Introduction

It is assumed that you have read Appendix J in the SIMION 7.0 manual in detail. This document and its examples illustrate the approaches that we have found to be effective strategies for creating and using geometry files. These are by no means the only way to approach these issues. Perhaps you will discover better approaches yourself. However, they have proved useful for our work and perhaps they will help you too. Note: Most figures have GEM files that you can examine and modify.

2D/3D Compatible Geometry Files

A geometry file contains the definitions of electrode/pole geometry in terms of a collection of fill commands described via a nested geometry language:

```
Pa_define(100,100,1,c,xy)       ; 2d array
; Pa_define(100,100,100,p,xyz); 3d array

E(0){fill{within{circle(0,0,50)}}}
```

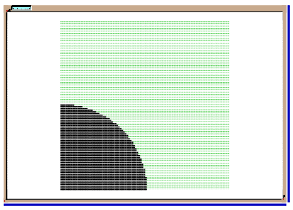

Figure 1 Array

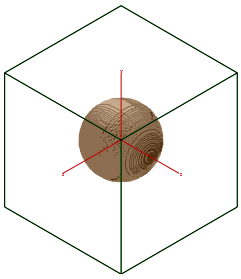

Figure 2 Intro01.gem

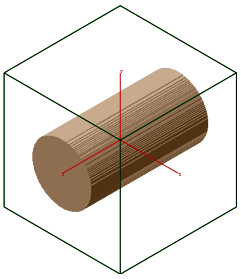

Figure 3 Intro01a.gem

Figure one shows modify's view of the array created by the geometry file defined above. Note that the Pa_define was for a 2D cylindrical array with mirroring in x as well as in y This allows the definition of a sphere (Figure 2) with an array of ½ the number of points required if x mirroring were not specified. Moral: Use symmetry where you can to reduce the number of array points required.

The surprise occurs in Figure 3. In this example the 2D array Pa_define was commented out and the 3D array Pa_define was uncommented. Instead of creating a sphere, a long cylinder was created. Why? First, the circle command defines a 2D circle or ellipse in x and y. All 2D commands ***always*** assume an infinite extent in z when projected into a 3D array. Thus a 3D cylinder would be the expected result of projecting a circle into a 3D volume (without any additional constraints). The fix is the sphere cmd (results shown in Figure 4).

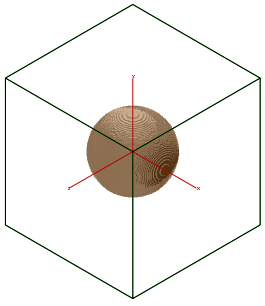

Figure 4 Intro01b.gem

```
Pa_define(100,100,1,c,xy)       ; 2d array
; Pa_define(100,100,100,p,xyz); 3d array

E(0){fill{within{sphere(0,0,0,50)}}}
```

Now projecting the fill into a 2D (z = 0) or 3D array produces a sphere. Note that the use of x, y, and z mirroring has reduced the 3D array size requirements to $1/8^{th}$ of what would be required if no mirroring were applied (take advantage of symmetry). However, the important lesson here is to ***always*** use complete 3D fills ***if*** both 2D and 3D arrays are to be used. This guarantees portability between 2D and 3D arrays (assuming the actual symmetry will support it).

The use of 2D/3D compatible geometry files (where appropriate) can be very helpful during the design phase. Using 2D arrays is often quite useful, because they refine more quickly, and thus can help speed up developmental iterations. Also 2D arrays can also define a higher resolution model with fewer points than a 3D array assuming the symmetry supports 2D general symmetry.

Always Use Physical Units

While it may be very tempting to define geometry in array grid units, it is ***not*** a very good idea. One of the powerful features of geometry files is the ability to project electrode/pole geometry into arrays of varying densities. When you define geometry in physical units (e.g. mm or inches) scaling is more straightforward and the implications of changing grid densities are more obvious.

