EGS5

How to Code Geometry

Writing Subroutine HOWFAR

Walter R. Nelson
Stanford Linear Accelerator Center

Introduction

- EGS5 User Codes require the user to define
  - `subroutine ausgab` for scoring results of interest
  - `subroutine howfar` to provide information about the nature of the geometry
- The most trivial `howfar` is a ***homogeneous, infinite*** medium. Namely,

```
subroutine howfar
implicit none
return
end
```

- Note: <u>always</u> put `implicit none` in your codes !!!
- The purpose of this tutorial is to show you how to write code that can be used for more complicated geometries

Useful Geometry References

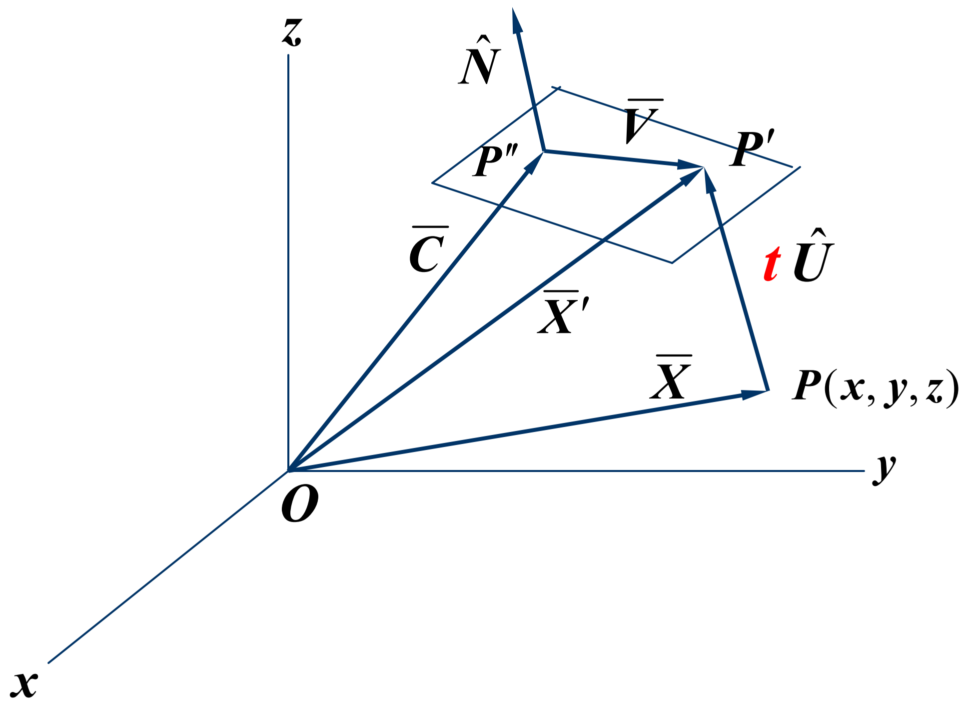
originally designed for EGS4
but…consistent with EGS5

- W. R. Nelson and T. M. Jenkins, "Geometry Methods and Packages", Chapter 17 in ***MONTE CARLO TRANSPORT OF ELECTRONS AND PHOTONS*** (Plenum Press, 1988)
- W. R. Nelson and T. M. Jenkins, "Writing SUBROUTINE HOWFAR for EGS4", SLAC-TN-87-4 (31 August 1988/Rev.)

Items Covered in this Tutorial

- General mathematical considerations (vectors)
- The role of key variables in EGS5
- Special geometry routines available with EGS5. These are Fortran files located in the directory `/egs5/auxcode/` and identified mnemonically (e.g., `plane1.f`, `cylndr.f`, etc.)
- The `common` files associated with these routines. These are Fortran files located in the directory `/egs5/auxcommons/` and also identified mnemonically (e.g., `pladta.f`, `cyldta.f`, etc.)
- Putting all the modules together to form the geometry

Mathematical Considerations



- Particle trajectories are described by the position and direction vectors

$$\overline{X} = x\hat{i} + y\hat{j} + z\hat{k} \quad \text{and} \quad \hat{U} = u\hat{i} + v\hat{j} + w\hat{k}$$

represented by `x(np)`, `y(np)`, `z(np)` and `u(np)`, `v(np)`, `w(np)`, respectively, and are passed in `common/STACK/` along with the stack pointer, `np`

