

```
> cad bviToSTL ../et/qc/Planar_Module2_data/ridges0_data/bvi/ridges0.bvi
```

in order to create an STL file in the same directory as the original BVI file.

2.6.4 Converting from OSG to STL

There is also a command for converting from OSG to STL formats. This command is useful when, for instance, an OSG file has been created, edited in Maya [11] or some other 3D modeling package, exported to an OSG [12], [13] file, and would like to simulate the results with Simion. This command is almost identical to those used for converting from BVI to OSG and STL. Here is an example of the command:

```
> cad osgToSTL ../et/qc/Planar_Module2_data/ridges0_data/bvi/ridges0.osg
```

which will create a ridges0.stl file in the same directory as the original OSG file.

2.6.5 Filtering BVI Files into Many Separate OSG Files

Finally, it is sometimes useful to separate out portions of a BVI file and then save those portions as individual OSG files. This allows a visualization tool such as Umbra to manipulate the geometry separately. This is important for the overall ion trap visualization, where each electrode is colored independently to show its voltage. The command to run this filtering process is as follows:

```
> cad filterToElectrodes bviFile.bvi filtersFile.txt
```

This, of course, assumes that the QC library was loaded and the cad object created as described above. The BVI file passed to this command is any BVI file created by the toolchain. The filter file contains a list of filter, OSG output file, and low quality OSG output file triplets. The OSG output file is a direct conversion of the BVI file passing through the filter and the low quality

OSG output file is a bounding volume representation of the BVI geometry passing through the filter. Two OSG files are created to allow a level of detail (LOD) system to be setup within Umbra. This essentially allows the lower quality geometry to be displayed when the camera is far enough away from the geometry that the details of the regular file cannot be seen. This allows for a great decrease in rendering time. As with the other file formats mentioned, the specifics of the filter file are described in Appendix A.

3. UMBRA VISUALIZATION OF PHYSICS SIMULATION

Having prepared the ion trap geometry for physical simulation, the next step is to run the simulation and gather the output for visualization within Umbra. As mentioned earlier, the Simion package is targeted by this toolchain. Simion is a far too complicated piece of software to be adequately described here, so please see the user manual provided with Simion for such a description. Instead, this portion of the document will describe an overview of the steps necessary to run the Simion simulation, to gather the output of the simulation, and to visualize that output in Umbra.

3.1 Simion

Simion takes as input the ion trap geometry and a program describing how voltages for individual electrodes within the ion trap change through time. A subdirectory should be created in a subdirectory under Simion to hold all of the files described below.

3.1.1 Filtering Geometry Input

Copy the STL file containing the ion trap geometry into the Simion subdirectory, and convert it to a PA file. To do this, right click on the file, select “Simion SL Toolkit Functions” and then follow the onscreen directions. Having done this, load the PA file in Simion, refine it, and fly ions through it. All of these details are described in the Simion user’s manual.

3.1.2 Electrode Voltages Input

The electrode voltage control is provided through the output of a Quartus II simulation. Please see the Quartus II documentation for details on using the software. Of importance here is the usage of the Quartus II output (Vector Wave Form, or VWF) files within Simion. The salient portion of a VWF file is a series of “transition lists” which describe the waveform for input and output signals is a format similar to many nested for() loops within a typical programming language. This format, of course, cannot be used directly in Simion. Instead, a program was written to convert data from a VWF file into a Simion array file.

This conversion is performed using a program called VWFConv. This is a Windows Console Application, run from the Windows command line as follows:

```
> vwfconv.exe inputFile outputFile [signal pattern = ""] \
              [output type (simion / c) = simion] \
              [low voltage = -1.65] [high voltage = 1.65] \
              [voltage multiplier = 1.0] [output only = true]
```

As before, “\” characters are used to represent continuation of the command and should not be typed. Options listed in brackets “[…]” are optional, with default values set as shown in the brackets. Here is what the options mean:

- `inputFile` is a VWF file
- `outputFile` has a meaning that depends on the output type (described below). If the output type is “`c`” then all of the output will go in outputFile. If the output type is “`simion`” then `outputFile` is used as a prefix that is appended to the signal names. Thus, each signal’s transition list will be stored in a file named `[outputFile]_signalName.txt`
- `signal pattern` is a pattern used to determine which signals from the input VWF file should be converted and stored in the output file(s). For instance, a pattern of “`vout`” selects all signal names containing “`vout`” for output. Any signal not containing “`vout`” would not be converted for output.
- `output type` can be either “`c`” or “`simion`” The first value stores the output as C-style arrays in a C header file. The second option creates Simion arrays in individual text files as output.
- `low voltage` specifies what voltage logical low represents
- `high voltage` specifies what voltage logical high represents
- `voltage multiplier` is a constant value that both low voltage and high voltage are multiplied by. Thus, logical low is actually (low voltage * voltage multiplier) and logical high is actually (high voltage * voltage multiplier).
- `output only` can be either “`0`” or “`1`”. 1 means that only output signals from the VWF file are converted, and `0` means that both input and output signals are converted. This option works in conjunction with the “`signal pattern`” option, such that a signal pattern of “`vout`” and output only set to `1` would select for conversion only those output signals containing “`vout`” as a portion of their name.

