# 6.0 Geometry File Tricks

# Important Concepts

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

# Important Concepts

- Fills tests are ALWAYS made in the reverse list order. The last fill in the list ALWAYS has the highest priority.

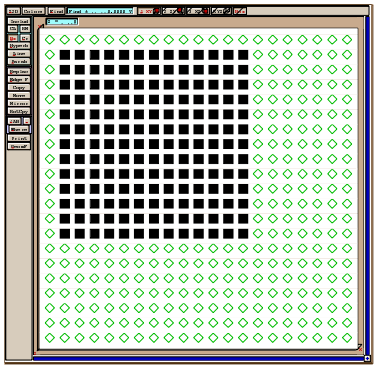
- Fill{inside{circle()}}
  - Fill{inside{box()}}
  - Fill{inside{sphere()}}

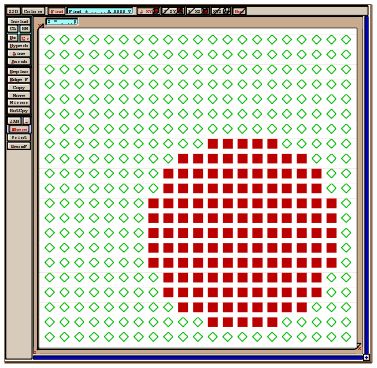
- The sphere fill has highest priority if more than one fill changes the array point's value.

# Important Concepts

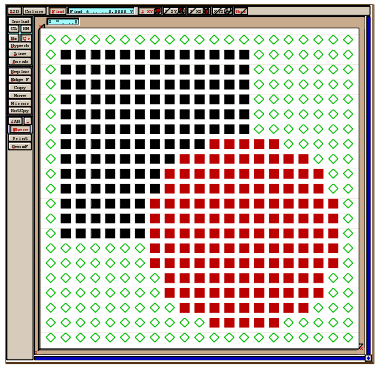
- ALL fill definitions are ALWAYS projected to their final locations in array coordinates BEFORE the actual decision to fill is made.
  - Any overlap conflicts are resolved at the final array point locations.

As defined:



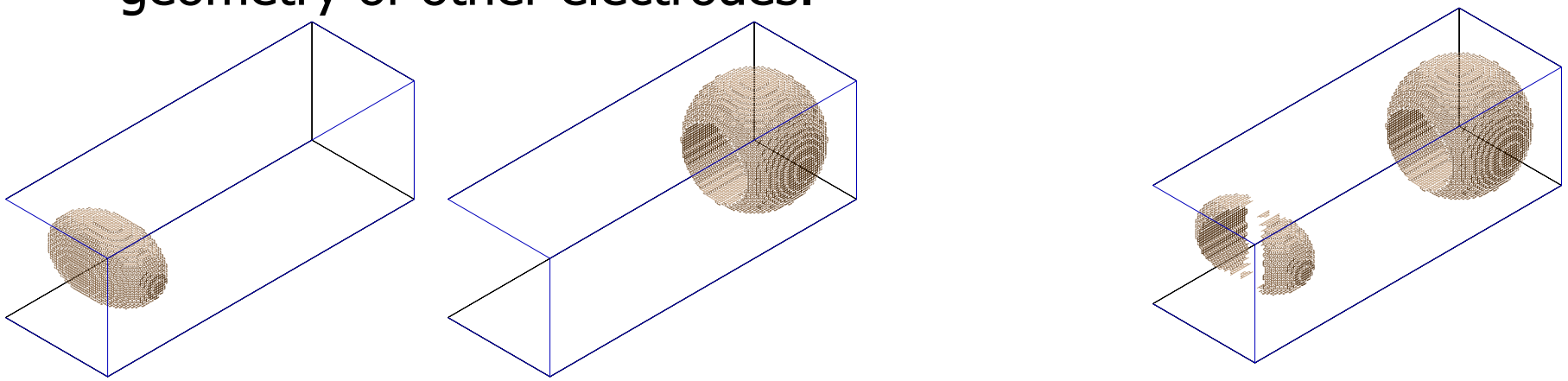


As located:



# Important Concepts

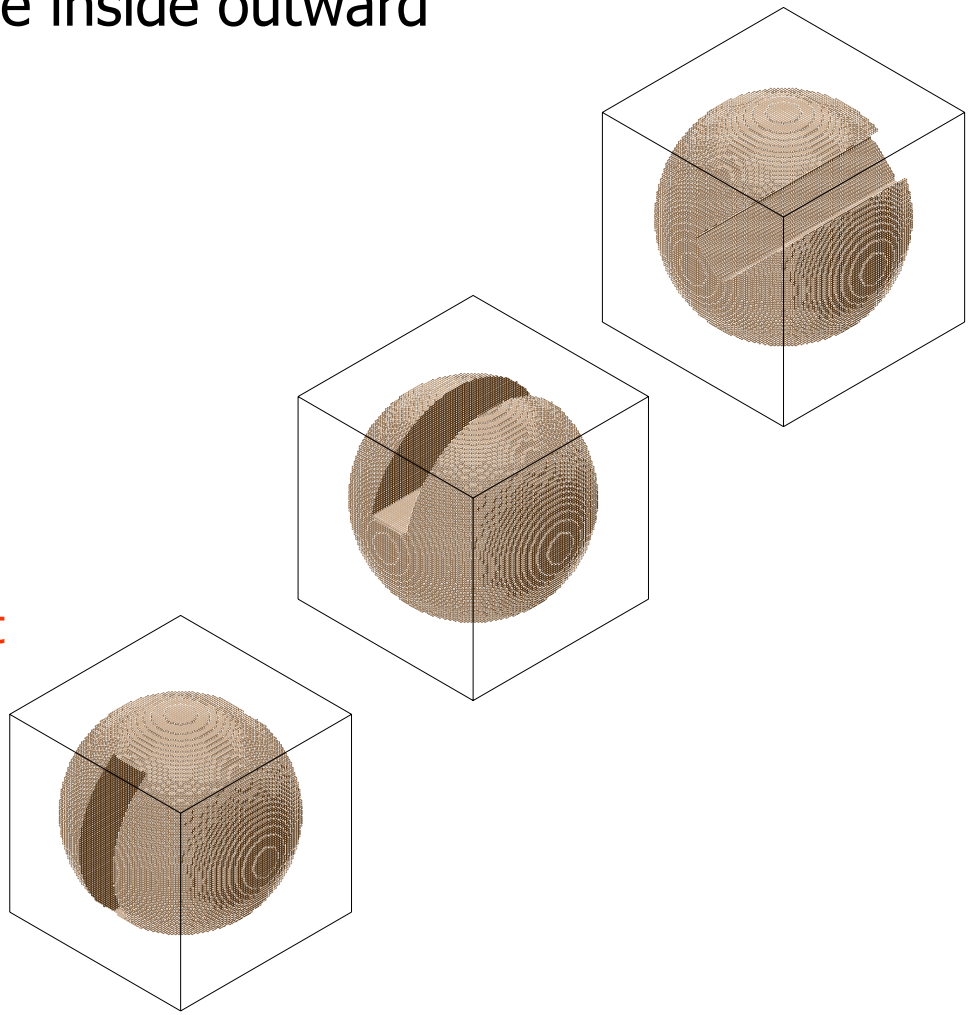
- ALL fill definitions are ALWAYS projected to their final locations in array coordinates BEFORE the actual decision to fill is made.
  - Unbounded volume fills can accidentally change the geometry of other electrodes.



# Important Concepts

Nested Languages work from the inside outward

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



# Important Concepts

Nested languages have nesting level rules:

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

```
;Base nesting level

;Locates work at any nesting level

;Fill nesting level
;Within nesting level
;Volume function definition

;Locates work at any nesting level

;Locates can be nested to define translation order

;Within nesting level
```

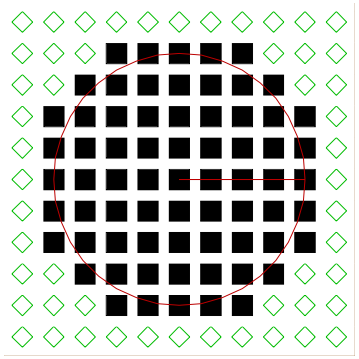
# Important Concepts

Boolean language rules:

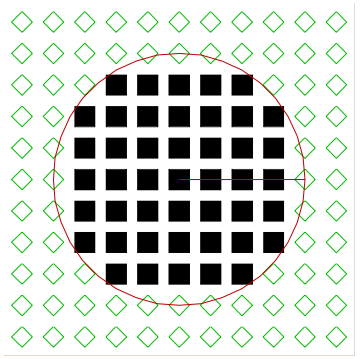
```
Fill
  {
  IF within  {  circle()  AND  box()  AND  polyline()  }  OR
      within_inside  {  testa  AND  testb  }  OR
      within_inside_or_on  {  testc  }

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

# Important Concepts

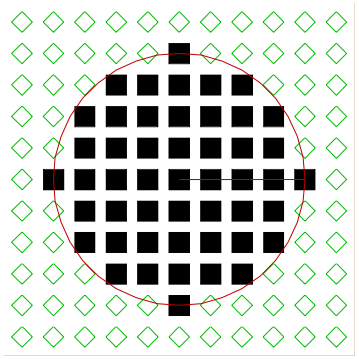
- Use within and notin alternatives for more precision



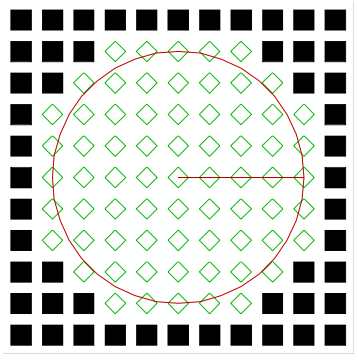
Use within,



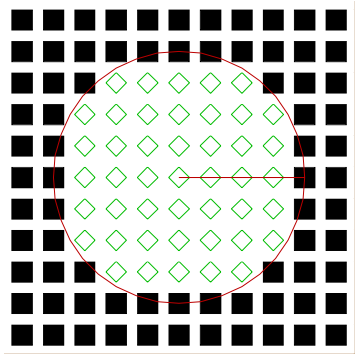
within_inside,



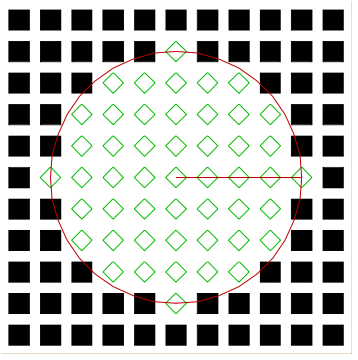
within_inside_or_on.



Use notin,



notin_inside,



notin_inside_or_on

# The Divide & Conquer Approach

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

# The Divide & Conquer Approach

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

# The Secret in Real Estate is:

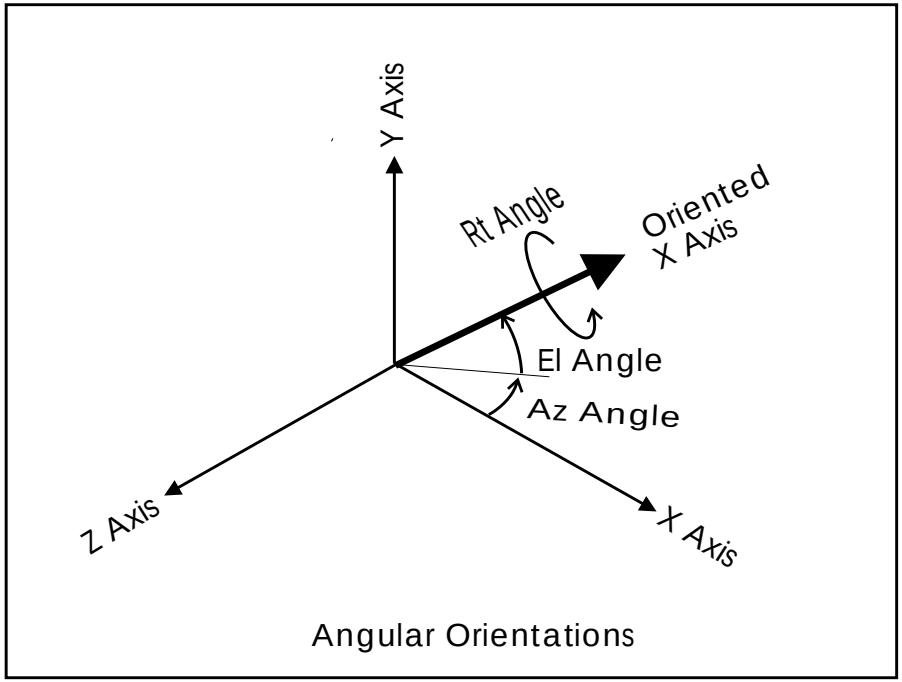