Subroutine HOWFAR

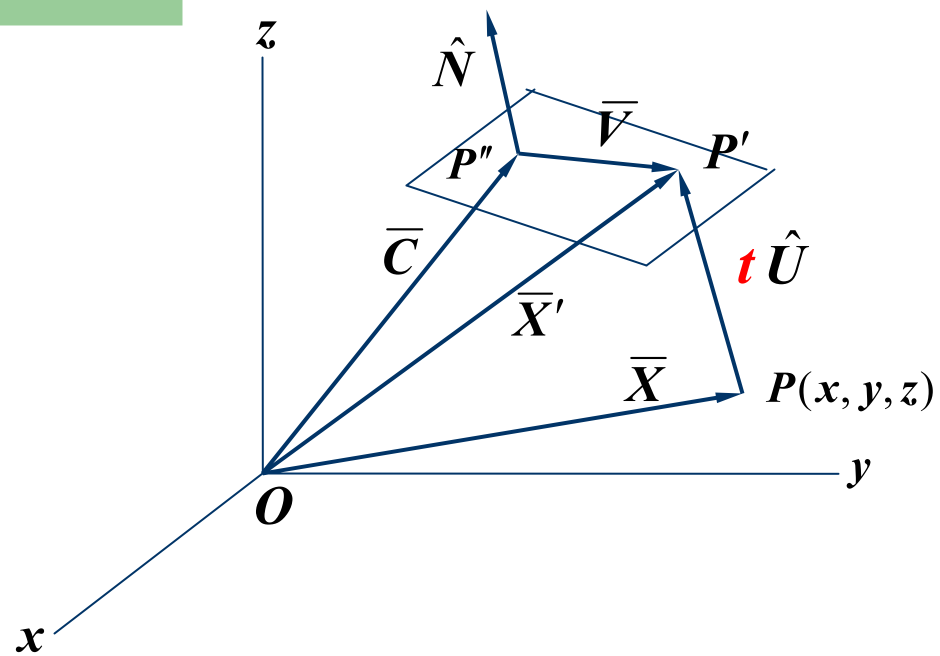
EGS5

…Math Considerations (cont.)

- The quantities $\overline{\mathbf{X}}(\mathbf{x},\mathbf{y},\mathbf{z})$ and $\hat{\mathbf{U}}(\mathbf{u},\mathbf{v},\mathbf{w})$, together with such things as particle **type**, **energy**, **weight**, ***time***, etc., define the ***state function*** of the particle (and are called ***stack variables***)
- In writing `subroutine howfar`, the problem becomes one of
  - determining the point of intersection, ***P′***, of the particle trajectory with any given surface,
  - which allows for the extraction the distance ***t*** **,**
  - and comparing ***t*** with `ustep` — the transport step about to be taken for the current particle being followed

referring to the figure again…

…Math Considerations (cont.)

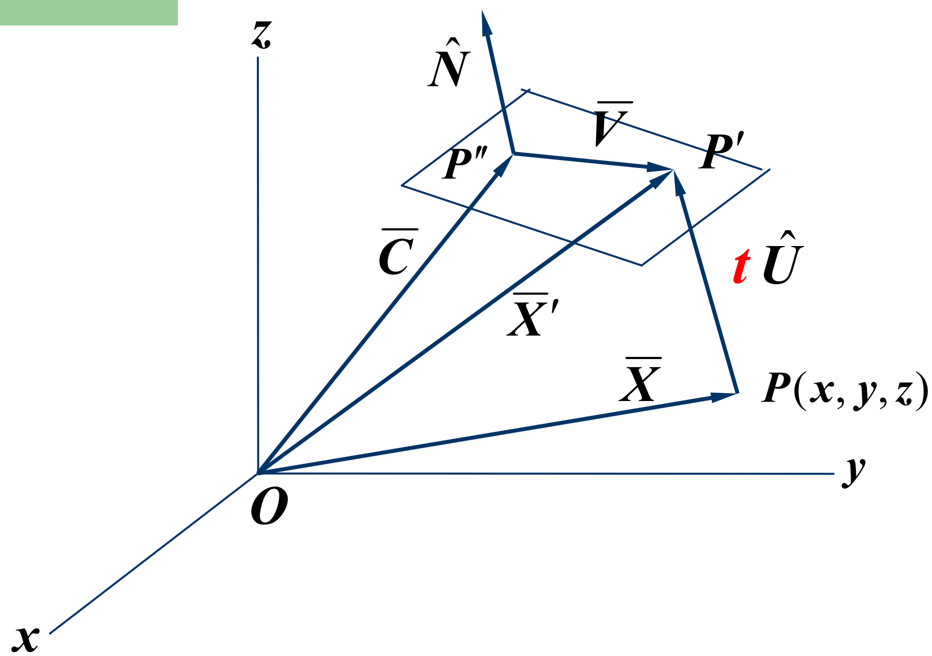


- A plane surface can be described by the vector to a point, ***P''***, on the surface and a unit vector normal to it,

$$\overline{C} = c_1\hat{i} + c_2\hat{j} + c_3\hat{k} \quad \text{and} \quad \hat{N} = n_1\hat{i} + n_2\hat{j} + n_3\hat{k}$$

which are the arrays **`pcoord(3:100)`** and **`pnorm(3:100)`**, respectively, that are passed in **`common/PLADTA/`** .