Here is an example command used to convert all of the output signals containing “vout” as part of their name to Simion array files:

```
> vwfconv.exe qcontrol.sim.vwf sim vout simion -2.0 2.0 1.5 1
```

Note that logical low takes on the value of -3.0V and logical high takes on the value of 3.0V

Next, copy all of the sim_*.txt files into the same Simion subdirectory that contains the STL and PA files created above.

Now, the Simion array files containing electrode voltages are available for use in Simion SL Toolkit programs. Of particular importance is the ability for Simion programs to edit electrode voltages (through the `fast_adjust` subroutine) while the simulation is running. Again, there are too many details to describe here, so please refer to the *Simion SL Toolkit Quick Start Notes* provided with Simion for specifics. A sample SL program, however, can be found in Appendix B.

Having written an SL program, save the program in the same directory and with the same name (but with a .sl extension) as the PA file created from the STL geometry. Then, right click on the SL file and select “Simion: Compile”. Now, the SL program will be used whenever a simulation is run using the PA file.

3.1.3 Collecting Simion Output

The ion motion from Simion simulations can be saved to a `.ion` file; it is this file format that Umbra uses to visualize the results of Simion simulations. To setup this functionality, go to the ion definition menu, click on the “`Define`” button next to “`Data Recording`,” and select “`Delimited`” (as opposed to “`Verbose`”). Then, click “`Ok`” and type in a filename (such as `trapFlight.ion`) where the ion data should be stored. This file will be used by Umbra.

3.2 Umbra

All of the pieces are now ready for the Umbra visualization. In particular, these pieces are the OSG geometry filtered to electrodes, the VWF file describing electrode voltages, and the ION file describing ion motion within the trap. This section will describe how to setup Umbra to collect all of these pieces together for visualization.

Setting up the visualization is slightly more complicated than the examples seen previously. However, it is still a rather straightforward process. A list of commands, with explanations of what they do, is presented below.

```
> umb::load qc
> qc::GDSLayoutController g
> g connect ir_scene scene
```

These three lines load the Umbra `QC` library, create a `GDSLayoutController` object called `g`, and tell `g` to use the default Umbra display window for drawing. This object gathers together and synchronizes each piece of the visualization.

```
> g loadGeometry electrodesFile.txt electrodeVoltagesFile.txt
```

The `loadGeometry` function takes two parameters. The first is a file describing which OSG files to load into Umbra. This is actually the same file as used when calling the `filterToElectrodes` command described above. The second parameter is a file that contains a mapping from electrodes to voltages. It tells which voltages (from the VWF file, loaded next) are applied to which electrodes. During visualization, the electrode colors will change based on the voltages. There are two options for this, as seen in Figure 5. Instantaneous voltages show the actual binary high (red)/low (blue) voltage on an electrode at each point in time, and averaged voltages show a continuous gradient from low to high based on the effective voltage of an electrode. See *Control Electrode Switching Frequency and Induced Micromotion in Trapped 9Be+* [4] for a thorough discussion of this process.

```
> g ic_low 0.0
> g ic_high 150.0
> g ic_mult 1.0
```

Set the voltages that correspond to values read in from the VWF file. These are needed since the VWF file will output binary values. The low value corresponds to zeroes in the VWF file and the high value corresponds to ones. The multiplier is a further scale factor applied to all voltages.

```
> g loadVoltages vwfFile.vwf signalPattern
```

Load a VWF file. The signal pattern works the same way as it did during the `VWFConv` application, meaning that only those signals with names containing `signalPattern` are loaded from the VWF file.

Properties of RF electrodes also need to be described. RF voltages are updated according to the equation: RF = `sin(time * omega.get) * scaledRF`
Where omega is angular velocity in radians/micro second and scaledRF is the RF voltage. In this code, omega is calculated from the frequencyHz parameter.

```
> # Setup RF
> g ic_frequencyHz 85.0E6
> g ic_theta 0.0
> g ic_scaledRF 900.0
> g ic_timeStepsPerCycle 100.0
```

FrequencyHZ is used to calculate omega, theta is a phase shift, and scaledRF is the RF voltage. TimeStepsPerCycle is a parameter used for synchronizing DC and RF electrode updates, and refers to the number of Umbra timesteps that should occur in the time it takes an RF electrode's voltage to complete one period. A value of 100 works well for this parameter.

```
> g loadIon ../et/qc/path.ion
```

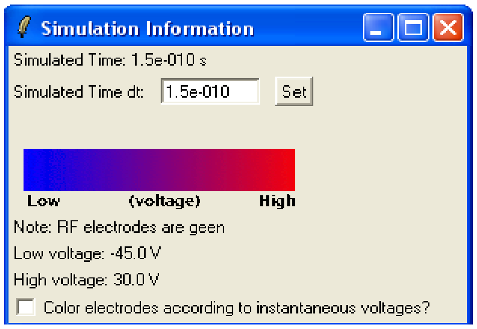
Loads an ion motion file that was output by Simion.

```
> g setExcelFile ../et/qc/excelOut.csv
> g excelPeriod 10
```

The electrode voltages and ion positions are gathered together and stored in an Excel file. This is useful for creating graphs of ion positions as electrode voltages change. The first line tells Umbra the Excel file to use for storing this output. The second line tells Umbra how many updates to wait between outputs to the Excel file. A value of 1 would output to the Excel file every timestep. Here, 10 means Umbra will output to the Excel file every 10th timestep.

```
> source ../et/qc/simionGui.tcl
> makeLayoutGUI
> umb::run 1
```

The first two lines load a simple GUI window that shows information on the visualization. An example of this GUI is shown in Figure 5. The final line tells Umbra to update one time, and is used to synchronize all of the data just loaded.