Let's assume that we want to project an object defined in inches into the workbench. The correct aggregate conversion constant must be 25.4 mm/inch (each physical inch is mapped into 25.4 mm). Assuming that we want 0.010 inch per array grid unit, the instance scale factor ***must be*** 25.4 * 0.010 or **0.254 mm/gu**, and the ***outer*** locate command's scaling factor that translates inches into array grid units ***must be*** the reciprocal of 0.010 inch/gu or **100 gu/inch**. Assuming we want a 3D array that encloses 0.25", Each dimension in pa_define should be: One Plus 0.25 inch times 100 gu/inch or 1 + (100 * 0.25) = 26 for nx, ny, and nz.

```
Pa_define(11,11,11,p,xyz)       ; 3d array - 0.025" per grid unit or 0.635 mm/gu
;Pa_define(26,26,26,p,xyz)      ; 3d array - 0.010" per grid unit or 0.254 mm/gu
;Pa_define(51,51,51,p,xyz)      ; 3d array - 0.005" per grid unit or 0.127 mm/gu
;Pa_define(101,101,101,p,xyz) ; 3d array - 0.0025" per grid unit or 0.0635 mm/gu

Locate(0,0,0,40)   ; 0.025" per grid unit or 0.635 mm/gu
;Locate(0,0,0,100)   ; 0.010" per grid unit or 0.254 mm/gu
;Locate(0,0,0,200)   ; 0.005" per grid unit or 0.127 mm/gu
;Locate(0,0,0,400)   ; 0.0025" per grid unit or 0.0635 mm/gu
{
E(0){fill{within{sphere(0,0,0,0.250)}}}
}
```

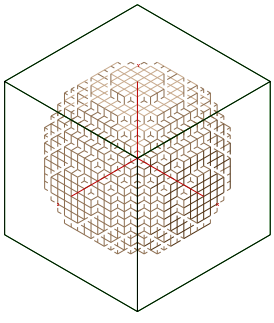

Figure 5 0.025"/gu intro02a.gem

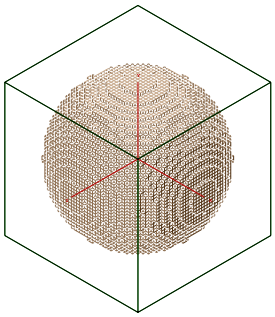

Figure 6 0.010"/gu intro02b.gem

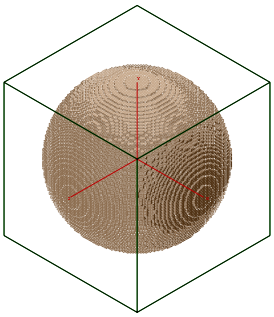

Figure 7 0.005"/gu intro02c.gem

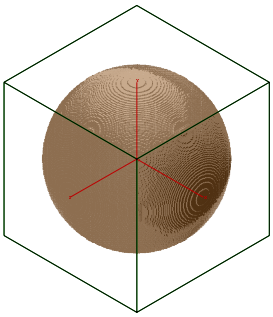

Figure 8 0.0025"/gu intro02d.gem

The four examples above show a 0.250” radius sphere projected into four different arrays of grid densities varying from 0.025” per grid unit to 0.0025” per grid unit. These examples were created by selectively uncommenting the four array pa_define (array defining) and four locate (scaling) commands in the above geometry file. Note that these lines work in pairs (e.g. top pa_define with top locate). The inline comments provide scaling information that can be used when projecting a properly scaled array instance image of the sphere into view’s workbench coordinates (e.g. scaling to use in mm/gu). This is a very powerful way to define geometry files because one GEM file can contain all the pa_defines and scaling locates you plan to use (instead of requiring a GEM file for each).