...Math Considerations (cont.)



- The condition for the intersection point (***P′***) to ***lie on the plane*** is obtained from the vector dot product $\overline{V} \cdot \hat{N} = 0$.
- From the diagram we see that

$$\overline{V} = \overline{X}' - \overline{C} = \overline{X} + t\hat{U} - \overline{C} \ ,$$

which leads to the equation…

…Math Considerations (cont.)

$$t = \frac{(\bar{C} - \bar{X}) \cdot \hat{N}}{\hat{U} \cdot \hat{N}} = \frac{(c_1 - x)n_1 + (c_2 - y)n_2 + (c_3 - z)n_3}{un_1 + vn_2 + wn_3}$$

- The following physical situations apply

  1) $t > 0$: particle travels ***away from*** the plane

  2) $t < 0$: particle travels ***towards*** the plane

- Note: The above equation is indeterminant when the denominator is 0, corresponding to the physical situation in which the particle travels ***parallel to*** the plane
- The algorithm that is used in EGS5, and which will be described next, is called **`subroutine plane1`**

subroutine plane1

subroutine plane1 (i.e., **plane1.f**) begins with an explanation header and the required declarations

```
! Input arguments:
! ----------------
!   nplan = ID number assigned to plane
!   iside =  1 normal points away from current region
!         = -1 normal points towards current region
! Output arguments:
! -----------------
!   ihit  =  1 trajectory will strike plane
!         =  2 trajectory parallel to plane
!         =  0 trajectory moving away from plane
!   tval  = distance to plane (when ihit=1)
! ----------------------------------------------------------------------
      subroutine plane1(nplan,iside,ihit,tval)
      implicit none
      include 'include/egs5_h.f'                ! Main EGS5 "header" file
      include 'include/egs5_stack.f'      ! COMMONs required by EGS5 code
      include 'auxcommons/aux_h.f'        ! Auxiliary-code "header" file
      include 'auxcommons/pladta.f'             ! Auxiliary-code COMMONs
      real*8 tval,tnum                                       ! Arguments
      integer nplan,iside,ihit
      real*8 udota,udotap                 ! Local variables
```

subroutine plane1(cont.)

The remainder of the coding is the actual algorithm:

```
                .
                .
                .
    udota = pnorm(1,nplan)*u(np) + pnorm(2,nplan)*v(np) + pnorm(3,nplan)*w(np)
    udotap = udota*iside
    if (udota .eq. 0.) then                 ! Traveling parallel to plane
      ihit = 2
    else if (udotap .lt. 0.) then             ! Traveling away from plane
     ihit = 0
    else                    ! Traveling towards plane---determine distance
      ihit = 1
    tnum = pnorm(1,nplan)*(pcoord(1,nplan) - x(np)) +
   *       pnorm(2,nplan)*(pcoord(2,nplan) - y(np)) +
   *       pnorm(3,nplan)*(pcoord(3,nplan) - z(np))
    tval = tnum/udota
    end if
    return
    end
```

Except for parameter **iside**, which allows for more efficient determination of *t*, the algorithm below is based precisely on the previous equation.

Specifications for `howfar`

- For every `call shower` invocation in the MAIN program of the User Code, particles that are being transported are placed on a ***stack***
- The ***current*** particle being tracked is identified on the ***stack*** by the pointer, `np` (in `common/STACK/`)
- There are three EGS variables that play an important role in **`howfar`**: `ustep`, `idisc`, and `irnew`
- These variables are passed in `common/EPCONT/`. The Fortran file (`epcont.f`) is located in `/egs5/include/`
- Of course, the `common` files associated with each of the geometry routines are passed in `common/PLADTA/`, etc.

… Specifications for `howfar` (cont.)