Simulation Information

Simulated Time: 1.5e-010 s

Simulated Time dt: 1.5e-010 Set

Low (voltage) High

Note: RF electrodes are geen

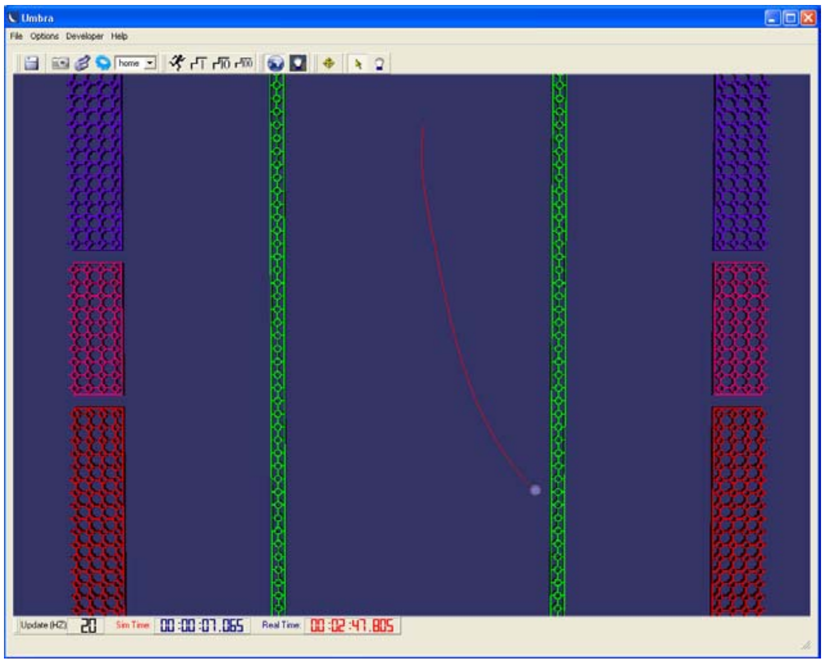
Low voltage: -45.0 V

High voltage: 30.0 V

☐ Color electrodes according to instantaneous voltages?

Figure 5. Simion layout GUI. The GUI displays the current simulation time and a gradient describing the mapping between electrode voltage and electrode color. The check box is used to toggle between instantaneous and averaged electrode colors. See Figure 6 for a picture of the visualization.

The overall Umbra visualization is seen in Figure 6. Notice the top 4 electrodes in the image are shades of purple. These colors match the voltage gradient seen in Figure 5 and are a visual representation of the electrode voltages. The green electrode in the middle is an RF electrode, and the DC electrodes on the bottom are red, indicating they are set at logical high, or 1.65V (as described by the GUI). Finally, the red line shows the ion motion as calculated in Simion. The blue sphere at the end of the line is a representation of an ion (not to scale). The visualization is fully synchronized with the Simion simulation, so the ion motion and electrode voltages are updated the same as they were in the initial simulation.



Update (HZ) 20 Sim Time 00:00:07.065 Real Time 00:02:47.805

Figure 6. Umbra visualization gathering together the ion trap geometry, electrode voltages from Quartus II, and ion motion from Simion.

4. UMBRA VISUALIZATION OF ELECTROSTATIC POTENTIALS WITHIN AN ION TRAP

To simulate ion motion, the Simion program calculates the electrostatic potentials within the ion trap. This information is used within Umbra to display equipotential surfaces. This helps develop an understanding of the environment within an ion trap and can be used to refine electrode voltages, ion shuttling routines, and trap geometry. Electrostatic potential surfaces are defined as the regions in space that have approximately the potential value defined by the isosurface: one side of the surface will have a lower potential and the other side will have a higher potential [14].

4.1 Gather the Electrostatic Potentials

During simulation, Simion continuously updates the PA files previously discussed to reflect the current potential at each grid point. By loading the Simion PA files and implementing the PA update algorithm in Umbra, access to the same potential data used to fly ions is achieved. To load this data into Simion, simply add the line

- `g loadPAs ../et/qc/paFiles_conddown.txt`

to a Simion script. The format of this file is described in the appendix, but it simply lists each of the PA files used by Simion. See the Simion manual for a detailed description on the PA file format.

4.2 Update the Electrostatic Potentials

During a Simion simulation or an Umbra visualization of that simulation, potentials are updated according to the linear differential equation: dV = dVe * k (0 <= k <= 1) where V is voltage and Ve is voltage on an electrode.