Define an Electrode and THEN Position it in the Array

We have found that the best way to define the geometry of a multiple electrode/pole object is to define each electrode separately at the origin (or some convenient location) and then scale, orient, and translate it to the desired location within the volume. Remember, define the electrode/pole object in mm or inches (or some physical unit). Moreover, the volume that you scale, orient, and translate these objects into should also be in the same physical units (e.g. mm or inches). As a last step, the ***outermost*** locate in the geometry file should be used to orientate, scale, and locate the entire defined physical volume and its included geometry into the desired array volume (3D arrays) or plane (2D arrays).

```
Pa_define(200,50,50,p,yz)       ; ¼ volume pa -- 10 mill resolution
;Pa_define(200,50,101,p,y)      ; ½ volume pa -- 10 mill resolution

Locate(0,0,0,100)       ;project physical volume into ¼ volume potential array (scale = 100 gu/inch)
;Locate(0,0,50,100)     ;project physical volume into ½ volume potential array (scale = 100 gu/inch)
    {
    locate(.2,0,0,1)            ;move ring 0.2” in positive x direction
            {
            e(1){rotate_fill(360){within{polyline(0,.25,.25,.30,.5,.5,0,.5)}}}
            }
    locate(1.5,0,0,1,45)        ;move ring 1.5” in x dir. and rotate 45 ccw degrees in azimuth
            {
            e(2){rotate_fill(360){within{circle(0,.4,.1)}}}
            }
    }
```

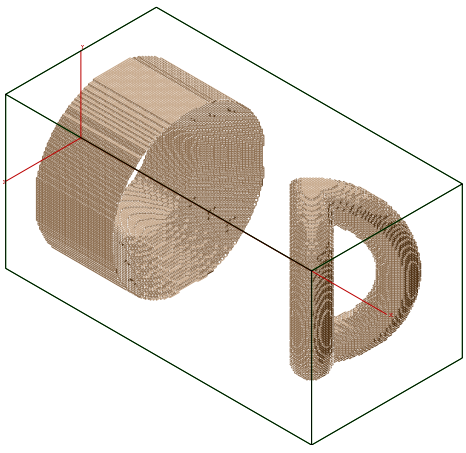

Figure 9 intro03a.gem

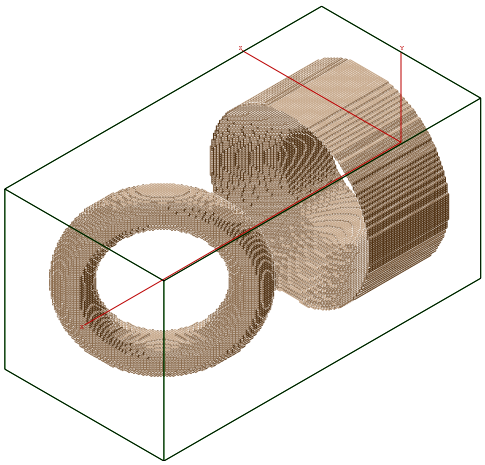

Figure 10 intro03b.gem

Figure 9 shows a projection into ¼ volume potential array with y and z mirroring. Note that the z mirroring causes the second ring to be bent into its z mirror image (oops). Figure 10 shows a projection into a ½ volume potential array (no mirroring in z). Now the second ring appears as it was intended. Mirroring must be used carefully to avoid unexpected results. ***Always*** examine the resulting geometry ***carefully*** to insure that you haven't inflicted something unexpected on yourself.

How to use a Single Fill to Define an Object

The geometry file language allows you great freedom in how you can define an object. In most cases an object (electrode/pole) can be defined with as a single fill by using the appropriate within and notin definitions:

```
Pa_define(101,101,101,p,n)      ; 10 mill planar non-mirrored

Locate(50,50,50,100)            ; center object and scale to 100 gu per inch
{
E(0){fill{
            within{sphere(0,0,0,.5)box3d(-1,-1,-1,1,0,1)}
            notin_inside{sphere(0,0,0,.4,.4,.2)}
            notin_inside{circle(0,0,.1)}
            notin_inside{locate(0,0,0,1,90){circle(0,0,.1)}}
    }}
}
```