- On entry to `howfar`, EGS has predetermined that it would like to transport the current particle by a straight-line distance, `ustep`
- `howfar` must then determine if `ustep` will carry the particle past the boundary towards which it is heading
- If it does carry it past, `howfar` must do two things:
  - Shrink `ustep` to the distance to the boundary, t
  - Set `irnew` to the "new" region in which the particle is expected to end up
- Otherwise, a `return` is simply made to the subroutine that called `howfar` (i.e., `electr` or `photon`)

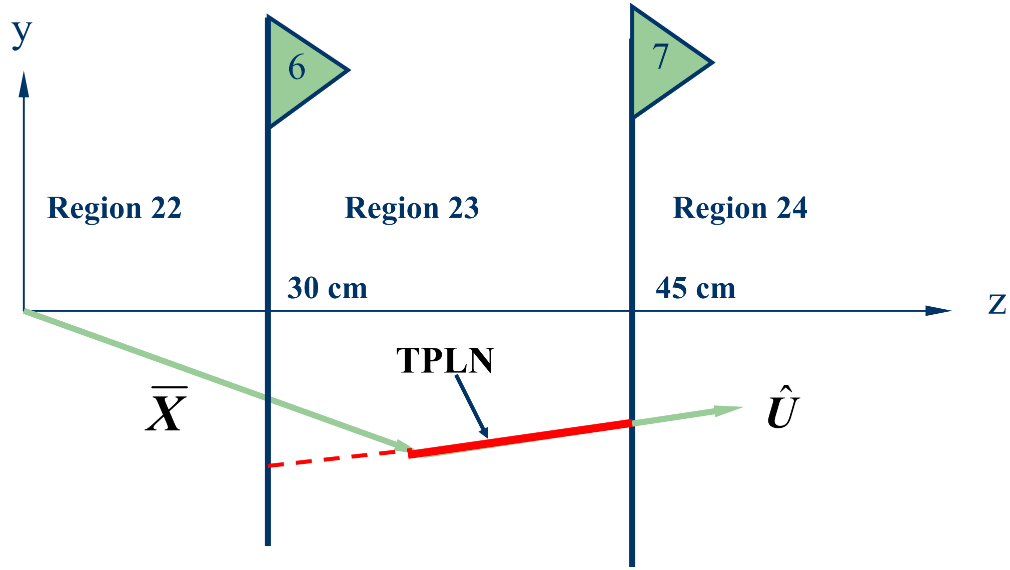
… Specifications for `howfar` (cont.)

- On occasion, a particle will end up in a region designated by the user as a *discard region*
- In such cases, the flag **`idisc`** is set equal to **unity** in **`howfar`** and a **`return`** is made to the calling subprogram
- The distance to the next boundary, t , can be determined by calling one of the following geometry subroutines:

  `plane1, plan2p, plan2x, cylndr, cyl2, sphere, sph2, cone, cone2, cone21`
- Two other geometry subprograms also are available to help in this task:
  - **`chgtr`** for changing **`ustep`** and **`irnew`** if needed
  - **`finval`** for getting the coordinates at the end of a projected transport

EGS5

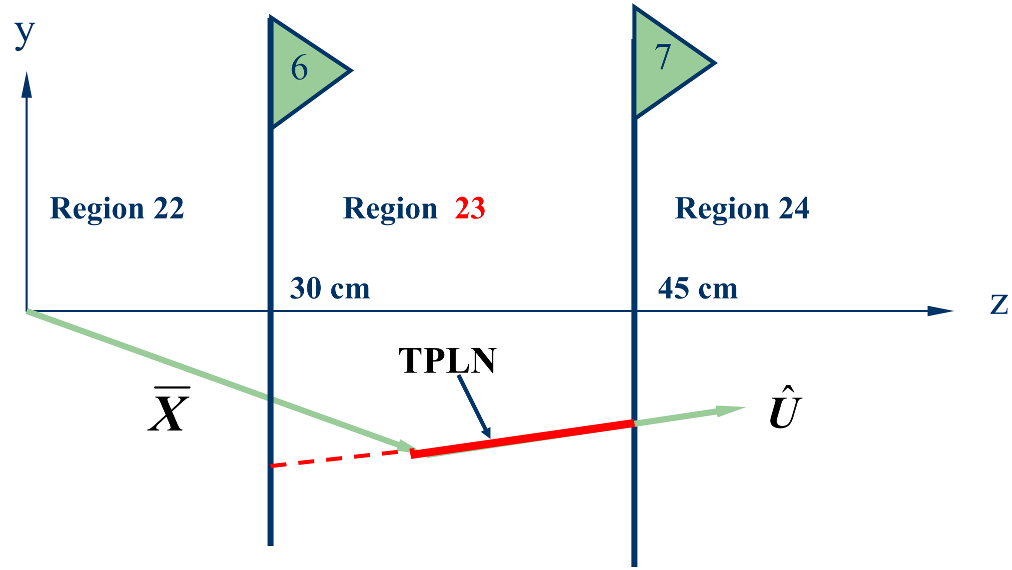
Using `plane1` and `chgtr`



- Consider the two parallel planes separating three regions
- The regions are identified by the numbers **22**, **23** and **24** and the planes by the numbers **6** and **7**
- The ***triangles*** enclosing the numbers **6** and **7** have a purpose—they point in the direction of the unit normal vectors and the user must define them with values (`pnorm`)

EGS5

… plane1 and chgtr (cont.)