The differential equation is implemented according to the following equations [15]:

Vnew = Adjusted point voltage (for base array)
Vold = Current point voltage (for base array)
Vref = Current point voltage (for electrode array)
Update equation: Vnew = Vold + F * Vref

where F is a constant scaling factor constant over all (x,y,z) and Vnew, Vold, and Vref are variables over (x,y,z) this becomes: Vnew(x,y,z) = Vold (x,y,z) + F * Vref(x,y,z)

Expanding and rewriting this expression, get

(Vnew - Vold) = (Venew - Veold) * (Vref(x,y,z) / Veref) such that 0 <= (Vref(x,y,z) / Veref) <= 1 and equals 1 on electrodes. This is the differential equation mentioned above. Note that the Veref used inside F is a constant that Simion apparently always sets to 10000V.

Pseudocode for the update procedure is:

```
1. Load the PA# file (the base array)
2. for each electrode's PA file (i.e. PA1, PA2, ...) {
3.   Compute the necessary scaling constant F.
4.   for each point (x,y,z) {
5.     Measure the voltage Vold(x,y,z) initially in the base array.
6.     Measure the voltage Vref(x,y,z) in the electrode array.
7.     Vnew(x,y,z) in the base array is set to
              Vold (x,y,z) + F * Vref(x,y,z).
```

4.3 Defining Equipotential Surfaces

To define an equipotential surface, simply insert the line

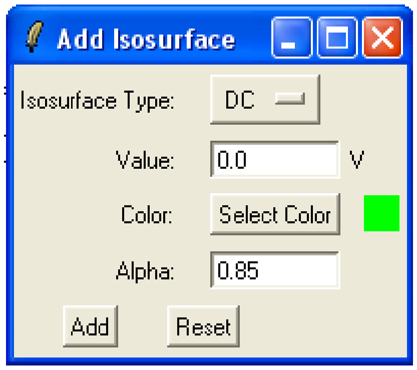
```
> g addIsosurface both 124.0 "1.0 0.0 0.0 0.85"
```

into your TCL script or console. The format is

```
      addIsosurface [type] [voltage] [color]
```

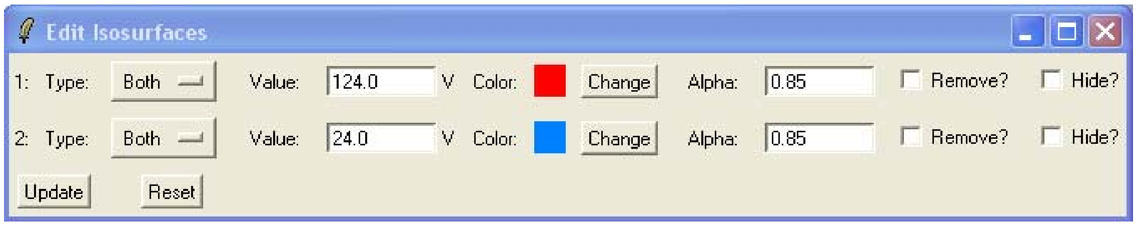
where type is one of “dc”, “rf”, or “both”. DC means only DC electrodes will contribute to the isosurface, RF means only RF electrodes will contribute, and BOTH means all electrodes will contribute. Voltage is the potential the isosurface will display, and color is an RGBA 4-tuple of values between 0.0 and 1.0 defining the isosurface color. The A portion is a measure of transparency, with 0.0 being completely transparent and 1.0 being completely opaque.

Isosurfaces can also be created using a GUI to set all of the same parameters, as seen in Figure 7 and
Figure 8. Both of these menus are accessed from the Toolbar “Ion Trap Sim” pull down menu.



Isosurface Type: DC
Value: 0.0 V
Color: 🟩
Alpha: 0.85

Figure 7. Add isosurface GUI. The GUI allows an isosurface to be added at a particular voltage influenced by DC, RF, or both types of electrodes and to define the isosurface color and transparency.



Edit Isosurfaces

1: Type: Both	Value: 124.0 V	Color: 🟥	Change	Alpha: 0.85	☐ Remove?	☐ Hide?
2: Type: Both	Value: 24.0 V	Color: 🟦	Change	Alpha: 0.85	☐ Remove?	☐ Hide?

Update Reset

Figure 8. Edit isosurface GUI. The GUI allows users to change all the properties of an existing isosurface, hide the isosurface, or remove it from the visualization. Changes only take effect when click update button is clicked, and can be reversed up to that point by clicking the reset button.

As many equipotential surfaces as desired can be defined, but beware that updating the surfaces can be time consuming. Figure 9 shows an equipotential surface as visualized in Umbra.