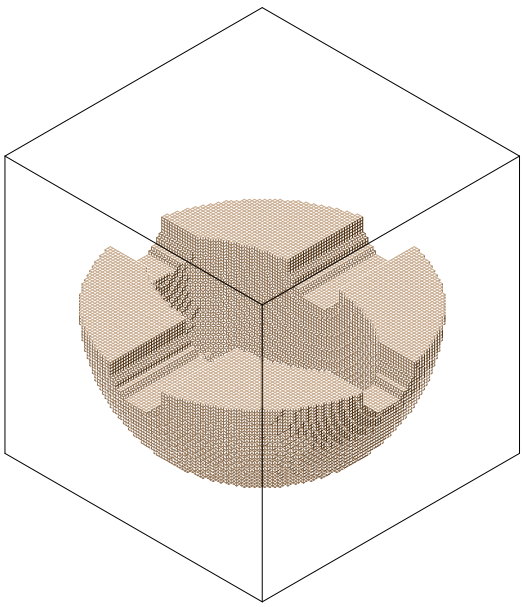

Figure 11 intro04a.gem

The GEM file above (shown in Figure 11) serves as an example of a non-trivial object created by a single fill. The first within contains both a sphere and box3d definition. The sphere definition defines a sphere with a radius of 0.5 inches. The box3d definition creates a box 1 inch high (y) by 2 inches wide (x) and deep (z). The box is positioned so its top is at y = 0 (its height is in the negative y). Whenever

a within (or notin) contains more than one definition function the resulting volume is the volume common (shared) by all the definition functions. In this case the common volume is the lower half of the sphere. Note: The box3d is larger than it needs to be. Only one dimension is critical ($y_{max} = 0$). The others were made much larger to facilitate its definition and insure that the entire lower hemisphere would be within.

The fill also contains three notin commands. Note that the notin commands used were notin_inside. These commands were used to insure that material was removed just to within the edge of the notin volume, but not on the edge or beyond it. This mimics what a drill bit or end mill would do. ***It is an important consideration when objects are to be projected into arrays of varying densities***.

The first notin command removes an ellipsoid volume from within the hemisphere. The second notin drills a 0.2" diameter hole centered on the z = 0 axis. The third notin drills a 0.2" diameter hole centered on the x = 0 axis. Note the use of a locate command to swing this hole 90° ccw in azimuth.

This is easier to understand if we work outward from the circle function. The circle is defined to be centered along the z axis. The locate command simply swings the circle ccw 90° in azimuth so it is centered along the x axis when interpreted by its notin function.

Notin_inside{box3d(-1,0,-1,1,1,1)}

Moreover, box3d in the within command could be removed and the notin command above could be added to perform the same function. The choice of how you want to define things is up to you. Clearly, you should use the approach that is most logical and obvious to you.

Defining an object with a single fill can become quite convoluted with collections of seemingly unrelated withins, notins, and locate commands that often compound the confusion. We humans seem to have 2D minds with a bit of 3D depth cues. Our experience indicates that arbitrarily defined 3D objects often tax our perceptual abilities.

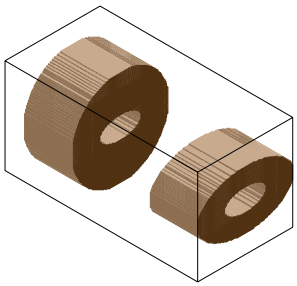

Figure 12
intro04b.gem

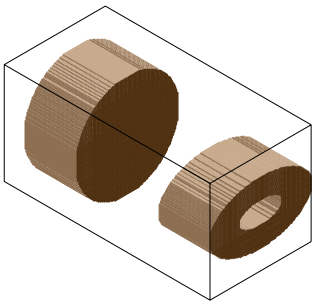

Figure 13
intro04c.gem

How to Use Multiple Fills to Define Objects

Other objects may require the use of multiple fills (to draw and selectively erase). Even if you could define an object with one fill, it may be easier for you to create the object with multiple fills.