- Assume that planes 6 and 7 are located at z = 30 and 45 cm, respectively

```
PCOORD(1,6)= 0.0     PCOORD(2,6)= 0.0     PCOORD(3,6)= 30.0
PNORM(1,6) = 0.0     PNORM(2,6) = 0.0     PNORM(3,6) = 1.0
PCOORD(1,7)= 0.0     PCOORD(2,7)= 0.0     PCOORD(3,7)= 45.0
PNORM(1,7) = 0.0     PNORM(2,7) = 0.0     PNORM(3,7) = 1.0
```

- If particles are initially <u>started</u> in region **23** and <u>discarded</u> when they leave this region, the following **howfar** will work nicely with EGS

EGS5

… plane1 and chgtr (cont.)

```
subroutine howfar
implicit none

include 'include/egs5_h.f'                   ! Main EGS "header" file
include 'include/egs5_epcont.f'            ! COMMONs required by EGS5
include 'include/egs5_stack.f'

real*8 tpln                                           ! Local variables
integer ihit

if (ir(np).ne.23) then
  idisc=1                    ! Discard particles outside region 23
else                            ! Track particles within region 23
  call plane1(7,1,ihit,tpln)           ! Check upstream plane first
  if (ihit.eq.1) then     ! Surface hit---make changes if necessary
    call chgtr(tpln,24)
  else if (ihit.eq.0) then                        ! Heading backwards
    call plane1(6,-1,ihit,tpln) ! ihit=1 a must ,so get tpln-value
    call chgtr(tpln,22)                   ! Make changes if necessary
  end if
end if

return
end
```

Subroutine HOWFAR

EGS5

subroutine chgtr

```
subroutine chgtr(tvalp,irnewp)
implicit none
include 'include/egs5_h.f'          ! Main EGS5 "header" file
include 'include/egs5_epcont.f'   ! COMMONs required by EGS5

real*8 tvalp                                    ! Arguments
integer irnewp

if (tvalp .le. ustep) then
  ustep = tvalp
  irnew = irnewp
end if

return
end
```

In the above, **subroutine chgtr** does the following:

- If **tvalp.le.ustep** → **ustep=tvalp** and **irnew=irnewp=24** (or **22**)
- Otherwise nothing is done

Also, **egs5_epcont.f** makes **ustep** and **irnew** available and **egs5_h.f** provides **MXAUS** (required by **egs5_epcont.f**)

subroutine plan2p*

(* Mnemonic for *two parallel* planes)

The **howfar** example that we have been following can be simplified even further with the aid of **subroutine plan2p**

```
subroutine howfar
implicit none

include 'include/egs5_h.f'                ! Main EGS5 "header" file
include 'include/egs5_epcont.f'          ! COMMONs required by EGS5
include 'include/egs5_stack.f'

if (ir(np).ne.23) then
  idisc=1                   ! Discard particles outside region 23
else                           ! Track particles within region 23
  call plan2p(7,24,1,6,22,-1)
end if

return
end
```

… `plan2p`(cont)

- First group of numbers **`(7,24,1)`** corresponds to checking the ***downstream*** plane and is equivalent to **`call plane1(7,1,ihit,tpln)`** followed by **`call chgtr(tpln,24)`**
- Second group **`(6,22,-1)`** corresponds to checking the ***upstream*** plane and is equivalent to **`call plane1(6,-1,ihit,tpln)`** followed by **`call chgtr(tpln,22)`**
- **`plan2p`** is efficient in that the second plane is only checked if necessary; namely, if the particle is really heading towards it (i.e., it makes sense to query the ***downstream*** plane first because that's the direction of the radiation "flow")