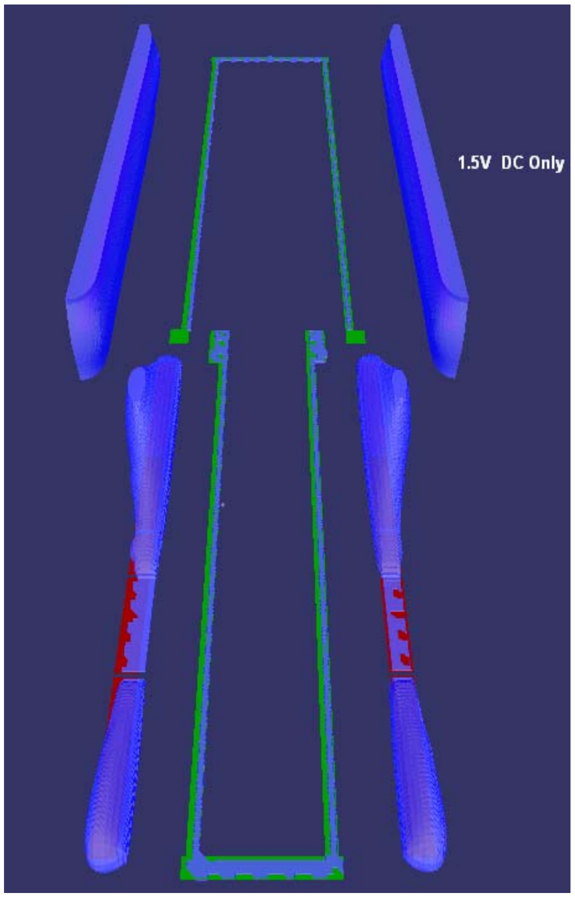


Figure 9. An equipotential surface as visualized in Umbra. The two dips in the surface near the bottom of the image are over an electrode with an effective voltage of 1.5V whereas the other DC electrodes have a higher voltage resulting in surfaces farther away.

4.4 Related Topics

The PA files Simion uses define a fine grid in order to accurately simulate ion motion, resulting in large datasets that need to be updated in order to calculate equipotential surfaces. A general suggestion is to use the fine grid for simulation and a coarser grid for initial visualizations. There is a downsampling feature which eliminates a number of grid points. A downsample factor of 1 retains all grid points, a factor of 2 will eliminate half of the points in each dimension, a factor of 4 will eliminate three-quarters of the points in each dimension, and so forth. Since this is three dimensional data, downsampling with a factor of 2 decreases the amount of data by a factor of 8. To downsample, there are two options:

1. Downsample the PA files as it is loaded into Umbra:
   ```
   > g downsampleFactor 2
   > g loadPAs ../et/qc/paFiles.txt
   ```

2. Downsample the PA files and save the result for future use
   ```
   > g downsampleFactor 2
   > g convertPAsToDBPA ../et/qc/paFilesConversionScript.txt
   ```

The conversion script file simply lists the PA files to downsample and the paths in which to store the downsampled PA files. See Appendix A for an example of this type of file. After performing the downsampled process, the resulting PA files are valid and can be used as though they were created by Simion.

Note that if when loading a previously downsampled file use a downsample factor of 1 to prevent further downsampling.

4.5 PA Filtering

It maybe desirable to define an area filter on the PA files. Filters define regions in the PA grid that should be continuously updated according to Simion's update equation. By filtering out some grid points, the user will experience faster Umbra updates and response times. If there is no interest in visualizing equipotential surfaces then a filter can be defined that will not pass anything. Filters can also be defined that only pass certain regions of the ion trap that are of interest. The default of not defining a filter will result in no data being filtered.

To load a filter, do the following:
```
> g loadPAFilter ../et/qc/rectangularFilter.txt
```

All filter regions are rectangular boxes, and a filter file can define one or more filter regions. See Appendix A for a description of the file format.

5. REFERENCES

[1] Simion 8.0. Software. Scientific Instrument Services, Inc., 2003-2006. http://www.simion.com/

[2] Umbra 4.0 Software. Sandia National Laboratories. http://www.sandia.gov/isrc/UMBRA.html

[3] Quartus II Software. Altera Corporation. http://www.altera.com/

[4] Hamlet, Jason R. "Control Electrode Switching Frequency and Induced Micromotion in Trapped 9Be+." Unpublished. (2007).

[5] Boolean GDSII Viewer/Editor. Klaas Holwerda. http://boolean.klaasholwerda.nl/bool.html

[6] General Polygon Clipper. University of Manchester. http://www.cs.man.ac.uk/~toby/alan/software/

[7] OpenSceneGraph. Open source graphics software used by Umbra. http://www.openscenegraph.org

[8] Shewchuk, Jonathan. *Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator*, in ``Applied Computational Geometry: Towards Geometric Engineering'' (Ming C. Lin and Dinesh Manocha, editors), volume 1148 of Lecture Notes in Computer Science, pages 203-222, Springer-Verlag, Berlin, May 1996. (From the First ACM Workshop on Applied Computational Geometry.)

[9] STL ASCII File Format. Gesellschaft für Optische Meβtechnik http://www.gom.com/CONTAINER/files/sp_stl_en.pdf

[10] TCL. Programming language, open source software. http://www.tcl.tk/

[11] Maya 7.0. Autodesk, Inc. http://usa.autodesk.com/

[12] Osg2Maya. Sung-Hee Lee. Maya plugin to import OpenSceneGraph files. http://www.cs.ucla.edu/~sunghee/osg2maya/user_guide.htm

[13] OSG Exporter-Maya. DIOsoft. http://www.diosoft.com/soft/osgmaya/

[14] Bourke, Paul. Polygonising a Scalar Field using Tetrahedrons (1997). http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/

[15] Manura, David. Personal communication with this lead Simion developer describing the explanation of Simion's update equation for electrostatic potentials.