However, defining an object with a single fill has the advantage of physical isolation. The scope of the single fill limits the range of effect of the withins and notins. An unbounded notin does not impact any other fill beyond the range of the withins of the fill it is in. The following multiple fill example serves to demonstrate:

```
Pa_define(201,50,50,p,yz)       ; 10 mill planar non-mirrored

Locate(0,0,0,100)                ; 100 gu per inch
{
locate(.2,0,0,1,-90)
     {                           ; fill number one
     e(1){fill{within{cylinder(0,0,0,.5,,.5)}}}
     }
locate(1.5,0,0,1,-90)
     {                           ; fill number two
     e(2){fill{within{cylinder(0,0,0,.5,.3,.5)}}}

                                 ;fill number three
     n(0){fill{within_inside{circle(0,0,.2,.1)}}}

                                 ;limited fill number three
;    n(0){fill{within_inside{cylinder(0,0,0.01,.2,.1,.52)}}}
     }
}
```

The GEM file above created Figure 12. It contains three fills. The first fill creates a solid cylinder. The second fill creates an ellipsoidal cylinder, and the third fill drills an elliptical hole through it with a circle command. Unfortunately, the hole also appears in the first electrode that was supposed to be solid. What happened? ***Fills occur in the order they are defined***. Thus fill one created the solid cylinder, but fill three drilled an elliptical hole through it, because a circle's depth is infinite (unless constrained in some manner). If the third fill is commented out and the following commented fill is uncommented the elliptical hole will only be bored through the second electrode (Figure 13). In this case, a cylinder command was used to limit the extent of the elliptical hole.

Notice that both third fills use a within_inside command. This command, like the notin_inside, works like a drill bit or end mill by retaining the electrode definition on the exact boundary's surface. However, the within_inside and notin_inside commands can also introduce unexpected results. If the cylinder that defines the elliptical hole has exactly the same length and position as the second fill, no hole will be visible from the outside (although a cutaway of electrode two would show the elliptical hole inside – Figure 14). The reason for this is that the front and rear end planes are exactly on the boundary and will retain their definition as electrode points from fill two. Thus the cylinder command used in the example above defines a cylinder that begins ***just <u>in front</u>*** of the second electrode and extends through and ***slightly <u>beyond</u>*** it. This insures that the elliptical hole goes ***<u>clear</u>*** though the second fill (but *<u>just</u>* barely).

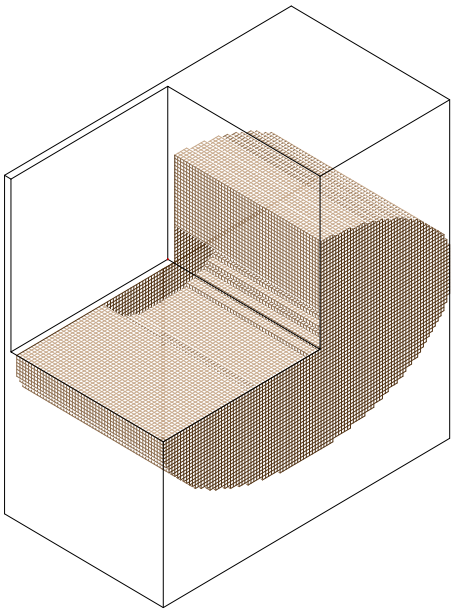

Figure 14 intro04d.gem

Thus if an object is to be defined by multiple fills, it is <u>imperative</u> that the erasing fills (e.g. n(0)) be bounded within the volume of the object, and that all the defined volumes take into account any subtle characteristics of the commands used.

A Useful Multiple Fill Strategy

We have found that the following multiple fill strategy is both powerful and very general. It defines the object progressively in much the same manner that a part would be created with a milling machine. For example, a rotate_fill might use a polyline command to create a complex cylindrically symmetric volume of revolution that defines the starting blank of the object. Next this blank is reoriented 90° so that erasing fills can be used to remove volumes parallel to the axis of rotation of the part. Perhaps the object is reoriented again to allow the erase fills to create off-axis beam holes. This process continues until all features (by additions or erasures) have been defined. Then a final locate orients and positions the object in its proper location in the physical volume the GEM file is defining.