- Location
- Location
- Location

# The Secret in Geometry Files is:

- Locates
- Locates
- Locates

# The Locate Command

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

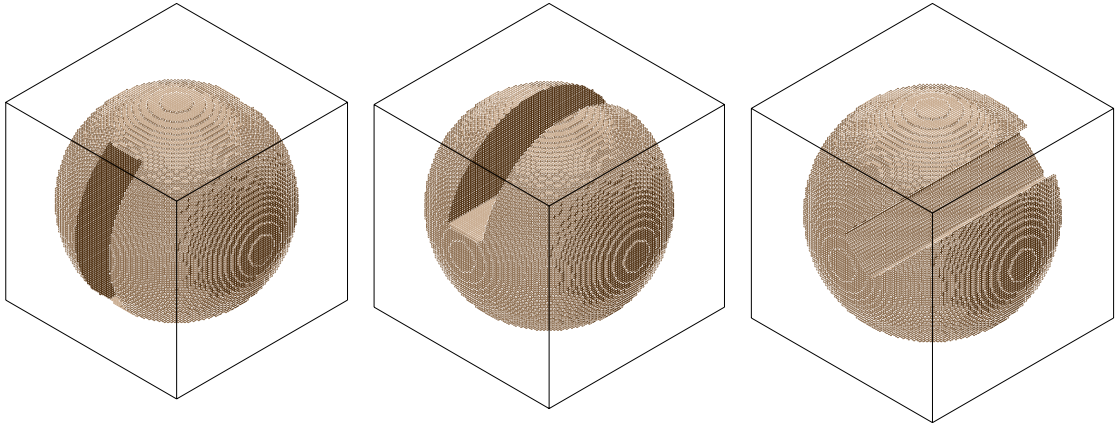


Angular Orientations

# Nested Locate Commands

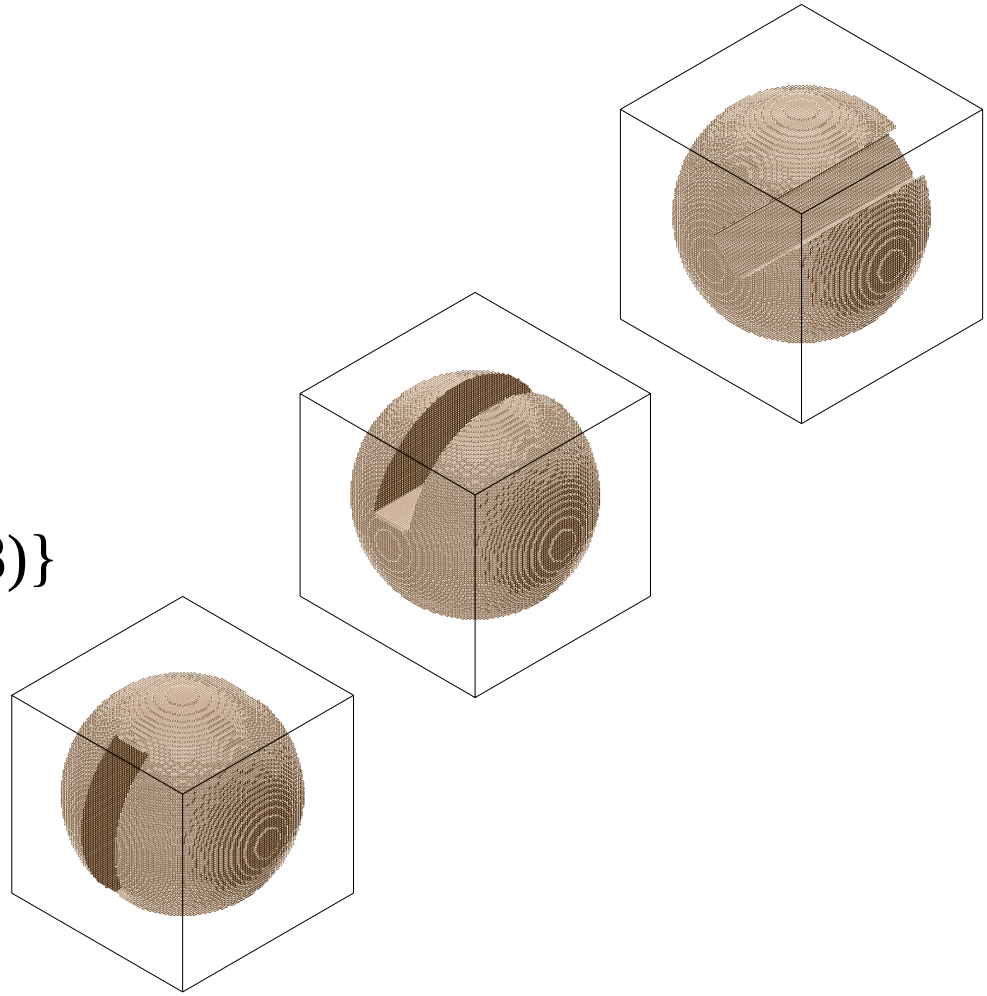
- No performance costs (locate cmds are combined)
- Insures the order that transforms are applied
- Supports progressive definitions

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



# Indent Nested Commands

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

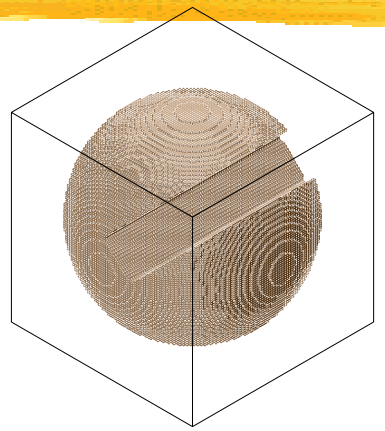


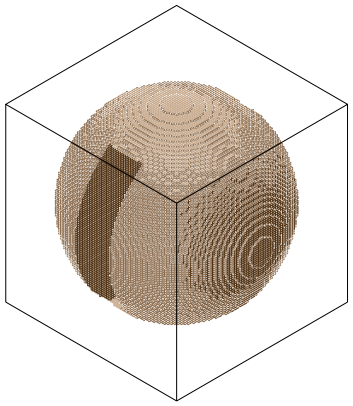
# Body Verses Machining Oriented Definitions

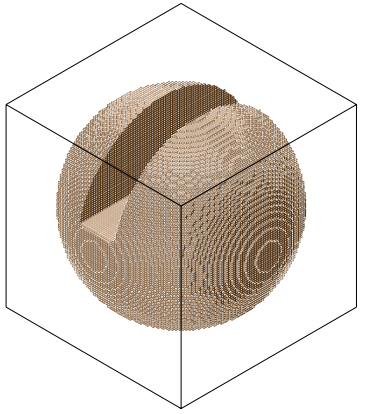
- You can move and orient either the entity (body), the cuts (machining), or both (can be confusing).