Multislab Example

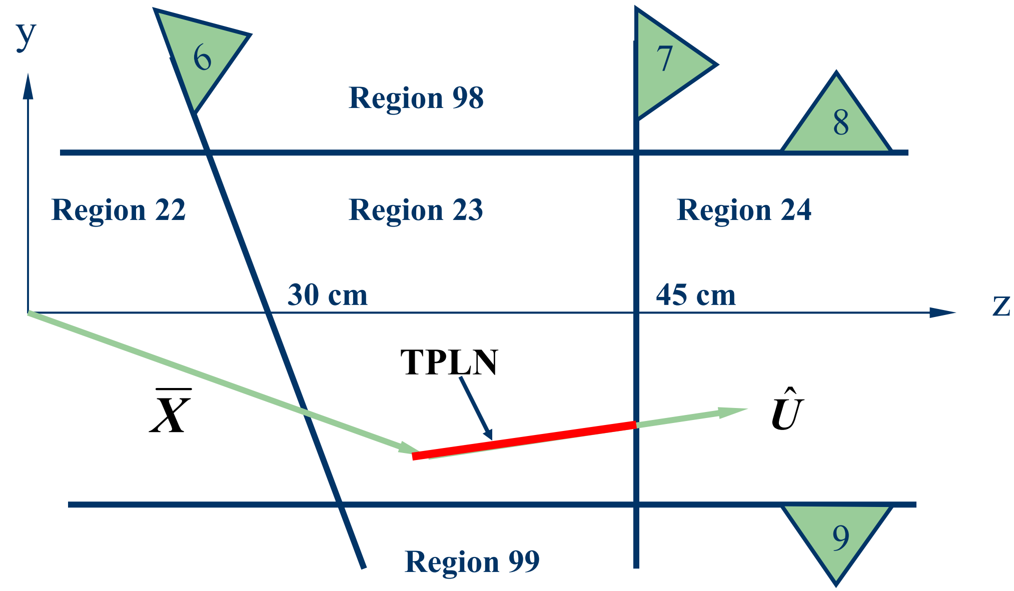
It is very simple to extend the previous howfar for many slabs

```
!---------------------------------------
! Multislab (NREG-1) shower calorimeter
!---------------------------------------
subroutine howfar
implicit none
include 'include/egs5_h.f'            ! Main EGS5 "header" file
include 'include/egs5_epcont.f'       ! COMMONs required by EGS5
include 'include/egs5_misc.f'                ! Required for nreg
include 'include/egs5_stack.f'

integer irl

irl=ir(np)                            ! Create a local variable

if (irl.eq.1 .or. irl.eq.nreg) then
  idisc=1         ! Discard in the upstream & downstream regions
else                  !Track particles within calorimeter proper

  call plan2p(irl,irl+1,1,irl-1,irl-1,-1)
end if

return
end
```

subroutine plan2x*

(* Mnemonic for *two crossing* planes)

- Consider the geometry below consisting of five regions formed by a pair of *parallel* and a pair of *crossing* planes:



- We will impose the conditions that all particles start out in region 23, but are ***discarded*** when they leave it

…the following **`howfar`** can be written

... plan2x (cont)

```
subroutine howfar

implicit none

include 'include/egs5_h.f'              ! Main EGS5 "header" file
include 'include/egs5_epcont.f'         ! COMMONs required by EGS5
include 'include/egs5_stack.f'

if (ir(np).ne.23) then
  idisc=1                    ! Discard particles outside region 23
else                           ! Track particles within region 23
  call plan2x(7,24,1,6,22,-1)
  call plan2p(8,98,1,9,99,1)
end if
return
end
```

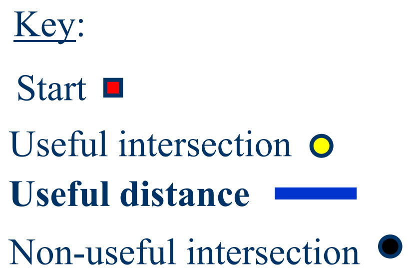
… `plan2x` (cont)

It is instructive to convince oneself that the following statements are true:

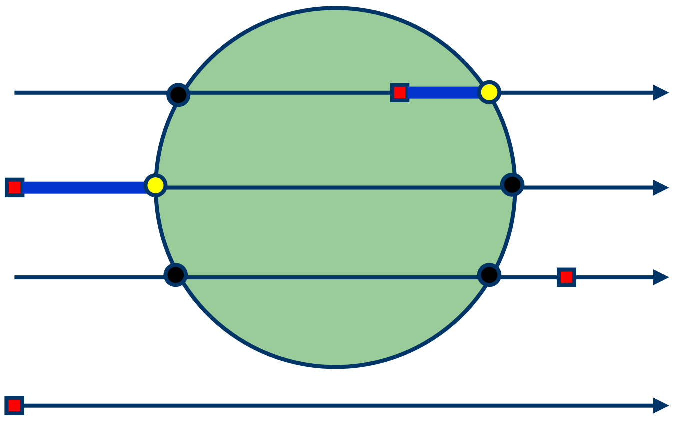
- **`plan2x`** and **`plan2p`** must both be called
- The order in which they are called is not important
- Pre-knowledge of the direction of radiation flow can increase the efficiency with **`plan2p`**---i.e., it makes sense to call the *preferred plane* first
- But with **`plan2x`** there is no preferred calling order
- **`ustep`** and **`irnew`** will always be properly selected