[16] Blaine, Matthew G., et. al. "Developing Key Capabilities for Quantum Computing: Trapped Ion and GaAs Approaches," SAND2008-6189. (2008).

APPENDIX A: FILE FORMAT DESCRIPTIONS

This appendix describes the formats for the various text input files described in the above sections. All of the files are ASCII text and have a rather straightforward format.

A.1 Filter File

This type of file is used by the `qc::loadGDS` command. It is also the type of file referenced by the `filtersFile` in the `qc::CADExtrusion filterToElectrodes` command.

The format is:
[One integer describing the number of polygons that follow]
[One integer describing the number of vertices in the first filter polygon]
[Two whitespace separated floating point values describing first vertex of first polygon]
…
[Two whitespace separated floating point values describing last vertex of first polygon]
[Repeat the above for each polygon in the filter]

An example filter file that describes two overlapping rectangular polygons is:

```
2
4
-0.490 -1.470
 0.490 -1.470
 0.490  1.490
-0.490  1.490
4
 0.480 1.450
 0.790 1.450
 0.790 2.490
 0.480 2.490
```

A.2 Extrusion Amounts File

This type of file is used to describe the amount to extrude each layer of the original GDSII file. It is used by the `qc::loadGDS` command.

The format is:
[One integer describing the number of layers that follow]
[For each layer, a whitespace separated string and floating point value]

The strings are layer names, and the floating point values are extrusion amounts. The extrusion amount is specified in the same units as the CGDS geometry, with a default of mm. A layer name of "default" is optional; if it appears, it specifies the amount of extrusion for any layers not appearing in the file. If it does not appear, the `qc::loadGDS` command will use a default value of 0.001. Additionally, the strings are matched against the beginnings of the actual CGDS filenames. Thus, referencing the example below, a CGDS file named `barrier0.cgds` starts with a substring that matches with `barrier`, so it is extruded by .0088 units.

An example extrusion amounts file is:

```
8
barrier     0.0088
ridges      0.003
conddown    0.003
resup       0.00075
resdown     0.00075
mvial       0.003
vial        0.003
default     0.003
```

A.3 Filters File used by qc::CADExtrusion filterToElectrodes

This file is used to specify the filters to be applied to a piece of BVI geometry to split it into several different OSG files. The original BVI file is loaded, passed through the first filter referenced by this file, and the results written to an OSG file. This process is then repeated for each of the remaining filters.

The format is:
[One integer describing the number of filter lines that follow]
[For each filter line, 3 whitespace separated strings]

The first string represents the filter file to use (this filter file is of the format described above), the second string is the name of the OSG file to output the geometry to, and the third string is the name of a low quality OSG file. The low quality OSG file will contain a bounding volume representation of the first OSG file.

An example filters file is:

```
5
filters/e0.txt electrodes/e0.osg electrodes/e0_low.osg
filters/e1.txt electrodes/e1.osg electrodes/e1_low.osg
filters/e2.txt electrodes/e2.osg electrodes/e2_low.osg
filters/e3.txt electrodes/e3.osg electrodes/e3_low.osg
filters/e4.txt electrodes/e4.osg electrodes/e4_low.osg
```

A.4 Electrode Voltages File

This type of file is used by the `qc::GDSLayoutController loadGeometry` command. It specifies which voltages (as read from a VWF file) should correspond to which electrodes (as read from a Filters File as described in the previous section). This file format has to be used carefully as all of the information is presented as indices (starting at 0) representing the voltages and electrodes. Thus, an electrode value of 0 is the first electrode, a value of 1 is the second electrode, and so on. These numbers match the ordering of electrodes in the filters file. Similarly, a voltage value of 0 is the first voltage, a voltage value of 1 is the second voltage, and so on. These numbers represent the order signals are found in the VWF file when reading the file from beginning to end, with voltage 1 (index 0) the first signal matching the `signalPattern`, voltage 2 (index 1) representing the second signal matching the `signalPattern`, and so on.

The format is:
[One integer describing the number of electrode / voltage pairs that follow]
[For each electrode/voltage pair, two space separated integer values]

The first integer value of each pair represents an electrode and the second integer value represents a voltage. A negative voltage value means the electrode is RF.

An example electrode voltage file is:

```
18
0 0
1 0
2 1
3 1
4 2
5 2
6 3
7 3
8 4
9 4
10 5
11 5
12 6
13 6
14 7
15 7
16 -1
17 -1
```

This file describes 18 electrodes. The first 16 are DC electrodes and the last two are RF electrodes.