  - The body oriented approach moves the entity and assumes that the machine is fixed.
  - The machine oriented approach moves the machining and assumes the body is fixed.
  - Moving both the entity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

# Machining Oriented Definitions

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

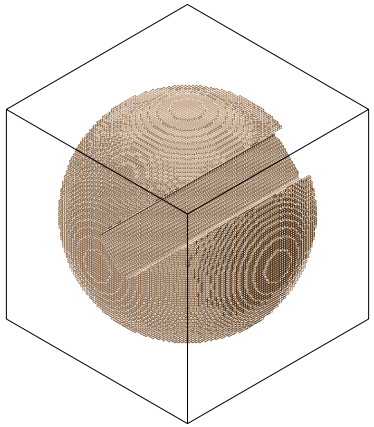


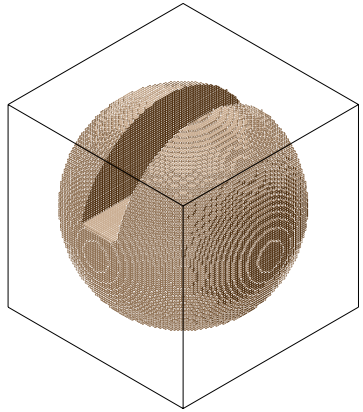




# Body Oriented Definitions

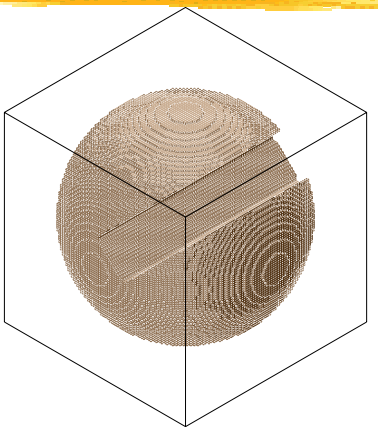
```
Locate(50,50,50,10)                    ;6:  Scale and center in array
{
fill{
    Locate(0,0,0,1,0,-45)              ;5:  Rotate sphere to slot angle
      {
       Locate(0,5)                     ;4:  Shift sphere to origin
         {
          notin{cent_box(0,0,2,8)}     ;3:  Drill slot at origin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