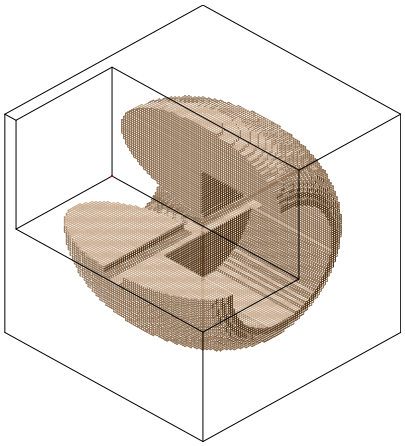

Figure 15 intro05a.gem

To implement this strategy successfully, one must use the nested nature of the geometry language to insure that each reorientation and successive fill occurs in the desired order and performs the desired tasks. Successful strategies with a nested language work much like the approaches one needs to take with RPN calculator languages and programming languages like Pascal. Problems are solved by working from the inside outward. The following RPN example is provided as an illustration:

```
R = sqrt (x^2 + y^2)        ; equation to convert to RPN

x enter *                   ; x^2
y enter *                   ; y^2
+                           ; (x^2 + y^2)
sqrt                        ; sqrt (x^2 + y^2)
```

The following is how the inside outward approach works to define an object within a geometry file:

```
Pa_define(101,101,101,p,n)      ;define array to use

Locate(0,50,50,100)             ;Step 6: center physical volume in array and scale to 100 gu/inch
    {
    locate(.5,0,0,1, -90)       ;Step 5:  twist back 90 degrees cw in az and position in volume
            {
            locate(0,0,0,1,90) ;Step 3:  now twist 90 degrees ccw in az for axis parallel holes
                    {
                            ;Step 1:  create elliptical ring (fill 1 – rotate fill)
                    e(1){rotate_fill(360){within{circle(0,.25,.4,.2)}}}

                            ;Step 2:  drill 0.050" hole radially through ring (fill 2 – fill)
                    n(0){fill{within_inside{cylinder(0,0,0.501,0.050,,1.02)}}}
                    }
                            ;Step 4:  create partial hole on axis (fill 3 – fill)
            n(0){fill{within_inside{cylinder(0,0,0,.25,,.401)}}}
            }
    }
```

The geometry file above created the object shown as a cutaway view in Figure 15. Note the use of step numbers in the comments above to help you see the sequence of events:

1. An ellipsoidal ring is created in arbitrary space.

2. A 50 thousandths radius hole is drilled radially through the ring (same orientation).

3. The object is twisted 90 degrees ccw in azimuth.

4. A cylinder command bores a hole in the now negative z region of the ring.

5. The object is twisted back 90 degrees cw in azimuth and its center is shifted 0.5” in positive x within the physical volume.

6. The physical volume is scaled and centered in the potential array’s volume. This locates the object in the center of the potential array.

```
Pa_define(101,101,101,p,n)          ;define array to use

Locate(0,50,50,100)   ;Step 6: center in array and scale to 100 gu/inch
{
locate(.5,0,0,1, -90)   ;Step 5:  twist back 90 degrees and position in volume
                  {
     locate(0,0,0,1,90);Step 3:  now twist 90 degrees for axis parallel holes
          {                ;Step 1:  create elliptical ring (fill 1)
          e(1){rotate_fill(360){within{circle(0,.25,.4,.2)}}}
                           ;Step 2:  drill 0.050" hole through on edge (fill 2)
          n(0){fill{within_inside{cylinder(0,0,0.501,0.050,,1.02)}}}
          }
                           ;Step 4:  create partial hole on axis (fill 3)
     n(0){fill{within_inside{cylinder(0,0,0,.25,,.401)}
                    locate(0,0,0,1,0,-120)
                     {
                     locate(0,0.45,0,1)
                          {
                          within_inside{cylinder(0,0,.401,.05,,.802)}
                          within_inside{sphere(0,0,0,.1)}
                          }
                     }
                    locate(0,0,0,1,0,0)
                     {
                     locate(0,0.45,0,1)
                          {
                          within_inside{cylinder(0,0,.401,.05,,.802)}
                          within_inside{sphere(0,0,0,.1)}
                          }
                     }
                    locate(0,0,0,1,0,120)
                     {
                     locate(0,0.45,0,1)
                          {
                          within_inside{cylinder(0,0,.401,.05,,.802)}
                          within_inside{sphere(0,0,0,.1)}
                          }
                     }
               }}
     }
```