A.5 PA File for Potential Updates

PA files are used by Simion to update electrostatic potentials within an ion trap. These potentials are then used to fly ions. Umbra uses the same files and update routines to generate equipotential surfaces. The Umbra file format is:
[One integer describing the number of PA files that follow]
[A list of PA file paths]

An example file is:

```
19
C:/_Sim7/_conddown/conddown.PA#
C:/_Sim7/_conddown/conddown.PA1
C:/_Sim7/_conddown/conddown.PA2
C:/_Sim7/_conddown/conddown.PA3
C:/_Sim7/_conddown/conddown.PA4
C:/_Sim7/_conddown/conddown.PA5
C:/_Sim7/_conddown/conddown.PA6
C:/_Sim7/_conddown/conddown.PA7
C:/_Sim7/_conddown/conddown.PA8
```

```
C:/_Sim7/_conddown/conddown.PA9
C:/_Sim7/_conddown/conddown.PA10
C:/_Sim7/_conddown/conddown.PA11
C:/_Sim7/_conddown/conddown.PA12
C:/_Sim7/_conddown/conddown.PA13
C:/_Sim7/_conddown/conddown.PA14
C:/_Sim7/_conddown/conddown.PA15
C:/_Sim7/_conddown/conddown.PA16
C:/_Sim7/_conddown/conddown.PA17
C:/_Sim7/_conddown/conddown.PA18
```

The first file is the base array and the following 18 files are arrays for each electrode. The base array can be viewed as an accumulator for the overall potential at each grid point while the electrode arrays hold the potential induced at each grid point from the individual electrodes.

A.6 Downsample PA Conversion File

This type of file is used for creating downsampled PA files. It lists the original files to be downsampled and the files where the downsample data should be stored. The format is:

[One integer describing the number of PA files that follow]
[A list of PA file paths followed the paths for storing the downsampled files]

An example file is:

```
19
C:/_Sim7/_STL_tall/conddown.PA0 C:/_Sim7/_STL_tall_down/conddown.dbpa0
C:/_Sim7/_STL_tall/conddown.PA1 C:/_Sim7/_STL_tall_down/conddown.dbpa1
C:/_Sim7/_STL_tall/conddown.PA2 C:/_Sim7/_STL_tall_down/conddown.dbpa2
C:/_Sim7/_STL_tall/conddown.PA3 C:/_Sim7/_STL_tall_down/conddown.dbpa3
C:/_Sim7/_STL_tall/conddown.PA4 C:/_Sim7/_STL_tall_down/conddown.dbpa4
C:/_Sim7/_STL_tall/conddown.PA5 C:/_Sim7/_STL_tall_down/conddown.dbpa5
C:/_Sim7/_STL_tall/conddown.PA6 C:/_Sim7/_STL_tall_down/conddown.dbpa6
C:/_Sim7/_STL_tall/conddown.PA7 C:/_Sim7/_STL_tall_down/conddown.dbpa7
C:/_Sim7/_STL_tall/conddown.PA8 C:/_Sim7/_STL_tall_down/conddown.dbpa8
C:/_Sim7/_STL_tall/conddown.PA9 C:/_Sim7/_STL_tall_down/conddown.dbpa9
C:/_Sim7/_STL_tall/conddown.PAa C:/_Sim7/_STL_tall_down/conddown.dbpaa
C:/_Sim7/_STL_tall/conddown.PAb C:/_Sim7/_STL_tall_down/conddown.dbpab
C:/_Sim7/_STL_tall/conddown.PAc C:/_Sim7/_STL_tall_down/conddown.dbpac
C:/_Sim7/_STL_tall/conddown.PAd C:/_Sim7/_STL_tall_down/conddown.dbpad
C:/_Sim7/_STL_tall/conddown.PAe C:/_Sim7/_STL_tall_down/conddown.dbpae
C:/_Sim7/_STL_tall/conddown.PAf C:/_Sim7/_STL_tall_down/conddown.dbpaf
C:/_Sim7/_STL_tall/conddown.PAg C:/_Sim7/_STL_tall_down/conddown.dbpag
C:/_Sim7/_STL_tall/conddown.PAh C:/_Sim7/_STL_tall_down/conddown.dbpah
C:/_Sim7/_STL_tall/conddown.PAi C:/_Sim7/_STL_tall_down/conddown.dbpai
```

A.7 PA File Filtering

This type of file is used for filtering PA files within Umbra. Filter files define one ore more rectangular regions that are allowed to pass through the filter. Regions that do not pass are not used. The format is:
[An integer describing the number of rectangular filters]
[List of rectangular boxes, each of the form LEFT BOTTOM RIGHT TOP]

For instance, to define a cube of .5 units per dimension centered at the origin, use left, bottom, right, top values of -.25 -.25 .25 .25

A sample file is:

```
    1
    0.0 0.0 0.0 0.0
```

which will pass nothing.

Another sample is:

```
    2
    -0.2615  0.3085  0.2915 1.4815
    -0.4815 -1.4615  0.4815 0.3085
```

which passes two separate regions of interest.