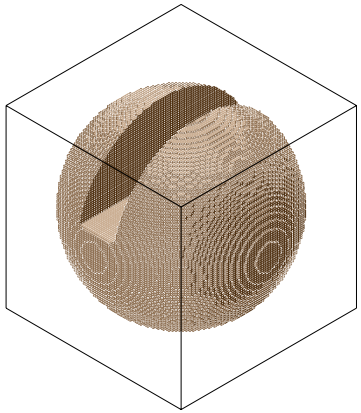




# Combined Approach Definition

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





# Using Notins or Erasing Fills?

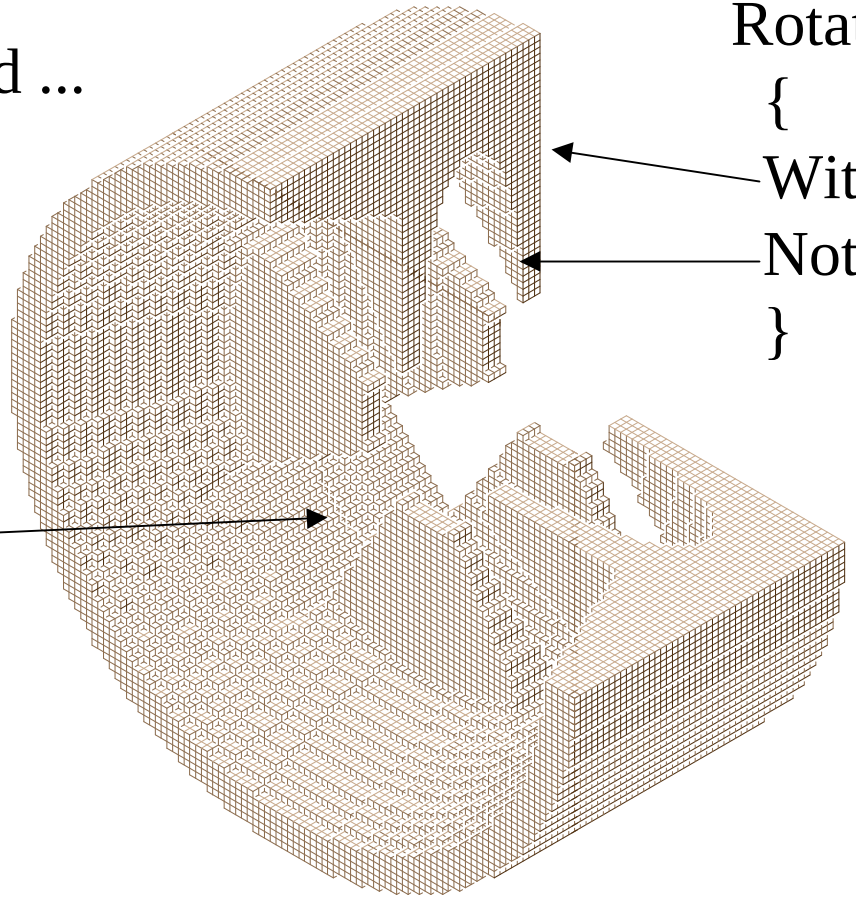
- With normal Fill commands it is best to use Notins to limit or remove material.
  - Notins can act exactly like a normal erasing fill.
  - 2D commands (e.g. circle) can be used in a Notin without fear of drilling infinite holes.

- With Rotate_Fills any non-cylindrically symmetric removal will normally required a subsequent erasing fill.
  - Erasing fills (e.g. n(0)) must be volume limited.
  - 2D commands are dangerous (unlimited volume).

# Using Notins and Erasing Fills with a Rotate_Fill

Then swing 90° az and ...

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



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

# The Role of Include Files

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

# Recommended Method for Using Include Files

Calling File:

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

Include File: draw out.gem

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

# Include File Examples