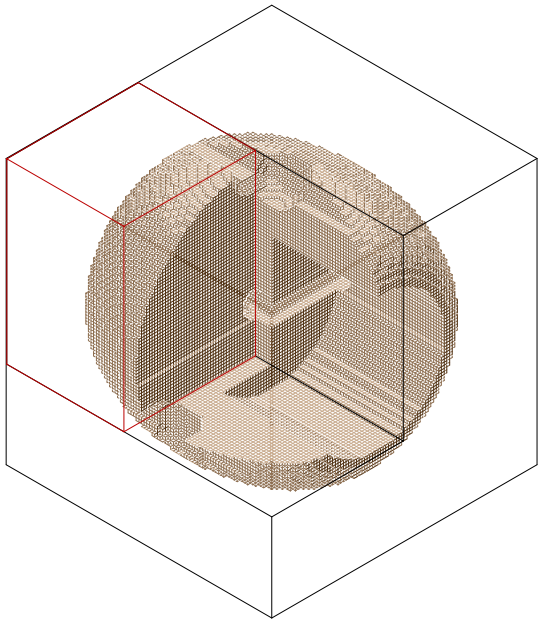

Figure 16 intro05b.gem

For the sake of discussion and to complicate your life further, let’s assume that the electrostatic lens element defined above is to be suspended between three rods with fixed spherical insulators on the rods used for holding the element in place. This means that material must be removed to allow rod clearance as well as a spherical seat to register each the spherical insulator. These features can be created in fill 3 (it has the correct orientation) by adding within statements. Remember that withins are ORed together. This means that one fill can in principle define a whole collection of volumes.

In this case we need to erase three clearance volumes required for the rod and sphere at 120° spacing intervals around the outside of the object. The following withins centered on the z axis serve as a starting point:

```
Within{cylinder(0,0,.401,.05,,.802)}     ;hole for through rod
within{sphere(0,0,0,.1)}                 ;depression for sphere
```

These withins were created on the axis of ring revolution to simplify their definitions. Now locate commands can be used to position them for an erasure.

```
Locate(0,0,0,1,0,-120)                  ;Step 3:  rotate within group –120 degrees in elevation
    {
    locate(0,0.45,0,1)                  ;Step 2:  shift within group up 0.45" in y
            {
            within{cylinder(0,0,.401,.05,,.802)};Step 1:  hole for through rod
            within{sphere(0,0,0,.1)}            ;         and depression for sphere
            }
    }
```

The inner locate merely shifts the within group up 0.45" in the positive y direction. This centers the holes of the within group on the outer radius of the ring object. The outer locate then uses a –120 elevation angle to swing the location of the erased holes 120 degrees cw in elevation from the vertical. Three copies of the above sequence with elevation angles of –120, 0, and 120 respectively will create the three desired erased volumes. The listing above created the electrode with erased mounting volumes shown in Figure 16.

Using Include GEM Files

The include file capability in geometry files allows geometry definitions to be divided into a collection of files. While this feature has a lot of uses, it can also obscure what's going on by hiding important features in the include files. It has been our experience that include files are primarily useful when they include a complete object (electrode) defined in a standard location (centered at the origin). The calling geometry file should define the fill type for the include file (e.g. e(1)) and also the locate commands needed to position the object in the physical volume. This approach allows each reference to an included object to designate its fill value, orientation, scaling and position. The following serves as an example how to use include files:

In this example (cutaway view shown in Figure 17) the file **intro06.gem** creates two copies of the object defined above by using the include file **includ06.gem**. The first include creates electrode number one (e(1)), and the second include create electrode number two (e(2)). Notice that the locate in the second include rotates the electrode 180° in azimuth and then twists it 120° so that the radial beam holes are not parallel. However both electrodes can be mounted between the three rods because the mounting groves are lined up.