APPENDIX B: SAMPLE SIMION SL PROGRAM

A Simion SL program provides the connection between the Quartus hardware simulation and the Simion physics simulation. The following is a sample Simion SL program that uses the transition lists from a VWF file (converted using `VWFConv`) to control the DC electrodes and a sinusoid to control the RF electrodes.

```
#===== variables
################################################
# Declare variables
################################################
#
static next_pe_update = 0.0   # next PE surface update time.
static numVoltages = 96999    # This needs to be modified for actual   #
                              length

adjustable[96999] vout0
adjustable[96999] vout1
adjustable[96999] vout2
adjustable[96999] vout3
adjustable[96999] vout4
adjustable[96999] vout5
adjustable[96999] vout6
adjustable[96999] vout7

static pe_update_each_usec = 0.000001

adjustable _percent_tune = 97.0           # percent of optimum tune
adjustable _amu_mass_per_charge = 100.0   # mass tune point
                                          #(in amu/charge)
adjustable _quad_axis_voltage =     -8.0  # quad axis voltage



#----- adjustable at beginning of flight
adjustable phase_angle_deg = 0.0    # entry phase angle of ion
adjustable frequency_hz = 1.1E6     # RF frequency of quad in (hz)
adjustable effective_radius_in_cm = 0.40 # effective quad radius (cm)

adjustable percent_energy_variation = 10.0  # randomized ion energy
                                            # variation (+- %)
adjustable cone_angle_off_vel_axis = 5.0    # randomized ion trajectory
                                            # cone angle (+- degrees)
adjustable random_offset_mm = 0.1           # randomized initial ion
                                            # offset position (in mm)
                                            # with mid-point at zero   #
                                            offset.
adjustable random_tob = 0.909091            # max randomized time of
                                            # birth over one cycle (in
                                            # usec)

#----- static variables
static scaled_rf =                 1.0         # scaled RF base
static omega =                     1.0         # frequency (in
                                               # radians/usec)
```

```
static theta =                     0.0    # phase offset (in radians)


################################################
# Load the electrode voltages
# This sets the voltages to values stored
# in arrays created by the VWFConv program
################################################
sub Initialize
      array_load(vout0, "electlong_Vout0.txt")
      array_load(vout1, "electlong_Vout1.txt")
      array_load(vout2, "electlong_Vout2.txt")
      array_load(vout3, "electlong_Vout3.txt")
      array_load(vout4, "electlong_Vout4.txt")
      array_load(vout5, "electlong_Vout5.txt")
      array_load(vout6, "electlong_Vout6.txt")
      array_load(vout7, "electlong_Vout7.txt")
endsub

################################################
# Adjust the voltage on the electrode
################################################

#===== subroutines

# Generate trap rf.
sub fast_adjust
      # Might have to scale this
      index = floor(ion_time_of_flight*1000000) + 1

      # Set electrode potentials.

      # Note: these can be adjusted during the simulation, so they
      # must be recalculated.

      #### Here we adjust DC electrodes

      if index <= numVoltages
          adj_elect01 = vout0[index]*2
          adj_elect02 = vout0[index]*2

          adj_elect03 = Vout1[index]*2
          adj_elect04 = Vout1[index]*2

          adj_elect05 = Vout2[index]*2
          adj_elect06 = Vout2[index]*2

          adj_elect07 = Vout3[index]*2
          adj_elect08 = Vout3[index]*2

          adj_elect09 = Vout4[index]*2
          adj_elect10 = Vout4[index]*2

          adj_elect11 = Vout5[index]*2
          adj_elect12 = Vout5[index]*2

          adj_elect13 = Vout6[index]*2
```

```
            adj_elect14 = Vout6[index]*2

            adj_elect15 = Vout7[index]*2
            adj_elect16 = Vout7[index]*2

        else
            adj_elect01 = vout0[numVoltages]*2
            adj_elect02 = vout0[numVoltages]*2

            adj_elect03 = Vout1[numVoltages]*2
            adj_elect04 = Vout1[numVoltages]*2

            adj_elect05 = Vout2[numVoltages]*2
            adj_elect06 = Vout2[numVoltages]*2

            adj_elect07 = Vout3[numVoltages]*2
            adj_elect08 = Vout3[numVoltages]*2

            adj_elect09 = Vout4[numVoltages]*2
            adj_elect10 = Vout4[numVoltages]*2

            adj_elect11 = Vout5[numVoltages]*2
            adj_elect12 = Vout5[numVoltages]*2

            adj_elect13 = Vout6[numVoltages]*2
            adj_elect14 = Vout6[numVoltages]*2

            adj_elect15 = Vout7[numVoltages]*2
            adj_elect16 = Vout7[numVoltages]*2

        endif

        ## Here we adjust RF electrodes

        rfvolts = scaled_rf * _amu_mass_per_charge
        dcvolts = rfvolts * _percent_tune / 100 * 0.1678399
        dcvolts = 0
        tempvolts = sin(ion_time_of_flight * omega + theta) * rfvolts + dcvolts
        adj_elect17 = _quad_axis_voltage + tempvolts
        adj_elect18 = _quad_axis_voltage + tempvolts

        adj_elect01 =  tempvolts
        adj_elect02 =  tempvolts


endsub

# Update potential energy surface display periodically.
sub other_actions
    if ion_time_of_flight >= next_pe_update
        next_pe_update = ion_time_of_flight + pe_update_each_usec
        update_pe_surface = 1
    endif
endsub
```

DISTRIBUTION

1	MS1082	Matthew Blain	01725
1	MS0401	Benjamin Hamlet	05527
1	MS0401	David Gallegos	05527
1	MS1072	Lyndon Pierson	05629
1	MS0899	Technical Library	09536 (electronic copy)