cylndr, cone and sphere

- The conic surface algorithms are basically all the same and **cone** and **sphere** may be used in **subroutine howfar** in the same manner as **cylndr**. Therefore, only **cylndr** will be described here.
- We will skip the math here, but the intersection of a vector with a conic surface leads to a quadratic equation, the solutions of which are both real and imaginary and correspond to actual ***physical*** solutions
- The following figure shows possible trajectories intersecting a cylinder



* Currently defined along the z-axis only



subroutine cylndr

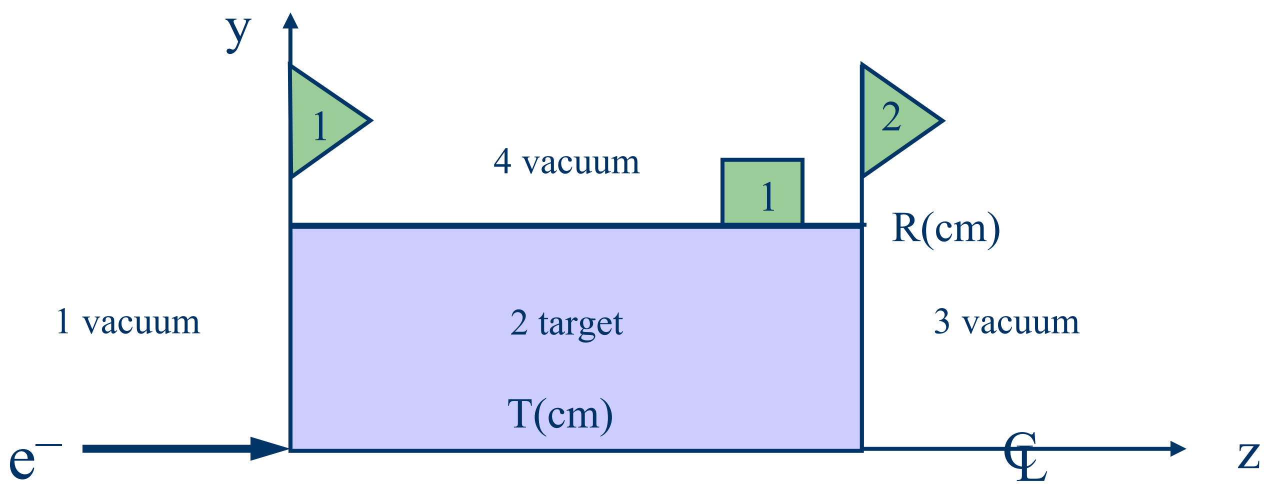
- The algorithm in `cylndr` was designed to take all the trajectory possibilities into account
- To accomplish the task, the user must determine whether the current particle location is ***inside*** or ***outside*** of the cylinder
- For `call cylndr(icyl,infl,ihit,tcyl)` the parameters are explained as follows

```
Input:   icyl       Cylinder identification number
         infl = 1   Current particle position is INSIDE cylinder
              = 0   Current particle position is OUTSIDE cylinder
Output:  ihit = 1   Particle trajectory HITS surface
              = 0   Particle trajectory MISSES surface
         tcyl       Distance (shortest) to surface (when IHIT=1)
```

- `tcyl` is the ***useful distance*** shown by the blue line segment in the previous slide

Cylinder-Slab Example

- Consider a cylindrical target struck by an electron beam



  - The cylinder of rotation about the z-axis is identified by box 1 and radius R
  - Planes 1 and 2 define the length of the target of thickness T
  - There are four regions of interest: the target (region 2) and three vacuum regions– upstream (region 1), downstream (region 3) and surrounding the target (region 4)
- The radius (squared) of the cylinder is defined in the `main` program of the User Code and passed in `common/CYLDTA`/

…Cylinder-Slab Example (cont.)