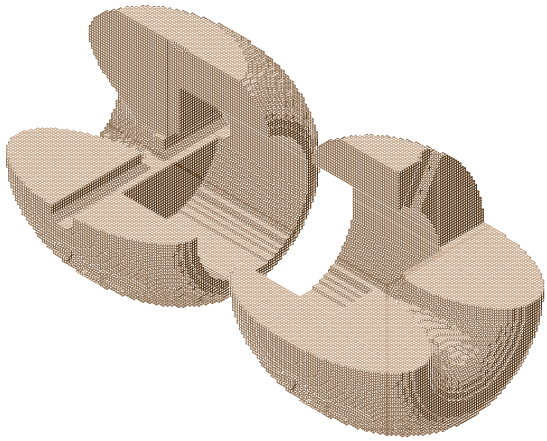

Figure 17 intro06.gem

```
;FILE = INTRO06.GEM

Pa_define(201,101,101,p,n)   ;define array to use

Locate(0,50,50,100)          ;Step 7: center in array and scale to 100 gu/inch
{
locate(.5,0,0,1,0,0,0)       ;Step 6:  position in volume
                {            ;set to 1 volt – electrode 1
                e(1){include(includ06.gem)}
                }
locate(1.5,0,0,1,180,0,120)  ;Step 6:  position in volume and orient (az = 180, rt = 120)
                {            ;set to 2 volts – electrode 2
                e(2){include(includ06.gem)}
                }
}
```

```
;FILE = INCLUD06.GEM

locate(0,0,0,1, -90)   ;Step 5:  twist back 90 degrees
    {
    locate(0,0,0,1,90) ;Step 3:  now twist 90 degrees for axis parallel holes
        {              ;Step 1:  create elliptical ring (fill passed by caller)
        rotate_fill(360){within{circle(0,.25,.4,.2)}}
                       ;Step 2:  drill 0.050" hole through on edge (fill 2)
        n(0){fill{within_inside{cylinder(0,0,0.501,0.050,,1.02)}}}
        }
                       ;Step 4:  create partial hole on axis (fill 3)
    n(0){fill{within_inside{cylinder(0,0,0,.25,,.401)}
                locate(0,0,0,1,0,-120)
                 {
                 locate(0,0.45,0,1)
                     {
                     within_inside{cylinder(0,0,.401,.05,,.802)}
                     within_inside{sphere(0,0,0,.1)}
                     }
                 }
                locate(0,0,0,1,0,0)
                 {
                 locate(0,0.45,0,1)
                     {
                     within_inside{cylinder(0,0,.401,.05,,.802)}
                     within_inside{sphere(0,0,0,.1)}
                     }
                 }
                locate(0,0,0,1,0,120)
                 {
                 locate(0,0.45,0,1)
                     {
                     within_inside{cylinder(0,0,.401,.05,,.802)}
                     within_inside{sphere(0,0,0,.1)}
                     }
                 }
            }}
    }
```

Other Considerations

If you are designing an electrode assembly, your simulations will be more accurate if the electrode placements and boundaries are reasonably integral with the grid spacing of the arrays that you plan to use. For example, if you are planning to model with arrays of 100 grid units per inch, then electrode dimensions and positioning should ***snap*** to 0.010” intervals.

Although this may seem overly constraining, we have found that this approach really doesn’t limit one’s design freedom very much. Electrode dimensions are integral to some spacing length (easy to specify and check). Moreover, electrode boundaries naturally match array point boundaries, improving SIMION’s ability to simulate your problem accurately.