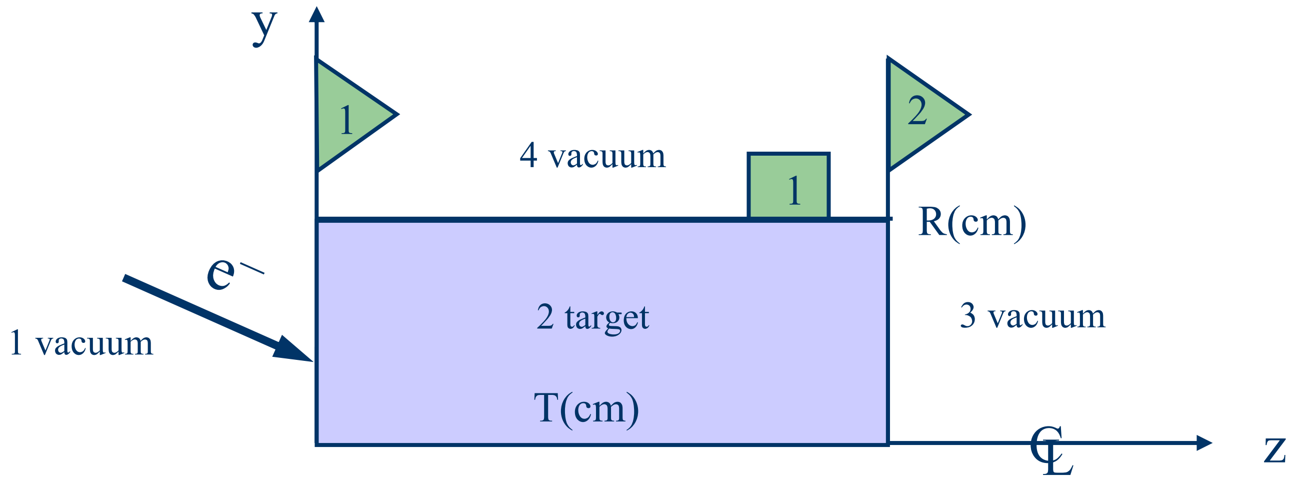
The following will work for the above geometry:

```
subroutine howfar
implicit none

include 'include/egs5_h.f'                ! Main EGS5 "header" file
include 'include/egs5_epcont.f'          ! COMMONs required by EGS5
include 'include/egs5_stack.f'

real*8 tcyl
integer ihit

if(ir(np).eq.2) then            ! Track particles within the target
  call cylndr(1,1,ihit,tcyl)           ! Check the cylinder surface
  if(ihit.eq.1) then
    call chgtr(tcyl,4)                       ! Change if necessary
  end if
  call plan2p(2,3,1,1,1,-1) ! Check the downstream plane first and
                            !   then the upstream one if necessary
else
  idisc=1                   ! Discard particles outside the target
end if

return
end
```

Subroutine HOWFAR

subroutine finval

Subroutine finval is useful for determining the final coordinates of a particle---for example, the coordinates at the ***point of intersection*** of a trajectory and a geometric surface. To illustrate this, consider the cylinder-slab geometry of the previous example:



Assume a *beam* is to the left in region 1 and particles are to be transported to the target. Which region do we enter, 2 or 4?

We have modified the previous cylinder-slab example to demonstrate how to handle this using **finval**.

… finval (cont)

```
subroutine howfar
implicit none
include 'include/egs5_h.f'                    ! Main EGS5 "header" file
include 'include/egs5_epcont.f'              ! COMMONs required by EGS5
include 'include/egs5_stack.f'
include 'auxcommons/aux_h.f'                      ! Required for MXCYLS
include 'auxcommons/cyldta.f'                     ! Required for cyrad2
real*8 tcyl,tpln,xf,yf,zf
integer ihit,irnxt
if(ir(np).eq.1 .and. w(np).gt.0.) then
  call plane1(1,1,ihit,tpln)
  if(ihit.eq.1) then
    call finval(tpln,xf,yf,zf)              ! Get final coordinates
      if(xf*xf + yf*yf.lt.cyrad2(1)) then
        irnxt=2
      else
        irnxt=4
      end if
    call chgtr(tpln,irnxt)
  end if
else if(ir(np).eq.2) then      ! Track particles within the target
  call cylndr(1,1,ihit,tcyl)           ! Check the cylinder surface
  if(ihit.eq.1) then
    call chgtr(tcyl,4)                            ! Change if necessary
  end if
  call plan2p(2,3,1,1,1,-1)      ! Check the downstream plane first
else
  idisc=1                    ! Discard particles outside the target
end if
return
end
```

Subroutine HOWFAR

Summary

- A set of geometry routines is available in the EGS5 distribution to aid in defining `subroutine howfar`
- This tutorial demonstrates how one uses these routines to create a relatively simple geometry
- The subroutines can also be used in a modular way to define very complex geometries
- Although originally written for the EGS4 Code System, which was coded in the Mortran3 language, the references at the beginning of this tutorial provide even more complex examples that can be used with EGS5